Guía del Programador Competitivo

Programación competitiva para todos.

Diego Fernando Rodriguez Castañeda

ISSN 978-958-8817-48-4

ISSN-e 978-958-8817-44-6

Editorial Universidad ECCI

Edición 1

2021

Bogotá, Colombia.

1. Agradecimientos

Luego de todos los años que llevo entrenando, aprendiendo, compitiendo y enseñando todo lo referente a la maravillosa área de la programación competitiva, he compartido muchos momentos con muchas personas, a todas ellas les debo un pedacito de toda la experiencia que tengo.

A mi padre, su compañera y mi familia Rodriguez, les agradezco toda la paciencia que me han tenido, toda la comprensión que me han demostrado, además su apoyo moral me ha permitido llegar hasta este punto, y más lejos aún en el futuro, también a Camilo y Daniel, mis copilotos en la vida.

Los ingenieros Alexander Sabogal, Mónica Barrios, Ana Rocio León, Efraín Ruiz, Fabio Vargas, German Salas, Fabio Hurtado, y Cesar Torres, todos ellos maestros y administrativos de la Universidad ECCI, les agradezco todas las enseñanzas, regaños, consejos y ayuda dentro del desarrollo de las diversas actividades en las cuales hemos participado, y todo el conocimiento que he adquirido durante las clases que ellos me han dictado.

Al grupo de programación competitiva de la Universidad ECCI, mis muchachos y estudiantes, los cuales he tenido el placer de poder entrenar dentro del arte de la programación competitiva, además de que es un orgullo verlos crecer, ser cada día mejores personas y profesionales.

A Aiden, una mujer maravillosa que quiero mucho, me apoya y cuida a diario, a ella le debo gran parte de la ejecución de este libro, y a su apoyo en la realización de las ilustraciones.

Les agradezco a todos, este logro es gracias a todos ustedes, dedico este libro a ustedes.

01110000011100100110000101101001011100110110010101101011011001010110101

2. Tabla de contenido

Capitulo 1. Introducción

Las competencias de programación a nivel mundial han demostrado ser una valiosa herramienta para el aprendizaje y desarrollo de habilidades relacionadas a la programación en función de resolución de problemas, muchas de estas competencias emplean diversos aplicativos webs para el entrenamiento y evaluación de los estudiantes.

Es un concurso de programación algorítmica para estudiantes universitarios. Equipos de uno a tres integrantes, que representan a su universidad, trabajan para resolver los problemas de la vida real, fomentando la colaboración, la creatividad, la innovación y la capacidad de actuar bajo presión. A través del entrenamiento y la competencia, los equipos se desafían unos a otros para elevar la presión y la rapidez con el cual realizan los problemas.

Grandes empresas de talla internacional utilizan estas competencias como principal medio de reclutamiento de programadores, con el fin de engrosar sus filas de profesionales dispuestos a trabajar con ellos, abriendo también espacios de pasantías y de nutrición de hoja de vida, estas oportunidades son de utilidad tanto para la empresa como para el programador, debido a que les otorga prestigio y reconocimiento en el mercado laboral.

Capitulo 2. Teoría inicial

2.1) Reglas básicas

Durante las competencias, dependiendo de la modalidad de la misma, existen ciertas reglas que se deben cumplir para mantener la igualdad entre los competidores y evitar cualquier tipo de trampas.

Competencias presenciales:

- Durante las competencias está prohibido el uso de elementos electrónicos tales como celulares, dispositivos de almacenamiento extraíbles, reproductores de música, auriculares, entre otros. De ser encontrado con alguno de estos, el participante será retirado del recinto y perderá el derecho a continuar en la competencia por intento de fraude.
- El ingreso a páginas web distintas a la plataforma oficial de la competencia será penalizado con la descalificación inmediata de la competencia.
- El consumo de bebidas y alimentos dentro de las aulas no está permitido.
- Durante la competencia los equipos no podrán hablar ni intercambiar material entre ellos. De ser sorprendidos por primera vez incumpliendo esta normal, se les hará una advertencia pero si ya es la segunda vez serán descalificados de la competencia.
- La copia de código de fuentes externas como páginas web o redes sociales se considera plagio.
- Está permitido el uso de material como libros de texto, códigos impresos, textos matemáticos, diccionarios, entre otros.
- Cada equipo tendrá a su disposición una sola máquina durante la competencia.
- En la competencia los equipos serán conformados por máximo tres competidores.

Competencias virtuales:

- Durante la competencia los equipos no podrán hablar ni intercambiar material entre ellos. De ser sorprendidos por primera vez incumpliendo esta normal, se les hará una advertencia pero si ya es la segunda vez serán descalificados de la competencia.

- En la competencia los equipos serán conformados por máximo tres competidores.
- La copia de código de fuentes externas como páginas web o redes sociales se considera plagio.
- No se admite cualquier tipo de ataques informáticos que intenten vulnerar la integridad de los sistemas usados en las competencias.

Durante la mayoría de las competencias, la forma de entrada y salida de información de cada ejercicio se realiza por medio de la consola (STDIN/STDOUT), aunque algunas competencias si utilizan entornos gráficos y entornos web (Uva Online Judge, 2012).

2.2) ¿Qué es la programación competitiva?

Dentro de las iniciativas en el área de las ciencias de la computación, un tema que año con año cobra más auge es la programación competitiva; el cual se evidencia en torneos en los cuales varios grupos de personas intentan resolver un problema a través de código y algoritmos.

El trabajo en equipo, pensamiento crítico y actuar bajo presión, además de otras actividades, son desarrolladas e impulsadas a través de estas competencias. Se trata de un deporte mental en el que participan e impulsan compañías multinacionales de software como Google, IBM y Facebook.

El desarrollo de las habilidades de programación de algoritmos se potencia a través de la colaboración, es por ello que colegios y universidades de todo el mundo buscan que los alumnos participen en este tipo de torneos.

A su vez, el impulso de talento es crucial para actividades cada vez más demandantes y demandadas, como es la programación. De ahí que empresas de gran relevancia nacional e internacional este interesadas en egresados de carreras relacionadas.

2.3) ¿Qué es un problema de programación competitiva?

Un problema de programación competitiva es un pequeño ejercicio de programación que puede abordar cualquier tema de la vida cotidiana, ciencias puras como la matemática o la física, un juego de mesa como el ajedrez o un rompecabezas.
Utilizando la programación se pueden resolver estos ejercicios y buscar una solución óptima al mismo utilizando matemática, algoritmos matemáticos especializados o incluso lógica básica.

2.4) Estructura de un problema de programación competitiva

Un problema se encuentra conformado de 4 partes principales:

1. Título: Contiene el nombre del problema, comúnmente da una vaga aproximación al tema del problema
2. Descripción: Contiene toda la información del problema, puede ser una historia, un evento, algo que le pasando a alguien y quiere que usted lo ayude a resolverlo. Puede contener algunas pistas de cómo resolver el ejercicio, variables que se pueden usar y de qué tipo pueden ser.
3. Variables y restricciones: muestra el o los tipos de datos que entraran y el o los tipos de datos que deben salir, además del tamaño que puede tener la entrada.
4. Ejemplo de entrada y salida: Muestra uno de todos los posibles casos de prueba que pueden evaluarse en el ejercicio, tener en cuenta que los ejemplos en los textos de ejercicio nunca serán los únicos, se debe usar la parte de variables para calcular cual será el caso más fácil, un caso intermedio y el caso más difícil.

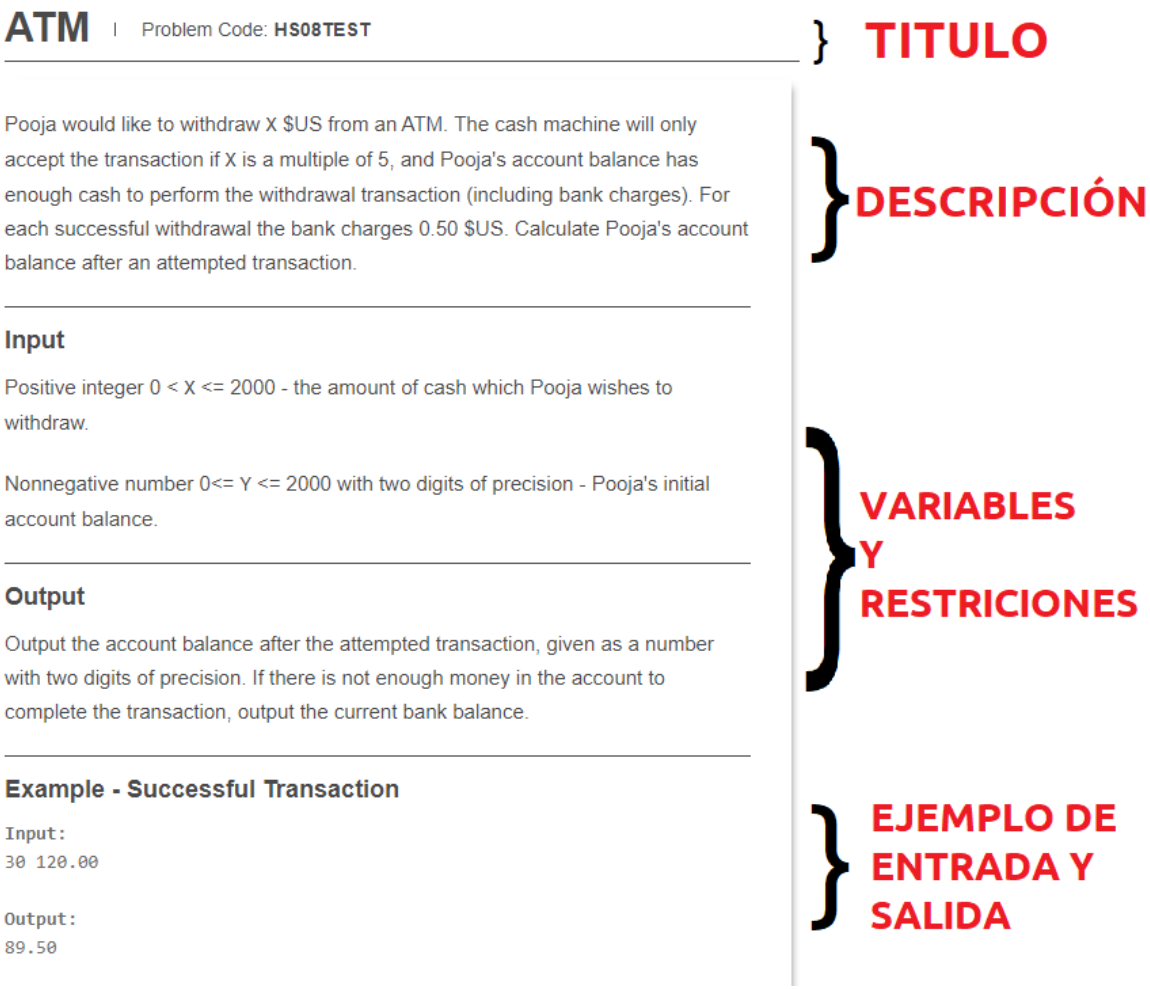


Ilustración 2-1: Partes principales de un problema de programación competitiva

2.5) Como abordar un problema

Comúnmente todos los ejercicios tienen dos o más formas de ser solucionados, utilizando diversos tipos de algoritmos o metodologías matemáticas de resolución de problemas, a continuación nombraremos algunas de las formas existentes para abordar un problema:

- Fórmulas matemáticas
- Fuerza bruta
- Aproximaciones de raíces

- Procesamiento de Strings
- Búsqueda de patrones
- Uso de estructuras de datos especializadas
- Algoritmos prediseñados modificados
- Enfoque geométrico
- Algoritmos ingenuos
- Uso de estructuras geometrías avanzadas
- Búsquedas
- Ordenamientos

Al leer toda la información de nuestro ejercicio, su deber como es identificar cual es la metodología más adecuada para la resolución de estos ejercicios, podemos apoyarnos de material previamente consultado. Entre cada metodología de resolución varia el uso de memoria, tiempo de ejecución y forma de entrada de la información a procesar, por lo que conocer los puntos fuertes y débiles de cada uno es crucial.

Por lo general las metodologías matemáticas son las más rápidas pero también suelen ser las más complicadas de implementar, ya que se basan en la búsqueda de sistemas de ecuaciones o el uso de teoremas matemáticos que son complicados y abstractos de entender, las metodologías de fuerza bruta son las más lentas pero las más seguras a la hora de encontrar una solución a un problema debido a que usan gran cantidad de recursos para asegurarse de dar con la respuesta, además de obtener las demás respuestas posibles a un problema cuando sus variables cambian.

Los desarrolladores y estudiantes resuelven muchas preguntas de codificación de estructuras de datos y algoritmos, pero la mayoría de ellos no comprende la importancia de esto. Muchos de ellos también tienen la opinión de que las estructuras de datos y los algoritmos solo ayudan en las entrevistas y después de eso, no hay uso de todas esas cosas complicadas en los trabajos diarios.

Es importante estar contento con aprender un nuevo lenguaje o framework y crear algunas aplicaciones con eso, pero una vez que ingrese a la industria del mundo real, se dará cuenta

de que su trabajo no es solo escribir el código y hacer que las cosas funcionen. Su verdadero trabajo es escribir la cantidad correcta de buen código, lo que significa que debe ser eficiente y robusto, y aquí viene el papel de las estructuras de datos y los algoritmos. Las estructuras de datos y los algoritmos no solo ayudan a obtener la lógica de su programa, sino que también ayudan a escribir el código eficiente para su software. Ya sea que hablemos sobre la complejidad del tiempo o la administración de la memoria, la refactorización del código o la reutilización del código, comprenderá su valor en cada parte de su aplicación.

En proyectos del mundo real, su cerebro debería ser capaz de escribir una solución rápida y eficiente para desafíos complicados, y solo puede hacerlo cuando haya practicado mucha programación. Comprenda que el lenguaje y los frameworks son solo herramientas, no le enseñarán habilidades para resolver problemas. Se desarrolla habilidad para resolver problemas cuando se practica mucha programación.

Cada desarrollador tiene sus propios trucos y siguen su propio patrón para resolver problemas de codificación, pero cuando se trata de nuevos desarrolladores, siempre tienen dudas sobre dónde comenzar. Muchos de ellos entienden los problemas, la lógica y los conceptos básicos de la sintaxis, también entienden los códigos de otras personas y pueden seguirlos, pero cuando se trata de resolver las preguntas por su cuenta, se atascan. No entienden cómo convertir sus pensamientos en código a pesar de que entienden la sintaxis o la lógica. Estos son algunos pasos simples que lo ayudarán a abordar una pregunta de codificación dentro de una competencia o dentro de una entrevista de trabajo.

Comprender y analizar el problema

No importa si ha visto el ejercicio que está realizando en el pasado o no, lea la pregunta varias veces y comprenda por completo. Ahora, piense en la pregunta y analícela cuidadosamente. A veces leemos algunas líneas y asumimos el resto de las cosas por nuestra cuenta, pero un ligero cambio en su pregunta puede cambiar muchas cosas en su código, así que tenga cuidado al respecto. Ahora tome un papel y escriba todo. ¿Qué se proporciona (entrada) y qué necesita averiguar (salida)?

Mientras se debe realizar las siguientes preguntas...

1. ¿Entendió el problema completamente?
2. ¿Sería capaz de explicar esta pregunta a otra persona?
3. ¿Qué y cuántas entradas se requieren?
4. ¿Cuál sería la salida para esas entradas?
5. ¿Necesita separar algunos módulos o partes del problema?
6. ¿Tiene suficiente información para resolver esa pregunta? De lo contrario, lea nuevamente la pregunta.

Por ejemplo: si se le da un vector y necesita devolver el vector que contiene solo números pares, primero analice el problema cuidadosamente. Mientras analiza el problema, debe hacerse algunas preguntas antes de saltar a la solución.

1. ¿Cómo identificar un número par? Divida ese número entre 2 y vea si su residuo es 0.
2. ¿Qué debo pasar a esta función? Un vector
3. ¿Qué contendrá ese vector? Uno o más números
4. ¿Cuáles son los tipos de datos de los elementos en el vector? Números
5. ¿Cuál es el objetivo final? El objetivo es devolver el vector de números pares. Si no hay números pares, devuelve un vector vacío.

Revise los datos de muestra y los ejemplos a fondo

Cuando intente comprender el problema, tome algunas entradas de muestra e intente analizar la salida. Tomar algunas entradas de muestra lo ayudará a comprender el problema de una mejor manera. También obtendrá la claridad de cuántos casos puede manejar su código y cuáles pueden ser la salida o el rango de salida posibles.

- Considere algunas entradas o datos simples y analice la salida.
- Considere algunos aportes complejos y más grandes e identifique cuál será el resultado y cuántos casos debe tomar para resolver el problema.

- Considere también los casos extremos. Analice cuál sería la salida si no hay entrada o si proporciona alguna entrada no válida.

Romper el problema

Cuando vea un problema de programación que es complejo o grande, en lugar de tener miedo y confundirse acerca de cómo resolver esa pregunta, divida el problema en fragmentos más pequeños y luego intente resolver cada parte del problema. A continuación se detallan algunos pasos que debe seguir para resolver las complejas preguntas de codificación:

- Haga un diagrama de flujo para el problema en cuestión.
- Divida el problema en subproblemas o fragmentos más pequeños.
- Resuelva los subproblemas. Haga funciones independientes para cada subproblema.
- Conecte las soluciones de cada subproblema llamándolos en el orden requerido o según sea necesario.
- Donde sea necesario, use clases y objetos

Escribir pseudocódigo

Antes de saltar a la solución, siempre es bueno escribir un pseudocódigo para su problema. Básicamente, el pseudocódigo define la estructura de su código y le ayudará a escribir cada línea de código que necesite para resolver el problema. Leer el pseudocódigo da una idea clara de lo que su código debe hacer. Muchas personas o programadores experimentados omiten este paso, pero cuando escribe pseudocódigo, el proceso de escribir el código final se vuelve más fácil para usted. Al final, solo tendrá que traducir cada línea de pseudocódigo en código real. Así que escriba cada paso y lógica en su pseudocódigo.

Reemplazar pseudocódigo con código real

Una vez que haya escrito el pseudocódigo, es hora de traducirlo al código real. Reemplace cada línea de su pseudocódigo en código real en el idioma en el que está trabajando. Si ha dividido su problema en subproblemas, anote el código de cada subproblema. Mientras escribe el código, tenga en cuenta tres cosas:

- El punto donde comenzó
- ¿Dónde está ahora mismo?
- ¿Cuál es su destino (resultado final)?

No olvide probar su código con conjuntos de datos de muestra (paso 2) para verificar si la salida real es igual a la salida esperada. Una vez que haya terminado con la codificación, puede deshacerse del pseudocódigo

Cuando se encuentre dentro de una entrevista de trabajo, escriba el código, vaya diciéndole al entrevistador cómo está abordando el problema.

- Dígale al entrevistador cómo está tratando de comenzar.
- Cuéntele al entrevistador acerca de su enfoque para resolver el problema.
- Discuta con el entrevistador sobre la parte más difícil que enfrenta en su problema.
- Informe al entrevistador sobre el enfoque para resolver cada subproblema para obtener el resultado final.
- Discuta los datos de la muestra o los casos de prueba con el entrevistador.
- Discuta sobre la mejor solución con el entrevistador.

Simplifique y Optimice su Código

Intenta siempre mejorar su código. Mire hacia atrás, analícelo una vez más e intente encontrar una solución mejor o alternativa. Se ha mencionado anteriormente que siempre debe intentar escribir la cantidad correcta de código correcto, así que siempre busque la solución alternativa que sea más eficiente que la anterior. Escribir la solución correcta a su

problema no es lo último que debe hacer. Explore el problema completamente con todas las soluciones posibles y luego escriba la solución más eficiente u optimizada para su código. Entonces, una vez que haya terminado de escribir la solución para su código, hay algunas preguntas que debe hacerse.

- ¿Se ejecuta este código para cada entrada posible, incluidos los casos límite?
- ¿Existe una solución alternativa para el mismo problema?
- ¿Es eficiente el código? ¿Puede ser más eficiente o se puede mejorar el rendimiento?
- ¿De qué otra forma puede hacer que el código sea más legible?
- ¿Hay más pasos o funciones adicionales que pueda realizar?

2.6) Códigos dentro del libro

Los códigos y algoritmos que se encuentran dentro de este documento han sido escritos luego de ser investigados en diferentes textos matemáticos y páginas web especializadas en programación competitiva como por ejemplo GeeksForGeeks®, LetCode® y Algorithmist®, los cuales sus códigos son de uso público y no contienen derechos de autor, han sido probados en distintos entornos de desarrollo, también en máquinas con distintas capacidades de procesamiento, cada código contiene comentarios sobre el uso o funcionalidad de cada sentencia, con el fin de que el usuario comprenda el fin de estas.
Cada código es la aproximación más cercana de cada algoritmo, y puede tener formas de ser optimizado, por lo que su efectividad no es del 100% para todos los casos, así que siempre se debe tener en cuenta que existe un margen de error, y debemos buscar medios para mitigarlos, ese es el fin de las competencias de programación, probar nuestra capacidad de resolución de todo tipo de problemas, ver con qué medios nos enfrentamos a ellos.

2.7) Manejo de archivos de código fuente

El nombre de los archivos fuente es Main, teniendo en cuenta la extensión de cada lenguaje. Cuando se disponga a enviar una solución, solo debe enviar el archivo fuente del ejercicio, no todo el proyecto.

Lenguaje	Nombre del archivo
JAVA	Main.java
PYTHON	Main.py
C++	Main.cpp
NOTA: En el caso específico de JAVA no se usa Package, coloque su archivo fuente en el default package	
NOTA: En algunas competencias, el nombre del archivo fuente tiene que ser como las instrucciones de la misma diga. Ejemplo**: A.java, problema1.cpp o sol.py**	

Tabla 2-1 Ejemplo de nombres de archivo fuente validos

2.8) Casos de prueba y tipos de casos de prueba

Existen 4 tipos de casos de prueba:

Tipo de caso	Descripción	Ejemplo
Caso único	Solo se utilizara una entrada por problema	N == 1
Caso bandera	El ejercicio seguirá ejecutándose hasta que una condición se cumpla	Mientras N sea diferente de 0

Caso fijo	Hay un número limitado de casos y el programa se ejecutara hasta que se cumpla este número	Mientras N sea menor a 10 Acumulando 1 por ciclo
Caso ilimitado o fin de archivo	El ejercicio seguirá leyendo y procesando entradas hasta que se encuentre el banderín EOF, se considera ese tipo como ilimitado.	Mientras siga habiendo entradas

Tabla 2-2: Tipos de casos de prueba

Los ejercicios tienen un tiempo límite de ejecución, si al enviar el ejercicio al juzgador la respuesta es TIME LIMIT, su algoritmo no es lo suficientemente rápido para resolver el problema, intente reducir ciclos, mejorar la entrada de datos o usar algoritmos mejor optimizados.

2.9) Terminología básica

Array: Un arreglo es un conjunto de datos o una estructura de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria RAM, sirve para almacenar datos en forma temporal.

Set: Un set o conjunto es una colección (contenedor) de ciertos valores, sin ningún orden concreto ni valores repetidos. Su correspondencia en las matemáticas sería el conjunto finito.

Variable: Una variable está formada por un espacio en el sistema de almacenaje comúnmente en la memoria principal de un ordenador y un nombre simbólico el cual sirve identificador que está asociado a dicho espacio. Ese espacio contiene una cantidad de información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa.

Número: Un número, es una abstracción que representa una cantidad o una magnitud. En matemáticas un número puede representar una cantidad métrica o más generalmente un elemento de un sistema numérico o un número ordinal que representará una posición dentro de un orden de una serie determinada.

Algoritmo: Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.

Programación: La programación es un proceso que se utiliza para idear y ordenar las acciones que se realizarán en el marco de un proyecto; al anuncio de las partes que componen un acto o espectáculo; a la preparación de máquinas para que cumplan con una cierta tarea en un momento determinado; a la elaboración de programas para la resolución de problemas mediante ordenadores, y a la preparación de los datos necesarios para obtener una solución de un problema

Tiempo de ejecución: Se denomina tiempo de ejecución (runtime en inglés) al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo. Este tiempo se inicia con la puesta en memoria principal del programa, por lo que el sistema operativo comienza a ejecutar sus instrucciones. El intervalo finaliza en el momento en que éste envía al sistema operativo la señal de terminación, sea ésta una terminación normal, en que el programa tuvo la posibilidad de concluir sus instrucciones satisfactoriamente, o una terminación anormal, en el que el programa produjo algún error y el sistema debió forzar su finalización.

Memoria: La memoria es el dispositivo que retiene, memoriza o almacena datos informáticos durante algún período de tiempo. La memoria proporciona una de las principales funciones de la computación moderna: el almacenamiento de información y conocimiento. Es uno de los componentes fundamentales de la computadora, que interconectados a la unidad central de procesamiento y los dispositivos de entrada/salida.

Compilador: Un compilador es un tipo de traductor que transforma un programa entero de un lenguaje de programación (llamado código fuente) a otro. Usualmente el lenguaje objetivo es código máquina, aunque también puede ser traducido a un código intermedio (bytecode) o a texto. A diferencia de los intérpretes, los compiladores reúnen diversos

elementos o fragmentos en una misma unidad (un programa ejecutable o una librería), que puede ser almacenada y reutilizada. Este proceso de traducción se conoce como compilación.

Interprete: Intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

Fuerza bruta: La búsqueda por fuerza bruta, búsqueda combinatoria, búsqueda exhaustiva o simplemente fuerza bruta, es una técnica trivial pero a menudo usada, que consiste en enumerar sistemáticamente todos los posibles candidatos para la solución de un problema, con el fin de chequear si dicho candidato satisface la solución al mismo.

Algoritmo voraz: Un algoritmo voraz (también conocido como ávido, devorador o greedy) es una estrategia de búsqueda por la cual se sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento. Normalmente se aplica a los problemas de optimización.

Divide y conquista: El término divide y conquista hace referencia a uno de los más importantes paradigmas de diseño algorítmico. El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original.

Programación dinámica: La programación dinámica es una técnica matemática que se utiliza para la solución de problemas matemáticos seleccionados, en los cuales se toma una serie de decisiones en forma secuencial. Proporciona un procedimiento sistemático para encontrar la combinación de decisiones que maximice la efectividad total, al descomponer

el problema en etapas, las que pueden ser completadas por una o más formas (estados), y enlazando cada etapa a través de cálculos recursivos.

Package: Un paquete en Java se usa para agrupar clases relacionadas. Piense en ello como una carpeta en un directorio de archivos. Usamos paquetes para evitar conflictos de nombres y para escribir un código mejor mantenible.

Clase: Una clase es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje.

Función: Una función, una subrutina o subprograma, como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica.

Librería: Las librerías son trozos de código hechas por terceros. Esto nos facilita mucho la programación y hace que nuestro programa sea más sencillo de hacer y luego de entender.

Tipado: Cualidad de los lenguajes de programación con la cual establecen el tipos de las variables (El tipo de dato que almacena dicha variable).

2.10) Librerías y funciones prohibidos

En las competencias de programación hay cierto tipo de librerías que no se pueden usar debido a su finalidad, ya que pueden usar requerimientos de hardware o software que pueden generar inestabilidad en las maquinas o simplemente hacer trampa dentro de la competencia, nombraremos las más importantes.

- Librerías de creación y manipulación de hilos (Threads).
- Librerías de manipulación de memoria (Malloc).
- Se evita el uso de punteros de memoria en las variables.
- Librerías de entornos gráficos (Existen excepciones de estas librerías).
- Librerías que no sean de grupo estándar del lenguaje, es decir librerías hechos por terceros ajenos.

- Librería de manejo de procesos.

2.11) 10 pasos para resolver cualquier problema

1. Lea el problema completamente al menos dos o tres veces (o la cantidad de veces que sean que te hagan sentir cómodo)
2. Identifique el tema al que pertenece el problema. ¿Es un problema de ordenamiento o coincidencia de patrones? ¿Puede usar la teoría de grafos? ¿Está relacionado con la teoría de números? etc.
3. Intente resolver el problema manualmente considerando 3 o 4 sets de datos de prueba.
4. Luego concéntrese en optimizar los pasos manuales. Intente hacerlo lo más simple posible.
5. Escriba el pseudocódigo y comentarios, además del código de cada paso. Una cosa que puede hacer es verificar después de escribir cada función. Use un buen IDE con un depurador, si es posible. No es necesario pensar mucho en la sintaxis. Solo concéntrese en la lógica y los pasos.
6. Reemplace los comentarios o pseudocódigo con código real. Siempre verifique si los valores y el código se comportan como se esperaba antes de pasar a la nueva línea de pseudocódigo.
7. Luego optimice el código real.
8. Cuide también las condiciones de restricción de cada variable.
9. Obtenga comentarios de sus compañeros de equipo, profesores y otros desarrolladores y, si es posible, haga su pregunta en StackOverflow. Intente aprender de las pautas de los demás y de lo que están manejando esos problemas. Un problema puede resolverse de varias maneras. Por lo tanto, no se decepcione si no puede pensar como un experto. Debe atenerse al problema y gradualmente será mejor y más rápido para resolver problemas como los demás.
10. Practique, practique y practique.

2.12) Resultados y veredictos de los jueces en línea

Luego de enviar su programa al juez en línea, se compilará y ejecutará en ese sistema, y el juez automático lo probará con algunas entradas y salidas, o quizás con una herramienta de juez específica. Pasados unos segundos o minutos, recibirá una de estas respuestas:

In Queue (QU): El juez está ocupado y no puede atender su presentación. Será juzgado lo antes posible.

Accepted o Correct (AC): ¡OK! ¡Su programa es correcto! Produjo la respuesta correcta en un tiempo razonable y dentro del límite de uso de memoria. ¡Felicidades!

Presentation Error (PE): las salidas de su programa son correctas pero no se presentan de la manera correcta. Buscar espacios, justificar, saltos de línea...

Wrong Answer (WA): No se alcanzó la solución correcta para las entradas. Las entradas y salidas que usan los jueces en línea para probar los programas no son públicas, por lo que tendrá que detectar el error por usted mismo.

Compile Error (CE): El compilador no pudo compilar su programa. Los mensajes de salida del compilador se le informan para que solucione el problema.

Runtime Error (RE): Su programa falló durante la ejecución (error de segmentación, excepción de punto flotante, mal lectura de datos, desbordamiento de variables… entre otros). La causa exacta no se informa al usuario para evitar el plagio. Asegúrese de que su programa devuelva un código 0 (Ejecución correcta) al shell. Si está utilizando Java, siga todas las especificaciones de envío.

Time Limit Exceeded (TLE): Su programa intentó ejecutarse durante demasiado tiempo; este error no le permite saber si su programa alcanzaría la solución correcta al problema o no.

Memory Limit Exceeded (MLE): Su programa intentó usar más memoria de la que permite el juez.

Output Limit Exceeded (OL): Su programa intentó escribir demasiada información. Esto suele ocurrir si entra en un bucle infinito.

Submission Error o Submit Failed (SE): El envío no se realizó correctamente. Esto se debe a algún error durante el proceso de envío o corrupción de datos.

Restricted Function (RF): Su programa está intentando utilizar una función que se considera perjudicial para el sistema del juez en línea. Si obtiene este veredicto, probablemente sepa por qué... (Está haciendo trampa o intentando hacer daño al juez).

Can't Be Judged (CJ): El juez no tiene entradas y salidas de prueba para el problema seleccionado. Al elegir un problema, asegúrese de que el juez pueda juzgarlo.

Capitulo 3. Los lenguajes de programación

En el mercado existen muchísimos lenguajes de programación que sirven para infinidad de áreas aplicadas, tales como la matemática, la medicina, la administración aeroespacial, el control de gobierno, entre otras, pero dentro de este tendremos en cuenta 3 lenguajes únicamente los cuales son:

JAVA 1.8.0: Es un lenguaje de programación de propósito general, concurrente, orientado a objetos, que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos diez millones de usuarios reportados (Oracle, 2019).

El entorno de desarrollo recomendado para este libro es Eclipse IDE.
Link de descarga:
https://www.eclipse.org/downloads

C++11: Es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.
Posteriormente se añadieron facilidades de programación genérica, que se sumaron a los paradigmas de estructurada y programación orientada a objetos. Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma (CPP Reference, 2019) .

El entorno de desarrollo recomendado es Zinjai 20190808.

Link de descarga:

http://zinjai.sourceforge.net/

PYTHON3: Es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma (Python Software Foundation., 2019) .

El entorno de desarrollo recomendado es Python 3.7 IDLE.

Link de descarga:

https://www.python.org/downloads/

Ilustración 3-1 Logos JAVA, C++ y Python, Tomados de: https://www.theverge.com, https://adictoalcodigo.blogspot.com y https://anthoncode.com respectivamente

Cada uno de estos lenguajes tiene fortalezas y debilidades cuando los comparamos en rendimiento, escalabilidad, plataforma, almacenamiento y tiempo de ejecución. Por lo que tener conocimiento de los 3 lenguajes es una gran ventaja ya que se tiene en cuenta cual

es más apto para la resolución de un ejercicio, y también permite mayor competitividad a nivel laboral.

3.1) Diferencias generales de los lenguajes

Debido a la metodología de ejecución de los diferentes lenguajes de programación, la cantidad de librerías, utilidades estándar de cada lenguaje y su dificultad al entender la sintaxis de las líneas de código, se evidencia que cada lenguaje tiene pros y contras en su utilización al momento de resolver ejercicios de programación competitiva, hay que recordar que Java es un lenguaje que utiliza una máquina virtual para su ejecución y que Python es un lenguaje interpretado, estas dos generan que ambos lenguajes sean ligeramente más lentos al ejecutarse.

Otra idea a tener en cuenta es el hecho de que Python no tiene problemas de desbordamiento de variables, debido a que todos los objetos que se crean dentro del código se les asigna memoria de forma dinámica y automática, lo que le permite realizar cálculos con números muy grandes, pero sacrificando rendimiento.

Java contiene una gran cantidad de librerías estándar para uso dentro de la programación competitiva, lo que le permite reducir líneas de código y agregar funcionalidades que optimicen diversas operaciones dentro de la ejecución del código.

Teniendo estas ideas en cuenta se puede realizar la siguiente comparación:

Velocidad de ejecución

$$PYTHON < JAVA < C++$$

Cantidad de librerías

$$JAVA > PYTHON > C++$$

Dificultad de código

$$PYTHON < JAVA < C++$$

Nota: estas conclusiones solo aplican cuando los lenguajes se utilizan en ejercicios de programación competitiva.

Python es más lento en tiempo de ejecución que Java, y Java es más lento que C++, por lo que C++ es el lenguaje más rápido. Java tiene muchas más librerías estándar que Python, y Python tiene muchas más librerías que C++, permitiendo que Java sea el lenguaje más polivalente, por último la dificultad de la sintaxis de Python es mucho menor a la de Java, y la dificultad de Java es mayor a la de C++, permitiendo que Python sea el lenguaje más sencillo de comprender al ser leído.

Un buen programador competitivo tiene conocimiento de estos tres lenguajes de programación de tal manera que al abordar un ejercicio pueda el decidir cuál de los tres lenguajes es el más adecuado para darle solución al mismo.

3.2) Consejos y trucos para programadores competitivos

¿Qué lenguaje usar? Esta es una pregunta frecuente frente a en qué idioma se debe preferir ser eficiente en la programación competitiva. Es algo de lo que uno no debería preocuparse, ya que lo que importa es la lógica, no el lenguaje. La mayoría de los lenguajes son más o menos iguales, pero hasta ahora el lenguaje más utilizado es C ++, a continuación se presenta un resumen de las ventajas y desventajas de cada lenguaje.

Python

• Simple y fácil: Python es simple, fácil de escribir (se necesita escribir menos) y tiene una gran colección de módulos con casi todas las funciones que pueda imaginar.

• Tipos de datos: generalmente se prefiere Python, ya que no tiene ningún límite superior en la memoria de los enteros. Además, no es necesario especificar qué tipo de datos es y cosas como esta hacen que sea más fácil de codificar, pero al mismo tiempo dificultan la compilación (en referencia al tiempo necesario para la compilación).

• Lento en la ejecución: los programas de Python son generalmente más lentos en comparación con Java. Python está prácticamente descartado en el inicio debido a su alto tiempo de ejecución.

• Python no está permitido en todas partes: Python no está permitido en concursos en varios portales de programación competitivos populares en línea.

Ahora nos quedamos principalmente con Java, C, C ++, ahora aquí se hace difícil comparar y depende principalmente del usuario, aquí se presentan los puntos buenos y malos de cada uno de ellos.

Java

• STL vs contenedores: STL en C ++ está muy bien diseñado, mientras que algunas personas aman Java Containers más que nada. Hay pocas situaciones en las que STL no tiene una solución directa. Por ejemplo, priority_queue en STL no admite la operación de tecla de disminución que se requiere para las implementaciones del algoritmo de ruta más corto de Dijkstra y el algoritmo de Prim

• El manejo de excepciones en Java es incomparable: el código Java proporciona un manejo de excepciones más fuerte en comparación con C ++. Por ejemplo, es más fácil rastrear una excepción ArrayIndexOutOfBound o una falla de segmentación en Java. C ++ / C puede darle respuestas incorrectas, pero Java es seguramente confiable en este contexto.

• El límite de tiempo excede: es posible que obtenga TLE debido a que Java es un poco más lento en el lado del límite de tiempo

• Grandes enteros y expresiones regulares: Java tiene algunas pocas ventajas con respecto a los concursos de programación. Biginteger, Expresiones regulares y biblioteca de geometría son algunas de ellas.

C++ y C

• La velocidad de C ++ es comparable a C: muchos programas de C también son programas de C ++ válidos, y dichos programas de C se ejecutan a una velocidad idéntica cuando se compilan

• C ++ no fuerza la programación orientada a objetos: el lenguaje C ++ contiene algunas extensiones de lenguaje que facilitan la programación orientada a objetos y C ++ no fuerza el diseño orientado a objetos en ninguna parte, simplemente lo permite.

• Una biblioteca estándar más grande: C ++ permite el uso completo de la biblioteca estándar C, así como C ++ incluye sus propias bibliotecas, incluida la Biblioteca de plantillas estándar. El STL contiene una serie de plantillas útiles, como la rutina de ordenamiento anterior. Incluye útiles estructuras de datos comunes como listas, mapas, conjuntos, etc. Al igual que la rutina de ordenamiento, las otras rutinas STL y estructuras de datos están "adaptadas" a las necesidades específicas que tiene el programador; todo lo que el programador tiene que hacer es completar los tipos.

Por ejemplo, si necesitamos implementar Binary Search para un problema, tendremos que escribir nuestra propia función, mientras que en C ++ Binary Search, la rutina STL se define como

- binary_search (startaddress, endaddress, valuetofind)

C ++ vs Java

• Los códigos Java son más largos, un programador necesita escribir más cuando programa en Java

• Java es detallado: en C ++, Entrada Salida es más simple simplemente escribiendo scanf / printf. En Java, a veces se necesita la clase BufferedReader, tediosa de programar.

• C ++ STL vs Java Containers: a la mayoría de los programadores les resulta más fácil usar STL.

• C ++ es más popular: ya sea el año de origen o la comodidad de uso, pero C ++ destaca a Java en términos de cantidad de usuarios que usan el lenguaje.

• C ++ ahorra tiempo: es un hecho bien conocido que Java es más lento que C ++. Generalmente necesitamos compilar y ejecutar programas muchas veces para probarlos. Lleva relativamente menos tiempo en C ++.

En resumen, C ++ es hasta la fecha el lenguaje más preferido seguido de Java cuando se trata de concursos de programación, pero siempre debe elegir un idioma con el que se sienta cómodo. Tener confianza en cualquier idioma es lo más importante. Nunca elija un idioma que haya "aprendido" recientemente o con el cual tenga poca experiencia, ya que será difícil expresarse en ese idioma.

Capitulo 4. ¿Cómo llegar a ser un maestro en programación competitiva?

Las competencias de programación a nivel internacional son desarrolladas para medir la capacidad de los estudiantes de programación de las distintas universidades y entes estudiantiles, entre ellas se permite utilizar todo tipo de medios escritos en los cuales basarse para el desarrollo de diferentes soluciones a problemas de la vida real que requieren un algoritmo eficiente.

Grandes empresas de desarrollo como Google, Microsoft, IBM entre otras utilizan estas competencias como principal medio de reclutamiento de personal, por lo que se hace evidente la necesidad de participación en ellas.

Hay muchas personas para quienes la programación es como un sueño embrujado. La programación no es más que un arte de hablar con las máquinas y decirles qué hacer, cuándo hacerlo y por qué hacerlo. La mayoría de los estudiantes escuchan esta palabra en el colegio. Para muchos de ellos, la programación comienza con "C" y termina en "C".

La programación competitiva es una forma avanzada de programación que se ocupa de problemas del mundo real. Aquí vemos nuestro código gobernando el mundo. Pero escribir dicho código requiere destreza con pasión.

Se tiene claro que un código es básicamente nuestra lógica detrás de cualquier problema en lenguaje de alto nivel. Pero la lógica por sí sola no es suficiente para escribir un código perfecto. Requiere una comprensión más profunda de términos técnicos como complejidad, sintaxis y el arte de crear grandes soluciones a través de los códigos más cortos posibles. Todo esto solo se puede lograr a través de la práctica. Pero si la práctica se fusiona con una buena guía, se puede llegar al dominio. Este objetivo se puede lograr a través de los siguientes pasos:

1. Obtenga una comprensión profunda

En primer lugar, estudie todos los conceptos del lenguaje de programación profundamente. Siempre use libros estándar. Hoy en día, hay muchas plataformas en línea disponibles donde los competidores de todo el mundo comparten sus conocimientos e intentan facilitar los conceptos.

2. Seguir un enfoque jerárquico

Intente comenzar a codificar utilizando problemas más simples. Antes de escribir directamente el código, primero haga un diagrama de flujo de la lógica que se está utilizando. Esto aumentará la cantidad de códigos correctos que no solo agudizarán tus habilidades sino que también aumentarán tu confianza.

3. Implementación en la vida real

Una vez que se acostumbre a los códigos y la programación básica, intente crear códigos que resuelvan tus problemas de la vida diaria. Estos pueden incluir la libreta de calificaciones de cualquier estudiante, el sistema de reserva de boletos, el sistema de administración de la biblioteca, etc. De esta manera obtendrá confianza.

4. Simplificar el código

Ahora el siguiente paso es acortar el código. Suponga que crea un código simple para el sistema de administración de una biblioteca. Ahora intente abreviarlo de tal manera que la misma tarea se pueda lograr de una manera mucho más simple y más corta. Primero puede ver el problema y crear su propio código. Ahora vea la solución óptima para aprender cómo se puede reducir. Esta es la fase más importante y la transición de la programación básica a la programación competitiva.

5. No se rinda, láncese a los leones

Ahora comience a participar en competencias de codificación. Las competiciones pueden ser en su escuela, en línea o a nivel nacional. Aquí encontrara personas igual que usted compitiendo y retándose unos con otros. Aquí, debe escribir la solución óptima y eso también en el menor tiempo posible. Obviamente, dado que es una competencia, es la

supervivencia del más apto. Un entorno competitivo amistoso hace que la tasa de aprendizaje sea más rápida e implica aprender mediante un mecanismo divertido. Además de esto, también obtendrá una calificación de acuerdo con sus presentaciones exitosas del código y las competiciones que gane, lo que mejora su perfil profesional.

6. Comience a difundir su conocimiento

Una vez que se convierta en un maratonista, no se guarde los conocimientos. Extiéndalos. Compártalos con sus estudiantes o aprendices, sus compañeros y a través del mundo. Su conocimiento aumenta más si lo comparte con otros.

Capitulo 5. Competencias locales, nacionales e internacionales.

5.1) ACIS/REDIS y CCPL

Ilustración 5-1 Logos ACIS/REDIS Tomado de: http://www.evaluamos.com/images2017A/16219-1.jpg

La Asociación Colombiana de Ingenieros de Sistemas (ACIS) es una organización sin ánimo de lucro que agrupa a más de 1500 profesionales en el área de sistemas. ACIS nació en 1975 agrupando entonces a un número pequeño de profesionales en sistemas. Con el transcurrir de los años, así como el panorama profesional para el área de los sistemas ha ido evolucionando, ACIS ha experimentado un desarrollo paralelo. Es el gremio de los Ingenieros de Sistemas participante en el desarrollo nacional.

Hoy en día, además de organizar eventos académicos de gran importancia a nivel nacional en el área de informática, la Asociación Colombiana de Ingenieros de Sistemas ha multiplicado sus campos de acción involucrándose en la mayoría de los debates sobre el desarrollo tecnológico de Colombia. ACIS se ha constituido en los últimos años como el gestor de eventos de gran reconocimiento que buscan cubrir las diferentes áreas tecnológicas de la Ingeniería de Sistemas como son el Salón de Informática, las Jornadas de Gerencia de Proyectos de TI, las Jornadas de Seguridad Informática, Cursos de Capacitación, MoodleMoot Colombia, entre otros (Asociación Colombiana de Ingenieros de Sistemas., 2019).

La Red de Programas de Ingeniería de Sistemas y Afines (REDIS) es una agrupación de quienes actúan como autoridades máximas de los programas de ingeniería de sistemas, o nombres afines, ofrecidos por instituciones de educación superior. Fue creada en Bogotá en el año 2001 y desde entonces se reúne por lo menos una vez al mes a tratar asuntos comunes de carácter académico y profesional, que implican colaboración y relación con los representantes de distintos sectores y gremios: Acis, Acofi, Fedesoft, Ministerio de las TIC, Ministerio de Industria y Comercio, y Ministerio de Educación Nacional, entre otros.

Ilustración 5-2 Logo CCPL Tomado de: www.programingleague.org

Colombian Collegiate Programming League (CCPL) es la liga de competencia en programación más grande de Colombia, realizando diferentes rondas por año en donde las diferentes entidades de educación superior envían equipos de estudiantes a competir, las competencias se realizan en diferentes sedes cada ronda y funcionan como entrenamiento para las competencias clasificatorias nacionales en Colombia.

5.2) ACM e ICPC

Ilustración 5-3 Logo ACM-ICPC Tomado de: http://www.cs.cornell.edu/acm/

Association for Computing Machinery (ACM) reúne a educadores, investigadores y profesionales de computación a nivel mundial para inspirar el diálogo, compartir recursos y abordar los desafíos del campo. Como la sociedad de computación más grande del mundo, ACM fortalece la voz colectiva de la profesión a través de un liderazgo sólido, la promoción de los más altos estándares y el reconocimiento de la excelencia técnica. ACM apoya el crecimiento profesional de sus miembros al brindar oportunidades para el aprendizaje de por vida, el desarrollo profesional y la creación de redes profesionales.

Fundada en los albores de la era de la computación, el alcance de ACM se extiende a cada parte del mundo, con más de la mitad de sus casi 100,000 miembros que residen fuera de EE. UU. Sus acciones mejoran la capacidad de ACM para aumentar el conocimiento de los importantes problemas técnicos, educativos y sociales de la informática en todo el mundo.

El Concurso Internacional de Programación Colegial (International Collegiate Programming Contest ICPC) es un concurso de programación algorítmica para estudiantes universitarios. Equipos de tres, que representan a su universidad, trabajan para resolver los problemas más reales del mundo, fomentando la colaboración, la creatividad, la innovación y la capacidad de desempeñarse bajo presión. A través del entrenamiento y la competencia, los

equipos se desafían entre sí para elevar el nivel de lo posible. En pocas palabras, es el concurso de programación más antiguo, más grande y más prestigioso del mundo (Zhu Jie-ao, 2005).

El ICPC se remonta a 1970, cuando los pioneros del Capítulo Alfa de la Sociedad de Honor de Ciencias de la Computación de la UPE organizaron la primera competencia. La iniciativa se difundió rápidamente dentro de los Estados Unidos y Canadá como un programa innovador para aumentar la ambición, la capacidad de resolución de problemas y las oportunidades de los estudiantes más sólidos en el campo de la informática (Combéfis, 2014).

Con el tiempo, el concurso se convirtió en una competencia de varios niveles con la primera ronda de campeonato llevada a cabo en 1977. Desde entonces, el concurso se ha convertido en una colaboración mundial de universidades que organizan competiciones regionales que hacen avanzar a los equipos a la ronda de campeonatos mundial anual, Las finales mundiales de ICPC (ICPC World Finals) (Amraii, 2006).

Ilustración 5-4 ICPC WORLD FINALS 2018 Tomado de: www.topcoder.com

El International Collegiate Programming Contest (ICPC) es el principal concurso mundial de programación realizado por y para las universidades del mundo. El ICPC está afiliado a la Fundación ICPC y tiene su sede en la Universidad de Baylor.

5.3) Google CodeJAM, HashMap y Kickstart

Ilustración 5-5 Inscripciones CODEJAM 2018 Tomado de: https://www.youtube.com/watch?v=ipdUjbK1_h8

Google Code Jam, HashMap y Kickstart son competencias de programación internacional organizada y administrada por Google, son de modalidad solitario, en equipo y entrenamiento respectivamente. La competencia CodeJAM comenzó en 2003 como un medio para identificar a los mejores talentos de ingeniería para un empleo potencial en Google. La competencia consiste en un conjunto de problemas algorítmicos que deben resolverse en un período de tiempo fijo.

Los competidores pueden usar cualquier lenguaje de programación y entorno de desarrollo para crear sus soluciones. De 2003 a 2007, Google Code Jam se implementó en la plataforma de Topcoder y tenía unas reglas bastante diferentes. Desde 2008, Google ha desarrollado su propia infraestructura dedicada para el concurso.

5.4) CodeChef

Ilustración 5-6 Logo CodeChef y Directi, Tomado de: www.codechef.com

CodeChef es una iniciativa educativa sin fines de lucro de Directi hasta mayo del 2020, una compañía india de software. Es una comunidad de programación global que fomenta el aprendizaje y la competencia amistosa, construida sobre la plataforma de programación competitiva más grande del mundo.

CodeChef fue creado como una plataforma para ayudar a los programadores a triunfar en el mundo de los algoritmos, la programación de computadoras y los concursos de programación. Realizan tres concursos destacados cada mes y otorgan premios y regalos a los ganadores como estímulo. Aparte de esto, la plataforma está abierta a toda la comunidad de programación para organizar sus propios concursos. Las principales instituciones y organizaciones de todo el mundo utilizan la plataforma para organizar sus concursos (Directi, 2019).

En el siguiente link se podrá encontrar un video tutorial explicativo sobre cómo utilizar CodeChef para entrenamiento de programación:

Solving your first problem in Java on CodeChef:

https://www.youtube.com/watch?v=gaPdjwuFZTs

5.5) Online Judge

Ilustración 5-7 Logo Online Judge 2019, Tomado de: https://onlinejudge.org/

Online Judge (Anteriormente Uva Online Judge) es un juez automatizado en línea para problemas de programación anteriormente alojado anteriormente por la Universidad de Valladolid. Su archivo de problemas tiene más de 14000 problemas y el registro de usuarios está abierto a todos. Actualmente hay más de 1000000 usuarios registrados. Un usuario puede enviar una solución en ANSI C (C89), C ++ (C ++ 98), Pascal, Java, C ++ 11 o Python.

Ilustración 5-8 Anterior logo de UVa Online Judge, Tomado de: http://nafischonchol.blogspot.com

OJ también organiza concursos. En el entorno del concurso, el usuario tiene un tiempo limitado para resolver un pequeño conjunto de problemas. El UVa OJ fue creado en 1995 por Miguel Ángel Revilla, matemático que enseñaba algoritmos en la Universidad de Valladolid en España, actualmente la plataforma se convirtió en independiente de esta universidad y es administrada por Miguel Ángel Revilla Jr.

Los siguientes links presentan video tutoriales explicando cómo buscar y realizar ejercicios en esta plataforma:

How to submit in UVa Online Judge:

https://www.youtube.com/watch?v=i56-7MIzyHY

Find Beginners Problem in UVa Online Judge:

https://www.youtube.com/watch?v=MZkhV8B9lnw

Capitulo 6. Comenzando a programar

Antes de ir de cabeza a programar competitivamente, necesita repasar ciertos temas base de las áreas de la programación, la lógica y la matemática, en este capítulo se encuentra un resumen de los temas clave a la hora de resolver problemas de programación competitiva usando lenguajes de programación.

6.1) Operadores matemáticos, lógicos y comparativos

¿Qué es un operador lógico?

Los operadores lógicos, utilizados en Informática, lógica proposicional y álgebra booleana, entre otras disciplinas, nos proporcionan un resultado a partir de que se cumpla o no una cierta condición, producen un resultado booleano, y sus operandos son también valores lógicos o asimilables a ellos (los valores numéricos son asimilados a verdadero o falso según su valor sea cero o distinto de cero). Esto genera una serie de valores que, en los casos más sencillos, pueden ser parametrizados con los valores numéricos 0 y 1. La combinación de dos o más operadores lógicos conforma una función lógica.

Los operadores lógicos son tres, dos de ellos son binarios, el último (negación) es unario. Tienen una doble posibilidad de representación depende el lenguaje:

- "Y" lógico -> && -> AND
- "O" lógico -> || -> OR
- Negación lógica -> ! -> NOT

Las expresiones conectadas con los operadores && y || se evalúan de izquierda a derecha, y la evaluación se detiene tan pronto como el resultado verdadero o falso es conocido.

¿Qué es una operación matemática?

Una operación matemática es un proceso mediante el cual se logra la transformación de una o más cantidades en otra cantidad llamada resultado.

Toda operación matemática tiene unos valores de entrada, y presenta una regla de definición que señala el tipo de proceso que se debe realizar para llegar al resultado.

Junto con los valores de entrada se coloca un símbolo que se denomina **operador matemático.**

Hay operaciones matemáticas que nos son muy conocidas, cuyos operadores también lo son: por ejemplo, el + que señala una suma, la rayita –que indica una resta, y los operadores de multiplicación *, división / y residuo %.

¿Qué es un operador comparativo?

Los operadores de comparación comparan dos expresiones y devuelven un valor booleano que representa la relación de sus valores. Existen operadores para comparar valores numéricos, operadores para comparar cadenas y operadores para comparar objetos.

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO	TIPO
+	Suma	5+6	11	Matemático
-	Resta	5-6	-1	Matemático
*	Multiplicación	5*6	30	Matemático
/	División	7/5	1	Matemático
%	Modulo (Residuo)	7%5	2	Matemático
&	AND	true & true	True	Lógico
|	OR	true | false	True	Lógico
!	NOT	!true	False	Lógico
^	XOR	true^true	False	Lógico
&&	AND de condición	5>4&&6>7	False	Lógico
||	OR de condición	5>4||6>7	True	Lógico
==	Igual que	7==38	False	Comparativo
!=	Distinto que	‘a’ != ‘k’	True	Comparativo
<	Menor que	‘G’ < ‘B’	False	Comparativo

>	Mayor que	‘b > ‘a’	True	Comparativo
<=	Menor o igual que	7.5 <= 7.38	False	Comparativo
>=	Mayor o igual que	38 >= 7	true	Comparativo

Tabla 6-1 Operadores lógicos, comparativos y matemáticos con ejemplo

6.2) Tipos de variables básicas

En programación, las variables son espacios reservados en la memoria que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa. Una variable corresponde a un área reservada en la memoria principal del ordenador.

Por buenas prácticas, el identificador de la variable (Nombre de la variable) debe ser mnemotécnico, es decir que debe reflejar el uso dentro del programa de la misma.

El tipo de dato informático es un atributo de una parte de los datos que indica al ordenador (y al programador) algo sobre la clase de datos sobre los que se va a procesar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar. Tipos de datos comunes son: enteros, cadenas alfanuméricas, fechas, horas, colores, coches o cualquier cosa que se nos ocurra. Por ejemplo, el tipo "int" representa un conjunto de enteros. Éste es un concepto propio de la informática, más específicamente de los lenguajes de programación, aunque también se encuentra relacionado con nociones similares de las matemáticas y la lógica.

Debido a que las variables contienen o apuntan a valores de tipos determinados, las operaciones sobre las mismas y el dominio de sus propios valores están determinadas por el tipo de datos en cuestión.

Tipo de dato lógico:

El tipo de dato lógico o booleano es en computación aquel que puede representar valores de lógica binaria, esto es 2 valores, que normalmente representan falso o verdadero. Se

utiliza normalmente en la programación, estadística, electrónica, matemáticas (álgebra booleana) y otras.

Tipo de dato entero:

El tipo de dato entero en computación se usa para representar un subconjunto finito de los números enteros. El mayor número que se puede representar depende del tamaño del espacio usado por el dato y la posibilidad (o no) de representar números negativos. Los tipos de dato entero disponibles y su tamaño dependen del lenguaje de programación usado así como la arquitectura en cuestión.

Tipo de dato carácter:

En terminología informática y de telecomunicaciones, un carácter es un símbolo que representa cada carácter de un lenguaje natural. Un ejemplo de carácter es una letra, un número o un signo de puntuación.

Cadena de caracteres:

En programación, una cadena de caracteres o frase (string en inglés) es una secuencia ordenada de longitud arbitraria (aunque finita) de elementos que pertenecen a un cierto alfabeto. En general, una cadena de caracteres es una sucesión de caracteres (letras, números u otros signos o símbolos).

Desde el punto de vista de la programación, si no se ponen restricciones al alfabeto, una cadena podrá estar formada por cualquier combinación finita de todo el juego de caracteres disponibles (las letras de la 'a' a la 'z' y de la 'A' a la 'Z', los números del '0' al '9', el espacio en blanco ' ', símbolos diversos '!', '@', '%', entre otros). Un caso especial de cadena es la que contiene cero caracteres, a esta cadena se le llama cadena vacía.

Variable	Tipo	JAVA	C++	PYTHON

Int	Numérico entero	2^32 bits	2^32 bits	Infinito
long	Numérico entero	2^64 bits	2^64 bits	No aplica
float	Numérico decimal	2^32 bits	2^32 bits	Infinito
double	Numérico decimal	2^64 bits	2^64 bits	No aplica
char	Carácter	2^16 bits	2^8 bits	Infinito
string	Cadena de caracteres	Infinito	Infinito	Infinito
byte	Binario	8 bits	No aplica	No aplica

Tabla 6-2: Tipos de variable y sus tamaños en tres lenguajes de programación

6.3) Lectura e impresión

Las lecturas e impresiones de nuestros ejercicios de programación competitiva, sus algoritmos base y sus modificaciones se realizan por la consola (STDIN, STDOUT), por lo que por lo general la programación de entornos gráficos para nuestros códigos no es válida, los siguientes códigos muestran ejemplos de cómo se realizan las entradas y salidas de diferentes tipos de variables.

Código

Ejemplo de entrada
6555
73653736353
463553
345543234
a
holamundo
101

JAVA

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class LecturaEscritura {
    //Variables Globales
    static int numeroentero;
    static long numeroenterolargo;
    static float numerodecimal;
    static double numerodecimallargo;
    static char caracter;
    static String cadena;
    static byte bits;

    public static void main(String[] args) throws IOException {
        System.out.println("Lectura\n");
        lecturaLenta();
        lecturaRapida();
        System.out.println("");
        System.out.println("Impresión \n");
        impresionNormal();
        impresionConFormato();
    }

    static void lecturaLenta() {
        //Objeto lector
        Scanner sc = new Scanner(System.in);
        //Lectura con tipo especificado
        numeroentero = sc.nextInt();
        numeroenterolargo = sc.nextLong();
        numerodecimal = sc.nextFloat();
        numerodecimallargo = sc.nextDouble();
        caracter = sc.next().charAt(0);
        cadena = sc.next();
        bits = sc.nextByte();
    }
    static void lecturaRapida() throws IOException {
        //Objeto lector
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        //Lectura como cadena y conversión a tipo
        numeroentero = Integer.parseInt(br.readLine());
        numeroenterolargo = Long.parseLong(br.readLine());
        numerodecimal = Float.parseFloat(br.readLine());
        numerodecimallargo = Double.parseDouble(br.readLine());
        caracter = br.readLine().charAt(0);
        cadena = br.readLine();
        bits = Byte.parseByte(br.readLine());
    }

    static void impresionNormal()
```

```
 {
        System.out.println(numeroentero);
        System.out.println(numeroenterolargo);
        System.out.println(numerodecimal);
        System.out.println(numerodecimallargo);
        System.out.println(caracter);
        System.out.println(cadena);
        System.out.println(bits);
    }

    static void impresionConFormato() {
        //Se especifica el tipo de dato o la forma en que se imprimira
        System.out.printf("%d \n", numeroentero);
        System.out.printf("%d \n", numeroenterolargo);
        System.out.printf("%f \n", numerodecimal);
        System.out.printf("%e \n", numerodecimal);
        System.out.printf("%f \n", numerodecimallargo);
        System.out.printf("%e \n", numerodecimallargo);
        System.out.printf("%s \n", caracter);
        System.out.printf("%s \n", cadena);
        System.out.printf("%s \n", bits);
    }
}
```

C++

```
//Unicas 2 librerias que se usan en c++ (Contienen todas)
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
//variables globales
int numeroentero;
long numeroenterolargo;
float numerodecimal;
double numerodecimallargo;
char caracter;
string cadena;

void lecturaLenta() {
        //Lectura con tipo especificado
        cin>>numeroentero;
        cin>>numeroenterolargo;
        cin>>numerodecimal;
        cin>>numerodecimallargo;
        cin>>caracter;
        cin>>cadena;
}
void lecturaRapida() {
        //Lectura como cadena y conversión a tipo (En c++ & es necesario)
        scanf("%d",&numeroentero);
        scanf("%lld",&numeroenterolargo);
        scanf("%f",&numerodecimal);
        scanf("%lf",&numerodecimallargo);
        scanf("%s",&caracter);
        scanf("%s",cadena.c_str());
```

```
}
void impresionNormal() {
        cout<<numeroentero<<endl;
        cout<<numeroenterolargo<<endl;
        cout<<numerodecimal<<endl;
        cout<<numerodecimallargo<<endl;
        cout<<caracter<<endl;
        cout<<cadena<<endl;
}
 void impresionConFormato() {
        //Se especifica el tipo de dato o la forma en que se imprimira
        printf("%d \n",numeroentero);
        printf("%lld \n",numeroenterolargo);
        printf("%f \n",numerodecimal);
        printf("%e \n",numerodecimal);
        printf("%lf \n",numerodecimallargo);
        printf("%e \n",numerodecimallargo);
        printf("%c \n",caracter);
        printf("%s \n",cadena.c_str());
}

int main (int argc, char *argv[]) {
        cout<<"Lectura\n"<<endl;
        lecturaLenta();
        cout<<"Impresión \n"<<endl;
        impresionNormal();
        cout<<endl;
        cout<<"Lectura\n"<<endl;
        lecturaRapida();
        cout<<"Impresión \n"<<endl;
        impresionConFormato();
        return 0;
}
```

PYTHON

```
#Los comentarios en Python se usa el # (Numeral)
#variables globales
#la identación define el orden
numeroentero = int(0)
numeroenterolargo = int(0)
numerodecimal = float(0)
numerodecimallargo = float(0)
caracter = ''
cadena = ""
print("Lectura")
#Lectura con tipo especificado
numeroentero = int(input())
numeroenterolargo = int(input())
numerodecimal = float(input())
numerodecimallargo = float(input())
caracter = input()
cadena = str(input())
print("Impresión")
```

```
print(numeroentero)
print(numeroenterolargo)
print(numerodecimal)
print(numerodecimallargo)
print(caracter)
print(cadena)
print("Impresión formateada")
#Se especifica el tipo de dato o la forma en que se imprimira
#obligatorio usar el %
print("%d" % numeroentero)
print("%d" % numeroenterolargo)
print("%f" % numerodecimal)
print("%e" % numerodecimal)
print("%f" % numerodecimallargo)
print("%e" % numerodecimallargo)
print("%c" % caracter)
print("%s" % cadena)
```

Existen diferentes tipos de formateadores de salida y entrada los cuales son utilizados en las lecturas e impresiones formateadas, estos formateadores son los siguientes:

Formateador	**Salida/Entrada**
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%ld	Entero de 32 bits (long)

%lu	Entero sin signo de 32 bits (unsigned long)
%e	La notación científica (mantisa / exponente), minúsculas (decimal precisión simple o doble)
%E	La notación científica (mantisa / exponente), mayúsculas (decimal precisión simple o doble)
%c	carácter (char)
%s	cadena de caracteres (string)

Tabla 6-3 Tipos de formateadores de salida y entrada

6.4) Condicionales y ciclos

Un condicional, como su nombre lo indica, es una condición para discernir entre una opción u otra, y en el proceso mental normalmente se manifiesta con un "Si"; por ejemplo: Si va a llover, coja el paraguas. Sintácticamente, IF es la palabra reservada para desencadenar el poder de los condicionales en el código. ELSE expresa "en el caso contrario". Siguiendo con el ejemplo anterior de la lluvia: if va a llover coja el paraguas else coja el vestido de baño.

Un bucle o ciclo, en programación, es una secuencia que ejecuta repetidas veces un trozo de código, hasta que la condición asignada a dicho bucle deja de cumplirse. Los tres bucles más utilizados en programación son el bucle while, el bucle for y el bucle do-while.

Ejemplo de entrada

5

JAVA

```
import java.util.Scanner;

public class CondicionalesCiclos {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        //Lectura
        int numero = sc.nextInt();
        //Estructura de un condicional
        //Dentro el parentesis va la condicion a revisar
        //Si se cumple se realiza las siguientes lineas
        //En caso contrario pasa al else
        //y se realiza las lineas siguientes del else
        if (numero % 2 == 0) {
            System.out.println("Es par");
        } else {
            System.out.println("Es impar");
        }
        System.out.println("");
        System.out.println("Ciclo for ");
        // variable iteradora, condicion , acumulador
        for (int i = 0; i < 10; i++) {
            System.out.println(i);
        }
        System.out.println("Ciclo while");
        //Variable, condicion, al final acumulador
        int acumulador = 0;
        while (acumulador < 10) {
            System.out.println(acumulador);
            acumulador++;

        }
        System.out.println("Ciclo do while");
        //Haga hasta que una condición se cumpla
        acumulador = 0;
        do {
            System.out.println(acumulador);
            acumulador++;
        } while (acumulador < 10);
        System.out.println("Ciclo for each");
        //Iterar entre todos los elementos que tenga una estructura
        //Sin importar el tamaño
        int numeros[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
        for (int o : numeros) {
            System.out.println(o);
        }

    }

}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int main(int argc, char *argv[]) {
    int número;
```

```
    cin>>número;
    //Estructura de un condicional
    //Dentro el parentesis va la condicion a revisar
    //Si se cumple se realiza las siguientes lineas
    //En caso contrario pasa al else
    //y se realiza las lineas siguientes del else
    if (número % 2 == 0) {
        cout << "Es par" << endl;
    } else {
        cout << "Es impar" << endl;
    }
    cout << endl;
    cout << "Ciclo for " << endl;
    // variable iteradora, condicion , acumulador
    for (int i = 0; i < 10; i++) {
        cout << i << endl;
    }
    cout << "Ciclo while" << endl;
    //Variable, condicion, al final acumulador
    int acumulador = 0;
    while (acumulador < 10) {
        cout << acumulador << endl;
        acumulador++;
    }
    cout << "Ciclo do while" << endl;
    //Haga hasta que una condición se cumpla
    acumulador = 0;
    do {
        cout << acumulador << endl;
        acumulador++;
    } while (acumulador < 10);
    cout << "Ciclo for each" << endl;
    //Iterar entre todos los elementos que tenga una estructura
    //Sin importar el tamaño
    int números[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    for (int o : números) {
        cout << o << endl;
    }
    return 0;
}
```

PYTHON

```
numero = 0;
numero = int(input())
#Estructura de un condicional
#Dentro el parentesis va la condicion a revisar
#Si se cumple se realiza las siguientes lineas
#En caso contrario pasa al else
#y se realiza las lineas siguientes del else
if (numero % 2 == 0):
    print("Es par")
else:
    print("Es impar")
print("Ciclo for ")
```

```
#variable iteradora\textcolor{Orange}{"Ciclo for each"}, rango inicio,final
for i in range (0, 10):
    print(i)
print("Ciclo while")
#Variable, condicion, al final acumulador
acumulador = 0;
while (acumulador < 10):
    print(acumulador)
    acumulador = acumulador + 1;
print("Ciclo for each")
#Iterar entre todos los elementos que tenga una estructura
#Sin importar el tamaño
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
for o in numeros:
    print(o)
```

6.5) Conversiones, parseos, y casteos de variables

El casting consiste en la conversión de tipos de datos similares (compatibles) entre sí, generalmente a través de la herencia.

Por otra parte, el parsing consiste en analizar el formato de una sentencia de texto, y obtener información si el formato es correcto.

JAVA

```
import java.math.BigDecimal;
import java.math.BigInteger;

public class Main {

    public static void main(String[] args) {
        int enteropequenio = 5;
        long enterogrande = 100L;
        float decimalpequenio = 3.4f;
        double decimalgrande = 3.4;
        String cadena = "125";
        char numerosolitario = '7';
        boolean booleano = true;
        // conversión a string
        System.out.println("------------A String----------");
        System.out.println(Integer.toString(enteropequenio));
        System.out.println(Long.toString(enterogrande));
        System.out.println(Float.toString(decimalpequenio));
```

```
System.out.println(Double.toString(decimalgrande));
System.out.println(Character.toString(numerosolitario));
System.out.println(Boolean.toString(booleano));
// parseo de string a entero
System.out.println("------------A números----------");
int basedeentrada = 10;
System.out.println(Integer.parseInt(cadena, basedeentrada));
System.out.println(Long.parseLong(cadena, basedeentrada));
System.out.println(Float.parseFloat(cadena));
System.out.println(Double.parseDouble(cadena));
System.out.println(Character.valueOf(cadena.charAt(0)));
System.out.println(Boolean.parseBoolean("true"));
// casteo a entero
System.out.println("------------A int----------");
System.out.println((int) enterogrande);
System.out.println((int) decimalpequenio);
System.out.println((int) decimalgrande);
System.out.println((int) numerosolitario);
System.out.println((int) (booleano ? 1 : 0));
//casteo a long
System.out.println("------------A long----------");
System.out.println((long) enteropequenio);
System.out.println((long) decimalpequenio);
System.out.println((long) decimalgrande);
System.out.println((long) numerosolitario);
System.out.println((long) (booleano ? 1 : 0));
// casteo a float
System.out.println("------------A float----------");
System.out.println((float) enteropequenio);
System.out.println((float) enterogrande);
System.out.println((float) decimalgrande);
System.out.println((float) numerosolitario);
System.out.println((float) (booleano ? 1 : 0));
// casteo a double
System.out.println("------------A double----------");
System.out.println((double) enteropequenio);
System.out.println((double) enterogrande);
System.out.println((double) decimalpequenio);
System.out.println((double) numerosolitario);
System.out.println((double) (booleano ? 1 : 0));
// casteo a char
System.out.println("------------A char----------");
System.out.println((char) enteropequenio);
System.out.println((char) enterogrande);
System.out.println((char) decimalpequenio);
System.out.println((char) numerosolitario);
System.out.println((char) (booleano ? 1 : 0));
//biginteger y big decimal (SOLO JAVA)
System.out.println("------------bigdecimal y big integer----------");
BigInteger numeroenorme = new BigInteger(cadena);
//mientras quepa en la variable
int cambio = numeroenorme.intValue();
BigDecimal decimalenorme = new BigDecimal(cadena);
//mientras quepa en la variable
```

```
        double cambio2 = decimalenorme.doubleValue();
        System.out.println(cambio);
        System.out.println(cambio2);
    }
}
```

C++

```
#include<bits/stdc++.h>
using namespace std;
typedef long long int ll;

int main() {
    int enteropequenio = 5;
    ll enterogrande = 100LL;
    float decimalpequenio = 3.4f;
    double decimalgrande = 3.4;
    string cadena = "125";
    char numerosolitario = '7';
    bool booleano = true;
    string cast;
    // conversión a string
    printf("------------A String----------\n");
    cast.push_back(numerosolitario);
    printf("%s\n", to_string(enteropequenio).c_str());
    printf("%s\n", to_string(enterogrande).c_str());
    printf("%s\n", to_string(decimalpequenio).c_str());
    printf("%s\n", to_string(decimalgrande).c_str());
    printf("%s\n", cast.c_str());
    printf("%s\n", to_string(booleano).c_str());
    printf("------------A Numero----------\n");
    printf("%d\n", stoi(cadena));
    printf("%lld\n", stoll(cadena));
    printf("%f\n", stof(cadena));
    printf("%f\n", stof(cadena));
    printf("%c\n", (cadena[0]));
    printf("%d\n", stoi(to_string(booleano).c_str()));
    printf("------------A int----------\n");
    printf("%d\n", (int) enterogrande);
    printf("%d\n", (int) decimalpequenio);
    printf("%d\n", (int) decimalgrande);
    printf("%d\n", (int) numerosolitario);
    printf("%d\n", (int) (booleano ? 1 : 0));
    printf("------------A Long----------\n");
    printf("%lld\n", (ll) enteropequenio);
    printf("%lld\n", (ll) decimalpequenio);
    printf("%lld\n", (ll) decimalgrande);
    printf("%lld\n", (ll) numerosolitario);
    printf("%lld\n", (ll) (booleano ? 1 : 0));
    printf("------------A Float----------\n");
    printf("%f\n", (float) enteropequenio);
    printf("%f\n", (float) enteropequenio);
    printf("%f\n", (float) decimalgrande);
    printf("%f\n", (float) numerosolitario);
```

```
    printf("%f\n", (float) (booleano ? 1 : 0));
    // casteo a char
    printf("------------A char----------\n");
    printf("%c\n", (char) enteropequenio);
    printf("%c\n", (char) enterogrande);
    printf("%c\n", (char) decimalpequenio);
    printf("%c\n", (char) numerosolitario);
    printf("%c\n", (char) (booleano ? 1 : 0));
}
```

PYTHON

```
entero = 5
decimal = 3.4
cadena = '125'
numsolitario = '7'
booleano = True


# Conversion a Cadena
print('-------------- A String -------------')
print(str(entero))
print(str(decimal))
print(str(numsolitario))
print(str(booleano))

# Conversion Cadena a Numero
print('---------- Cadena a Numero ----------')
print(int(cadena))
print(int(cadena[0]))
print(float(cadena))
print(bool(booleano))

# Conversion a Entero
print('--------------- A Int ---------------')
print(int(decimal))
print(ord(numsolitario))
print(int(booleano))

# Conversion a Decimal
print('-------------- A Float --------------')
print(float(entero))
print(float(cadena))
print(float(numsolitario))
print(float(booleano))

# Conversion a Caracter
print('-------------- A Char ---------------')
print(chr(65))
print(chr(100))
```

6.6) Control de excepciones

El manejo de excepciones es una técnica de programación que permite al programador controlar los errores ocasionados durante la ejecución de un programa informático. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, por ejemplo retornando un mensaje de error o devolviendo un valor por defecto.

Ejemplo de entrada

5

JAVA

```java
import java.util.Scanner;
public class ControlExcep {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numero;
        //Intenta guardar el dato en la variable
        //Si falla controla la excepcion y
        //realiza otra cosa sin terminar la ejecución
        try {
            numero = sc.nextInt();
            System.out.println(numero);
        } catch (Exception e) {
            System.out.println("El dato insertado no es un número");
            System.out.println("O es muy grande para un tipo int");
        }
    }
}
```

C++

No es necesario controlar las excepciones, no almacena nada nuevo por lo que el programa mostrara el valor que anteriormente tenía esa posición de memoria

PYTHON

```python
numero = 0
#Intenta guardar el dato en la variable
#Si falla controla la excepción y
#realiza otra cosa sin terminar la ejecución
try:
    numero = int(input())
```

```
    print (numero)
except:
    print ("El dato insertado no es un numero")
    print ("O es muy grande para un tipo int")
```

6.7) Complejidad de tiempo (Eficiencia Algorítmica)

En Ciencias de la Computación, el término eficiencia algorítmica es usado para describir aquellas propiedades de los algoritmos que están relacionadas con la cantidad de recursos utilizados por el algoritmo. Un algoritmo debe ser analizado para determinar el uso de los recursos que realiza.

Notación	Nombre	Ejemplos
O(1)	Constante	Determinar si un número es par o impar. Usar una tabla de consulta que asocia constante/tamaño. Usar una hash para obtener un elemento
O(Log n)	Logarítmico	Buscar un elemento específico en un Array utilizando un árbol binario de búsqueda o un árbol de búsqueda balanceado, así como todas las operaciones en un Heap binomial.
O($\sqrt{n}$)	Radical	Buscar un elemento específico en un Array utilizando búsqueda por saltos

O(n)	Lineal	Buscar un elemento específico en una lista desordenada o en un árbol degenerado (peor caso).
O(n Log n)	Log Linear o Quasilinear	Ejecutar una transformada rápida de Fourier; Heap Sort, Quick Sort (caso mejor y promedio), o Merge Sort
O(N^2)	Cuadrático	Multiplicar dos números de n dígitos por un algoritmo simple. Bubble Sort (caso peor o implementación sencilla), Shell Sort, Quick Sort (caso peor).
O(C^n), C>1	Exponencial	Encontrar la solución exacta al problema del viajante utilizando programación dinámica. Determinar si dos sentencias lógicas son equivalentes utilizando una búsqueda por fuerza bruta
O(N^3)	Cubico	Algoritmos de búsqueda de ciclos triples, recorridos de grafos.

Tabla 6-4 Ejemplos de complejidad de tiempo

6.8) ¿Qué afecta la complejidad de tiempo?

- La cantidad de memoria requerida por el código del algoritmo.

- La cantidad de memoria requerida para almacenar los datos de entrada.
- La cantidad de memoria requerida para los datos de salida.
- La cantidad de memoria requerida en cuanto a espacio de trabajo del algoritmo para realizar los cálculos y asignaciones.
- Memoria Caché (usualmente RAM-estática): esta ópera a una velocidad comparable a la del CPU.
- Memoria física principal (usualmente RAM-dinámica): esta ópera un tanto más lenta que el CPU.
- Memoria virtual (usualmente en disco): esta da la impresión de una gran cantidad de memoria utilizable y opera en el orden de los miles más lenta que el CPU.
- La velocidad de la CPU
- El factor de tiempo propio de cada lenguaje, recuerde que cada lenguaje tiene tiempos de ejecución diferentes dependiendo de cómo funcione, de si es interpretado y de la prioridad que tenga en tiempo del procesador

6.9) Conteo de tiempo de ejecución

El tiempo de ejecución es el período en el que un programa es ejecutado por el sistema operativo. El período comienza cuando el programa es llevado a la memoria primaria y comienzan a ejecutarse sus instrucciones. El período finaliza cuando el programa envía la señal de término (normal o anormal) al sistema operativo.

Suele decirse también que un programa se encuentra "corriendo" mientras está siendo ejecutado. Otros tiempos de un programa son el tiempo de compilación, el tiempo de enlazado y el tiempo de carga.

JAVA

```
/*Calculo del tiempo de ejecución de un ciclo de 100000 números
Varia de maquina en máquina, y dependiendo de lo que se
```

```
encuentre en ejecución en la maquina*/

public class CalculoTiempo {

    public static void main(String[] args) {
        long timeBefore = System.currentTimeMillis();
        final int N = 100000;
        for (int i = 0; i <= N; i++) {
            System.out.println(i);
        }
        long timeAfter = System.currentTimeMillis();
        System.out.println((timeAfter - timeBefore) + "ms");
    }
}
```

6.10) String Matching y expresiones regulares

Una expresión regular, o expresión racional, es una secuencia de caracteres que conforma un patrón de búsqueda. También son conocidas como regex por su contracción de las palabras inglesas regular expression. Son principalmente utilizadas para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

Lógicos:

- x|y: x o y
- xy: x seguido de y
- (): Agrupación

Intervalos de caracteres:

- [abc]: Cualquiera de los caracteres entre corchetes. Pueden especificarse rangos, por ejemplo ([a-d] que equivale a [abcd]).
- [^abc]: Cualquier carácter que no esté entre los corchetes.
- [a-zA-Z]: a a la z o A a la Z (Rango).
- [a-z&&[def]]: d,e, o f (Intersección)
- [a-b&&[^bc]]: (Substracción)

Intervalos de caracteres predefinidos:

- **.:** Cualquier carácter individual, salvo el de salto de línea

- \d: Cualquier carácter de digito, equivalente a [0-9]
- \D: Cualquier carácter que no sea del digito, equivalente a [^0-9]
- \s: Cualquier carácter individual de espacio en blanco (Espacios, tabulaciones, saltos de página o saltos de línea)
- \S: Cualquier carácter individual que no sea un espacio en blanco
- \w: Cualquier carácter alfanumérico
- \W: Cualquier carácter que no sea alfanumérico

6.11) Lectura especializada y alta velocidad de procesamiento

Existen ejercicios dentro de programación competitiva en los cuales la cantidad de entradas es tan grande que usar los medios convencionales de lectura el algoritmo no es lo suficientemente rápido, retornando un Time Limit Exceeded (Tiempo límite excedido), es decir la solución no se encuentra lo suficientemente optimizada o no es lo suficientemente rápida, en este capítulo presentamos nuevas formas de lectura y procesamiento de entradas que pueden ayudarnos a resolver este problema.

JAVA Fast Reader

```
// FastReader (Lectura rapida con tokenizer)
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;

public class Main {

    static class FastReader {

        BufferedReader br;
        StringTokenizer st;
```

```
        public FastReader() {
            br = new BufferedReader(new InputStreamReader(System.in));
        }

        String next() {
            while (st == null || !st.hasMoreElements()) {
                try {
                    st = new StringTokenizer(br.readLine());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            return st.nextToken();
        }

        int nextInt() {
            return Integer.parseInt(next());
        }

        long nextLong() {
            return Long.parseLong(next());
        }

        double nextDouble() {
            return Double.parseDouble(next());
        }

        String nextLine() {
            String str = "";
            try {
                str = br.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return str;
        }
    }

    public static void main(String[] args) {
        FastReader s = new FastReader();
        int n = s.nextInt();
        int k = s.nextInt();
        int count = 0;
        while (n-- > 0) {
            int x = s.nextInt();
            if (x % k == 0) {
                count++;
            }
        }
        System.out.println(count);
    }
}
```

JAVA UltraReader

(Puede fallar en algunos juzgadores)

```java
// UltraReader (Lectura ultra rapida usando streams y buffers)

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class Main {

    static class UltraReader {

        final private int BUFFER_SIZE = 1 << 16;
        private DataInputStream din;
        private byte[] buffer;
        private int bufferPointer, bytesRead;

        public UltraReader() {
            din = new DataInputStream(System.in);
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }

        public UltraReader(String file_name) throws IOException {
            din = new DataInputStream(new FileInputStream(file_name));
            buffer = new byte[BUFFER_SIZE];
            bufferPointer = bytesRead = 0;
        }

        public String readLine() throws IOException {
            byte[] buf = new byte[64]; // line length
            int cnt = 0, c;
            while ((c = read()) != -1) {
                if (c == '\n') {
                    break;
                }
                buf[cnt++] = (byte) c;
            }
            return new String(buf, 0, cnt);
        }

        public int nextInt() throws IOException {
            int ret = 0;
            byte c = read();
            while (c <= ' ') {
                c = read();
            }
            boolean neg = (c == '-');
            if (neg) {
                c = read();
            }
            do {
                ret = ret * 10 + c - '0';
            } while ((c = read()) >= '0' && c <= '9');
```

```
        if (neg) {
            return -ret;
        }
        return ret;
    }

    public long nextLong() throws IOException {
        long ret = 0;
        byte c = read();
        while (c <= ' ') {
            c = read();
        }
        boolean neg = (c == '-');
        if (neg) {
            c = read();
        }
        do {
            ret = ret * 10 + c - '0';
        } while ((c = read()) >= '0' && c <= '9');
        if (neg) {
            return -ret;
        }
        return ret;
    }

    public double nextDouble() throws IOException {
        double ret = 0, div = 1;
        byte c = read();
        while (c <= ' ') {
            c = read();
        }
        boolean neg = (c == '-');
        if (neg) {
            c = read();
        }

        do {
            ret = ret * 10 + c - '0';
        } while ((c = read()) >= '0' && c <= '9');

        if (c == '.') {
            while ((c = read()) >= '0' && c <= '9') {
                ret += (c - '0') / (div *= 10);
            }
        }

        if (neg) {
            return -ret;
        }
        return ret;
    }

    private void fillBuffer() throws IOException {
```

```
            bytesRead = din.read(buffer, bufferPointer = 0, BUFFER_SIZE);
            if (bytesRead == -1) {
                buffer[0] = -1;
            }
        }

        private byte read() throws IOException {
            if (bufferPointer == bytesRead) {
                fillBuffer();
            }
            return buffer[bufferPointer++];
        }

        public void close() throws IOException {
            if (din == null) {
                return;
            }
            din.close();
        }
    }

    public static void main(String[] args) throws IOException {
        UltraReader s = new UltraReader();
        int n = s.nextInt();
        int k = s.nextInt();
        int count = 0;
        while (n-- > 0) {
            int x = s.nextInt();
            if (x % k == 0) {
                count++;
            }
        }
        System.out.println(count);
    }
}
```

C++ cin optimizado

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    // Se agregan las dos lineas de abajo permitiendo que CIN tenga la
    //velocidad de SCANF
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int n, k, t;
    int cnt = 0;
    cin >> n >> k;
    for (int i=0; i<n; i++)
    {
        cin >> t;
        if (t % k == 0)
            cnt++;
```

```
    }
    cout << cnt << "\n";
    return 0;
}
```

C++ INT FastScan

```
//Lectura rapida de numeros enteros (Solo valido para int)
#include <bits/stdc++.h>
using namespace std;

void fastscan(int &number) {
    //Variable para indicar el signo del número
    bool negative = false;
    register int c;
    number = 0;
    // Extraer el caracter actual de buffer
    c = getchar();
    if (c == '-') {
        // El numero es negativo
        negative = true;
        //Extraer el siguiente caracter del buffer
        c = getchar();
    }
    //se sigue extrayendo caracteres si son enteros
    // es decir su valor de ASCII esta entre '0'(48) y '9' (57)
    for (; (c > 47 && c < 58); c = getchar())
        number = number * 10 + c - 48;
    //si la entrada escaneada tiene signo negativo
    //se niega el valor del numero
    if (negative)
        number *= -1;
}

int main() {
    int number;
    fastscan(number);
    cout << number << "\n";
    return 0;
}
```

PYTHON FastInOut

```
# Se importa los modulos de lectura y escritura estandar
from sys import stdin
from sys import stdout
# Entrada de toda la linea
n = stdin.readline()
# Llenando una lista por medio de entrada de toda la linea
arr = [int(x) for x in stdin.readline().split()]
#Inicializamos variable
summation = 0
# Calcular sum
for x in arr:
    summation += x
```

```
# imprimir respuesta a través de write,
# el método write escribe solo strings
# por lo que necesitamos convertir cualquier
# dato en string para usarlo
stdout.write(str(summation))
```

PYTHON números en una sola línea

```
#Lectura de multiples numeros que estan en una misma linea
import sys
def get_ints(): return map(int, sys.stdin.readline().strip().split())
a,b,c,d = get_ints()
```

6.12) Importancia de las pruebas en la programación competitiva

Aunque la práctica es la única forma de garantizar un mayor rendimiento en los concursos de programación, tener algunos trucos bajo la manga garantiza la optimización y una depuración rápida.

Muchas veces al resolver problemas, enfrentamos problemas como el Tiempo límite excedido (Time limit Exceeded), la solución incorrecta (Wrong answer), el error de tiempo de ejecución (Runtime Error) y el límite de memoria excedido (Memory limit exceded) porque después de diseñar el algoritmo no probamos la eficiencia, la corrección, la complejidad del tiempo y la ejecución del algoritmo en un gran conjunto de entradas para fines de prueba.

Aquí es donde las pruebas de estrés vienen al rescate. La prueba de esfuerzo es la forma común de encontrar el error en un algoritmo.

Las pruebas de esfuerzo ayudan a encontrar un algoritmo eficiente y a corregir los problemas que puedan presentar los algoritmos, pero también ven por qué ciertos

enfoques no funcionan. En particular, es fácil esbozar soluciones codiciosas intuitivas a cualquier problema, pero tales soluciones a menudo no funcionan en la realidad.

- Busque el caso más sencillo del ejercicio y pruébelo
- Busque un caso intermedio y pruébelo
- Busque el caso más grande (Según las restricciones del ejercicio) y pruébelo.
- Compruebe si los resultados son los correctos, en caso de que no, realice una depuración paso a paso para verificar en que línea de código ha fallado el algoritmo.
- Si luego de estos pasos no logra resolver el problema, pruebe otro enfoque desde el principio
- No se desespere si no logra encontrar el error, la paciencia permite mejor flujo de ideas.

6.13) Ejercicios iniciales

Aquí podremos encontrar ejercicios de programación competitiva de nivel básico, realícelos con el fin de familiarizarse con ellos, puede usar traductor para entenderlos, pero procure realizarlos por mérito propio, luego de varios intentos, puede investigar la solución de los mismos en su buscador de confianza.

Para buscar la página de los ejercicios de CodeChef, utilice el siguiente link y cambie los "???" por el alias del ejercicio:

https://www.codechef.com/problems/???

1. **FLOW002**
2. **FLOW006**
3. **START01**
4. **FLOW001**
5. **INTEST**
6. **HS08TEST**
7. **FLOW007**
8. **LUCKFOUR**
9. **FLOW004**
10. **DIFFSUM**

Capitulo 7. Estructuras de datos

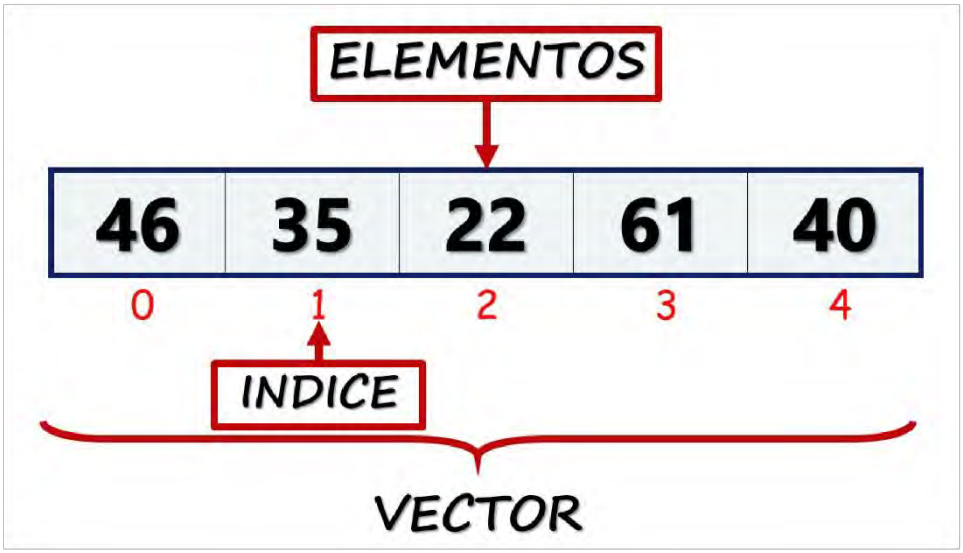
Una estructura de datos es una colección de valores, la relación que existe entre estos valores y las operaciones que podemos hacer sobre ellos se refieren a cómo los datos están organizados y cómo se pueden administrar. Una estructura de datos describe el formato en que los valores van a ser almacenados, cómo van a ser accedidos y modificados, pudiendo así existir una gran cantidad de estructuras de datos.

Lo habitual es que un vector tenga una cantidad fija de memoria asignada, aunque dependiendo del tipo de vector y del lenguaje de programación un vector podría tener una cantidad variable de datos. En este caso, se les denomina vectores dinámicos, en oposición, a los vectores con una cantidad fija de memoria asignada se los denomina vectores estáticos. El uso de vectores dinámicos requiere realizar una apropiada gestión de memoria dinámica. Un uso incorrecto de los vectores dinámicos, o mejor dicho, una mala gestión de la memoria dinámica, puede conducir a una fuga de memoria. Al utilizar vectores dinámicos siempre habrá que liberar la memoria utilizada cuando ésta ya no se vaya a seguir utilizando.

Lenguajes más modernos y de más alto nivel, cuentan con un mecanismo denominado recolector de basura que permiten que el programa decida si debe liberar el espacio basándose en si se va a utilizar en el futuro o no un determinado objeto.

Vector tamaño fijo (Array):

Vector



Guía del programador competitivo

Ilustración 7-1 Ejemplo de vector

Se le denomina vector o arreglo (en inglés Array) a una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila.

Estas estructuras de datos son adecuadas para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible. Por el contrario, si los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado utilizar una lista, ya que esta estructura puede cambiar de tamaño fácilmente durante la ejecución de un programa.

Vector tamaño dinámico o lista (ArrayList):

En programación, un arreglo dinámico o Array dinámico, es un Array de elementos que crece o mengua dinámicamente conforme los elementos se agregan o se eliminan. Se suministra como librerías estándar en muchos lenguajes modernos de programación.

Vector sin repetición (Set):

Es una colección desordenada de objetos en la que no se pueden almacenar valores duplicados.

Matriz (Matrix):

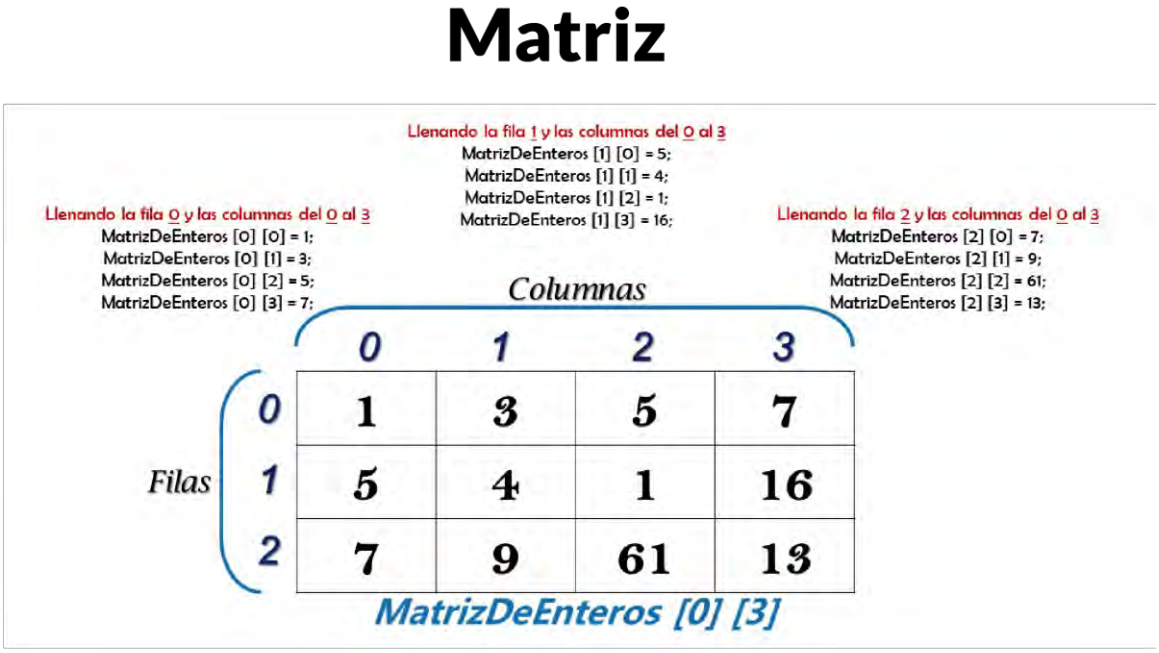


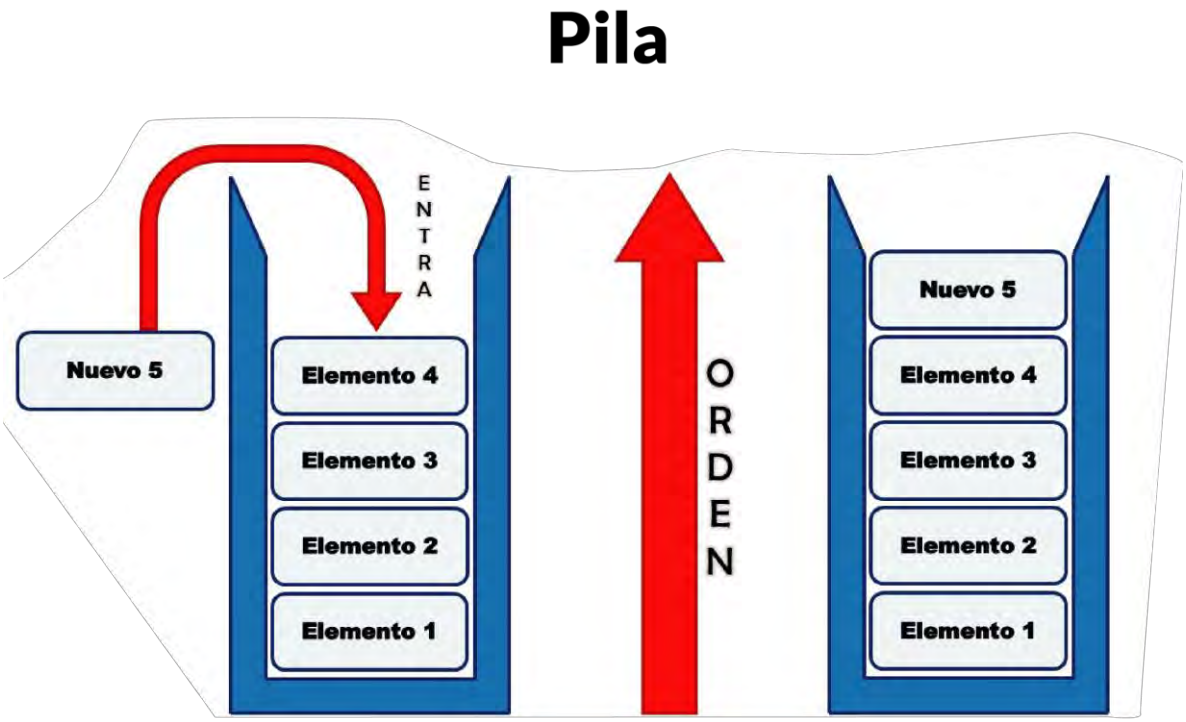
Ilustración 7-2 Ejemplo de matriz

Es una tabla bidimensional de números consistentes en cantidades abstractas con las que se pueden realizar diferentes operaciones, como por ejemplo la suma, multiplicación y descomposición de las mismas de varias formas, lo que también las hace un concepto clave en el campo del álgebra lineal. Las matrices se utilizan para describir sistema de ecuaciones lineales, realizar un seguimiento de los coeficientes de una aplicación lineal y registrar los datos que dependen de varios parámetros.

Diccionario o mapa (HashMap, TreeMap o Map):

Los mapas son contenedores asociativos que almacenan elementos de forma mapeada. Cada elemento tiene un valor clave y un valor asignado. No hay dos valores asignados que puedan tener los mismos valores clave.

Pila (Stack):



Guía del programador competitivo

Ilustración 7-3 Ejemplo de pila

Es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, el modo de acceso a sus elementos es de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»). Esta estructura se aplica en multitud de problemas en el área de informática debido a su simplicidad y capacidad de dar respuesta a numerosos procesos.

Cola (Queue):

Cola

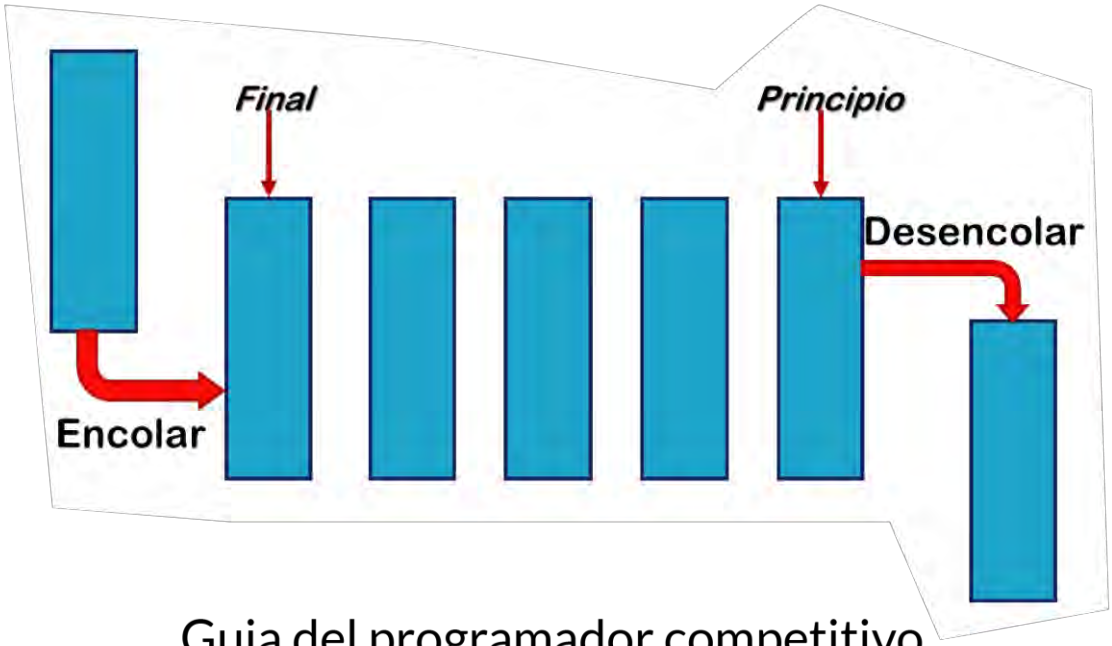


Ilustración 7-4 Ejemplo de cola

Es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.

JAVA

```
import java.util.*;

public class Main {

    public static Random r = new Random();
    public static int i, j;
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Inserte cantidad de datos");
        int n = sc.nextInt();
        System.out.println("Que estructura quiere utilizar? (Los números son
aleatorios");
        System.out.println("1) Vector fijo ");
        System.out.println("2) Vector dinámico");
        System.out.println("3) Vector sin repetición");
```

```
        System.out.println("4) Matriz");
        System.out.println("5) Diccionario");
        System.out.println("6) Pila");
        System.out.println("7) Cola");
        int aux = sc.nextInt();
        switch (aux) {
            case 1:
                vectorfijo(n);
                break;
            case 2:
                vectordinamico(n);
                break;
            case 3:
                vectorsinrepeticion(n);
                break;
            case 4:
                matriz(n);
                break;
            case 5:
                Diccionario(n);
                break;
            case 6:
                pila(n);
                break;
            case 7:
                cola(n);
                break;
        }
    }
    public static void vectordinamico(int n) {
        ArrayList<Integer> arreglo = new ArrayList<>();
        for (i = 0; i < n; i++) {
            arreglo.add(r.nextInt());
        }
        for (i = 0; i < n; i++) {
            System.out.println(arreglo.get(i));
        }
    }
    public static void vectorfijo(int n) {
        int[] números = new int[n];
        for (i = 0; i < n; i++) {
            números[i] = r.nextInt();
        }
        for (i = 0; i < n; i++) {
            System.out.println(números[i]);
        }
    }
    public static void matriz(int n) {
        int[][] matrix = new int[n][n];
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                matrix[i][j] = r.nextInt(50);
            }
        }
```

```
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println("");
        }
    }
    public static void vectorsinrepeticion(int n) {
        HashSet<Integer> sinrep = new HashSet<>();
        for (i = 0; i < n; i++) {
            sinrep.add(r.nextInt(50));
        }
        Iterator it = sinrep.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
    public static void Diccionario(int n) {
        TreeMap<String, String> dicc = new TreeMap<>();
        dicc.put("Programar", "Utilizacion de codigo para ejecutar un
programa");
        dicc.put("Futbol", "Deporte con una pelota y arcos");
        dicc.put("helado", "crema helada");
        dicc.put("Sargento mayor Johnson", "Practicar con palos y piedras");
        for (String concepto : dicc.keySet()) {
            String key = concepto;
            String value = dicc.get(concepto);
            System.out.println(key + "->" + value);
        }
    }
    public static void pila(int n) {
        Stack<Integer> mipila = new Stack<>();
        for (i = 0; i < n; i++) {
            mipila.push(r.nextInt(50));
        }
        while (!mipila.isEmpty()) {
            System.out.println(mipila.pop());
        }
    }
    public static void cola(int n) {
        Queue<Integer> micola = new LinkedList<>();
        for (i = 0; i < n; i++) {
            micola.offer(r.nextInt(50));
        }
        while (!micola.isEmpty()) {
            System.out.println(micola.poll());
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
```

```
int i, j;
void vectordinamico(int n) {
    vector<int> arreglo;
    for (i = 0; i < n; i++) {
        arreglo.push_back(rand() % 100);
    }
    for (i = 0; i < n; i++) {
        cout << arreglo.at(i) << endl;
    }

}

void vectorfijo(int n) {
    int números[n];
    for (i = 0; i < n; i++) {
        números[i] = rand() % 100;
    }
    for (i = 0; i < n; i++) {
        cout << números[i] << endl;
    }
}

void matriz(int n) {
    int matrix [n][n];
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            matrix[i][j] = rand() % 100;
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

void vectorsinrepeticion(int n) {
    set<int> sinrep;
    for (i = 0; i < n; i++) {
        sinrep.insert(rand() % 100);
    }
    for (auto itr = sinrep.begin(); itr != sinrep.end(); ++itr) {
        cout << *itr << endl;
    }
}

void Diccionario(int n) {

      map <string, int> mapa;

    int cantidad;
    cin>>cantidad;
    for (int i = 0; i < cantidad; i++) {
        string dato1;
```

```
        int dato2;
        cin >> dato1>>dato2;
        mapa.insert(type_pair(dato1, dato2));
    }
    cout << endl;
    for (type_pair j : mapa) {
        cout << j.first << " " << j.second << endl;
    }
}

void pila(int n) {
    stack<int> mipila;

    for (i = 0; i < n; i++) {
        mipila.push(rand() % 100);
    }
    while (!mipila.empty()) {
        cout << mipila.top() << endl;
        mipila.pop();
    }
}

void cola(int n) {
    queue<int> micola;
    for (i = 0; i < n; i++) {
        micola.push(rand() % 100);
    }
    while (!micola.empty()) {
        cout << micola.front() << endl;
        micola.pop();
    }
}

int main(int argc, char *argv[]) {
    cout << "Inserte cantidad de datos" << endl;
    int n;
    cin>>n;
    cout << "Que estructura quiere utilizar? (Los números son aleatorios" <<
endl;
    cout << "1) Vector fijo " << endl;
    cout << "2) Vector dinamico" << endl;
    cout << "3) Vector sin repeticion" << endl;
    cout << "4) Matriz" << endl;
    cout << "5) Diccionario" << endl;
    cout << "6) Pila" << endl;
    cout << "7) Cola" << endl;
    int aux;
    cin>>aux;
    switch (aux) {
        case 1:
            vectorfijo(n);
            break;
        case 2:
            vectordinamico(n);
```

```
            break;
        case 3:
            vectorsinrepeticion(n);
            break;
        case 4:
            matriz(n);
            break;
        case 5:
            Diccionario(n);
            break;
        case 6:
            pila(n);
            break;
        case 7:
            cola(n);
            break;
    }
    return 0;
}
```

PYTHON

```
from collections import deque
import random
def vector(n):
    arreglo = []
    for i in (0, n):
        arreglo.append(random.randint(0, 100))

    for i in (arreglo):
        print(i)

def matriz(n):
    matrix = []
    for i in range(n):
        row = []
        for j in range(n):
            row.append(random.randint(0, 100))
        matrix.append(row)
    for i in range(n):
        for j in range(n):
            print(matrix[i][j]+" ")
        print()

def vectorsinrepeticion(n):
    sinrep = {0}
    for i in range(n):
        sinrep.add(random.randint(0, 100))
    sinrep.remove(0)
    for i in sinrep:
        print(i)

def Diccionario(n):
    dicc = {}
```

```
        dicc['Programar'] = 'Utilizacion de codigo para ejecutar un programa
        dicc['Futbol'] = 'Deporte con una pelota y arcos'

        for concepto, definicion in dicc.items():
            print(f'{concepto} -> {definicion}')

def pila(n):
    mipila = []
    for i in range(n):
        mipila.append(random.randint(0, 100))
    while(len(mipila) != 0):
        print(mipila.pop())

def cola(n):
    micola = deque([])
    for i in range(n):
        micola.append(random.randint(0, 100))

    while(len(micola) != 0):
        print(micola.popleft())
print("Inserte cantidad de datos")
n = int(input())
#No existe switch en python
print("Vector")
vector(n)
print("Vector sin repetición")
vectorsinrepeticion(n)
print("Diccionario")
Diccionario(n)
print("Pila")
pila(n)
print("Cola")
cola(n);
print("Matriz")
matriz(n)
```

Capitulo 8. Ecuaciones, formulas e identidades matemáticas.

Igualdad matemática es la proposición de equivalencia existente entre dos expresiones algebraicas conectadas a través del signo = en la cual, ambas expresan el mismo valor.

La relación de igualdad establecida en una expresión de esta índole se emplea para denotar que dos objetos matemáticos expresan el mismo valor.

- 10 – 1 = 9

La igualdad matemática es una expresión que está formada por dos miembros. El miembro de la derecha, al lado izquierdo del signo igual y el miembro de la izquierda, al lado derecho del signo de igualdad. La solución del enunciado anterior nos revela el planteamiento de igualdad de las expresiones.

Una Identidad es una igualdad algebraica en la que aparecen números y letras que siempre se cumple, sean cuales sean los valores de las incógnitas.

Una ecuación es una igualdad algebraica que es cierta para algunos valores de las incógnitas y falsa para otros.

Por tanto, la diferencia entre identidad y ecuación es que la identidad siempre es cierta, mientras que la ecuación no.

El valor o valores de la incógnita que hacen que la igualdad se cumpla se llaman solución de la ecuación.

A continuación tenemos varias ecuaciones, igualdades e identidades conocidas.

- $A = \pi r^2$ Area del circulo
- $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ formula cuadrática
- $r = \frac{1}{2}at^2 + v_o t + r_0$ movimiento acelerado

- $\left(1+\frac{1}{n}\right)^n = e$ limite
- $\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}$ multiplicación de fracciones
- $\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$ Potencia de un cociente
- $(a^n)^m = a^{nm}$ Potencia de una potencia
- $a^n a^m = a^{n+m}$ Primera ley de los exponentes
- $(ab)^n = a^n b^n$ Segunda ley de los exponentes
- $\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$ Suma de fracciones
- $S = 4\pi r^2$ Area de la superficie de una esfera
- $A = \frac{1}{2}bh$ Area de un triángulo
- $V = \frac{4}{3}\pi r^3$ Volumen de una esfera
- $\sum_{i=1}^{n} \quad c = nc$ Sumatoria constante
- $\sum_{i=k}^{n} \quad c = (n-k+1)c$ Sumatoria constante desde k
- $\sum_{i=1}^{n} \quad i = \frac{n(n+1)}{2}$ Sumatoria de 1 hasta n
- $\sum_{i=0}^{n} \quad i^2 = \frac{n(n+1)(2n+1)}{6}$ Sumatoria de los cuadrados de 0 a n
- $\sum_{i=0}^{n} \quad i^3 = (\frac{n(n+1)}{2})^2$ Sumatoria de los cubos de 0 a n
- $\sum_{i=0}^{n} \quad i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$ Sumatorias de las potencias de 4 de 0 a n
- $\sum_{i=0}^{n} \quad 2^i = 2^{n+1} - 1$ Sumatoria de las potencias de 2 de 0 a n
- $\sum_{i=0}^{n} \quad a^i = \frac{1+a^{n+1}}{1-a}$ Sumatoria de las potencias de a desde 0 a n
- $a = \frac{a}{b}$
- $log\ log\ \frac{a}{b} = log\ log\ a\ - log\ log\ b$
- $log\ log\ (axb) = log\ log\ a + log\ log\ b$

- $G = \left(\frac{x1+x2+x3}{3}, \frac{y1+y2+y3}{3}\right)$Baricentro de un triángulo
- $P(n,r) = \frac{n!}{(n-r)!}$ Número de permutaciones de n objetos tomando r objetos sin repetición
- $C(n,r) = \frac{n!}{r!(n-r)!}$Número de combinaciones de n objetos tomando r objetos
- $P(n,r) = n^r$Número de permutaciones de n objetos tomando r objetos con repetición
- $C_n = \frac{(2n)!}{(n+1)!n!}$ n Número de Catalan
- $\frac{a}{Sin(A)} = \frac{b}{Sin(B)} = \frac{c}{Sin(C)}$ Ley del seno
- $c^2 = a^2 + b^2 - 2ab * Cos(C)$ Ley del coseno

Capitulo 9. Algoritmos de Búsquedas

En las ciencias de la computación, los algoritmos de búsqueda son un conjunto de instrucciones que se encuentran diseñados para localizar un elemento con ciertas características dentro de una muestra, en estos casos dentro de una estructura de datos con el fin de utilizar este elemento o simplemente demostrar que existe dentro de estas muestras.

9.1) Binary Search

Ilustración 9-1 Ejemplo de búsqueda binaria

La búsqueda binaria, también conocida como búsqueda de intervalo medio o búsqueda logarítmica, es un algoritmo de búsqueda que encuentra la posición de un valor en un arreglo ordenado. Compara el valor con el elemento en el medio del arreglo, si no son iguales, la mitad en la cual el valor no puede estar es eliminada y la búsqueda sigue en la mitad restante hasta que el valor se encuentre.

La búsqueda binaria es computada en el peor de los casos en un tiempo logarítmico, realizando O (log n) comparaciones, donde n es el número de elementos del arreglo y log es el logaritmo.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(log n) **Promedio:** O(log n)

JAVA

```java
// Implementación Java de una búsqueda binaria recursiva

public class BinarySearch {
    /*Retorna el indice de x si se encuentra presente
    en arr[1,2,....r]*, si no retorna -1*/
    static int binarySearch(int arr[], int l, int r, int x) {
        if (r >= l) {
            int mid = l + (r - l) / 2;
            /*Si el elemento esta presente en el medio*/
            if (arr[mid] == x) {
                return mid;
            }
            /*Si el elemento es más pequeño que la mitad
            entonces solo puede estar presente en el
            subarreglo izquierdo*/
            if (arr[mid] > x) {
                return binarySearch(arr, l, mid - 1, x);
            }
            //Si no el elemento solo pued estar presente
            //en el subarreglo derecho
            return binarySearch(arr, mid + 1, r, x);
        }
        //Si llegamos aqui, el elemento no esta presente
        //en el arreglo
        return -1;
    }

    public static void main(String args[]) {
        int arr[] = {2, 3, 4, 10, 40};
        int n = arr.length;
        int x = 10;
        int result = binarySearch(arr, 0, n - 1, x);
        if (result == -1) {
            System.out.println("Elemento no presente");
        } else {
            System.out.println("Element encontrado en la posición " + result);
        }
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
#include <cstdlib>

int binarysearch(int arr[], int left, int r, int x) {
    if (r >= left) {
        int mid = left + (r - left) / 2;
        if (arr[mid] == x) {
            return mid;
        }
        if (arr[mid] > x) {
            return binarysearch(arr, left, mid - 1, x);
        } else {
            return binarysearch(arr, mid + 1, r, x);
        }
    }
    return -1;
}

int main() {
    int arr[] = {1, 2, 3, 5, 6};
    int n = sizeof (arr) / sizeof (arr[0]);
    int x = 3;
    int resultado = binarysearch(arr, 0, n - 1, x);
    if (resultado == -1) {
        printf("elemento no presente");
    } else {
        printf("elemento encontrado en el indice: %d", resultado);
    }
    return 0;
}
```

PYTHON

```python
def BinarySearch(arr, inicio, fin, numerobuscado):
    if(fin >= inicio):
        medio = int(inicio + (fin-1) / 2)
        if(arr[medio] == numerobuscado):
            return medio
        if(arr[medio] > numerobuscado):
            return BinarySearch(arr, inicio, medio-1, numerobuscado)
        else:
            return BinarySearch(arr, medio + 1, fin, numerobuscado)
    return -1

arr = [1, 2, 3, 4, 5, 6]
tam = len(arr)
numerobuscado = 5
resultado = int (BinarySearch(arr, 0, tam-1, numerobuscado))
if(resultado == -1):
    print("Elemento no encontrado")
else:
    print("Elemento encontrado en la posicion", resultado)
```

9.2) Exponential Search

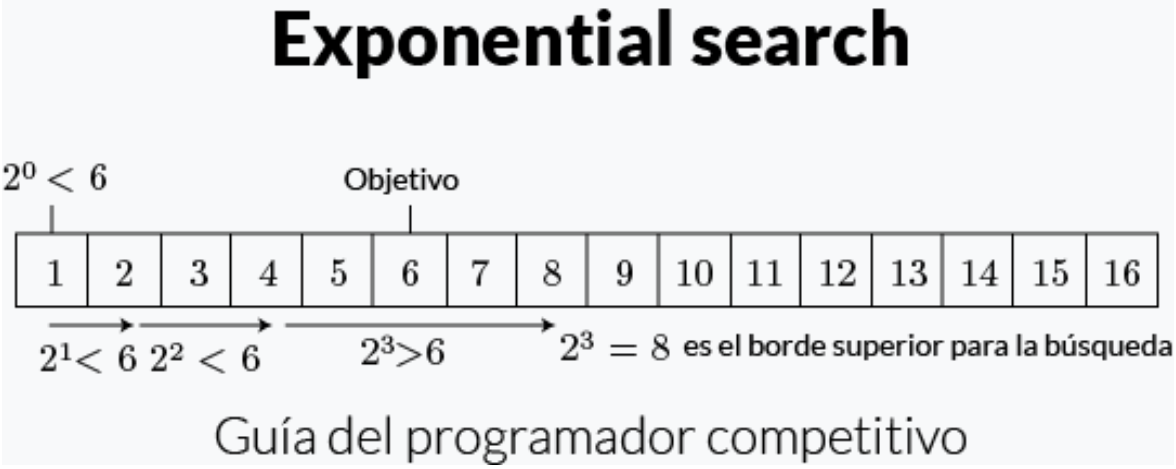


Ilustración 9-2 Buscando un dato usando exponential search

El nombre de este algoritmo de búsqueda puede ser engañoso, ya que funciona en tiempo O (Log n). El nombre proviene de la forma en que busca un elemento. La búsqueda binaria exponencial es particularmente útil para búsquedas ilimitadas, donde el tamaño del vector es infinito.

La búsqueda exponencial implica dos pasos:

1) Encuentra el rango donde el elemento está presente
2) Hacer una búsqueda binaria en el rango encontrado arriba.

La idea es comenzar con el tamaño del subarreglo 1, comparar su último elemento con x, luego probar el tamaño 2, luego el 4 y así sucesivamente hasta que el último elemento de un subconjunto no sea mayor que el tamaño del vector.

Una vez que encontramos un índice i (después de duplicar repetidamente i), sabemos que el elemento debe estar presente entre i / 2 - i, i / 2 Porque se encontró un valor mayor en la iteración anterior.

Podemos tener en cuenta que:

- La búsqueda binaria exponencial es particularmente útil para búsquedas ilimitadas, donde el tamaño del vector es infinito.
- Funciona mejor que la Búsqueda binaria para vectores limitados, y también cuando el elemento a buscar está más cerca del primer elemento.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(log n) **Promedio:** O(log n)

JAVA

```java
//Programa Java para encontrar un elemento
//x en un array ordenado usando Exponential Search

import java.util.Arrays;

public class ExponentialSearch {

    /* Retorna posición de la primerra ocurrencia
    de x en un arreglo*/
    static int exponentialSearch(int arr[],int n, int x) {
        // Si z esta presente en la primera localización en si misma
        if (arr[0] == x) {
            return 0;
        }
        /*Encuentra ranfo para la busqueda binaria
        por repetidos dobles*/
        int i = 1;
        while (i < n && arr[i] <= x) {
            i = i * 2;
        }
        //llama la busqueda binaria para el rango encontrado
        return Arrays.binarySearch(arr, i / 2,
                Math.min(i, n), x);
    }

    public static void main(String args[]) {
        int arr[] = {2, 3, 4, 10, 40};
        int x = 10;
        int result = exponentialSearch(arr, arr.length, x);
        System.out.println((result < 0)
                ? "El elemento no esta presente en el array"
                : "Elemento encontrado en : "
                + result);
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

int binarySearch(int arr[], int l, int r, int x) {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x) {
            return mid;
        }
        if (arr[mid] > x) {
            return binarySearch(arr, l, mid - 1, x);
        } else {
            return binarySearch(arr, mid + 1, r, x);
        }
    }

    return -1;
}

int exponentialSearch(int arr[], int n, int x) {
    if (arr[0] == x) {
        return 0;
    }
    int i = 1;
    while (i < n && arr[i] <= x) {
        i = i * 2;
    }
    return binarySearch(arr, i / 2, min(i, n), x);
}

int main(int argc, char const *argv[]) {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof (arr) / sizeof (arr[0]);
    int x = 5;
    int result = exponentialSearch(arr, n, x);
    if (result < 0) {
        cout << "El resultado no esta presente en el Array" << endl;
    } else {
        cout << "Elemento encontrado en el indice " << result << endl;
    }

    return 0;
}
```

PYTHON

```python
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write
```

```
def binary_Search(arr, left, right, x):
    if right >= left:

        mid = left + (right - left) // 2

        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_Search(arr, left, mid-1, x)
        else:
            return binary_Search(arr, mid + 1, right, x)

    else:
        return -1


def exponential_Search(arr, n, x):
    if arr[0] == x:
        return 0

    i = 1
    while i < n and arr[i] <= x:
        i = i * 2

    return binary_Search(arr, i // 2, min(i, n), x)


arr = list(map(int, rl().split()))
x = int(rl())
result = exponential_Search(arr, len(arr)-1, x)

if result == -1:
    wr('Dato no encontrado')
else:
    wr(f'Dato encontrado en la posicion {result}')
```

9.3) Array Max/Min Search

Ilustración 9-3 El mas grande y el mas pequeño en un arreglo

Tenemos un Array en donde queremos buscar el valor más pequeño y el valor más grande del mismo de forma eficiente haciendo la menor cantidad de comparaciones, para resolver este problema podemos emplear la siguiente aproximación.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
/* Programa Java que busca el máximo y el mínimo
en un array*/
public class ArrayMaxMinSearch {
    /*Clase Pair es usada para retornar
    dos valores de getMinMax()*/
    static class Pair {
        int min;
```

```
        int max;
    }

    static Pair getMinMax(int arr[], int n) {
        Pair minmax = new Pair();
        int i;
        /*Si es el unico elemento, lo retorna como min y max*/
        if (n == 1) {
            minmax.max = arr[0];
            minmax.min = arr[0];
            return minmax;
        }
        /*Si hay más de un elemento,
        entones inicializa min y más*/
        if (arr[0] > arr[1]) {
            minmax.max = arr[0];
            minmax.min = arr[1];
        } else {
            minmax.max = arr[1];
            minmax.min = arr[0];
        }
        for (i = 2; i < n; i++) {
            if (arr[i] > minmax.max) {
                minmax.max = arr[i];
            } else if (arr[i] < minmax.min) {
                minmax.min = arr[i];
            }
        }
        return minmax;
    }

    public static void main(String args[]) {
        int arr[] = {1000, 11, 445, 1, 330, 3000};
        int arr_size = 6;
        Pair minmax = getMinMax(arr, arr_size);
        System.out.printf("\nEl minimo elemento es %d", minmax.min);
        System.out.printf("\nEl maximo elemento es %d", minmax.max);
    }

}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

struct pairMaxMin {
    int min;
    int max;
};

static pairMaxMin getMinMax(int arr[], int n) {
    pairMaxMin minmax;
    int i;
```

```
    if (n == 1) {
        minmax.max = arr[0];
        minmax.min = arr[0];
        return minmax;
    }
    if (arr[0] > arr[1]) {
        minmax.max = arr[0];
        minmax.min = arr[1];
    } else {
        minmax.max = arr[1];
        minmax.min = arr[0];
    }
    for (i = 2; i < n; i++) {
        if (arr[i] > minmax.max) {
            minmax.max = arr[i];
        } else if (arr[i] < minmax.min) {
            minmax.min = arr[i];
        }
    }

    return minmax;
}

int main(int argc, char** argv) {
    int arr[] = {12, 3, 4, 345, 65, 43};
    int len = sizeof (arr) / sizeof (arr[0]);
    pairMaxMin minmax = getMinMax(arr, len);
    printf("el minimo elemento es %d\n", minmax.min);
    printf("el maximo elemento es %d\n", minmax.max);
    return 0;
}
```

PYTHON

```
class Pair:
    max,min = int(),int()

def  getmaxmin(lista,n):
    maxmin = Pair()
    i = int()
    if n==1:
        maxmin.max = lista[0]
        maxmin.max = lista[0]
        return maxmin
    if lista[0]>lista[1]:
        maxmin.max = lista[0]
        maxmin.min = lista[1]
    else:
        maxmin.max = lista[1]
        maxmin.min = lista[0]
    for i in range(2,n):
        if lista[i]>maxmin.max:
            maxmin.max = lista[i]
        elif lista[i]<maxmin.min:
```

```
            maxmin.min = lista[i]
    return maxmin

lista = [1, 422, 789, 2333, 2444]
n = len(lista)
resultado = getmaxmin(lista,n)
print("El minimo es ",resultado.min)
print("El maximo es ",resultado.max)
```

9.4) Fibonacci Search

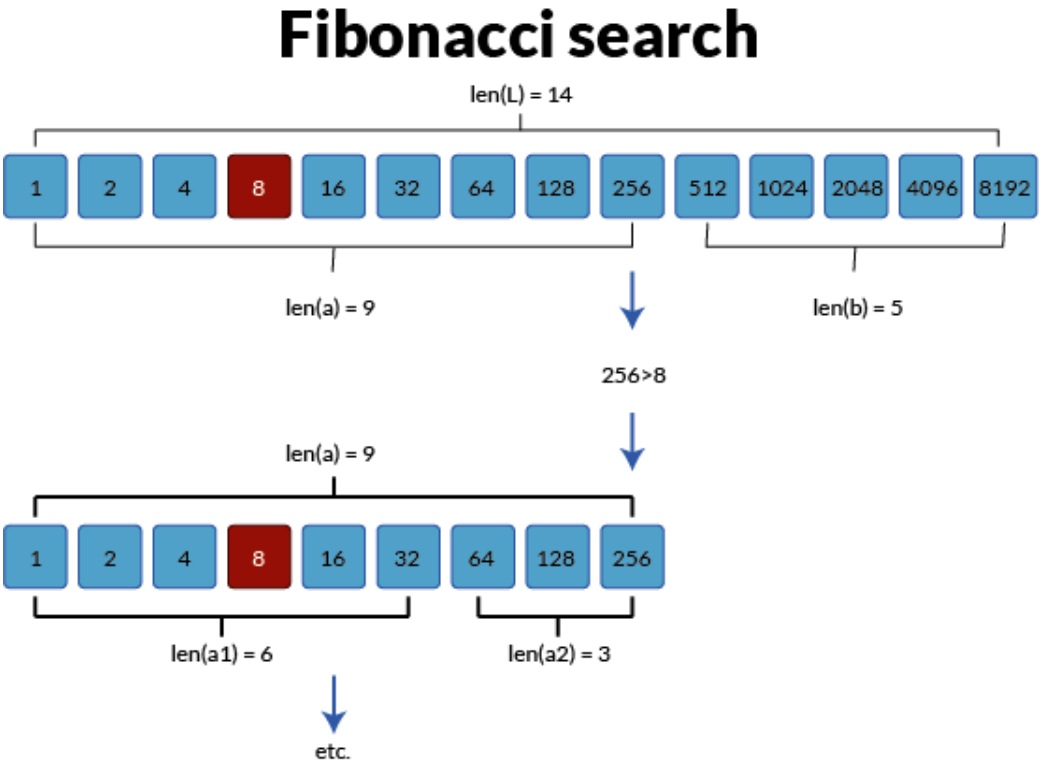


Ilustración 9-4 Busqueda de un dato usado Fibonacci search

La búsqueda de Fibonacci es una técnica basada en la comparación que utiliza los números de Fibonacci para buscar un elemento en un array ordenado.

Similitudes con la búsqueda binaria:

- Trabaja con arrays ordenados.
- Es un algoritmo de dividir y conquistar.
- Tiene una complejidad de registro y tiempo.

Diferencias con la búsqueda binaria:

- La búsqueda de Fibonacci divide un array dado en partes desiguales
- La búsqueda binaria usa el operador de división para dividir el rango. La búsqueda de Fibonacci no usa /, pero usa + y -. El operador de la división puede ser costoso en algunas CPU.
- La búsqueda de Fibonacci examina elementos relativamente más cercanos en pasos subsiguientes. Por lo tanto, cuando el vector de entrada es grande y no cabe en la memoria caché de la CPU o incluso en la RAM, la búsqueda de Fibonacci puede ser útil.
- La idea es encontrar primero el número de Fibonacci más pequeño que sea mayor o igual a la longitud del vector dado.

Complejidad de tiempo

Mejor caso : O(√ n) **Peor caso :** O(√ n) **Promedio:** O(√ n)

JAVA

```
// Programa Java para Fibonacci Search
public class FibonacciSearch {

    //Función de utilidad para biscar el minimo
    //de dos elementos
    public static int min(int x, int y) {
        return (x <= y) ? x : y;
    }

    /* Regresa el indice de x si esta presente, si no retorna -1*/
    public static int fibonacciSearch(int arr[],
            int x, int n) {
        /* Inicializa los números fibonacci*/
        int fibMMm2 = 0; // (m-2)esimo número fibonacci.
        int fibMMm1 = 1; // (m-1)esimo número fibonacci.
        int fibM = fibMMm2 + fibMMm1; // mesimo fibonacci.
        /* FibM va a ser almacenado como el más pequeño
        fibonacci más grande o igual a n*/
```

```
        while (fibM < n) {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
        }
        // Marca el rango eliminado del frente
        int offset = -1;
        /* Si bien hay elementos a inspeccionar.
        Tenga en cuenta que comparamos arr [fibMm2] con x.
        Cuando fibM se convierte en 1, fibMm2 se convierte en 0 */
        while (fibM > 1) {
            //Revisa si fibMm2 es una ubicación valida
            int i = min(offset + fibMMm2, n - 1);
            /*Si x es más grande que el valor
            en el indice fibMm2, fuera del subarray
            desde offset a i*/
            if (arr[i] < x) {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }
            /*Si x es más grande que el valor en indice
            fibMm2, corta el subarreglo despues de i+1*/
            else if (arr[i] > x) {
                fibM = fibMMm2;
                fibMMm1 = fibMMm1 - fibMMm2;
                fibMMm2 = fibM - fibMMm1;
            } /* Elemento encontrado, retorna indice*/
            else {
                return i;
            }
        }
        /* Comparando el ultimo elemento con x*/
        if (fibMMm1 == 1 && arr[offset + 1] == x) {
            return offset + 1;
        }
        /*Elemento no encontrado, retorna -1*/
        return -1;
    }

    public static void main(String[] args) {
        int arr[] = {10, 22, 35, 40, 45, 50,
            80, 82, 85, 90, 100};
        int n = 11;
        int x = 85;
        System.out.println("Found at index: "
                + fibonacciSearch(arr, x, n));
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
```

```
using namespace std;

static int fibonacciSearch(int arr[], int x, int n) {
    int fibMMm2 = 0;
    int fibMMm1 = 1;
    int fibM = fibMMm2 + fibMMm1;
    while (fibM < n) {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }
    int offset = -1;
    while (fibM > 1) {
        int i = std::min(offset + fibMMm2, n - 1);
        if (arr[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        } else if (arr[i] > x) {
            fibM = fibMMm2;
            fibMMm1 -= fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        } else {
            return i;
        }
    }
    if (fibM == 1 && arr[offset + 1] == x) {
        return offset + 1;
    }
    return -1;
}

int main(int argc, char** argv) {
    //
    int arr[] = {10, 22, 35, 40, 45, 50, 80, 82, 85, 90, 100};
    int n = sizeof (arr) / sizeof (arr[0]);
    int x = 85;
    printf("Encontrado en la posicion %d", fibonacciSearch(arr, x, n));
    return 0;
}
```

PYTHON

```
def menor(x, y):
    return min(x, y)

def FibonacciSearch(lista, x, n):
    if x > lista[n-1]:
        return -1
    Fm2 = int(0)
    Fm1 = int(1)
    Fms = int(Fm2 + Fm1)
    while Fms < n:
        Fm2 = Fm1
```

```
        Fm1 = Fms
        Fms = int(Fm2 + Fm1)
    offset = int(-1)
    while Fms > 1:
        i = int(menor(offset + Fm2, n - 1))
        if lista[i] < x:
            Fms = Fm1
            Fm1 = Fm2
            Fm2 = Fms - Fm1
            offset = i
        elif lista[i] > x:
            Fms = Fm2
            Fm1 = Fm1 - Fm2
            Fm2 = Fms - Fm1
        else:
            return i

    if Fm1 == 1 and lista[offset + 1] == x:
        return offset + 1
    else:
        return -1


lista = [10, 22, 33, 45, 89, 99, 100]
n = int(len(lista))
x = int(input())
if FibonacciSearch(lista, x, n) != -1:
    print("Encontrado en la poscion ", FibonacciSearch(lista, x, n))
else:
    print("No encontrado")
```

9.5) Jump Search

Jump Search

Guía del programador competitivo

Ilustración 9-5 Ejemplo búsqueda por saltos

Al igual que Binary Search, Jump Search es un algoritmo de búsqueda para vectores ordenados. La idea básica es verificar menos elementos (que la búsqueda lineal) saltando hacia delante con pasos fijos u salteando algunos elementos en lugar de buscar todos los elementos.

En el peor de los casos, tenemos que realizar saltos de n / m y, si el último valor verificado es mayor que el elemento a buscar, realizamos comparaciones de m-1 más para la búsqueda lineal. Por lo tanto, el número total de comparaciones en el peor de los casos será ((n / m) + m-1). El valor de la función ((n / m) + m-1) será mínimo cuando m = √n. Por lo tanto, el mejor tamaño de paso es m = √n.

Puntos importantes:

- Trabaja solo arreglos ordenados.
- El tamaño óptimo de un bloque a saltar es (√ n). Esto hace que la complejidad del tiempo de Jump Search O (√ n).

- La complejidad temporal de la búsqueda por salto es entre la búsqueda lineal ((O (n)) y la búsqueda binaria (O (Log n)).
- La búsqueda binaria es mejor que la búsqueda por salto, pero la búsqueda por salto tiene la ventaja de que retrocedemos solo una vez (la búsqueda binaria puede requerir saltos O (Log n), considere una situación en la que el elemento a buscar es el elemento más pequeño o más pequeño que el más pequeño). Entonces, en un sistema donde el salto hacia atrás es costoso, usamos Jump Search.

Complejidad de tiempo

Mejor caso : O(√ n) **Peor caso :** O(√ n) **Promedio:** O(√ n)

JAVA

```java
//Programa Java que implementa Jump Search
public class JumpSearch {

    public static int jumpSearch(int[] arr, int x) {
        int n = arr.length;
        //Buscando el tamaño del bloque que sera
        //saltado
        int step = (int) Math.floor(Math.sqrt(n));
        /* Buscando el bloque donde el elemento
        esta presente (Si esta presente)*/
        int prev = 0;
        while (arr[Math.min(step, n) - 1] < x) {
            prev = step;
            step += (int) Math.floor(Math.sqrt(n));
            if (prev >= n) {
                return -1;
            }
        }
        /*Realizando una busqueda linear para x en
        el bloque empezando con prev*/
        while (arr[prev] < x) {
            prev++;
            /*Si nosotros alcanzamos el siguiente bloque
            o el fin del array el elemento no esta presente*/
            if (prev == Math.min(step, n)) {
                return -1;
            }
        }
        // Si el elemento fue encontrado
        if (arr[prev] == x) {
            return prev;
```

```
        }
        return -1;
    }

    public static void main(String[] args) {
        int arr[] = {0, 1, 1, 2, 3, 5, 8, 13, 21,
            34, 55, 89, 144, 233, 377, 610};
        int x = 55;
        //Encontrar el indice de 'x' usando Jump Search
        int index = jumpSearch(arr, x);
        // Imprime el indice donde x fue encontrado
        System.out.println("\nNúmero " + x
                + " esta en el indice " + index);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

static int jumpSearch(int arr[], int x, int n) {
    int step = int(std::floor(std::sqrt(n)));
    int prev = 0;
    while (arr[std::min(step, n) - 1] < x) {
        prev = step;
        step += int(std::floor(std::sqrt(n)));
        if (prev >= n) {
            return -1;
        }
    }
    while (arr[prev] < x) {
        prev++;
        if (prev == std::min(step, n)) {
            return -1;
        }
    }
    if (arr[prev] == x) {
        return prev;
    }
    return -1;

}

int main(int argc, char const *argv[]) {
    int arr[] = {1, 2, 3, 4, 5};
    int n = sizeof (arr) / sizeof (arr[0]);
    int x = 5;
    int result = jumpSearch(arr, x, n);
    if (result < 0) {
        cout << "El elemento no esta presente en el Array" << endl;
    } else {
        cout << "Elemento encontrado en el indice " << result << endl;
    }
```

```
    return 0;
}
```

PYTHON

```
import math

def jumpSearch(lista, x):
    tamanio = len(lista)
    salto = int(math.floor(math.sqrt(tamanio)))
    previo = int(0)
    while(lista[min(salto, tamanio)-1] < x):
        previo = salto
        salto = salto + int(math.floor(math.sqrt(tamanio)))
        if(previo >= tamanio):
            return -1
    while(lista[previo] < x):
        previo = previo + 1
        if(previo == min(salto, tamanio)):
            return -1
    if(lista[previo] == x):
        return previo
    return -1

lista = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
numeroabuscar = int(input())
resultado = jumpSearch(lista, numeroabuscar)
if(resultado == -1):
    print('No encontrado')
else:
    print('Encontrado en el indice ', resultado)
```

9.6) Minimum Absolute Sum Pair

Teniendo un vector de números, se buscan dos elementos cuya suma es la más cercana a cero. Para cada elemento de dicho vector encuentre la suma con cada otro elemento del mismo vector y compare las sumas. Por último, devolver la suma mínima encontrada.

Complejidad de tiempo

Mejor caso : $O(n^2)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```java
//Programa JAVA que busque el par de números
//que sumados de la minima suma absoluta

public class MinAbsSumPair {
//función que busca el par

    static void minAbsSumPair(int arr[], int arr_size) {
        int l, r, min_sum, sum, min_l, min_r;
        /*El array debe tener al menos 2 elementos*/
        if (arr_size < 2) {
            System.out.println("Invalid Input");
            return;
        }
        /* Inicialización de los valores*/
        min_l = 0;
        min_r = 1;
        min_sum = arr[0] + arr[1];
        for (l = 0; l < arr_size - 1; l++) {
            for (r = l + 1; r < arr_size; r++) {
                sum = arr[l] + arr[r];
                if (Math.abs(min_sum) > Math.abs(sum)) {
                    min_sum = sum;
                    min_l = l;
                    min_r = r;
                }
            }
        }
        System.out.println(" Los dos elementos los cuales "
                + "tienen la suma minima son "
                + arr[min_l] + " y " + arr[min_r]);
    }

    public static void main(String[] args) {
        int arr[] = {1, 60, -10, 70, -80, 85};
        minAbsSumPair(arr, 6);
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

void minAbsSumPair(int arr[], int n) {
    int l, r, min_sum, sum, min_l, min_r;
    if (n < 2) {
        printf("Entrada no valida");
    }
    min_l = 0;
    min_r = 1;
    min_sum = arr[0] + arr[1];
```

```
    for (l = 0; l < n - 1; ++l) {
        for (r = l + 1; r < n; ++r) {
            sum = arr[l] + arr[r];
            if (std::abs(min_sum) > std::abs(sum)) {
                min_sum = sum;
                min_l = l;
                min_r = r;
            }
        }
    }
    printf("Los numero que tienen la minima suma son %d %d", arr[min_l],
arr[min_r]);
}

int main(int argc, char const *argv[]) {
    int arr[] = {1, -2, -10, 70, -80, 85};
    int n = sizeof (arr) / sizeof (arr[0]);
    minAbsSumPair(arr, n);
    return 0;
}
```

PYTHON

```
def minAbsSumPair(arr,n):
    l,r,min_sum,sum,min_l,min_r = int(),int(),int(),int(),int(),int()
    if n<2:
        print("Array no tiene suficientes valores")
        return
    min_l = int(0)
    min_r = int (1)
    min_sum = arr[0]+arr[1]
    for l in range(n-1):
        for r in range(l+1,n):
            sum = arr[l]+arr[r]
            if abs(min_sum)>abs(sum):
                min_sum=sum
                min_l=l
                min_r=r
    print(f"la minima suma es: {arr[min_l]} y {arr[min_r]}")

arr = [1,60,-10,70,-80,85]
minAbsSumPair(arr,len(arr))
```

9.7) Missing Number Search

Ilustración 9-6 Búsqueda del número perdido

Se le da una lista de n-1 enteros y estos enteros están en el rango de 1 a n. No hay duplicados en la lista pero nos falta uno de los enteros en la lista.

Los pasos para resolver este problema son los siguientes:

1) XOR todos los elementos del array, que el resultado de XOR sea X1.
2) XOR todos los números del 1 al n, sea XOR sea X2.
3) XOR de X1 y X2 da el número que falta, es decir el resultado.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
public class MissingNo {

    public static void main(String[] args) {
        int[] arr = {1,2,3,5,6,7,8};
        int missed = getMissingNo(arr, arr.length);
        System.out.println("el numero que falta es: "+ missed);
    }

    static int getMissingNo(int a[], int n) {
        int x1 = a[0];
        int x2 = 1;
        for (int i = 1; i < n; i++) {
            x1 ^= a[i];
```

```
        }
        for (int i = 2; i <= n + 1; i++) {
            x2 ^= i;
        }
        return x1 ^ x2;
    }

}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

static int MissingNoSearch(int arr[], int n) {
    int x1 = arr[0];
    int x2 = 1;
    for (int i = 1; i < n; ++i) {
        x1 = x1^arr[i];
    }
    for (int i = 2; i <= n + 1; ++i) {
        x2 = x2^i;
    }
    return (x1^x2);
}

int main(int argc, char const *argv[]) {
    int arr[] = {1, 3, 4, 5, 6, 7};
    int n = sizeof (arr) / sizeof (arr[0]);
    printf("El numero faltante es : %d", MissingNoSearch(arr, n));
    return 0;
}
```

PYTHON

```
def getMissingNo(a,n):
    x1 = a[0]
    x2 = int(1)
    for i in range(1,n):
        x1^=a[i]
    for i in range(2,n+2):
        x2^=i
    return x1^x2

lista = [1,2,3,4,5,7,8]
missed = getMissingNo(lista,len(lista))
print("Numero perdido: "+ str(missed))
```

9.8) Difference Pair Search

Dada un vector sin orden y un número n, busque si existe un par de elementos en el vector cuya diferencia es n.

El método más simple es ejecutar dos bucles, el bucle externo selecciona el primer elemento (elemento más pequeño) y el bucle interno busca el elemento seleccionado por el bucle externo más n. La complejidad del tiempo de este método es O (n ^ 2).

Podemos usar el ordenamiento y la búsqueda binaria para mejorar la complejidad del tiempo a O (nLogn). El primer paso es ordenar el array en orden ascendente. Una vez que el array esté ordenado, recorra el array de izquierda a derecha y, para cada elemento arr [i], la búsqueda binaria de arr [i] + n en arr [i + 1...n-1]. Si se encuentra el elemento, devuelva el par.
Tanto el primer como el segundo paso toman O (nLogn). Así que la complejidad global es O (nLogn).

El segundo paso del algoritmo anterior se puede mejorar a O (n). El primer paso sigue siendo el mismo. La idea para el segundo paso es tomar dos variables de índice i y j, inicializarlas como 0 y 1 respectivamente. Ahora ejecuta un bucle lineal. Si arr [j] - arr [i] es más pequeño que n, debemos buscar un arr mayor [j], así que aumente j. Si arr [j] - arr [i] es mayor que n, debemos buscar un arr mayor [i], entonces incremente i.

Complejidad de tiempo

Mejor caso : O(nLog(n)) **Peor caso :** O(nLog(n)) **Promedio:** O(nLog(n))

JAVA

```
// Programa Java para buscar un par
// Dada una diferencia
```

```java
public class PairDifferenceSearch {
    //La función asume que el array esta ordenado
    static boolean findPair(int arr[], int n) {
        int size = arr.length;
        // Inicializa la posición de dos elementos
        int i = 0, j = 1;
        // Busca por el par
        while (i < size && j < size) {
            if (i != j && arr[j] - arr[i] == n) {
                System.out.print("Par Encontrado: "
                        + "( " + arr[i] + ", " + arr[j] + " )");
                return true;
            } else if (arr[j] - arr[i] < n) {
                j++;
            } else {
                i++;
            }
        }
        //No encuentra el par
        System.out.print("No hay tal par");
        return false;
    }

    public static void main(String[] args) {
        int arr[] = {1, 8, 30, 40, 100};
        int n = 60;
        findPair(arr, n);
    }
}
```

C++

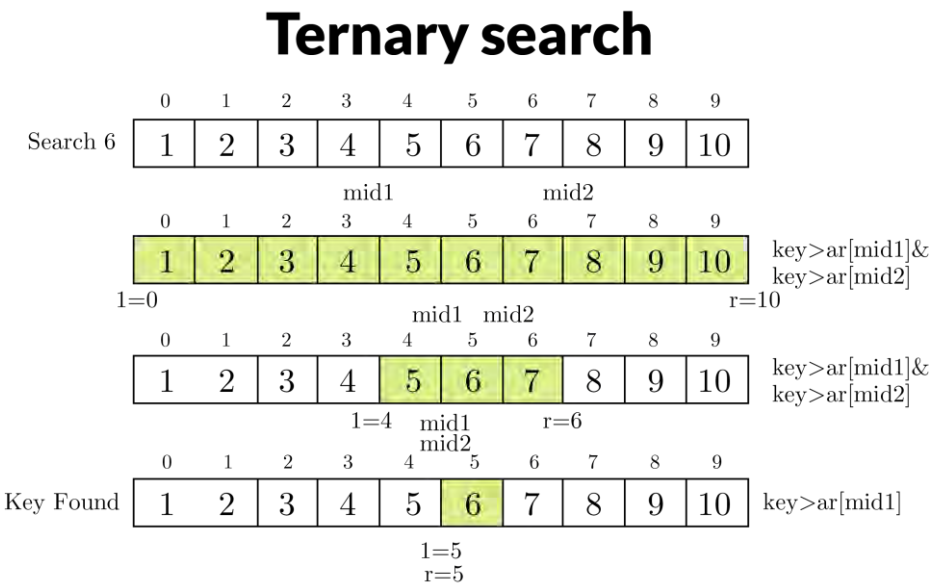
```cpp
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

static bool findPair(int arr[], int n, int size) {
    int i = 0, j = 1;
    while (i < size && j < size) {
        if (i != j && arr[j] - arr[i] == n) {
            printf("par encontrado (%d %d)", arr[i], arr[j]);
            return true;
        } else if (arr[j] - arr[i] < n) {
            j++;
        } else {
            i++;
        }
    }
    printf("no hay par que de el numero");
    return false;
}

int main(int argc, char const *argv[]) {
    int arr[] = {1, 8, 30, 40, 100};
```

```
    int n = 7;
    int size = sizeof (arr) / sizeof (arr[0]);
    findPair(arr, n, size);
    return 0;
}
```

PYTHON

```
def FindPair(arr,x):
    size = len(arr)
    i,j = 0,1
    while i<size and j<size:
        if i != j and arr[j] - arr[i] == x:
            print(f"par encontrado en: ({arr[i]} - {arr[j]})")
            return True
        elif arr[j]-arr[i]<x:
            j+=1
        else:
            i+=1
    print("No existe el par que cumpla la suma de n")
    return False

arr = [1,8,30,40,100]
x = int(60)
FindPair(arr,x)
```

9.9) Ternary Search



Guía del programador competitivo

Ilustración 9-7 Buscando un dato usando ternary search

La búsqueda ternaria es un algoritmo de dividir y conquistar que se puede usar para encontrar un elemento en un array. Es similar a la búsqueda binaria donde dividimos el array en dos partes pero en este algoritmo dividimos el array dado en tres partes y determinamos cuál tiene la clave (elemento buscado). Podemos dividir el array en tres partes tomando mid1 y mid2, que se pueden calcular como se muestra a continuación.

mid1 = l + (r-l) / 3

mid2 = r - (r-l) / 3

Inicialmente, l y r serán iguales a 0 y n-1 respectivamente, donde n es la longitud de la matriz.

El array debe ordenarse para realizar una búsqueda ternaria en ella.

Complejidad de tiempo

Mejor caso : O(nLog(n)) **Peor caso :** O(nLog(n)) **Promedio:** O(nLog(n))

JAVA

```
//Programa Java para ilustrar
//recursivamente la aproximación
```

```java
//de una búsqueda ternaria

public class TernarySearch {
    //Función que realiza la busqueda ternaria
    static int ternarySearch(int l, int r, int key, int ar[]) {
        if (r >= l) {
            //encontrar el mid1 y mid 2
            int mid1 = l + (r - l) / 3;
            int mid2 = r - (r - l) / 3;
            //Verificar si la key esta presente en algun medio
            if (ar[mid1] == key) {
                return mid1;
            }
            if (ar[mid2] == key) {
                return mid2;
            }
            /*Desde que la key no este presente en el mid
            verifica en cada region si esta presente
            luego repite la operación de busqueda
            en esa región*/
            if (key < ar[mid1]) {
                //La key yace entre 1 y mid1
                return ternarySearch(l, mid1 - 1, key, ar);
            } else if (key > ar[mid2]) {
                //La key yace entre mid2 y r
                return ternarySearch(mid2 + 1, r, key, ar);
            } else {
                //La key  yace entre mid 1 y mid 2
                return ternarySearch(mid1 + 1, mid2 - 1, key, ar);
            }
        }
        // key no encontrada
        return -1;
    }

    public static void main(String args[]) {
        int l, r, p, key;
        //Cree el array y ordenelo si no lo está
        int ar[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        // Indice inicial
        l = 0;
        // Tamaño del arreglo
        r = 9;
        // Buscando por el 5
        // key que va a ser buscada en el array
        key = 5;
        // Busca usando Ternary Search
        p = ternarySearch(l, r, key, ar);
        // Imprime el resultado
        System.out.println("Indice de " + key + " es " + p);
        // Buscando por el 50
        // Key a ser buscada en el array
        key = 50;
        // Buscar usando Ternary Search
```

```
        p = ternarySearch(l, r, key, ar);
        // Imprime el resultado
        System.out.println("Index of " + key + " is " + p);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

static int ternarySearch(int left, int r, int key, int arr[]) {
    if (r >= left) {
        int mid1 = left + (r - left) / 3;
        int mid2 = r - (r - left) / 3;
        if (arr[mid1] == key) {
            return mid1;
        }
        if (arr[mid2] == key) {
            return mid2;
        }
        if (key < arr[mid1]) {
            return ternarySearch(left, mid1 - 1, key, arr);
        } else if (key > arr[mid2]) {
            return ternarySearch(mid2 + 1, r, key, arr);
        } else {
            return ternarySearch(mid1 + 1, mid2 - 1, key, arr);
        }

    }
    return -1;
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = sizeof (arr) / sizeof (arr[0]);
    int l = 0;
    int r = 9;
    int x = 5;
    int result = ternarySearch(l, r, x, arr);
    if (result < 0) {
        printf("el numero %d no se encuentra en el array\n", x);
    } else {
        printf("indice encontrado en la posicion %d\n", result);
    }
    return 0;
}
```

PYTHON

```
def TernarySearch(l,r,key,arr):
    if r >= l:
        mid1 = l+(r-l)//3
        mid2 = r-(r-l)//3
```

```
        if arr[mid1]==key:
            return mid1
        if arr[mid2] == key:
            return mid2
        if key<arr[mid1]:
            return TernarySearch(l,mid1-1,key,arr)
        elif key>arr[mid2]:
            return TernarySearch(mid2+1,r,key,arr)
        else:
            return TernarySearch(mid1+1,mid2-1,key,arr)

    return -1

l,r,resultado,key = int(),int(),int(),int()
arr = [1,2,3,4,5,6,7,8,9,10]
l = int(0)
r = len(arr)-1
key = 7
resultado=TernarySearch(l,r,key,arr)
if resultado==-1:
    print("No existe el dato en el arreglo")
else:
    print("Encontrado en el indice: "+str(resultado))
```

9.10) Problemas de repaso

Ejercicios en Online Judge

246-Count on Cantor	10277-Boastin' Red Socks
714-Copying Books	10611-The Playboy Chimp
978-Lemmings Battle!	10152-ShellSort

Capitulo 10. Algoritmos de Ordenamiento

Los algoritmos de ordenamiento son una secuencia de instrucciones que permiten ordenar de mayor a menor o viceversa elementos que se encuentran dentro de una estructura de datos que permita esta acción, es decir se realiza dada una relación de orden, las relaciones de orden más conocidas son el orden numérico y el orden lexicográfico. Existen diversos algoritmos de ordenamientos los cuales difieren en la metodología usada y en el uso de recursos como lo son el tiempo de ejecución y el almacenamiento en memoria.

10.1) Bubble Sort

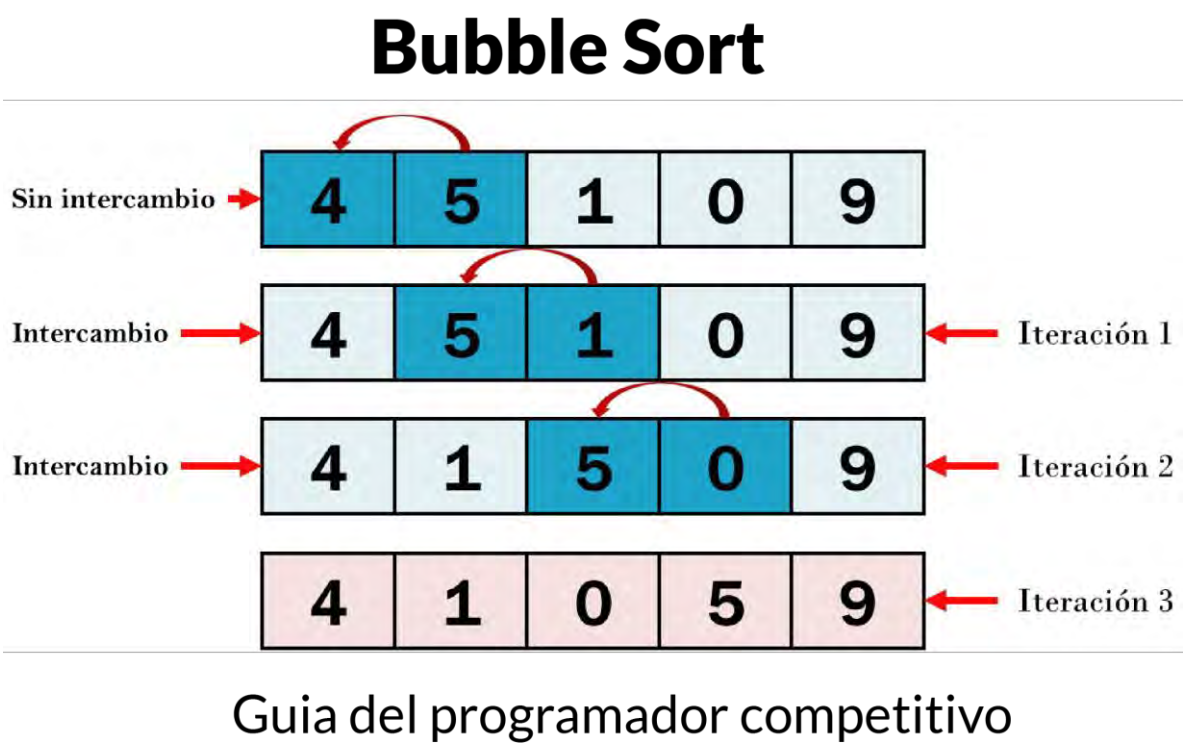


Ilustración 10-1 Ejemplo ordenamiento burbuja

Ordenamiento burbuja es el algoritmo de ordenamiento más simple que existe, funciona intercambiando repetidamente los elementos adyacentes si están en orden incorrecto.
Si existen demasiadas recursiones puede llegar a ser demasiado demorado o resultar en un error en tiempo de ejecución.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```java
//Programa java que realiza Bubble Sort Recursivo

import java.util.Arrays;

public class RecursiveBubbleSort {

    static void bubbleSort(int arr[], int n) {
        // Caso base
        if (n == 1) {
            return;
        }
        //Un paso de Bubble Sort, luego de este
        //paso, el elemento más largo es movido
        //  hasta el final
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1]) {   //intercambia arr[i], arr[i+1]
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
        bubbleSort(arr, n - 1);
    }

    public static void main(String[] args) {
        int arr[] = {64, 34, 25, 12, 22, 11, 90};
        bubbleSort(arr, arr.length);
        System.out.println("Array ordenado : ");
        System.out.println(Arrays.toString(arr));
    }
}
```

C++ (Ciclico, no recursivo)

```cpp
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;
typedef long long int ll;

void bubble_sort(int arr[], int len) {
    if (len == 1) {
        return;
    }
    for (int j = 0; j < len; j++) {
        for (int i = 0; i < len - 1; i++) {
            if (arr[i] > arr[i + 1]) {
```

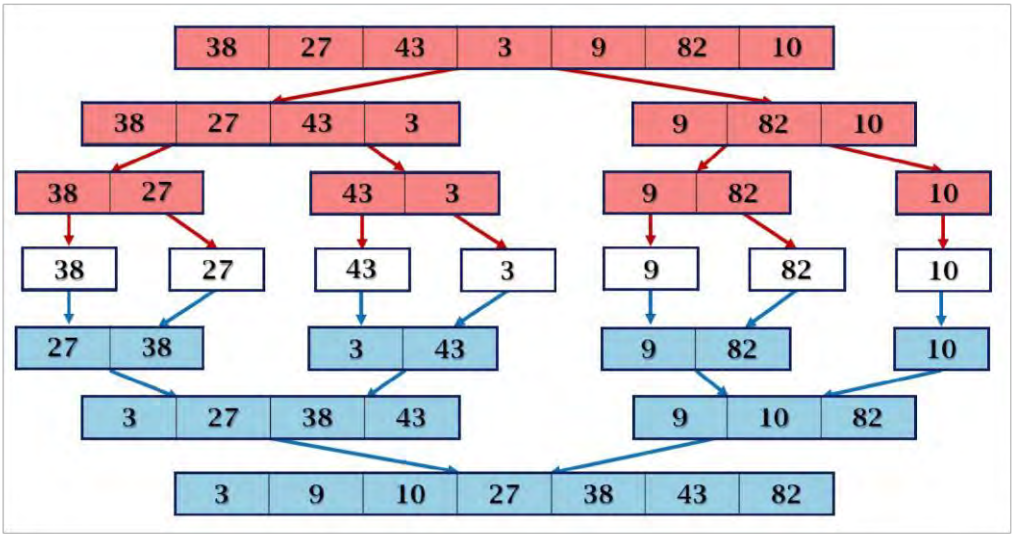
```
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {9, 7, 6, 4, 3, 2, 1, 7, 8, 43, 43, 4, 54, 54, 3, 234, 1, 23};
    int len = sizeof (arr) / sizeof (arr[0]);
    bubble_sort(arr, len);
    for (int i = 0; i < len; ++i) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

PYTHON

```
def BubbleSort(arr, n):
    if n == 1:
        return
    for i in range(n-1):
        if arr[i] > arr[i + 1]:
            temp = arr[i]
            arr[i] = arr[i + 1]
            arr[i + 1] = temp
    BubbleSort(arr, n-1)
arr = [4,5,8,9,10,2]
BubbleSort(arr, len(arr))
print(arr)
```

10.2) Merge Sort

Merge sort



Guía del programador competitivo

Ilustración 10-2 Ejemplo de ordenamiento por unión

Al igual que Quick Sort, Merge Sort es un algoritmo de Dividir y Conquistar. Divide el arreglo de entrada en dos mitades, se llama a sí misma para las dos mitades y luego combina las dos mitades clasificadas. La función merge () se usa para fusionar dos mitades.

El merge (arr, l, m, r) es un proceso clave que asume que arr [l...m] y arr [m + 1...r] están ordenados y combina los dos subarreglos ordenados en uno solo.

Merge Sort es útil para ordenar listas enlazadas en tiempo O (nLogn). En el caso de listas enlazadas, el caso es diferente principalmente debido a la diferencia en la asignación de memoria de los arrays y las listas enlazadas. A diferencia de los arrays, los nodos de listas enlazadas pueden no estar adyacentes en la memoria. A diferencia del array, en la lista enlazada, podemos insertar elementos en el medio en O (1) espacio adicional y O (1) tiempo. Por lo tanto, la operación de fusión de merge sort se puede implementar sin espacio adicional para las listas vinculadas.

En arrays, podemos hacer acceso aleatorio ya que los elementos son continuos en la memoria.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```
/*Programa java para Merge Sort*/
public class MergeSort {

    /*Une dos subarrays de arr[]
    Primer subarray es arr[1...m]
    Segundo subarray es arr[m+1..r]*/
    static void merge(int arr[], int l, int m, int r) {
        //Encuentra tamaños de dos subarrays a ser unidos
        int n1 = m - l + 1;
        int n2 = r - m;
        /* Crear arrays temporales */
        int L[] = new int[n1];
        int R[] = new int[n2];
        /*Copia datos en los arrays temporales*/
        for (int i = 0; i < n1; ++i) {
            L[i] = arr[l + i];
        }
        for (int j = 0; j < n2; ++j) {
            R[j] = arr[m + 1 + j];
        }
        /*Une los arreglos temporales*/
        // Indices iniciales del los dos subarrays
        int i = 0, j = 0;
        //indice inicial de array unido
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }
        /*Copia los elementos restanes de L[] si hay*/
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }
        //Copia los elementos restanes de R[] si hay
        while (j < n2) {
            arr[k] = R[j];
```

```
            j++;
            k++;
        }
    }

    static void sort(int arr[], int l, int r) {
        if (l < r) {
            //Encuentra el punto medio
            int m = (l + r) / 2;
            // Ordena los dos subarrays
            sort(arr, l, m);
            sort(arr, m + 1, r);
            // Une los subarrays
            merge(arr, l, m, r);
        }
    }

    //Imprimir array
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        int arr[] = {12, 11, 13, 5, 6, 7};
        System.out.println("Array dado");
        printArray(arr);
        sort(arr, 0, arr.length - 1);
        System.out.println("\nArray ordenado");
        printArray(arr);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1];
    int R[n2];
    for (int i = 0; i < n1; ++i) {
        L[i] = arr[l + i];
        for (int j = 0; j < n2; j++) {
            R[j] = arr[m + 1 + j];
        }
    }
    int i = 0, j = 0;
    int k = l;
```

```
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    cout << "\n";
}

void sort(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        sort(arr, l, m);
        sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {

    /*
    int arr[]={3,41,1,34,41,2,4,0};
    int n=sizeof(arr)/sizeof(arr[0]);
    sort(arr,0,n-1);
    printArray(arr,n);
     */
    //Mediante lectura
    string str;
    getline(cin, str);
    string intermediate;
    vector<int> vec;
    stringstream check1(str);
    while (getline(check1, intermediate, ' ')) {
        vec.push_back(atoi(intermediate.c_str()));
```

```
    }
    int arr[vec.size()];
    for (int i = 0; i < vec.size(); ++i) {
        arr[i] = vec[i];
    }
    sort(arr, 0, vec.size() - 1);
    printArray(arr, vec.size());
    return 0;
}
```

PYTHON

```
def merge(arr, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid
    L, R = [], []
    for i in range(n1):
        L.append(arr[left + i])
    for i in range(n2):
        R.append(arr[mid + 1 + i])
    i, j = int(0), int(0)
    k = left
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr, left, right):
    if left < right:
        m = (left + right) // 2
        mergeSort(arr, left, m)
        mergeSort(arr, m + 1, right)
        merge(arr, left, m, right)

arr = [int(x) for x in input().split()]
mergeSort(arr, 0, len(arr)-1)
print(arr)
```

10.3) Binary Insertion Sort

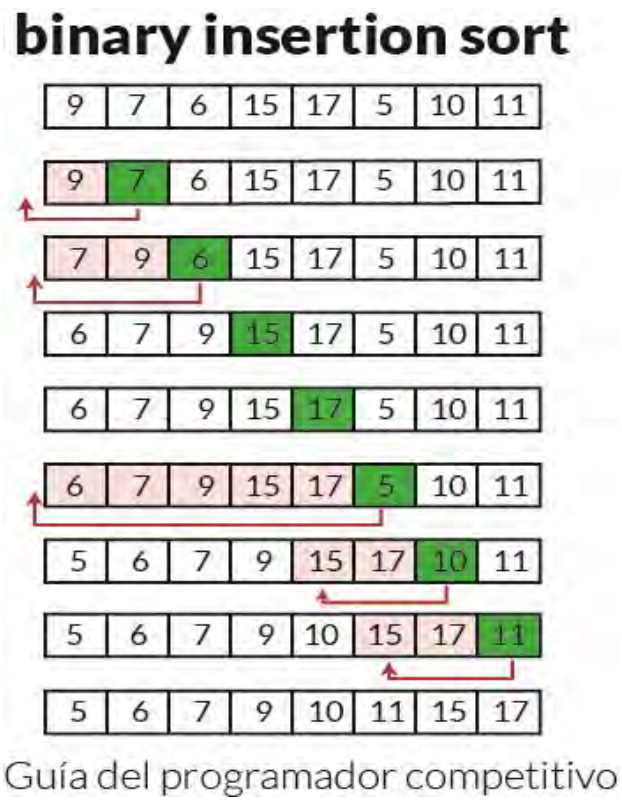


Ilustración 10-3 Ordenamiento de vector por medio de binary sort

Podemos usar la búsqueda binaria para reducir el número de comparaciones en el ordenamiento de inserción normal. El ordenamiento de inserción binaria utiliza la búsqueda binaria para encontrar la ubicación adecuada para insertar el elemento seleccionado en cada iteración.

En el ordenamiento de inserción normal, se necesitan comparaciones O(n) (en la iteración n) en el peor de los casos. Podemos reducirlo a O (log n) mediante la búsqueda binaria.

El algoritmo en su conjunto aún tiene un tiempo de ejecución en el peor de los casos de O (n^2) debido a la serie de intercambios necesarios para cada inserción.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```java
// Programa java implementando
// Binary Insertion Sort

import java.util.Arrays;

public class BinaryInsertionSort {

    public static void main(String[] args) {
        int[] arr = {37, 23, 0, 17, 12, 72, 31,
            46, 100, 88, 54};
        sort(arr);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    static void sort(int array[]) {
        for (int i = 1; i < array.length; i++) {
            int x = array[i];
            // Encontrar ubicación para insertar usando
            // Binary Search
            int j = Math.abs(Arrays.binarySearch(array, 0, i, x) + 1);
            System.arraycopy(array, j, array, j + 1, i - j);
            // Colocar elemento en su correcta localización
            array[j] = x;
        }
    }
}
```

C++

```cpp
#include <iostream>

using namespace std;

int binarySearch(int arr[], int i, int low, int high) {
    if (high <= low) {
        return (i > arr[low]) ? (low + 1) : low;
    }
    int mid = (low + high) / 2;
    if (i == arr[mid]) {
        return mid + 1;
    }
    if (i > arr[mid]) {
        return binarySearch(arr, i, mid + 1, high);
    }
    return binarySearch(arr, i, low, mid - 1);
}

void insertionSort(int arr[], int n) {
    int i, loc, j, selected;
    for (i = 1; i < n; ++i) {
```

```
        j = i - 1;
        selected = arr[i];
        loc = binarySearch(arr, selected, 0, j);
        while (j >= loc) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = selected;
    }
}

int main() {
    int arr[] = {-63, 8, -7, -15, -19, 17, 19, 110};
    int n = sizeof (arr) / sizeof (arr[0]);
    insertionSort(arr, n);
    cout << "Arreglo ordenado:" << endl;
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
    return 0;
}
```

PYTHON

```
def partition(arr, low, high):
  pivot = arr[high]
  i = low - 1
  for j in range(low, high):
    if(arr[j] < pivot):
      i = i + 1
      temp = arr[i]
      arr[i] = arr[j]
      arr[j] = temp
  temp = arr[i + 1]
  arr[i + 1] = arr[high]
  arr[high] = temp
  return i + 1


def quickSort(arr, low, high):
  if(low < high):
    pivot = partition(arr, low, high)
    quickSort(arr, low, pivot-1)
    quickSort(arr, pivot + 1, high)


lista = [int(x) for x in input().split()]
quickSort(lista, 0, len(lista)-1)
print(*lista)
```

10.4) Quick Sort

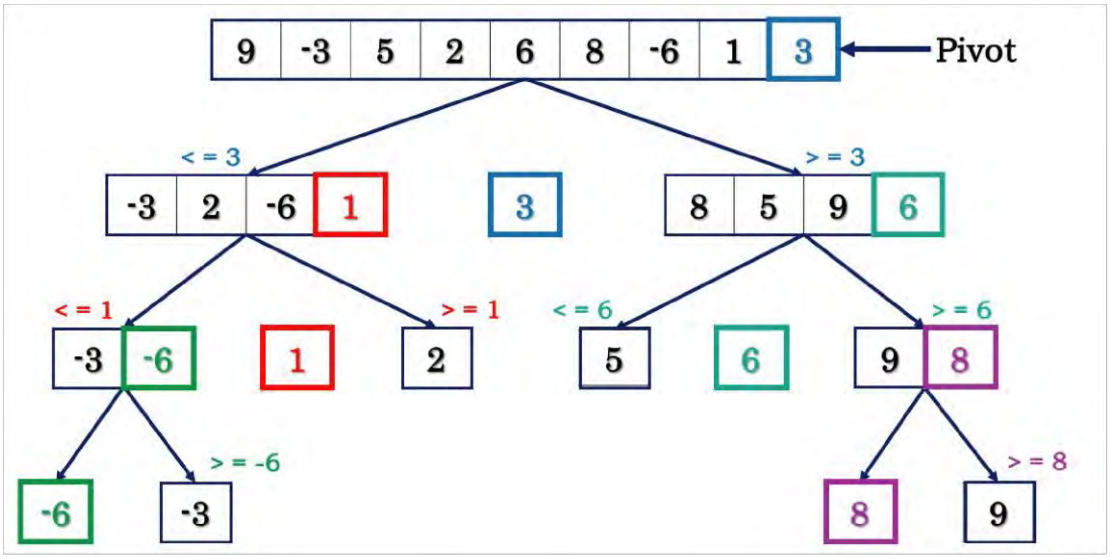


Ilustración 10-4 Ejemplo ordenamiento rápido

Quick Sort es un algoritmo de Dividir y Conquistar. Selecciona un elemento como pivote y divide el array dado alrededor del pivote seleccionado. Hay muchas versiones diferentes de quick Sort que seleccionan pivote de diferentes maneras.

- Elije siempre el primer elemento como pivote.
- Siempre elije el último elemento como pivote.
- Elige un elemento aleatorio como pivote.
- Elije la mediana como pivote.

El proceso clave en quick Sort es partition (). El destino de las particiones es, dada un array y un elemento x del array como pivote, coloca x en su posición correcta en el array ordenado y coloca todos los elementos más pequeños (más pequeños que x) antes de x, y coloca todos

los elementos mayores (mayores que x) después de X. Todo esto debe hacerse en tiempo lineal.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** $O(n^2)$ **Promedio:** O(nlog(n))

JAVA

```java
// Programa java para la implementación de QuickSort

public class QuickSort {

    /* Esta función toma el ultimo elemento como
    pivote, coloca el pivote en la posición correcta
    en el array ordenado, y coloca todos los más
    pequeñps (Más pequeños que el pivote) a la izquierda
    del pivote, y todos los más grandes a la derecha del
    pivote*/
    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1); // Indice del elemento más pequeño
        for (int j = low; j < high; j++) {
            /*Si el actual elemento es más pequeño
            p igual que el pivote*/
            if (arr[j] <= pivot) {
                i++;
                //intercambia arr[i] y arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        // Intercambia arr[i+1] y arr[high] (o pivote)
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    /*arr[] --> Array que sera ordenado,
      low  --> indice inicial,
      high  --> indice final */
    static void sort(int arr[], int low, int high) {
        if (low < high) {
            /* pi esta particionando indices, arr[pi] is
              now at right place */
            int pi = partition(arr, low, high);
            // Recursivamente ordena elementos antes de
            // la partición y despues de la partición
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
```

```
        }
    }
    /* Imprimir array */
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        int arr[] = {10, 7, 8, 9, 1, 5};
        int n = arr.length;
        sort(arr, 0, n - 1);
        System.out.println("Array ordenado");
        printArray(arr);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void print(int arr[], int n) {
```

```
    for (int i = 0; i < n; ++i) {
        printf("%d ", arr[i]);
    }
    cout << "\n";
}

int main(int argc, char const *argv[]) {
    int arr[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
    int n = sizeof (arr) / sizeof (arr[0]);
    quickSort(arr, 0, n - 1);
    print(arr, n);
    return 0;
}
```

PYTHON

```
def Partition(arr, low, high):
    pivot = arr[high]
    i = low-1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            temp = arr[i]
            arr[i] = arr[j]
            arr[j] = temp
    temp = arr[i + 1]
    arr[i + 1] = arr[high]
    arr[high] = temp
    return i + 1

def QuickSort(arr, low, high):
    if low < high:
        pivot = Partition(arr, low, high)
        QuickSort(arr, low, pivot-1)
        QuickSort(arr, pivot + 1, high)
arr = [int(x) for x in input().split()]
QuickSort(arr, 0, len(arr)-1)
print(arr)
```

10.5) Radix Sort

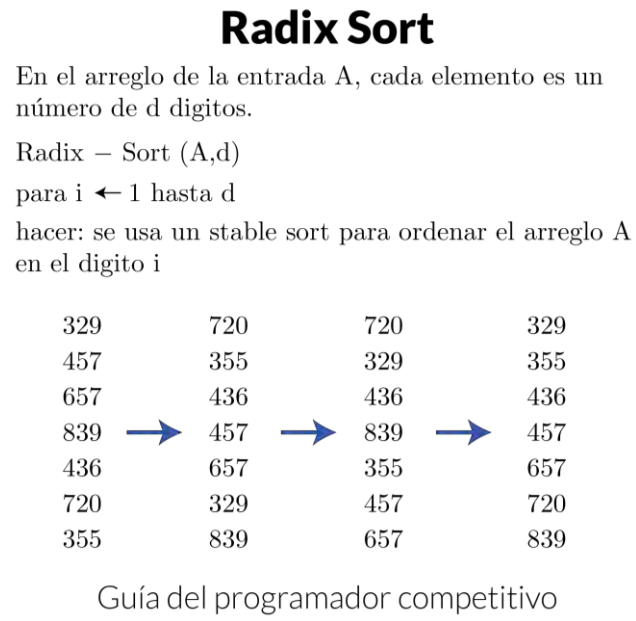


Ilustración 10-5 Ejemplo de radix sort

Radix sort es un algoritmo de ordenamiento que clasifica los números según las posiciones de sus dígitos. Básicamente, utiliza el valor posicional de los dígitos en un número. A diferencia de la mayoría de los otros algoritmos de ordenamiento, como Merge Sort, Insertion Sort y burbuja, no compara los números.

Radix sort utiliza un algoritmo de ordenamiento estable como subrutina para ordenar los dígitos. Aquí hemos usado una variación de conteo como una subrutina que usa la raíz para ordenar los dígitos en cada posición. El ordenamiento conteo es un algoritmo de ordenamiento estable y funciona bien en la práctica.

La clasificación por radix funciona clasificando los dígitos del dígito menos significativo al dígito más significativo.

Si tenemos bits log2n para cada dígito, el tiempo de ejecución de Radix parece ser mejor que Quick Sort para una amplia gama de números de entrada. Los factores constantes ocultos en la notación asintótica son mayores para Radix Sort y Quick-Sort usa cachés de hardware de manera más efectiva.

Complejidad de tiempo

Mejor caso : O(dígitos*n) **Peor caso :** O(dígitos*n) **Promedio:** O(dígitos*n)

JAVA

```java
//Implementación java de Radix Sort

import java.util.*;

public class RadixSort {

    // Una función de utilidad que obtiene
    //El maximo valor en arr[]
    static int getMax(int arr[], int n) {
        int mx = arr[0];
        for (int i = 1; i < n; i++) {
            if (arr[i] > mx) {
                mx = arr[i];
            }
        }
        return mx;
    }

    /*Una función que realiza conteo de ordenamiento en
    arr[] de acuerdo al digito
    representado como exp*/
    static void countSort(int arr[], int n, int exp) {
        int output[] = new int[n]; // Array de salida
        int i;
        int count[] = new int[10];
        Arrays.fill(count, 0);
        // Almacena el conteo de las ocurrencias en count[]
        for (i = 0; i < n; i++) {
            count[(arr[i] / exp) % 10]++;
        }
        // cambia count[i] de tal manera que ahora
        //contenga la actual posicion de este digito en
        //output[]
        for (i = 1; i < 10; i++) {
            count[i] += count[i - 1];
        }
        //Contruye el array de salida
        for (i = n - 1; i >= 0; i--) {
            output[count[(arr[i] / exp) % 10] - 1] = arr[i];
            count[(arr[i] / exp) % 10]--;
        }
        // copia el array de salida a arr[], ahora arr[]
        // contiene los números ordenados de acuerdo al digito
        //actual
        for (i = 0; i < n; i++) {
            arr[i] = output[i];
        }
```

```
    }

    static void radixsort(int arr[], int n) {
        // Encuentra el maximo número para conocer el número
        // de digitos
        int m = getMax(arr, n);
        for (int exp = 1; m / exp > 0; exp *= 10) {
            countSort(arr, n, exp);
        }
    }

    //Imprimir el array
    static void print(int arr[], int n) {
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    public static void main(String[] args) {
        int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
        int n = arr.length;
        radixsort(arr, n);
        print(arr, n);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

int getMax(int arr[], int n) {
    int mx = arr[0];
    for (int i = 1; i < n; ++i) {
        if (arr[i] > mx) {
            mx = arr[i];
        }
    }
    return mx;
}

void countSort(int arr[], int n, int exp) {
    int output[n];
    int count[10];
    memset(count, 0, sizeof (count));
    for (int i = 0; i < n; ++i) {
        count[(arr[i] / exp) % 10]++;
    }
    for (int i = 0; i < 10; ++i) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; --i) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
```

```
    }
    for (int i = 0; i < n; ++i) {
        arr[i] = output[i];
    }
}

void radixSort(int arr[], int n) {
    int m = getMax(arr, n);
    for (int exp = 1; m / exp > 0; exp *= 10) {
        countSort(arr, n, exp);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i) {
        printf("%d ", arr[i]);
    }
    cout << "\n";
}

int main(int argc, char const *argv[]) {
    int arr[] = {9, 78, 6, 5, 32, 1, 1, 4, 5, 45};
    int n = sizeof (arr) / sizeof (arr[0]);
    radixSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

PYTHON

```
# Solo con Numeros Positivos

def getMax(arr, n):
    maxim = arr[0]
    for i in range(1, n):
        if(arr[i] > maxim):
            maxim = arr[i]
    return maxim


def countSort(arr, n, exp):
    output = [0 for x in range(n)]
    count = [0 for x in range(10)]
    i = 0
    for i in range(0, n):
        count[(arr[i] // exp) % 10] += 1
    for i in range(1, 10):
        count[i] += count[i-1]
    for i in range(n-1, -1, -1):
        output[count[(arr[i] // exp) % 10]-1] = arr[i]
        count[(arr[i] // exp) % 10] -= 1
    for i in range(0, n):
        arr[i] = output[i]
```

```
def radixSort(arr, n):
    m = getMax(arr, n)
    exp = 1
    while m // exp > 0:
        countSort(arr, n, exp)
        exp *= 10


lista = [int(x) for x in input().split()]
radixSort(lista, len(lista))
print(*lista)
```

10.6) Shell Sort

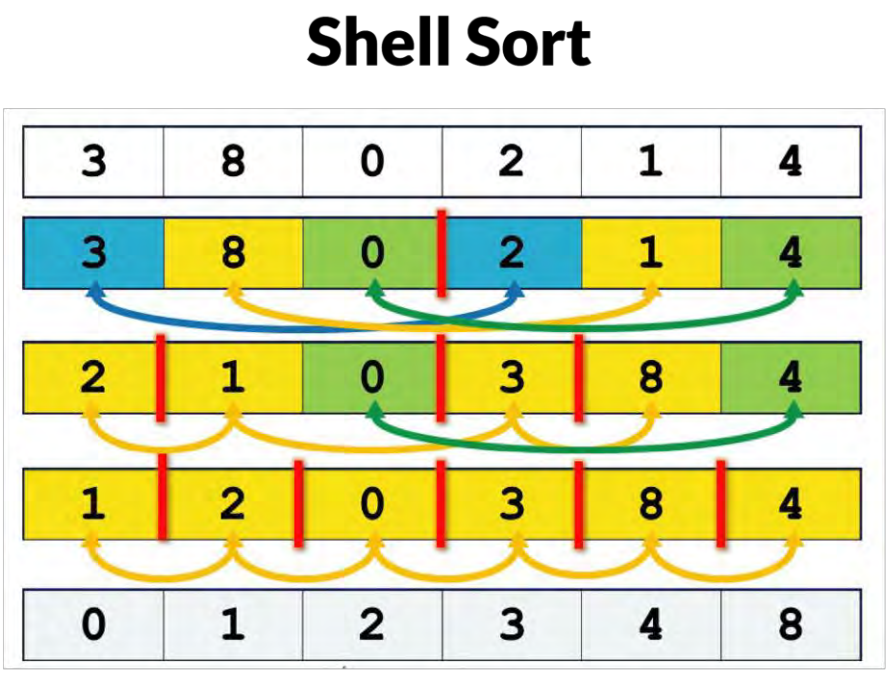


Ilustración 10-6 Ejemplo ordenamiento cascara

Shell Sort es principalmente una variación de Insertion Sort. En el ordenamiento por inserción, movemos los elementos solo una posición adelante. Cuando un elemento tiene que moverse mucho más adelante, hay muchos movimientos involucrados. La idea de shell Sort es permitir el intercambio de elementos lejanos. En shell Sort, hacemos el array para

un gran valor de h. Continuamos reduciendo el valor de h hasta que se convierte en 1. Se dice que un array está ordenado por h si todas las sublistas de cada elemento h están ordenadas.

La complejidad de tiempo de la implementación de shell Sort es O (n^2). En la implementación la brecha se reduce a la mitad en cada iteración.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(n^2) **Promedio:** O(nlog(n))

JAVA

```
// Implementación java de ShellSort

public class ShellSort {

    /*Imprimir el array */
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
    //Función ordenadora Shell Sort
    static void sort(int arr[]) {
        int n = arr.length;
        //Inicia con un gran salto, luego reduce el salto
        for (int gap = n / 2; gap > 0; gap /= 2) {
     //Realiza un insertion sort con salto
            // Los primeros elementros del sato a[0..gap-1]
            // estan ya en un orden de salto que sigue agregando
            //un elemento más hasta que el array entero este ordenado
            for (int i = gap; i < n; i += 1) {
                //Agrega a[i] a los elementos que estan en el ordenamiento
                // con salto
                int temp = arr[i];
                int j;
                for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                    arr[j] = arr[j - gap];
                }
                arr[j] = temp;
            }
        }

    }

    public static void main(String args[]) {
```

```
        int arr[] = {12, 34, 54, 2, 3};
        System.out.println("Array antes de ordenar");
        printArray(arr);
        sort(arr);
        System.out.println("Array despues de ordenar");
        printArray(arr);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
typedef long long int ll;
using namespace std;

void printArray(vector <int> myvec, int n) {
    for (int i = 0; i < n; ++i) {
        printf("%d ", myvec[i]);
    }
}

int sort(vector <int> myvec, int n) {
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i += 1) {
            int temp = myvec[i];
            int j;
            for (j = i; j >= gap && myvec[j - gap] > temp; j -= gap) {
                myvec[j] = myvec[j - gap];
            }
            myvec[j] = temp;
        }
    }
    printArray(myvec, n);
    return 0;
}

int main(int argc, char** argv) {
    int arr[] = {3, 4, 1, 3, 54, 6, 6, 4, 3, 2, 3, 1, 21};
    int n = sizeof (arr) / sizeof (arr[0]);
    vector<int> myvec;
    myvec.insert(myvec.begin(), arr, arr + n);
    sort(myvec, n);
    return 0;
}
```

PYTHON

```
def ShellSort(arr):
    n = len(arr)
    grap = n // 2
    while grap > 0:
        i = grap
        while i < n:
            temp = arr[i]
```

```
            j = i
            while j >= grap and arr[j-grap] > temp:
                arr[j] = arr[j-grap]
                j -= grap
            arr[j] = temp
            i += 1
        grap //= 2

arr = [12, 56, 89, 47, 89, 23]
ShellSort(arr)
print(*arr)
```

10.7) Tim Sort

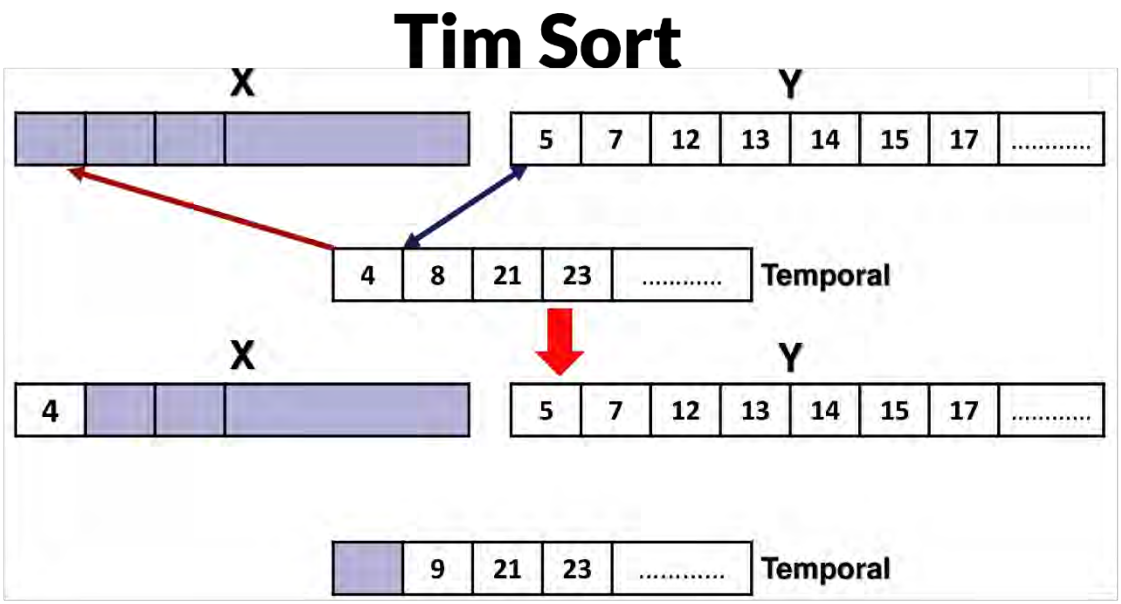


Ilustración 10-7 Ejemplo de ordenamiento Tim

TimSort es un algoritmo de ordenamiento basado en Insertion Sort y Merge Sort.

- Es un algoritmo de ordenamiento estable funciona en tiempo O (n Log n)

- Se utiliza en Arrays.sort () de Java, así como en Python sorted() y C++ sort()
- Primero clasifica las piezas pequeñas utilizando el ordenamiento de Inserción, luego fusiona las piezas utilizando ordenamiento por fusión (Merge Sort).

Dividimos el array en bloques conocidos como Run. Ordenamos esas ejecuciones utilizando el ordenamiento por inserción una por una y luego las combinamos utilizando la función de combinación utilizada en Merge Sort. Si el tamaño del array es menor que run, entonces el array se clasifica simplemente utilizando el ordenamiento de inserción.

El tamaño de run puede variar de 32 a 64, dependiendo del tamaño del array. Tenga en cuenta que la función de combinación funciona bien cuando los arreglos secundarios de tamaño son potencias de 2. La idea se basa en el hecho de que el ordenamiento por inserción funciona bien para arreglos pequeños.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```java
// Programa Java que realiza TimSort

public class TimSort {

    static int RUN = 32;
    // esta función ordena el arreglo
    //desde el índice izquierdo hasta
    //a la derecha el índice que es de tamaño más alto RUN
    public static void insertionSort(int[] arr, int left, int right) {
        for (int i = left + 1; i <= right; i++) {
            int temp = arr[i];
            int j = i - 1;
            while (arr[j] > temp && j >= left) {
                arr[j + 1] = arr[j];
                j--;
                 if (j < 0) {
                    break;
                }

            }
            arr[j + 1] = temp;
        }
    }
```

```
public static void merge(int[] arr, int l, int m, int r) {
    //Array original esta separado en dos partes
    // array derecho e izquierdo
    int len1 = m - l + 1, len2 = r - m;
    int[] left = new int[len1];
    int[] right = new int[len2];
    for (int x = 0; x < len1; x++) {
        left[x] = arr[l + x];
    }
    for (int x = 0; x < len2; x++) {
        right[x] = arr[m + 1 + x];
    }
    int i = 0;
    int j = 0;
    int k = l;
    // despues de comparar,
    // after comparing,unimos los dos array
    // en un subarray más largo
    while (i < len1 && j < len2) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }
    //Copia los elementos restates de la izquiera, si hay
    while (i < len1) {
        arr[k] = left[i];
        k++;
        i++;
    }
    // Copia los elementos restates de la derecha, si hay
    while (j < len2) {
        arr[k] = right[j];
        k++;
        j++;
    }
}

public static void timSort(int[] arr, int n) {
    //Ordena individualmente los subarrays de
    //tamaño RUN
    for (int i = 0; i < n; i += RUN) {
        insertionSort(arr, i, Math.min((i + 31), (n - 1)));
    }
    //Comienza a unir de tamaño run, luego
    //podra unir de tamaño 64, 128, 256 y asi...
    for (int size = RUN; size < n; size = 2 * size) {
        // Toma un punto inicial del subarray izquierdo
        // nosotros vamos a unir arr[left..left+size-1] y
```

```
            //arr[left+size, left+2*size-1], luego de cada union
            // nosotros incrementamos izquiera en  2*size
            for (int left = 0; left < n; left += 2 * size) {
                //Encontramos punto de finalizacion de
                // el subarray izquierdo, mid+1
                // mid+1 es el punto inicial del
                //subarray derecho
                int mid = Math.min((left + size - 1), (n - 1));
                int right = Math.min((left + 2 * size - 1), (n - 1));
                // une sub array arr[left.....mid] y
                // arr[mid+1....right]
                merge(arr, left, mid, right);
            }
        }
    }

    // Imprimir el array
    public static void printArray(int[] arr, int n) {
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.print("\n");
    }

    public static void main(String[] args) {
        int[] arr = {5, 21, 7, 23, 19};
        int n = arr.length;
        System.out.print("El array dado es\n");
        printArray(arr, n);
        timSort(arr, n);
        System.out.print("Array ordenado\n");
        printArray(arr, n);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
#define MAX 256
typedef long long int ll;
using namespace std;
int RUN = 32;

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
}

void insertionSort(int arr[], int left, int rigth) {
    for (int i = left + 1; i <= rigth; ++i) {
        int temp = arr[i];
        int j = i - 1;
        while (arr[j] > temp && j >= left) {
```

```
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}

void merge(int arr[], int l, int m, int r) {
    int len1 = m - l + 1, len2 = r - m;
    int left[len1];
    int rigth[len2];
    for (int x = 0; x < len1; ++x) {
        left[x] = arr[l + x];
    }
    for (int x = 0; x < len2; ++x) {
        rigth[x] = arr[m + 1 + x];
    }
    int i = 0, j = 0, k = l;
    while (i < len1 && j < len2) {
        if (left[i] <= rigth[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = rigth[j];
            j++;
        }
        k++;
    }
    while (i < len1) {
        arr[k] = left[i];
        k++;
        i++;
    }
    while (j < len2) {
        arr[k] = rigth[j];
        k++;
        j++;
    }
}

void timSort(int arr[], int n) {
    for (int i = 0; i < n; i += RUN) {
        insertionSort(arr, i, std::min((i + 31), (n - 1)));
    }
    for (int size = RUN; size < n; size = 2 * size) {
        for (int left = 0; left < n; left += 2 * size) {
            int mid = std::min((left + size - 1), (n - 1));
            int rigth = std::min((left + 2 * size - 1), (n - 1));
            merge(arr, left, mid, rigth);
        }
    }
    printArray(arr, n);
}
```

```
int main(int argc, char** argv) {
    int arr[] = {3, 4, 5, 6, 3, 1, 1, 23, 5, 546, 34, 3, 2, 2};
    int len = sizeof (arr) / sizeof (arr[0]);
    timSort(arr, len);
    return 0;
}
```

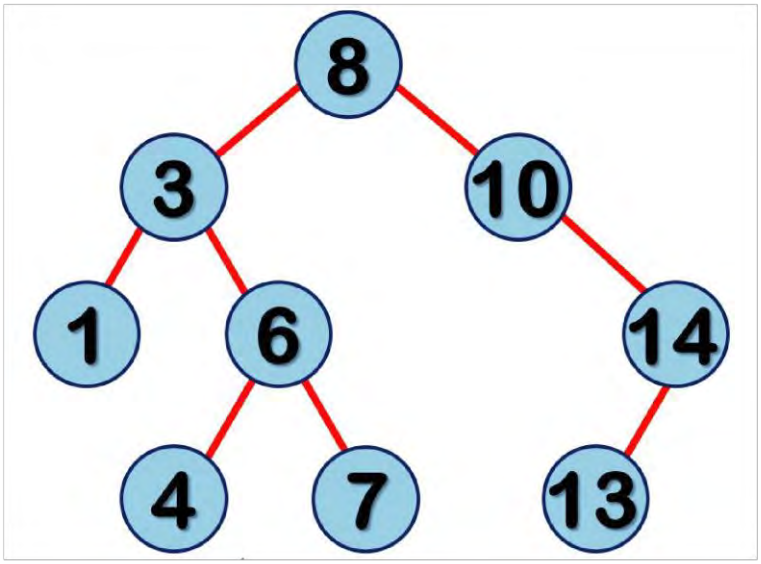
PYTHON

```
from sys import stdout
wr = stdout.write

RUN = 32
def insertionSort(arr, left, right):
    for i in range(left + 1, right + 1):
        temp = arr[i]
        j = i - 1
        while j >= left and arr[j] > temp:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = temp


def merge(arr, l, m, r):
    len1 = m - l + 1
    len2 = r - m
    left = [0 for x in range(len1)]
    right = [0 for x in range(len2)]
    for i in range(len1):
        left[x] = arr[l + x]
    for i in range(len2):
        right[x] = arr[m + 1 + x]
    i = 0
    j = 0
    k = l
    while i < len1 and j < len2:
        if left[i] <= right[i]:
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            j += 1
        k += 1
    while i < len1:
        arr[k] = left[i]
        k += 1
        i += 1
    while j < len2:
        arr[k] = right[j]
        k += 1
        j += 1


def timSort(arr, n):
    i = 0
```

```
    while i < n:
        insertionSort(arr, i, min((i + 31), (n - 1)))
        i += RUN
    size = RUN
    while size < n:
        left = 0
        while left < 2:
            mid = min ((left + size - 1), (n - 1))
            right = min((left + 2 * size - 1), (n - 1))
            merge(arr, left, mid, right)
            left += 2 * size
        size = 2 * size


def printArray(arr, n):
    for i in range(n):
        wr(f'{arr[i]} ')
    wr('\n')

arr = [int(x) for x in input().split()]
n = len(arr)
timSort(arr, n)
printArray(arr, n)
```

10.8) Tree Sort

Tree Sort



Guía del programador competitivo

Ilustración 10-8 Vista abstracta de un árbol binario de búsqueda

Ordenamiento árbol es un algoritmo de clasificación que se basa en la estructura de datos del Binary Search Tree (Árbol de búsqueda binaria). Primero crea un árbol de búsqueda binario a partir de los elementos de la lista de entrada o un array luego realiza un recorrido inorden en el árbol de búsqueda binario creado para obtener los elementos ordenados.

Para recorrer un árbol binario no vacío en inorden, hay que realizar las siguientes operaciones recursivamente en cada nodo:

1. Atraviese el sub-árbol o nodo izquierdo
2. Visite la raiz
3. Atraviese el sub-árbol o nodo derecho

Complejidad de tiempo

Mejor caso : $O(nlog(n))$ **Peor caso :** $O(n^2)$ **Promedio:** $O(nlog(n))$

JAVA

```
//Código java que implementa Tree Sort
```

```
public class TreeSort {

    //Clase conteniendo hijo izquierdo
    //y derecho del nodo actual, y
    // el valor del nodo (Key)
    static class Node {
        int key;
        Node left, right;
        public Node(int item) {
            key = item;
            left = right = null;
        }
    }
    // Raiz del arbol de busqueda
    //binaria
    static Node root;
    static void init() {
        root = null;
    }
    static void insert(int key) {
        root = insertRec(root, key);
    }

    /*Una función recursiva que inserta
    nuevos valores al arbol de busqueda binaria*/
    static Node insertRec(Node root, int key) {
        /*Si el arbol esta vacio
        Retorne un nuevo nodo*/
        if (root == null) {
            root = new Node(key);
            return root;
        }
        /*De otra forma, baja por el arbol*/
        if (key < root.key) {
            root.left = insertRec(root.left, key);
        } else if (key > root.key) {
            root.right = insertRec(root.right, key);
        }
        /*Retorna la raiz*/
        return root;
    }

    /*Una función que realiza recorrido
    inorden atraves del arbol*/
    static void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.key + " ");
            inorderRec(root.right);
        }
    }

    static void treeins(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
```

```
            insert(arr[i]);
        }
    }

    public static void main(String[] args) {
        int arr[] = {5, 4, 7, 2, 11};
        treeins(arr);
        inorderRec(root);
    }
}
```

C++

```
#include <bits/stdc++.h>
#include <cstdlib>
using namespace std;

struct Node {
    int key;
    struct Node *left, *right;
};

struct Node *newNode(int item) {
    struct Node *temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

void storeSorted(Node *root, int arr[], int &i) {
    if (root != NULL) {
        storeSorted(root->left, arr, i);
        arr[i++] = root->key;
        storeSorted(root->right, arr, i);
    }
}

Node* insert(Node* node, int key) {
    if (node == NULL) {
        return newNode(key);
    }
    if (key < node->key) {
        node->left = insert(node->left, key);
    } else if (key > node->key) {
        node->right = insert(node->right, key);
    }
    return node;
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i) {
        printf("%d ", arr[i]);
    }
}
```

```
void treeSort(int arr[], int n) {
    struct Node *root = NULL;

    root = insert(root, arr[0]);
    for (int i = 1; i < n; i++)
        root = insert(root, arr[i]);

    int i = 0;
    storeSorted(root, arr, i);
    printArray(arr, n);
}

int main() {
    /*
    int arr[]={2,3,5,1,32,4,50};
    int n = sizeof(arr)/sizeof(arr[0]);
    treeSort(arr,n);
     */
    string str;
    getline(cin, str);
    string intermediate;
    vector<int> vec;
    stringstream check1(str);
    while (getline(check1, intermediate, ' ')) {
        vec.push_back(atoi(intermediate.c_str()));
    }
    int arr[vec.size()];
    for (int i = 0; i < vec.size(); ++i) {
        arr[i] = vec[i];
    }
    treeSort(arr, vec.size());
    return 0;
}
```

PYTHON

```
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write


class Hoja:

    def __init__(self, item=None):
        self.key = item
        self.right = None
        self.left = None

root = Hoja()

def init():
```

```
    global root
    root = None

def insert(key):
    global root
    root = insertRec(root, key)

def insertRec(raiz, key):

    if raiz == None:
        raiz = Hoja(key)
        return raiz

    if key < raiz.key:
        raiz.left = insertRec(raiz.left, key)
    elif key >= raiz.key:
        raiz.right = insertRec(raiz.right, key)

    return raiz

def inOrder(actual):
    if actual != None:
        inOrder(actual.left)
        wr(f'{actual.key} ')
        inOrder(actual.right)


def insertInTree(arr):
    for i in range(len(arr)):
        insert(arr[i])

arr = list(map(int, rl().split()))
init()
insertInTree(arr)
inOrder(root)
wr('\n')
```

10.9) Bitonic Sort

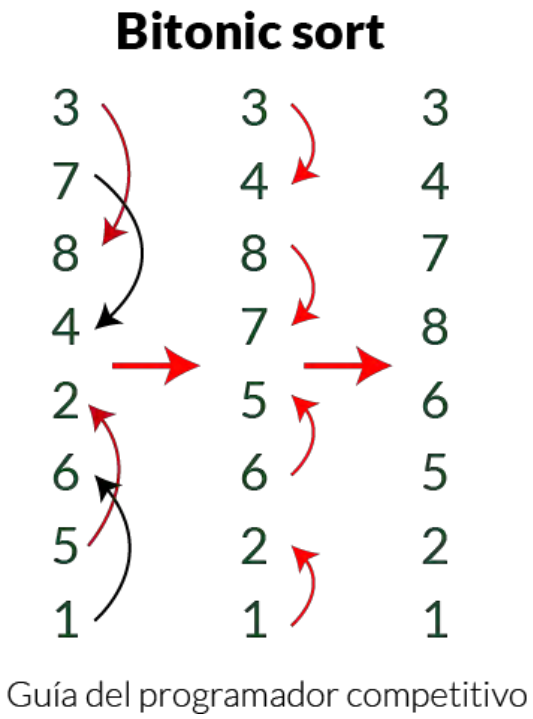


Ilustración 10-9 Ejemplo de ordenamiento bitónico

Bitonic Sort es un algoritmo clásico para ordenamiento.

El ordenamiento bitónico hace comparaciones O (n Log 2n).

El número de comparaciones realizadas por Bitonic Sort son más que los algoritmos de clasificación populares como Merge Sort [hace O (nLogn) comparaciones], pero Bitonic es mejor para la implementación en paralelo porque siempre comparamos elementos en una secuencia predefinida y la secuencia de comparación no. Depende de los datos. Por lo tanto, es adecuado para la implementación en hardware y array de procesado paralelo

Para entender el ordenamiento bitónico, primero debemos entender qué es la secuencia bitónica y cómo hacer una secuencia dada bitónica.

Una secuencia es bitónica si primero aumenta, luego disminuye. En otras palabras, una matriz arr [0...n-i] es bitonica si existe un índice i donde 0 <= i <= n-1 tal que

$$x0 <= x1 \ldots .. <= xi \text{ y } xi> = xi + 1 \ldots ..> = xn-1$$

Para formar una secuencia ordenada de longitud n a partir de dos secuencias ordenadas de longitud n / 2, se requieren comparaciones de log (n) (por ejemplo: log (8) = 3 cuando el

tamaño de la secuencia. Por lo tanto, el número de comparaciones T (n) de La clasificación completa está dada por:

$$T(n) = \log(n) + T(n / 2)$$

La solución de esta ecuación de recurrencia es

$$T(n) = \log(n) + \log(n) - 1 + \log(n) - 2 + \ldots + 1 = \log(n) \cdot (\log(n) + 1) / 2$$

Cada etapa de la red de ordenamiento consiste en n / 2 comparaciones. Por lo tanto un total de O (n log2n) comparaciones.

Complejidad de tiempo

Mejor caso : $O(\log^2(n))$ **Peor caso :** $O(\log^2(n))$ **Promedio:** $O(\log^2(n))$

JAVA

```
/* Programa java para Bitonic Sort*/
/* Nota: Este programa solo funciona si el tamaño
de la entrada es una potencia de 2*/
public class BitonicSort {

    /* El parámetro dir indica la dirección de ordenamiento,
       ASCENDIENDO o DESCENDIENDO; si (a [i]> a [j]) está de acuerdo
       con la dirección, entonces a[i] y a[j] son
        intercambiados*/
    static void compAndSwap(int a[], int i, int j, int dir) {
        if ((a[i] > a[j] && dir == 1)
                || (a[i] < a[j] && dir == 0)) {
            // intercambiando elementos
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }

    static void bitonicMerge(int a[], int low, int cnt, int dir) {
        if (cnt > 1) {
            int k = cnt / 2;
            for (int i = low; i < low + k; i++) {
                compAndSwap(a, i, i + k, dir);
            }
            bitonicMerge(a, low, k, dir);
            bitonicMerge(a, low + k, k, dir);
        }
    }

    static void bitonicSort(int a[], int low, int cnt, int dir) {
        if (cnt > 1) {
```

```
            int k = cnt / 2;
            // Ordena en orden ascendente, dir es 1
            bitonicSort(a, low, k, 1);
            // ordena en orden descendente, dir es 0
            bitonicSort(a, low + k, k, 0);
            // une toda la secuencia en orden ascendente
            bitonicMerge(a, low, cnt, dir);
        }
    }

    static void sort(int a[], int N, int up) {
        bitonicSort(a, 0, N, up);
    }

    /*Imprimir array */
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        int a[] = {3, 7, 4, 8, 6, 2, 1, 5};
        int up = 1;
        sort(a, a.length, up);
        System.out.println("\nArray ordenado");
        printArray(a);
    }
}
```

C++

```
#include <iostream>

using namespace std;

void compAndSwap(int arr[], int i, int j, int dir) {
    if ((arr[i] > arr[j] && dir == 1) || (arr[i] < arr[j] && dir == 0)) {
        swap(arr[i], arr[j]);
    }
}

void bitonicMerge(int arr[], int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        for (int i = low; i < low + k; i++) {
            compAndSwap(arr, i, i + k, dir);
        }
        bitonicMerge(arr, low, k, dir);
        bitonicMerge(arr, low + k, k, dir);
    }
}
```

```
void bitonicSort(int arr[], int low, int cnt, int dir) {
    if (cnt > 1) {
        int k = cnt / 2;
        bitonicSort(arr, low, k, 1);
        bitonicSort(arr, low + k, k, 0);
        bitonicMerge(arr, low, cnt, dir);
    }
}

void sorted(int arr[], int N, int up) {
    bitonicSort(arr, 0, N, up);
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; ++i) {
        if (i == n - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
}

int main() {
    int arr[] = {-5, -6, -1, -4, 8, 100, -90, -15, 35};
    int up = 1;
    int tam = sizeof (arr) / sizeof (arr[0]);
    sorted(arr, tam, up);
    cout << "Array ordenado" << endl;
    printArray(arr, tam);
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def compAndSwap(arr, i, j, direc):
    if (arr[i] > arr[j] and direc == 1) or (arr[i] < arr[j] and direc == 0):
        arr[i], arr[j] = arr[j], arr[i]

def bitonicMerge(arr, low, cnt, direc):
    if cnt > 1:
        k = cnt // 2
        for i in range(low, low+k):
            compAndSwap(arr, i, i+k, direc)

        bitonicMerge(arr, low, k, direc)
        bitonicMerge(arr, low+k, k, direc)


def bitonicSort(arr, low, cnt, direc):
```

```
    if cnt > 1:
        k = cnt // 2
        bitonicSort(arr, low, k, 1)
        bitonicSort(arr, low+k, k, 0)
        bitonicMerge(arr, low, cnt, direc)


def sortB(arr, n, up):
    bitonicSort(arr, 0, n, up)


def printArray(arr):
    for i in arr:
        wr(f'{i} ')
    wr('\n')


arr = list(map(int, rl().split()))
up = 1  # 0 = Mayor a Menor
printArray(arr)
sortB(arr, len(arr), up)
printArray(arr)
```

10.10) Cocktail Sort

Cocktail Sort

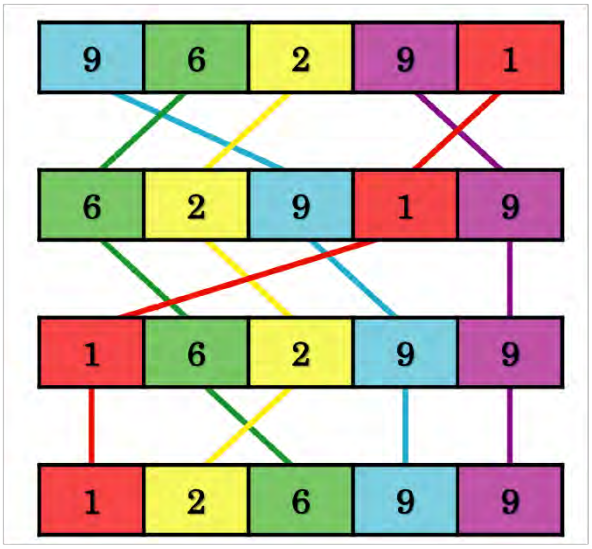


Ilustración 10-10 Ejemplo de ordenamiento coctel

Cocktail Sort es una variación de Bubble Sort. El algoritmo de ordenamiento de burbuja siempre atraviesa elementos de la izquierda y mueve el elemento más grande a su posición correcta en la primera iteración y el segundo más grande en la segunda iteración, y así sucesivamente. El ordenamiento de cóctel atraviesa un vector dado en ambas direcciones alternativamente.

Las complejidades de tiempo son iguales, pero Cocktail se desempeña mejor que Bubble Sort. Por lo general, el cóctel es menos de dos veces más rápido que el tipo burbuja. Considere el ejemplo (2, 3, 4, 5, 1). La clasificación de burbuja requiere cuatro recorridos de matriz para este ejemplo, mientras que la clasificación de cóctel requiere solo dos recorridos.

Complejidad de tiempo

Mejor caso : $O(n)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```
// Programa java implementando Cocktail sort
```

```java
public class CocktailSort {

    static void cocktailSort(int a[]) {
        boolean swapped = true;
        int start = 0;
        int end = a.length;
        while (swapped == true) {
            // Resetea la bandera intercambiada al entrar
            // en el ciclo, porque puede ser true de la
            // anterior iteración
            swapped = false;
            // Ciclo del fondo a encima igual
            // que bubble sort
            for (int i = start; i < end - 1; ++i) {
                if (a[i] > a[i + 1]) {
                    int temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                    swapped = true;
                }
            }
            // Si nada fue movido, array esta ordenado
            if (swapped == false) {
                break;
            }
            // de otra forma, resetea la bandera intercambiada
            // de tal forma que pueda ser usada en el
            //siguiente proceso
            swapped = false;
            // Mueve el punto final atras en uno, porque
            // el item al final esta en la derecha completa
            end = end - 1;
            // from top to bottom, doing the
            // same comparison as in the previous stage
            for (int i = end - 1; i >= start; i--) {
                if (a[i] > a[i + 1]) {
                    int temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                    swapped = true;
                }
            }
            // Incrementa el punto inicial, por que
            // la ultima fase pudo haber movido el siguiente
            // más pequeño número en la derecha completa
            start = start + 1;
        }
    }
    /* Imprime el array*/
    static void printArray(int a[]) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            System.out.print(a[i] + " ");
        }
```

```
        System.out.println();
    }
    public static void main(String[] args) {
        int a[] = {5, 1, 4, 2, 8, 0, 2};
        cocktailSort(a);
        System.out.println("Arreglo ordenado");
        printArray(a);
    }
}
```

C++

```
#include <iostream>

using namespace std;

void cocktailSort(int arr[], int n) {
    bool swapped = true;
    int start = 0;
    int ended = n;
    while (swapped == true) {
        swapped = false;
        for (int i = start; i < ended - 1; ++i) {
            if (arr[i] > arr[i + 1]) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
                swapped = true;
            }
        }
        if (swapped == false) {
            break;
        }
        swapped = false;
        ended = ended - 1;
        for (int i = ended - 1; i >= start; i--) {
            if (arr[i] > arr[i + 1]) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
                swapped = true;
            }
        }
        start += 1;
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
```

```
    }
}

int main() {
    int arr[] = {-5, -90, 5, 9, 7, -15, 8, 2, -8, 0, 1, 32, -35};
    int tam = sizeof (arr) / sizeof (arr[0]);
    cocktailSort(arr, tam);
    cout << "Arreglo ordenado" << endl;
    printArray(arr, tam);
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def cocktailSort(arr):
    swapped = True
    start = 0
    end = len(arr)
    while swapped:  # Sort Izq a Der
        swapped = False
        for i in range(start, end-1):
            if arr[i] > arr[i+1]:  # Cambiar < o >
                arr[i], arr[i+1] = arr[i+1], arr[i]
                swapped = True
        if not swapped:
            break

    swapped = False
    end -= 1
    for i in range(end-1, start-1, -1):  # Sort Der a Izq
        if arr[i] > arr[i+1]:  # Cambiar < o >
            arr[i], arr[i+1] = arr[i+1], arr[i]
            swapped = True

    start += 1


def printArray(arr):
    for i in arr:
        wr(f'{i} ')
    wr('\n')


arr = list(map(int, rl().split()))
printArray(arr)
cocktailSort(arr)
printArray(arr)
```

10.11) Comb Sort

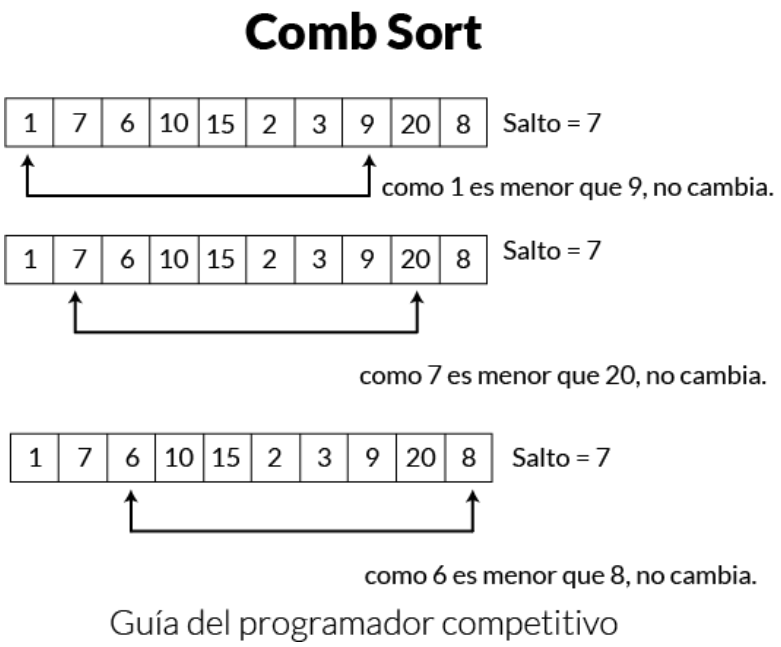


Ilustración 10-11 Ordenando un vector por medio de comb sort

Comb Sort es principalmente una mejora sobre Bubble Sort. El ordenamiento de burbuja siempre compara valores adyacentes. Así que todas las inversiones se eliminan una por una. Comb Sort mejora en Bubble Sort usando un espacio de tamaño mayor que 1. El espacio comienza con un gran valor y se reduce en un factor de 1.3 en cada iteración hasta que alcanza el valor 1. Por lo tanto, Comb Sort funciona mejor que Bubble Sort.

Aunque funciona mejor que Bubble Sort en promedio, el peor de los casos sigue siendo O (n^2).

Complejidad de tiempo

Mejor caso : $O(nlog(n))$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2/2^{incrementos})$

JAVA

```
// Programa java implementando Comb Sort
```

```java
public class CombSort {

    // Para buscar espacio entre elementos
    static int getNextGap(int gap) {
        gap = (gap * 10) / 13;
        if (gap < 1) {
            return 1;
        }
        return gap;
    }

    static void sort(int arr[]) {
        int n = arr.length;
        // Inicializa espacio
        int gap = n;
        boolean swapped = true;
        /*Mantiene ejecutando mientras gap es más que 1 y la
        ultima iteración causa un intercambio*/
        while (gap != 1 || swapped == true) {
            gap = getNextGap(gap);
            /*Inicializa swapped como falso, aso
            podemos verificar si el intercambio paso
            o no*/
            swapped = false;
            //Compara todos los elementos con el espacio actual
            for (int i = 0; i < n - gap; i++) {
                if (arr[i] > arr[i + gap]) {
                    //Intercambia arr[i] y arr[i+gap]
                    int temp = arr[i];
                    arr[i] = arr[i + gap];
                    arr[i + gap] = temp;
                    // Se intercambió
                    swapped = true;
                }
            }
        }
    }

    public static void main(String args[]) {
        int arr[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0};
        sort(arr);
        System.out.println("Array ordenado");
        for (int i = 0; i < arr.length; ++i) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

C++

```cpp
#include <iostream>

using namespace std;

int getNextGap(int gap) {
```

```
    gap = (gap * 10) / 13;
    if (gap < 1) {
        return 1;
    }
    return gap;
}

void sorted(int arr[], int n) {
    int gap = n;
    bool swapped = true;
    while (gap != 1 || swapped == true) {
        gap = getNextGap(gap);
        swapped = false;
        for (int i = 0; i < n - gap; i++) {
            if (arr[i] > arr[i + gap]) {
                int temp = arr[i];
                arr[i] = arr[i + gap];
                arr[i + gap] = temp;
                swapped = true;
            }
        }
    }
}

int main() {
    int arr[] = {8, 4, 1, 56, 3, -44, 23, -6, 28, 0, -10};
    int tam = sizeof (arr) / sizeof (arr[0]);
    sorted(arr, tam);
    cout << "Array ordenado" << endl;
    for (int i = 0; i < tam; i++) {
        if (i == tam - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
    return 0;
}
```

PYTHON

```
from sys import stdin,stdout
rl = stdin.readline
wr = stdout.write

def getNextGap(gap):
    gap = (gap * 10) // 13
    if gap < 1:
        return 1
    return gap

def combSort(arr, n):
```

```
    gap = n
    swapped = True

    while gap != 1 or swapped:
        gap = getNextGap(gap)
        swapped = False
        for i in range(n-gap):
            if arr[i] > arr[i+gap]:
                arr[i], arr[i+gap] = arr[i+gap], arr[i]
                swapped = True

arr = list(map(int, rl().split()))
combSort(arr, len(arr))
print(*arr)
```

10.12) Counting Sort

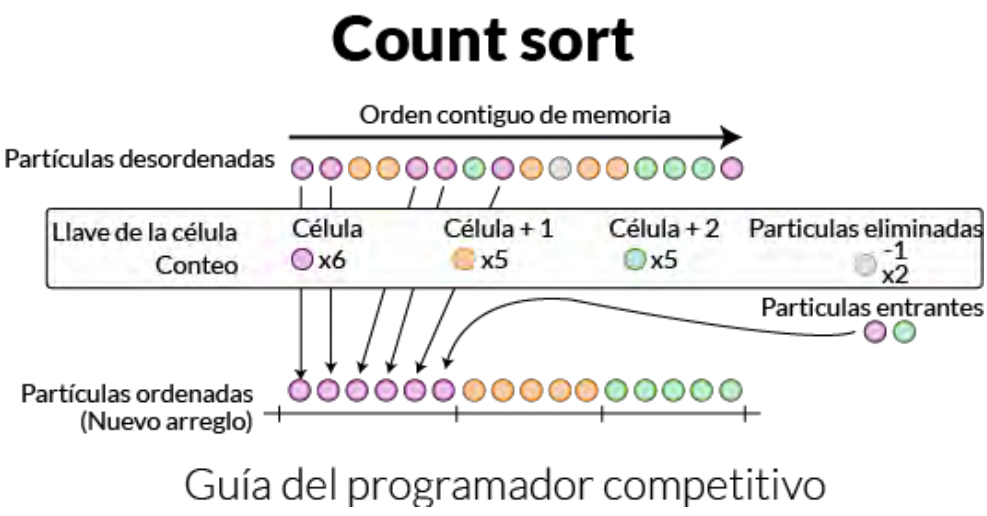


Ilustración 10-12 Ordenando un vector por medio de counting sort

Ilustración 10-13 Vectores a ordenar y vector de conteo

El ordenamiento por conteo es una técnica de ordenamiento basada en claves entre un rango específico. Funciona contando el número de objetos que tienen valores clave distintos (un tipo de hashing). Luego, hacer algo de aritmética para calcular la posición de cada objeto en la secuencia de salida.

1. El ordenamiento de conteo es eficiente si el rango de datos de entrada no es significativamente mayor que el número de objetos a clasificar.
2. No es una comparación basada en la clasificación. La complejidad del tiempo de ejecución es O (n) con espacio proporcional al rango de datos.
3. A menudo se usa como una sub-rutina para otro algoritmo de clasificación como la clasificación por radix.
4. EL ordenamiento de conteo usa un hashing parcial para contar la ocurrencia del objeto de datos en O (1).
5. El ordenamiento de conteo también se puede extender para que funcione con entradas negativas.

Complejidad de tiempo

Mejor caso : O(n+k) **Peor caso :** O(n+k) **Promedio:** O(n+k)

JAVA

```java
// Implementación java de Counting Sort

public class CountingSort {

    static void sort(char arr[]) {
        int n = arr.length;
        // El array de caracteres que sera el array
        // ordenado de salida
        char output[] = new char[n];
        // Crea un array de conteo para almacenar
        // conteo individual de caracteres e inicializar
        // array de conteo en 0
        int count[] = new int[256];
        for (int i = 0; i < 256; ++i) {
            count[i] = 0;
        }
        // Almacena el conteo de cada caracter
        for (int i = 0; i < n; ++i) {
            ++count[arr[i]];
        }
        // Cambian count[i] de forma que ahora contenga la
        //posición actual de este caracter en el array de saliday
        for (int i = 1; i <= 255; ++i) {
            count[i] += count[i - 1];
        }
        // Construye el array de caracteres de salida
        // Para hacerlo estable lo hacemos en orden inverso
        for (int i = n - 1; i >= 0; i--) {
            output[count[arr[i]] - 1] = arr[i];
            --count[arr[i]];
        }
        //Copia el array de salida a arr, asi arr ahora
        // contiene los caracteres ordenados
        for (int i = 0; i < n; ++i) {
            arr[i] = output[i];
        }
    }

    public static void main(String args[]) {
        char arr[] = {'g', 'e', 'e', 'k', 's', 'f', 'o',
            'r', 'g', 'e', 'e', 'k', 's'};
        sort(arr);
        System.out.print("array de caracteres ordenado: ");
        for (int i = 0; i < arr.length; ++i) {
            System.out.print(arr[i]);
        }
    }
}
```

C++

```cpp
#include <iostream>
```

```cpp
#include <string.h>

using namespace std;

void sorted(char arr[], int n) {
    char output[n];
    int counting[256];
    memset(counting, 0, sizeof (counting));
    for (int i = 0; i < n; i++) {
        ++counting[arr[i]];
    }
    for (int i = 1; i <= 255; ++i) {
        counting[i] += counting[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        output[counting[arr[i]] - 1] = arr[i];
        --counting[arr[i]];
    }
    for (int i = 0; i < n; ++i) {
        arr[i] = output[i];
    }
}

int main() {
    string entrada = "alkjslkjkdjaskjdkasjddf";
    char arr[entrada.size()];
    strcpy(arr, entrada.c_str());
    int tam = sizeof (arr) / sizeof (arr[0]);
    sorted(arr, tam);
    cout << "Array de caracteres ordenado: " << endl;
    for (int i = 0; i < tam; i++) {
        if (i == tam - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
    return 0;
}
```

PYTHON

```python
from sys import stdin, stdout
rl = stdin.readline
wl = stdout.write


def countSort(arr, n):

    output = [''] * n
    count = [0] * 256

    for i in arr:
        count[ord(i)] += 1
```

```
    for i in range(1, 256):
        count[i] += count[i-1]

    for i in range(n-1, -1, -1):
        output[count[ord(arr[i])]-1] = arr[i]
        count[ord(arr[i])] -= 1

    for i in range(n):
        arr[i] = output[i]


arr = list(map(str, rl().split()))
countSort(arr, len(arr))
print(*arr)
```

10.13) Cycle Sort

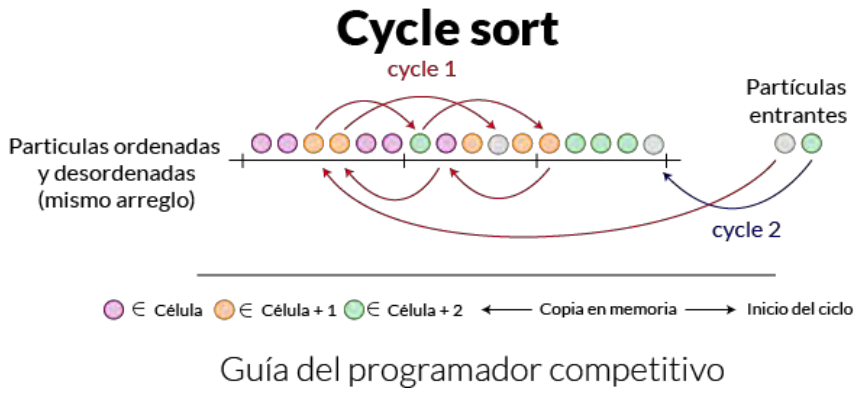


Ilustración 10-14 Ordenando un vector por medio de cycle sort

El ordenamiento por ciclos es un algoritmo de ordenamiento en el lugar, un ordenamiento de comparación que es teóricamente óptima en términos del número total de escrituras en el array original.

Es óptimo en términos de cantidad de escrituras de memoria. Minimiza el número de escrituras de memoria para ordenar (cada valor se escribe cero veces, si ya está en su posición correcta, o se escribe una vez en su posición correcta).

Complejidad de tiempo

Mejor caso : $O(n^2)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```
//Programa java implementando Cycle sort

public class CycleSort {

    public static void cycleSort(int arr[], int n) {
  //Contador de escrituras en memoria
        int writes = 0;
        //Pasar poe los elementros del array
        //y ponerlos en el lugar correcto
        for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
            // Inicializa item como punto inicial
            int item = arr[cycle_start];
            // Encuentra posicion donde nosotros pondremos el item
            // Basicamente contamos todos los elementos más pequeños
            //a la derecha del item
            int pos = cycle_start;
            for (int i = cycle_start + 1; i < n; i++) {
                if (arr[i] < item) {
                    pos++;
                }
            }
            //Si item esta ya en la posicion correcta
            if (pos == cycle_start) {
                continue;
            }
            //Ignorar todos los elementos duplicados
            while (item == arr[pos]) {
                pos += 1;
            }
            //Ponemos el item en su posición correcta
            if (pos != cycle_start) {
                int temp = item;
                item = arr[pos];
                arr[pos] = temp;
                writes++;
            }
            //Rota el resto del ciclo
            while (pos != cycle_start) {
                pos = cycle_start;
                // Encuentra posicion donde poner el elemento
                for (int i = cycle_start + 1; i < n; i++) {
```

```
                    if (arr[i] < item) {
                        pos += 1;
                    }
                }
                //Ignora todos los duplicados
                while (item == arr[pos]) {
                    pos += 1;
                }
                // Ponemos el item en su posición correcta
                if (item != arr[pos]) {
                    int temp = item;
                    item = arr[pos];
                    arr[pos] = temp;
                    writes++;
                }
            }
        }
    }

    public static void main(String[] args) {
        int arr[] = {1, 8, 3, 9, 10, 10, 2, 4};
        int n = arr.length;
        cycleSort(arr, n);
        System.out.println("Luego de ordenar : ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

C++

```
#include <iostream>

using namespace std;

void cycleSort(int arr[], int n) {
    int writes = 0;
    for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
        int item = arr[cycle_start];
        int pos = cycle_start;
        for (int i = cycle_start + 1; i < n; i++) {
            if (arr[i] < item) {
                pos++;
            }
        }
        if (pos == cycle_start) {
            continue;
        }
        while (item == arr[pos]) {
            pos += 1;
        }
        if (pos != cycle_start) {
            int temp = item;
            item = arr[pos];
```

```
            arr[pos] = temp;
            writes++;
        }
        while (pos != cycle_start) {
            pos = cycle_start;
            for (int i = cycle_start + 1; i < n; i++) {
                if (arr[i] < item) {
                    pos += 1;
                }
            }
            while (item == arr[pos]) {
                pos += 1;
            }
            if (item != arr[pos]) {
                int temp = item;
                item = arr[pos];
                arr[pos] = temp;
                writes++;
            }
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cout.tie(NULL);
    int arr[] = {1, 8, 3, 9, 10, -10, -5, -4, -8, -90, 90, 80};
    int n = sizeof (arr) / sizeof (arr[0]);
    cycleSort(arr, n);
    cout << "Luego de ordenar:" << endl;
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write


def cycleSort(arr, n):

    writes = 0
    for cyclestart in range(n-2):
        item = arr[cyclestart]
        pos = cyclestart

        for i in range(cyclestart+1, n):
```

```
            if arr[i] < item:
                pos += 1

        if pos == cyclestart:
            continue

        while item == arr[pos]:
            pos += 1

        if pos != cyclestart:
            item, arr[pos] = arr[pos], item
            writes += 1

        while pos != cyclestart:
            pos = cyclestart
            for i in range(cyclestart+1, n):
                if arr[i] < item:
                    pos += 1
            while item == arr[pos]:
                pos += 1
            if item != arr[pos]:
                item, arr[pos] = arr[pos], item
                writes += 1


arr = list(map(int, rl().split()))
cycleSort(arr, len(arr))
print(*arr)
```

10.14) 3 Way Merge Sort

Merge sort envuelve separación recursiva del array en dos partes, ordenamiento y finalmente unión de las mismas. Una variante de merge sort es llamado merge sort de tres vías donde en vez de separar el array en dos partes se separa en tres. Merge sort recursivamente rompe los arrays en arrays de tamaño medio, Merge de tres vías hace lo mismo en arrays de tamaño de un tercio.

Complejidad de tiempo: En caso de Merge Sort se tiene la ecuación T(n) = 2T(n/2) + O(n). Igualmente en caso de Merge de tres vías se tiene la ecuación(n) = 3T(n/3) + O(n) por lo que la aproximación más cercana de su complejidad es: O**(n log $_3$n).**

Complejidad de tiempo

Mejor caso : $O(n\log_3 n)$ **Peor caso :** $O(n\log_3 n)$ **Promedio:** $O(n\log_3 n)$

JAVA

```java
//Programa java que realiza Merge Sort de tres vias

public class MergeSort3Way {

    public static void mergeSort3Way(Integer[] gArray) {
        //Si el arrat tiene tamaño 0, retorna null
        if (gArray == null) {
            return;
        }
        //Creando duplicado del array dado
        Integer[] fArray = new Integer[gArray.length];
        //Copiando elementos del array dado al
        //array duplicado
        System.arraycopy(gArray, 0, fArray, 0, fArray.length);
        mergeSort3WayRec(fArray, 0, gArray.length, gArray);
        //Copiando elementos del array duplicado al
        //array dado
        System.arraycopy(fArray, 0, gArray, 0, fArray.length);
    }

    /*Realizando el algoritmo Merge Sort en el array
    dado para los valores en el rando de los indices
    [low,high], low es el menor y high es el mayor
    Exclusivo*/
    public static void mergeSort3WayRec(Integer[] gArray,
            int low, int high, Integer[] destArray) {
        //Si el array tiene tamaño 1 no haga nada
        if (high - low < 2) {
            return;
        }
        // Separando el array en tres partes
        int mid1 = low + ((high - low) / 3);
        int mid2 = low + 2 * ((high - low) / 3) + 1;
        // Ordenando las tres partes recursivamente
        mergeSort3WayRec(destArray, low, mid1, gArray);
        mergeSort3WayRec(destArray, mid1, mid2, gArray);
        mergeSort3WayRec(destArray, mid2, high, gArray);
        // Uniendo las tres partes
        merge(destArray, low, mid1, mid2, high, gArray);
    }

    /*Une los rangos ordenados (low, mid1),(mid1,mid2)
    y (mid2, high), mid 1 es el primer indice punto
    medio en todo el rando para unir mid2 es segundo*/
    public static void merge(Integer[] gArray, int low,
            int mid1, int mid2, int high,
            Integer[] destArray) {
```

```
        int i = low, j = mid1, k = mid2, l = low;
        // escoje el más pequeño de los más pequeños en los
        //tres rangos
        while ((i < mid1) && (j < mid2) && (k < high)) {
            if (gArray[i].compareTo(gArray[j]) < 0) {
                if (gArray[i].compareTo(gArray[k]) < 0) {
                    destArray[l++] = gArray[i++];
                } else {
                    destArray[l++] = gArray[k++];
                }
            } else {
                if (gArray[j].compareTo(gArray[k]) < 0) {
                    destArray[l++] = gArray[j++];
                } else {
                    destArray[l++] = gArray[k++];
                }
            }
        }
        while ((i < mid1) && (j < mid2)) {
            if (gArray[i].compareTo(gArray[j]) < 0) {
                destArray[l++] = gArray[i++];
            } else {
                destArray[l++] = gArray[j++];
            }
        }
        while ((j < mid2) && (k < high)) {
            if (gArray[j].compareTo(gArray[k]) < 0) {
                destArray[l++] = gArray[j++];
            } else {
                destArray[l++] = gArray[k++];
            }
        }
        while ((i < mid1) && (k < high)) {
            if (gArray[i].compareTo(gArray[k]) < 0) {
                destArray[l++] = gArray[i++];
            } else {
                destArray[l++] = gArray[k++];
            }
        }
        while (i < mid1) {
            destArray[l++] = gArray[i++];
        }
        while (j < mid2) {
            destArray[l++] = gArray[j++];
        }
        while (k < high) {
            destArray[l++] = gArray[k++];
        }
    }

    public static void main(String args[]) {
        Integer[] data = new Integer[]{45, -2, -45, 78,
            30, -42, 10, 19, 73, 93};
        mergeSort3Way(data);
```

```
        System.out.println("Despues de merge sort de 3 vias: ");
        for (int i = 0; i < data.length; i++) {
            System.out.print(data[i] + " ");
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

void mergeArr(int gArray[], int low, int mid1, int mid2, int high, int
arrdest[]) {
    int i = low, j = mid1, k = mid2, l = low;
    while ((i < mid1) && (j < mid2) && (k < high)) {
        if (gArray[i] < gArray[j]) {
            if (gArray[i] < gArray[k]) {
                arrdest[l++] = gArray[i++];
            } else {
                arrdest[l++] = gArray[k++];
            }
        } else {
            if (gArray[j] < gArray[k]) {
                arrdest[l++] = gArray[j++];
            } else {
                arrdest[l++] = gArray[k++];
            }
        }
    }
    while ((i < mid1) && (j < mid2)) {
        if (gArray[i] < gArray[j]) {
            arrdest[l++] = gArray[i++];
        } else {
            arrdest[l++] = gArray[j++];
        }
    }
    while ((j < mid2) && (k < high)) {
        if (gArray[j] < gArray[k]) {
            arrdest[l++] = gArray[j++];
        } else {
            arrdest[l++] = gArray[k++];
        }
    }
    while ((i < mid1) && (k < high)) {
        if (gArray[i] < gArray[k]) {
            arrdest[l++] = gArray[i++];
        } else {
            arrdest[l++] = gArray[k++];
        }
    }
    while (i < mid1) {
        arrdest[l++] = gArray[i++];
    }
```

```
    while (j < mid2) {
        arrdest[l++] = gArray[j++];
    }
    while (k < high) {
        arrdest[l++] = gArray[k++];
    }
}

void mergeSort3wayRec(int arr[], int low, int high, int arrdest[]) {
    if (high - low < 2) {
        return;
    }
    int mid1 = low + ((high - low) / 3);
    int mid2 = low + 2 * ((high - low) / 3) + 1;
    mergeSort3wayRec(arrdest, low, mid1, arr);
    mergeSort3wayRec(arrdest, mid1, mid2, arr);
    mergeSort3wayRec(arrdest, mid2, high, arr);

    mergeArr(arrdest, low, mid1, mid2, high, arr);
}

void mergeSort3way(int arr[], int n) {
    if (n == 0) {
        return;
    }
    int farr[n];
    for (int i = 0; i < n; i++) {
        farr[i] = arr[i];
    }
    mergeSort3wayRec(farr, 0, n, arr);
    for (int i = 0; i < n; i++) {
        arr[i] = farr[i];
    }
}

void printArr(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {1, 3, 54, 255, 2, 5, 3, 1, 5, 7, 4, 299, 3, 54, 64, 73};
    int N = sizeof (arr) / sizeof (arr[0]);
    mergeSort3way(arr, N);
    cout << "arreglo ordenados" << endl;
    printArr(arr, N);
}
```

10.15) Pigeon Hole Sort

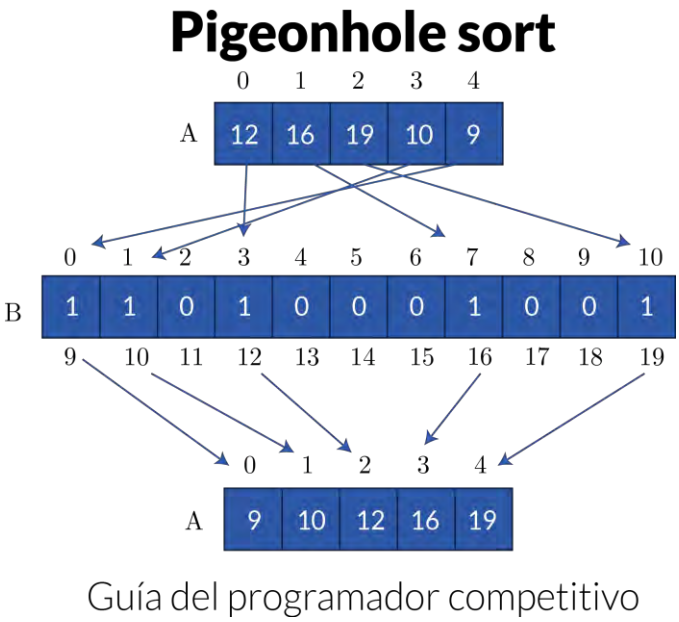


Ilustración 10-15 Ordenando un vector por medio de pigeonhole sort

Ordenamiento nido de Paloma es un algoritmo de ordenamiento que es adecuado para ordenar listas de elementos donde el número de elementos y el número de valores de los mismos son aproximadamente lo mismo.

Este requiere un tiempo de O(n + Rango) donde n es el número de elementos en el array de entrada y Rango es el número de posibles valores en el array. Este algoritmo es similar a Counting sort, pero difiere en la forma en que mueve los datos, dos veces.

Complejidad de tiempo

Mejor caso : O(N+n) **Peor caso :** O(N+n) **Promedio:** O(N+n)

N=rango

JAVA

```
/*Programa java que implementa Pigeonhole Sort*/
import java.util.*;

public class PigeonholeSort {
```

```
    static void pigeonhole_sort(int arr[],int n) {
        int min = arr[0];
        int max = arr[0];
        int range, i, j, index;
        for (int a = 0; a < n; a++) {
            if (arr[a] > max) {
                max = arr[a];
            }
            if (arr[a] < min) {
                min = arr[a];
            }
        }
        range = max - min + 1;
        int[] phole = new int[range];
        Arrays.fill(phole, 0);
        for (i = 0; i < n; i++) {
            phole[arr[i] - min]++;
        }
        index = 0;
        for (j = 0; j < range; j++) {
            while (phole[j]-- > 0) {
                arr[index++] = j + min;
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {8, 3, 2, 7, 4, 6, 8};
        System.out.print("Array ordenado : ");
        pigeonhole_sort(arr, arr.length);
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

C++

```
#include <iostream>
#include <string.h>
#include <vector>

using namespace std;

void pigeonHoleSort(int arr[], int n) {
    int minim = arr[0];
    int maxim = arr[0];
    int range, i;
    for (int a = 0; a < n; a++) {
        if (arr[a] > maxim) {
            maxim = arr[a];
        }
        if (arr[a] < minim) {
            minim = arr[a];
```

```
        }
    }
    range = maxim - minim + 1;

    //Crear un arreglo de vectores de tamamio range.
    //Cada vector representa un agujero que estara
    //para contener los elementos que coincidentes
    vector<int> pHole[range];
    memset(pHole, 0, range);

    //Atravesar el arreglo de entrada y colocar cada elemento
    //en su respectivo agujero
    for (i = 0; i < n; i++) {
        pHole[arr[i] - minim].push_back(arr[i]);
    }

    //Recorrer todos los agujeros uno a uno.
    //Para cada agujero, se toma sus elementos y se
    //colocan en el arreglo dado.
    int index = 0;
    for (i = 0; i < range; i++) {
        for (int k : pHole[i]) {
            arr[index++] = k;
        }
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cout.tie(NULL);
    int arr[] = {-8, 4, -5, 8, -6, 90, 70, -95, -100, 105, 97, -1000};
    cout << "Arreglo ordenado:" << endl;
    int n = sizeof (arr) / sizeof (arr[0]);
    pigeonHoleSort(arr, n);
    for (int i = 0; i < n; i++) {
        if (i == n - 1) {
            cout << arr[i] << endl;
            break;
        }
        cout << arr[i] << " ";
    }
    return 0;
}
```

PYTHON

```
from sys import stdin
from sys import stdout

in_, out = stdin.readline, stdout.write

def gen(arr): yield from arr

datos_int = lambda: list(map(int, in_().strip().split()))
```

```
def pigeonSort(arr, n):
    mn = arr[0]
    mx = arr[0]
    for k in range(n):
        if arr[k] > mx:
            mx = arr[k]
        if arr[k] < mn:
            mn = arr[k]
    range_ = mx - mn + 1
    phole = [0] * range_
    for i in range(n):
        phole[arr[i] - mn] += 1
    index = 0
    for j in range(range_):
        while phole[j]:
            arr[index] = j + mn
            index += 1
            phole[j] -= 1

lista = datos_int()
n = len(lista)
pigeonSort(lista, n)
for i in gen(lista):
    out(f"{i} ")
print()
```

10.16) Problemas de repaso

Ejercicios en Online Judge

450-Little Black Book

741-Burrows Wheeler Decoder

885-Telephone Directory Alphabetization

895- Word Problem

10041- Vito's Family

10107-What is the Median?

10152-CDV

Ejercicios en CodeChef

FLOW007

FLOW017

PBR01 FLOW016

Capitulo 11. Patrones y manejo de Strings

Una cadena de caracteres (String) es una secuencia ordenada de elementos que pertenecen a un lenguaje o alfabeto, las cuales se ven como una oración o una formula, más coloquialmente como una frase de texto, estas pueden contener valores alfabéticos, numéricos, símbolos, espacios en blanco y cualquier carácter imprimible, son almacenados en un vector en el cual cada espacio de este es un carácter, y la suma de todos los espacios conforman a la cadena de caracteres. Dentro de estas cadenas se pueden ejecutar diversos algoritmos que permiten la búsqueda de patrones, palabras y subcadenas con características previamente descritas a su ejecución.

11.1) Algoritmo de Knuth-Morris-Pratt

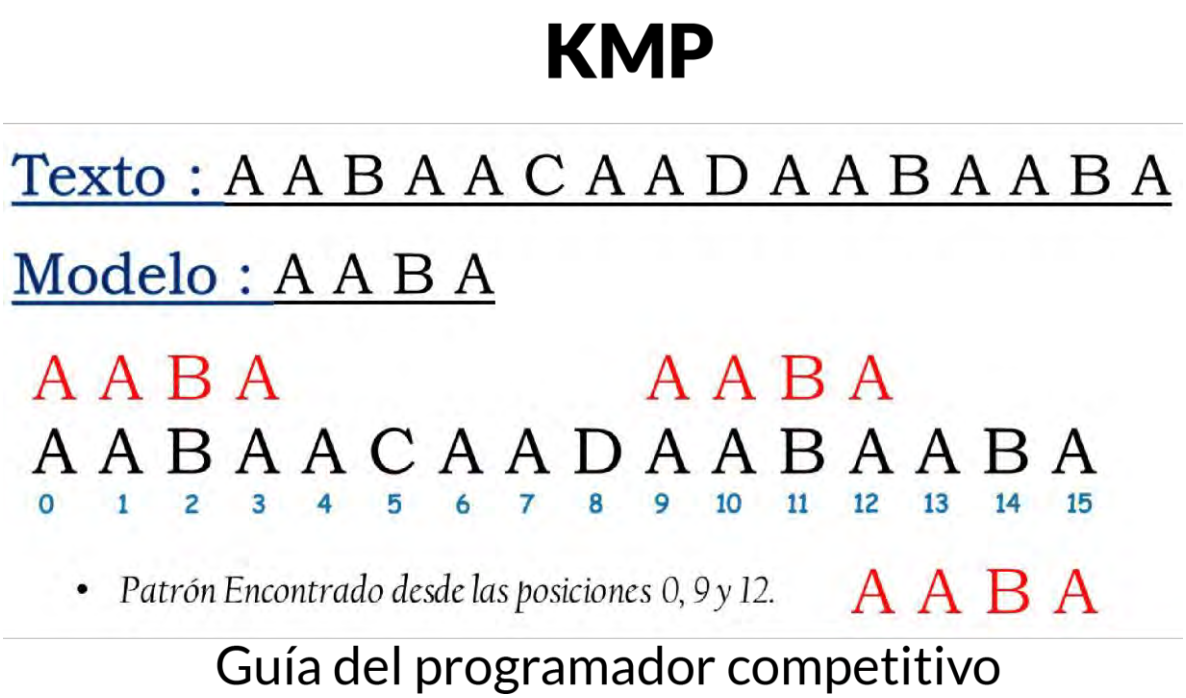


Ilustración 11-1 Búsqueda de patrón KMP

Dado un texto txt[0…n-1] y un patrón pat[0...m-1] busque todas las ocurrencias de pat en txt e imprímalas, asuma que n es mayor que m.

El algoritmo KMP usa la propiedad de degeneración (Usar los subpatrones que se puedan crear a partir del patrón) del patrón y mejora el peor caso de complejidad de O(n). La idea básica detrás de KMP es: Cuando detectemos una no coincidencia (luego de algunas coincidencias), ya sabemos algunos de los caracteres en el texto de la siguiente ventana. Tomamos ventaja de esta información para evadir coincidencias de caracteres que sabemos que de todas maneras coincidirán.

Revisión del preprocesado:

KMP preprocesa pat[] y construye un lps[] auxiliar de tamaño m (El mismo tamaño que el patrón) el cual es usado para saltar caracteres mientras se coteja.

El nombre lps indica el prefijo propio más largo el cual es también sufijo. Un prefijo propio es un prefijo con todo el string no permitido. Por ejemplo, prefijos de "ABC" son "","A","AB" Y "ABC", prefijos propios son "", "A" y "AB". Sufijos del string son "", "C","BC" y "ABC".

Buscamos por lps en subpatrones. Más claramente nos enfocamos en los subsring de patrones que son prefijos y sufijos. Por cada subpatron pat[0...i] donde i=0 hasta m-1, lps[i] almacena la longitud del máximo prefijo propio coincidente el cual también es sufijo del subpatron pat[0...i].

Algoritmo de búsqueda:

A diferencia de los algoritmos ingenuos, donde deslizamos el patrón uno por uno y comparamos todos los caracteres en cada movimientos, usamos un valor de lps[] para decidir los nuevos caracteres que serán cotejados, la idea es no cotejar un carácter que sabemos que va a aparecer.

¿Cómo usamos lps [] para decidir las siguientes posiciones o número de caracteres que serán saltadas?

Comenzamos la comparación de pat [j] con j=0 con los caracteres de la actual ventana de texto. Seguimos cotejando caracteres de txt[i] y pat [j] y seguimos incrementando i y j mientras pat[j] y txt [i] sigan coincidiendo.

Cuando vemos una no coincidencia, sabemos que los caracteres pat[0…j-1] coinciden con txt[i-j…..i-1] (Nótese que j inicia en 0 e incrementa solo cuando hay una coincidencia).

También sabemos que lps[j-1] está contando los caracteres de pat[o…j-1] que son prefijo propio y sufijo.

De estos puntos podemos concluir que nosotros no necesitaremos cotejar estos lps[j-1] caracteres con txt porque sabemos que de todas formas esos caracteres coincidirán.

Complejidad de tiempo

Mejor caso : On) **Peor caso :** O(n+m) **Promedio:** O(m+n)

JAVA

```
/* Programa java implementando el algoritmo
de busqueda KMP*/
public class KMPStringMatching {

    static void KMPSearch(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();
        // Crea lps[] el cual podra mantener el
        // más largo prefijo sufijo para patrón
        int lps[] = new int[M];
        int j = 0; // indice de pat[]
        // Preprocesa el patrón (Calcula lps[])
        computeLPSArray(pat, M, lps);
        int i = 0; // indice para txt[]
        while (i < N) {
            if (pat.charAt(j) == txt.charAt(i)) {
                j++;
                i++;
            }
            if (j == M) {
                System.out.println("Encontrado patrón "
                        + "en indice " + (i - j));
                j = lps[j - 1];
            } // no coincide luego de j veces
            else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                /*No coincide lps[0..lps[j-1]] caracteres
                ellos coincidiran de todos modos*/
                if (j != 0) {
                    j = lps[j - 1];
                } else {
                    i = i + 1;
                }
            }
        }
    }

    static void computeLPSArray(String pat, int M, int lps[]) {
        // Longitus del anterior prefijo sufijo más largo
        int len = 0;
```

```
        int i = 1;
        lps[0] = 0; // lps[0] es siempre 0
        // el ciclo calcula lps[i] para i = 1 hasta M-1
        while (i < M) {
            if (pat.charAt(i) == pat.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            } else // (pat[i] != pat[len])
            {
                // Esto es dificl, considere el ejemplo
                // AAACAAAA y i = 7. la idea es similar
                // el paso de busqueda
                if (len != 0) {
                    len = lps[len - 1];
                    // Tambien, note que no incrementamos
                    //i aqui
                } else {
                    // if (len == 0)
                    lps[i] = len;
                    i++;
                }
            }
        }
    }

    public static void main(String args[]) {
        String txt = "ABABDABACDABABCABAB";
        String pat = "ABABCABAB";
        KMPSearch(pat, txt);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

void computeLPSArray(string pat, int M, int lps[]) {
    int len = 0;
    int i = 1;
    lps[0] = 0;
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        } else {
            if (len != 0) {
                len = lps[len - 1];
            } else {
                lps[i] = len;
                i++;
            }
```

```
        }
    }
}

void KMPsearch(string pat, string txt) {
    int M = pat.size();
    int N = txt.size();
    int lps[M];
    int j = 0;
    computeLPSArray(pat, M, lps);
    int i = 0;
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }
        if (j == M) {
            cout << "Patron encontrado en el indice " << (i - j) << endl;
            j = lps[j - 1];
        } else if (i < N && pat[j] != txt[i]) {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
}

int main() {
    string txt = "lalalalalala";
    string pat = "lal;
    KMPsearch(pat, txt);
}
```

PYTHON

```
def KMPsearch(pat, txt):
    M = len(pat)
    N = len(txt)
    lps = [None for x in range(M)]
    j = 0
    computeLPSArray(pat, M, lps)
    i = 0
    while i < N:
        if pat[j] == txt[i]:
            i += 1
            j += 1
        if j == M:
            print("Encontrado patrÃ³n (", pat, ") en el indice ", (i - j), "-",
((i - j) + M - 1))
            j = lps[j - 1]
        elif i < N and pat[j] != txt[i]:
            if j != 0:
                j = lps[j-1]
```

```
            else:
                i += 1

def computeLPSArray(pat, M, lps):
    leng = 0
    i = 1
    lps[0] = 0
    while i < M:
        if pat[i] == pat[leng]:
            leng += 1
            lps[i] = leng
            i += 1
        else:
            if leng != 0:
                leng = lps[leng-1]
            else:
                lps[i] = leng
                i += 1

def variasBusquedas(arr, txt):
    for i in range(len(arr)):
        KMPsearch(arr[i], txt)

txt = "lalalalalalalalalalala"
arr = ["la", "lal", "lala"]
variasBusquedas(arr, txt)
```

11.2) Algoritmo de Rabin-Karp

Rabin Karp

Guía del programador competitivo

Ilustración 11-2 Búsqueda de patrón por Rabin-Karp

Dado un texto txt[0...n-1] y un patrón pat[0...m-1] busque todas las ocurrencias de pat en txt e imprímalas, asuma que n es mayor que m.

Como los algoritmos ingenuos, Rabin-Karp también desliza el patrón uno por uno, pero a diferencia de este, RK coteja el valor hash del patrón con el valor hash del actual substring de texto y su los valores hash coinciden entonces solo empieza a cotejar caracteres individuales, entonces RK necesita calcular los valores hash de los siguientes strings.

1) El string patrón
2) Todos los substrings del texto de tamaño m.

Desde que necesitamos calcular eficientemente los valores hash de todos los substrings de tamaño m del texto, debemos tener una función hash la cual tenga la siguiente propiedad. El hash del siguiente cambio debe ser eficientemente calculable desde el hash actual y el siguiente carácter en el texto o podemos decir hashtxt[s+1….. s+m])= rehash (txt[s+m], hash(txt[s….s+m-1]) y rehash debe ser una operación O(1).

La función hash sugerida por Rabin y Karp calcula un valor entero, el valor entero para un string es un valor número de un string, por ejemplo el número si todos los posibles

caracteres son de 1 a 10, el valor número de “122” seria 122. El número de posibles caracteres es mayor que 10 (256 en general) y la longitud el patrón puede ser larga. Entonces los valores numéricos no pueden ser prácticamente almacenados como un entero. Sin embargo el valor número es calculado usando matemática modular para asegurar que los valores hash pueden ser almacenados en una variable entra (puede caber en palabras de memoria). Para hacer rehashing, necesitamos tomar el más significante digito y añadirlo nuestro nuevo digito significante para el valor hash, el rehashing se realiza con la siguiente formula

- hash(txt[s+1 .. s+m]) = (d (hash(txt[s .. s+m-1]) – txt[s]*h) + txt[s + m]) mod q
- hash(txt[s .. s+m-1]) : Valor hash en cambio s.
- hash(txt[s+1 .. s+m]) : Valor has en nuevo cambio (cambio s+1)
- d:Número de caracteres en el alfabeto
- q: Un número primo
- h: d^(m-1)

Esto es matemática simple, calculamos el valor decimal de la actual ventana desde la ventana anterior.

Por ejemplo el tamaño del patrón es 3 y el string es “23456”

Se calcula el valor de la primera ventana el cual es 234 (String ventana es “234”).

¿Cómo puedes calcular el valor de la siguiente ventana “345”? se puede hacer (234 – 2*100)*10 + 5 y obtener 345.

El promedio y mejor tiempo de ejecución en un caso de RK es de O(n+m) pero su peor caso es O(nm). El peor caso de Rabin-Karp ocurre cuando todos los caracteres del patrón y el texto tienen los mismos valores hash de todos los substrings de txt, por ejemplo pat[]=”AAA” y txt[]=”AAAAAAA”.

Complejidad de tiempo

Mejor caso : O**(**m+n) **Peor caso :** O**(**n*m**)** **Promedio:** O**(**n*m**)**

JAVA

```java
/*Implementación java del algoritmo de
Rabin Karp*/
public class RabinKarp {
// d es elnúmero de caracteres en el alfabeto de entrada

    public final static int d = 256;

    /* pat -> patrón
        txt -> texto
        q -> Un número primo
     */
    static void search(String pat, String txt, int q) {
        int M = pat.length();
        int N = txt.length();
        int i, j;
        int p = 0; // valor hash  del patrón
        int t = 0; // valor hash  del txt
        int h = 1;
        // el valor de h debe ser "pow(d, M-1)%q"
        for (i = 0; i < M - 1; i++) {
            h = (h * d) % q;
        }
        /*Calcula el valor hash del patron y primera
        ventana de texto*/
        for (i = 0; i < M; i++) {
            p = (d * p + pat.charAt(i)) % q;
            t = (d * t + txt.charAt(i)) % q;
        }
        //Desliza el patrón por encima del texto uno por uno
        for (i = 0; i <= N - M; i++) {
            /*Verifica los valores hash de la actual ventana de text
            y patrón. Si el valor hash coincide entonces solo revisa
            los caratceres uno por uno*/
            if (p == t) {
                /*Revisa por caracteres uno por uno*/
                for (j = 0; j < M; j++) {
                    if (txt.charAt(i + j) != pat.charAt(j)) {
                        break;
                    }
                }
                // Si p==t y pat[0...M-1] = txt[i, i+1, ...i+M-1]
                if (j == M) {
                    System.out.println("Patron encontrado en el indice " + i);
                }
            }
            // Calcula el valor hash de la siguiente ventana de texto
            // Remueve el digito lider, y final
            if (i < N - M) {
                t = (d * (t - txt.charAt(i) * h) + txt.charAt(i + M)) % q;
                /*Nosotros obtendremos un valor negativo de t
                convirtiendolo a positivo*/
                if (t < 0) {
```

```
                    t = (t + q);
                }
            }
        }
    }

    public static void main(String[] args) {
        String txt = "EQUIPO ARTEMIS";
        String pat = "ARTE";
        int q = 101; // un número primo
        search(pat, txt, q);
    }
}
```

C++

```
using namespace std;
#include<bits/stdc++.h>
#include<cstdlib>
#define FAST
ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(0);cout<<setprecision(25
);
int d = 256;

void rabinKarp(string pat, string txt, int q) {
    int M = pat.size();
    int N = txt.size();
    int i, j; //indices
    int p = 0;
    int t = 0;
    int h = 1; //tamanio del hash
    for (i = 0; i < M - 1; i++) {
        h = (h * d) % q;
    }
    for (i = 0; i < M; i++) {
        p = (d * p + pat[i]) % q;
        t = (d * i + txt[i]) % q;
    }
    for (i = 0; i <= N - M; i++) {
        if (p == t) {
            for (j = 0; j < M; j++) {
                if (txt[i + j] != pat[j]) {
                    break;
                }
            }
            if (j == M) {
                cout << "Patron encontrado en el indice " << i << " " << (i + M
- 1) << endl;
            }
        }
        if (i < N - M) {
            t = (d * (t - txt[i] * h) + txt[i + M]) % q;
            if (t < 0) {
                t = (t + q);
            }
```

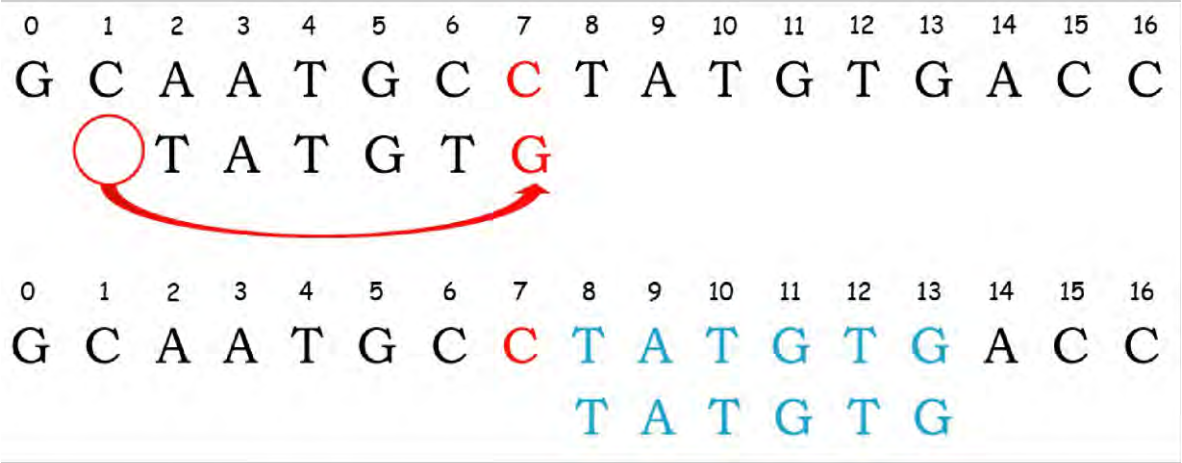
```
        }
    }
}

int main() {
    FAST
    string txt = "anita lava la tina";
    string pat = "la";
    rabinKarp(pat, txt, d);
}
```

PYTHON

```
from sys import stdin
from sys import stdout
d = 256


def rabinKarp(pat, txt, q):
    M = len(pat)
    N = len(txt)
    i, j, p, t, h = int(0), int(0), int(0), int(0), int(1)
    for i in range(M-1):
        h = (h * d) % q
    for i in range(M):
        p = (d * p + ord(pat[i])) % q
        t = (d * t + ord(txt[i])) % q
    for i in range(N-M + 1):
        if p == t:
            for j in range(M):
                if txt[i + j] != pat[j]:
                    break
            j += 1
            if j == M:
                stdout.write(f'Patron en {str(i)} - {str(i+M-1)}\n')
        if i < N-M:
            t = (d * (t-ord(txt[i]) * h) + ord(txt[i + M])) % q
            if t < 0:
                t = t + q


txt = 'Anita lava la tina'
pat = 'la'
rabinKarp(pat, txt, d)
```

11.3) Algoritmo de Boyer-Moore

Boyer Moore



Guia del programador competitivo

Ilustración 11-3 Ejemplo de búsqueda de patrón Boyer-Moore

Cuando realizamos una búsqueda de un string en Notepad, Word, buscador o una base de datos, los algoritmos de búsquedas de patrones son usados para buscar los resultados. Un enunciado de este problema podría ser:

Dado un texto txt[0…n-1] y un patrón pat[0...m-1] busque una función que imprima todas las ocurrencias de pat en txt, se asume que n es mayor que m.

Como KMP y Boyer Moore también preprocesa el patrón, Boyer Moore es la combinación de las siguientes dos aproximaciones:

1) Heurística de caracteres malos
2) Heurística de buenos sufijos

Ambas de las dos heurísticas de arriba pueden también ser usadas independientemente para buscar un patrón en un texto. Primero entendamos como estas dos aproximaciones trabajan juntos en Boyer Moore. Si tomamos un vistazo a cualquier algoritmo ingenuo, desliza el patrón sobre el texto carácter por carácter. KMP hace preprocesado sobre el patrón de tal forma que el patrón puede ser cambiado más de una vez.

El algoritmo de Boyer Moore hace preprocesado por la misma razón. Este procesa el patrón y crea arrays diferentes para cada heurística. En cada paso desliza el patrón por el máximo de deslices sugeridos por las dos heurísticas, entonces usa la mejor de las dos heurísticas en cada paso.

A diferencia de los anteriores algoritmos de búsqueda de patrones, Boyer Moore comienza el cotejamiento desde el último carácter del patrón.

Heurística de caracteres malos

La idea de esta heurística es simple. El carácter del texto que no coincida con el carácter actual del patrón es llamado un carácter malo. Con esta no coincidencia nosotros movemos el patrón hasta:

1) La no coincidencia se convierta en un cotejamiento positivo.
2) Patrón P se mueve después del carácter no coincidente.

Caso 1: No coincidencia se convierte en una coincidencia

Buscamos la posición de la última ocurrencia del carácter no coincidente en el patrón y si el carácter no coincidente existe en el patrón, entones movemos el patrón de tal manera que quede alineado con el carácter no coincidente en el texto T**.**

Caso 2: El patrón se mueve pasado el carácter no coincidente

Nosotros buscamos la posición de la última ocurrencia del carácter no coincidente en el patrón y si el carácter no existe deberíamos mover el patrón pasado ese carácter.

Complejidad de tiempo

Mejor caso : O**(**n**)** **Peor caso :** O**(**3n**)** **Promedio:** O**(**n**)**

JAVA

```
/*Programa java para heuristica de malos caracteres
usando el algoritmo de Boyer Moore*/

public class BoyerMoore {

    static int NO_OF_CHARS = 256;
    //Una función de utilidad para obtener el maximo de
```

```java
    //dos enteros
    static int max(int a, int b) {
        return (a > b) ? a : b;
    }
    //El preprocesado del algoritmo
    static void badCharHeuristic(char[] str, int size, int badchar[]) {
        int i;
        // Inicializa todas las ocurrencias en  -1
        for (i = 0; i < NO_OF_CHARS; i++) {
            badchar[i] = -1;
        }
        /*Llena el actual valor de la ultima ocurrencia
        de un caracter*/
        for (i = 0; i < size; i++) {
            badchar[(int) str[i]] = i;
        }
    }

    /* Una función de busqueda de patrón que usa
     la heuristica de mal caracter*/
    static void search(char txt[], char pat[]) {
        int m = pat.length;
        int n = txt.length;
        int badchar[] = new int[NO_OF_CHARS];
        /* Llena el arreglo de malos caracteres llamando
        la función de preprocesado para el patrón dado*/
        badCharHeuristic(pat, m, badchar);
        int s = 0;  // s  es cambiado del patron con respecto al texto
        while (s <= (n - m)) {
            int j = m - 1;
            /* Mantiene reduciento el indice j para el patrón
            mientras los caracteres del patrón y el texto
            estan coincidiendo en s*/
            while (j >= 0 && pat[j] == txt[s + j]) {
                j--;
            }
            /*Si el patrón esta presente en el actual
            cambio, entonces el indice j se convertira en -1
            luego del ciclo de arriba*/
            if (j < 0) {
                System.out.println("Patrón encontrado en cambio = " + s);
                s += (s + m < n) ? m - badchar[txt[s + m]] : 1;
            } else {
                s += max(1, j - badchar[txt[s + j]]);
            }
        }
    }
    public static void main(String[] args) {
        char txt[] = "ABAAABCDABCABC".toCharArray();
        char pat[] = "ABC".toCharArray();
        search(txt, pat);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
const int NO_OF_CHARS = 256;

int MAX(int a, int b) {
    return a > b ? a : b;
}

void badCharHeuristic(char str[], int siz, int badchar[]) {
    int i;
    memset(badchar, -1, sizeof badchar);
    for (i = 0; i < siz; i++) {
        badchar[(int) str[i]] = i;
    }
}

void boyerMoore(char txt[], char pat[]) {
    int m = strlen(pat);
    int n = strlen(txt);
    int badchar[NO_OF_CHARS];
    badCharHeuristic(pat, m, badchar);
    int s = 0;
    while (s <= (n - m)) {
        int j = m - 1;
        while (j <= 0 && pat[j] == txt[s + j]) {
            j--;
        }
        if (j > 0) {
            cout << "Patron encontrado encontrado en salto " << s << endl;
            s += (s + m < m) ? badchar[txt[s + m]] : 1;
        } else {
            s += MAX(1, j - badchar[txt[s + j]]);
        }
    }
}

int main() {
    char txt[] = "lalalalalalalala";
    char pat[] = "lala";
    boyerMoore(txt, pat);
}
```

PYTHON

```python
from sys import stdout
wr = stdout.write

NO_OF_CHARS = 256

def badChar(string, size, badchar):
    for i in range(size):
        badchar[ord(string[i])] = i
```

```
def BMsearch(txt, pat):

    M = len(pat)
    N = len(txt)
    badchar = [0] * NO_OF_CHARS
    badChar(pat, M, badchar)
    s = 0

    while s <= N-M:
        j = M-1
        while j >= 0 and pat[j] == txt[s+j]:
            j -= 1
        if j < 0:
            wr(f'"{"".join(txt[s:s+M])}" encontrado en el indice ({s} - {s+M-
1})\n')
            s += M - badchar[ord(txt[s+M])] if s + M < N else 1
        else:
            s += max(1, j - badchar[ord(txt[s+j])])


txt = 'anitalavalatina'
pat = 'al'
BMsearch(txt, pat)
```

11.4) Patrón en anagramas

Anagramas

Word	Rearrange the letters
DORMITORY	DIRTY ROOM
PRESBYTERIAN	BEST IN PRAYER
ASTRONOMER	MOON STARER
DESPERATION	A ROPE ENDS IT
THE EYES	THEY SEE
GEORGE BUSH	HE BUGS GORE
THE MORSE CODE	HERE COME DOTS
ANIMOSITY	IS NO AMITY

Ilustración 11-4 Ejemplo de anagramas según una palabra inicial

Dado un texto txt [0...n-1] y un patrón pat [0...m-11], use una función que imprima todas las ocurrencias de pat [] y sus permutaciones (o anagramas) en txt [], se asume que n es menor que m.

La complejidad de tiempo esperada es de O(n).

Este problema es ligeramente diferente a la búsqueda de patrones estándar, aquí necesitamos buscar por anagramas también. Por lo tanto no podemos aplicar directamente la búsqueda de patrones estándar de algoritmos como KMP, Rabin Karp o Boyer Moore.

Podemos conseguir una complejidad de tiempo de O(n) asumiendo que el tamaño del alfabeto está arreglado en los 256 caracteres ASCII. La idea es usar dos arrays de conteo:

1) El primer array de conteo almacena la frecuencia de los caracteres en el patrón.
2) El Segundo array de conteo almacena la frecuencia en la actual ventana de texto.

Una cosa importante a tener en cuenta es, la complejidad de tiempo de comparar dos arrays de conteo es O(1) como el número de elementos en ellos. Estos son los pasos de este algoritmo:

1) Almacena conteos de frecuencia del patrón en el primer array de conteo countP []. También almacena conteo de frecuencias de la primera ventana de texto en el array countTW[].
2) Ahora ejecuta un ciclo de i=M hasta N-1 haciendo lo siguiente en cada ciclo:

a) Si los dos arrays de conteo son idénticos, hemos encontrado una ocurrencia.

b) Incrementa el conteo del actual carácter del texto en countTW[]

c) Decrementa conteo del primer carácter en la ventana anterior en countWT[]

3) La última ventana no es revisada por este ciclo, se revisa explícitamente.

Complejidad de tiempo

Mejor caso : O**(n)** **Peor caso :** O**(**n**)** **Promedio:** O**(**n**)**

JAVA

```
// Programa java que busca todos los anagramás
// de un patrón en un texto

public class AnagramsPattern {

    static final int MAX = 256;
    // Esta función retorna true si los contenidos
    // de arr1[] y arr2[] son iguales, de otra forma es falso

    static boolean compare(char arr1[], char arr2[]) {
        for (int i = 0; i < MAX; i++) {
            if (arr1[i] != arr2[i]) {
                return false;
            }
        }
        return true;
    }
    // Esta función busca todas las permutaciones de
    // pat[] en txt[]

    static void search(String pat, String txt) {
        int M = pat.length();
        int N = txt.length();
        // countP[]:  Almacena el conteo de todos
        // los caracteres del patrón
```

```
        // countTW[]: Almacena el conteo de los caracteres
        // la ventana de texto
        char[] countP = new char[MAX];
        char[] countTW = new char[MAX];
        for (int i = 0; i < M; i++) {
            (countP[pat.charAt(i)])++;
            (countTW[txt.charAt(i)])++;
        }
        //Atravesar atravez de los caracteres restantes de patrón
        for (int i = M; i < N; i++) {   //Compara conteos de la ventana actual
            // de texto con los conteos de pattern[]
            if (compare(countP, countTW)) {
                System.out.println("Encontrado en indice "
                        + (i - M));
            }
            // Agreca el actual caracter a la ventana actual
            (countTW[txt.charAt(i)])++;
            // Remueve el primer caracter de la anterior ventana
            countTW[txt.charAt(i - M)]--;
        }
        // Revisa por la ultima ventana en el texto
        if (compare(countP, countTW)) {
            System.out.println("Encontrado en indice "
                    + (N - M));
        }
    }

    public static void main(String args[]) {
        String txt = "BACDGABCDA";
        String pat = "ABCD";
        search(pat, txt);
    }
}
```

C++

```
#include <iostream>
#include <string.h>

using namespace std;

const int MAX = 256;

bool compare(char arr1[], char arr2[]){
    for(int i = 0; i < MAX; i++){
        if(arr1[i] != arr2[i]){
            return false;
        }
    }
    return true;
}

void anagramsSearch(string pat, string txt){
    int M = pat.size();
    int N = txt.size();
```

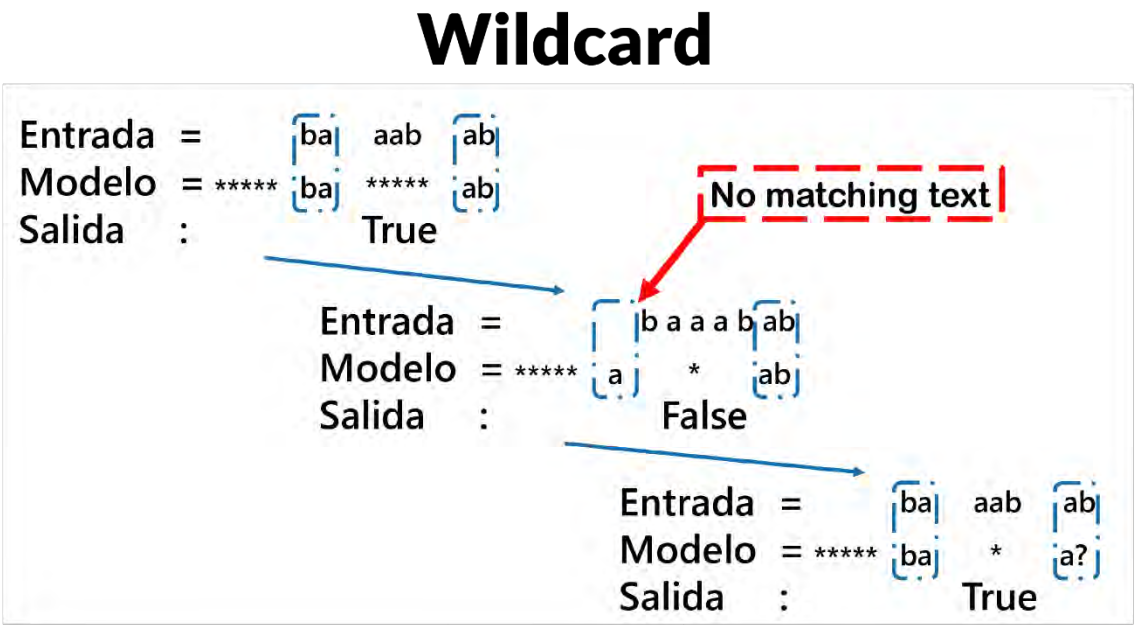
```
    char countP[MAX];
    char countTX[MAX];
    memset(countP,0 , MAX);
    memset(countTX, 0, MAX);
    for(int i = 0; i < M; i++){
        countP[pat[i]]++;
        countTX[txt[i]]++;
    }
    for(int i = M; i < N; i++){
        if(compare(countP, countTX)){
            cout << "Encontrado en el indice "<<(i - M)<<" Anagrama:
"<<txt.substr(i - M, M)<<endl;
        }
        countTX[txt[i]]++;
        countTX[txt[i - M]]--;
    }
    if(compare(countP, countTX)){
        cout<<"Encontrado en el indice "<<(N - M) <<" Anagrama: "<< txt.substr(N
- M, N) << endl;
    }
}

int main()
{
    string txt = "anitalavalatinaanilegustviajarnitatani";
    string pat = "anita";
    anagramsSearch(pat, txt);
    return 0;
}
```

PYTHON

```
from sys import stdout
wr = stdout.write

MAX = 256

def anagramsSearch(pat, txt):

    M = len(pat)
    N = len(txt)
    countP = [0] * MAX
    countTW = [0] * MAX

    for i in range(M):
        countP[ord(pat[i])] += 1
        countTW[ord(txt[i])] += 1
    for i in range(M, N):
        if countP == countTW:
            wr(f'"{txt[i-M:i]}" encontrado en el indice ({i-M} - {i-1})\n')
        countTW[ord(txt[i])] += 1
        countTW[ord(txt[i-M])] -= 1
    if countP == countTW:
        wr(f'"{txt[N-M:N]}" encontrado en el indice ({N-M} - {N-1})\n')
```

```
txt = 'BACDGABCDAABDCDBA'
pat = 'ABDC'
anagramsSearch(pat, txt)
```

11.5) Wildcard



Guía del programador competitivo

Ilustración 11-5 Ejemplo de Wildcards

Dado un texto y un patrón de wildcards, se debe implementar un algoritmo que encuentre si el patrón wildcard concuerda con el texto, el cotejamiento debe cubrir todo el texto.

El patrón wildcard puede incluir los caracteres '?' y '*':

'?' – Coteja cualquier carácter solitario.

'*' – Coteja cualquier secuencia de caracteres (Incluyendo una secuencia vacía).

Cada ocurrencia de '?' en el patrón wildcard puede ser reemplazado con cualquier otro carácter y cada ocurrencia de '*' con una secuencia de caracteres tales que el patrón de wildcard se convierta en idéntico al string de entrada luego del reemplazo.

Consideremos cualquier carácter en el patrón.

Caso 1: El carácter es '*'

Aquí dos cosas pueden pasar:

- Podemos ignorar el carácter '*' y movernos al siguiente carácter del patrón.
- El carácter '*' coincide con uno o más caracteres en el texto, aquí podemos movernos al carácter siguiente en el string.

Caso 2: El carácter es '?':

Podemos ignorar el actual carácter en el texto y movernos al siguiente carácter en el patrón y texto.

Caso 3: El carácter no es un carácter wildcard

Si el carácter actual en el texto coincide con el actual carácter en el patrón, podemos movernos al siguiente carácter en el patrón y el texto, si no concuerdan, el patrón wildcard y el texto.

Complejidad de tiempo

Mejor caso : O(n*m) **Peor caso :** O(n*m) **Promedio:** O(n*m)

JAVA

```
//Programa java que implementa el cotejamiento
// de patrones con Wildcard

import java.util.Arrays;

public class WildcardPattern {
    //Función que coteja str con el patron wildcard

    static boolean strmatch(String str, String pattern,
            int n, int m) {
        // Patron vacio colo puede coincidir
        // con string vacio
        if (m == 0) {
            return (n == 0);
        }
        // Tabla de busqueda para almacenar resultados
        // de subproblemás
        boolean[][] lookup = new boolean[n + 1][m + 1];
        //Inicializa la tabla en falso
        for (int i = 0; i < n + 1; i++) {
            Arrays.fill(lookup[i], false);
```

```
        }
        lookup[0][0] = true;
        //Solo '*' puede coincidri con string vacio
        for (int j = 1; j <= m; j++) {
            if (pattern.charAt(j - 1) == '*') {
                lookup[0][j] = lookup[0][j - 1];
            }
        }
        // Llena la tabla
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= m; j++) {
                /*Dos casos que podemos ver de '*'
                a) Ignoramos '*' y pasamos al siguiente
                caracter en el patrón
                b) '*' coincide con un iesimo
                caracter en una entrada*/

                if (pattern.charAt(j - 1) == '*') {
                    lookup[i][j] = lookup[i][j - 1]
                            || lookup[i - 1][j];
                } /*Actuales caracteres son considerados
                como coincidentes en dos casos
                a) actual caracter de patron es '?'
                b) caracteres actuales coinciden*/
                else if (pattern.charAt(j - 1) == '?'
                        || str.charAt(i - 1) == pattern.charAt(j - 1)) {
                    lookup[i][j] = lookup[i - 1][j - 1];
                } // Si el caracter no coincide
                else {
                    lookup[i][j] = false;
                }
            }
        }
        return lookup[n][m];
    }

    public static void main(String args[]) {
        String str = "baaabab";
        String pattern = "*****ba*****ab";
        //Casos de prueba
        // String pattern = "ba*****ab";
        // String pattern = "ba*ab";
        // String pattern = "a*ab";
        // String pattern = "a*****ab";
        // String pattern = "*a*****ab";
        // String pattern = "ba*ab****";
        // String pattern = "****";
        // String pattern = "*";
        // String pattern = "aa?ab";
        // String pattern = "b*b";
        // String pattern = "a*a";
        // String pattern = "baaabab";
        // String pattern = "?baaabab";
        // String pattern = "*baaaba*";
```

```
        if (strmatch(str, pattern, str.length(),
                pattern.length())) {
            System.out.println("Si");
        } else {
            System.out.println("No");
        }
    }
}
```

C++

```
using namespace std;
#include<bits/stdc++.h>
#include<cstdlib>
#define FAST ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(0);
#define Afill(x,y) memset(x,y,sizeof x)
bool strMatch(string str,string pat,int n,int m){
    if(m==0){
        return n==0;
    }
    bool lookup[n+1][m+1];
    for(int i=0;i<n+1;i++){
        Afill(lookup[i],false);
    }
    lookup[0][0]=true;
    for(int j=1;j<=m;j++){
        if(pat[j-1]=='*'){
            lookup[0][j]=lookup[0][j-1];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(pat[j-1]=='*'){
                lookup[i][j]=lookup[i][j-1]||lookup[i-1][j];
            }else if(pat[j-1]=='?' || str[i-1]==pat[j-1]){
                lookup[i][j]=lookup[i-1][j-1];
            }else{
                lookup[i][j]=false;
            }
        }
    }
    return lookup[n][m];
}
int main() {
    FAST
    string str="baaaaababa";
    string pat="ba*******?";
    if(strMatch(str,pat,str.size(),pat.size())){
        cout<<"Si"<<endl;
    }else{
        cout<<"No"<<endl;
    }
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write


def wildCardStrMatch(strn, pat, n, m):

    if m == 0:
        return n == 0

    lookup = [[False for x in range(m+1)] for x in range(n+1)]
    lookup[0][0] = True

    for i in range(1, m+1):
        if pat[i-1] == '*':
            lookup[0][i] = lookup[0][i-1]

    for i in range(1, n+1):
        for j in range(1, m+1):
            if pat[j-1] == '*':
                lookup[i][j] = lookup[i][j-1] or lookup[i-1][j]
            elif pat[j-1] == '?' or strn[i-1] == pat[j-1]:
                lookup[i][j] = lookup[i-1][j-1]
            else:
                lookup[i][j] = False

    return lookup[n][m]


txt = 'anitalavalatina'
pat = '?nit?lava*'

if wildCardStrMatch(txt, pat, len(txt), len(pat)):
    wr('Iguales\n')
else:
    wr('Diferentes\n')
```

11.6) Algoritmo de Manacher

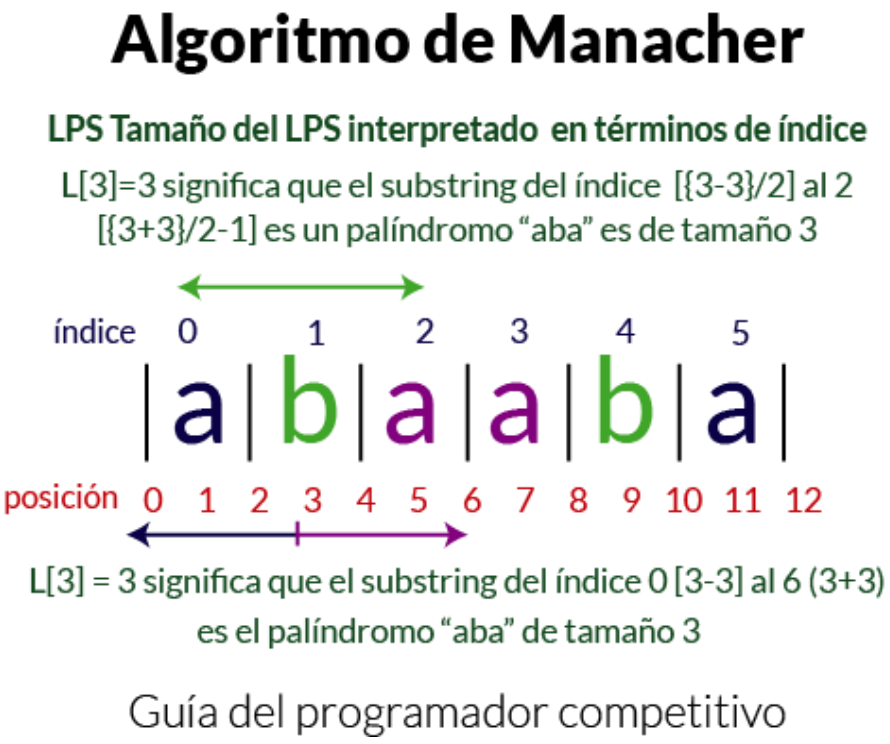


Ilustración 11-6 Ejemplo de aplicación del algoritmo de Manacher

Dado un string, encuentre el substring más largo que sea palíndromo.

Si el string dado es "abaaba" la salida deberá ser "abaaba"

Vamos a considerar el string "abababa".

Aquí el centro del string es el 4to carácter con índice 3, si cotejamos más caracteres en la izquierda y derecha del centro todos los caracteres coinciden y el string es palíndromo.

Considere el string "abaaba" de tamaño par. Este string es palíndromo alrededor de la posición entre el 3er y 4to carácter.

Para encontrar el substring más largo palíndromo de un string de tamaño N, una vía es tomar cada posible 2*N+1 centros (las N posiciones de caracteres, N-1 entre dos caracteres y dos posiciones en los fines de derecha e izquierda), haga que el carácter coincida en ambas direcciones en cada 2*N+1 centros y siga rastreando por LPS (Longest palindromic string).

Si el string dado es "abababa" la salida debe ser "abababa"

Si el string dado es "abcbabcbabcba" la salida debe ser "abcbabcba".

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa java implementando el algoritmo de Manacher
// LPS: longest palindrome string

public class ManacherAlgorithm {

    static char text[];

    public static void main(String[] args) {
        text = "babcbabcbaccba".toCharArray();
        findLongestPalindromicString();
        text = "abaaba".toCharArray();
        findLongestPalindromicString();
        text = "abababa".toCharArray();
        findLongestPalindromicString();
        text = "abcbabcbabcba".toCharArray();
        findLongestPalindromicString();
        text = "caba".toCharArray();
        findLongestPalindromicString();
        text = "abacdfgdcaba".toCharArray();
        findLongestPalindromicString();
        text = "abacdfgdcabba".toCharArray();
        findLongestPalindromicString();
        text = "abacdedcaba".toCharArray();
        findLongestPalindromicString();
    }

    static int min(int a, int b) {
        int res = a;
        if (b < a) {
            res = b;
        }
        return res;
    }

    static void findLongestPalindromicString() {
        int N = text.length;
        if (N == 0) {
            return;
        }
        N = 2 * N + 1; //Conteo de posición
        int L[] = new int[N]; //LPS tamaño de array
        L[0] = 0;
        L[1] = 1;
        int C = 1; //Posición central
        int R = 2; //posición Centro derecho
        int i = 0; //Posición actual derecho
        int iMirror; //Posición actual izquierda
        int maxLPSLength = 0;
        int maxLPSCenterPosition = 0;
        int start = -1;
```

```
        int end = -1;
        int diff = -1;
        //Descomentar para imprimir tamaño del arreglo LPS
        //printf("%d %d ", L[0], L[1]);
        for (i = 2; i < N; i++) {
            iMirror = 2 * C - i;
            L[i] = 0;
            diff = R - i;
            if (diff > 0) {
                L[i] = min(L[iMirror], diff);
            }
            /*Intente expandir palíndromo centrado en currentRightPosition i
            Aquí para posiciones impares, comparamos caracteres y
            si coinciden, aumente la longitud de LPS en UNO
            Si la posición es igual, solo incrementamos LPS en UNO sin*/
            try {
                while (((i + L[i]) < N && (i - L[i]) > 0)
                        && (((i + L[i] + 1) % 2 == 0)
                        || (text[(i + L[i] + 1) / 2] == text[(i - L[i] - 1) /
2]))) {
                    L[i]++;
                }
            } catch (Exception e) {
            }
            //Comparación de cualquier caracter
            if (L[i] > maxLPSLength) {
                maxLPSLength = L[i];
                maxLPSCenterPosition = i;
            }
            if (i + L[i] > R) {
                C = i;
                R = i + L[i];
            }
            //Descomentar para imprimir tamaño del arreglo LPS
            //printf("%d ", L[i]);
        }
        start = (maxLPSCenterPosition - maxLPSLength) / 2;
        end = start + maxLPSLength - 1;
        System.out.println("LPS del string es " + String.copyValueOf(text) + ":
");
        for (i = start; i <= end; i++) {
            System.out.printf("%c", text[i]);
        }
        System.out.println("");
    }
}
```

C++

```
using namespace std;
#include<bits/stdc++.h>
#include<cstdlib>
#define FAST ios_base::sync_with_stdio(false);cin.tie(NULL);
#define string_valueof(x) puts(x);
```

```
void findLPS(char text[], int N) {
    if (N == 0) {
        return;
    }
    N = 2 * N + 1;
    int L[N];
    L[0] = 0;
    L[1] = 1;
    int C = 1;
    int R = 2;
    int i_mirror;
    int max_LPS_len = 0;
    int max_LPS_centerPos = 0;
    int start = -1, endLPS = -1, diff = -1;
    for (int i = 2; i < N; i++) {
        i_mirror = 2 * C - i;
        L[i] = 0;
        diff = R - i;
        if (diff > 0) {
            L[i] = std::min(L[i_mirror], diff);
        }
        try {
            while (((i + L[i]) < N and (i - L[i]) > 0) and (((i + L[i] + 1) % 2
== 0) || text[(i + L[i] + 1) / 2] == text[(i - L[i] - 1) / 2])) {
                L[i]++;
            }
        } catch (exception& e) {
        }
        if (L[i] > max_LPS_len) {
            max_LPS_len = L[i];
            max_LPS_centerPos = i;
        }
        if (i + L[i] > R) {
            C = i;
            R = i + L[i];
        }
    }
    start = (max_LPS_centerPos - max_LPS_len) / 2;
    endLPS = start + max_LPS_len - 1;
    cout << "LPS del string es ";
    string_valueof(text);
    cout << " : ";
    for (int j = start; j <= endLPS; j++) {
        cout << text[j];
    }
    cout << endl;
    /*
    cout<<"inicio : "<<start<<"    -    final : "<<endLPS<<endl;
    for(int j=0;j<N;j++){
        cout<<L[j]<<" ";
    }
     */
}
```

```
int main() {
    FAST;
    char text[1000];
    gets(text);
    int N = strlen(text);
    findLPS(text, N);
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write


def findLPS():

    N = len(text)
    if N == 0:
        return

    N = 2 * N + 1
    L = [0] * N
    L[0] = 0
    L[1] = 1
    C, R = 1, 2
    iMirror = maxLPSlen = maxLPScenterPos = 0
    start = end = diff = -1

    for i in range(2, N):
        iMirror = 2 * C - i
        L[i] = 0
        diff = R - i
        if diff > 0:
            L[i] = min(L[iMirror], diff)
        try:
            while ((i + L[i]) < N and (i - L[i]) > 0) and (((i + L[i] + 1) % 2
== 0) or (text[(i + L[i] + 1) // 2] == text[(i - L[i] - 1) // 2])):
                L[i] += 1
        except:
            pass

        if L[i] > maxLPSlen:
            maxLPSlen = L[i]
            maxLPScenterPos = i

        if i + L[i] > R:
            C = i
            R = i + L[i]

    start = (maxLPScenterPos - maxLPSlen) // 2
    end = start + maxLPSlen - 1

    wr(f'LPS del string {"".join(text)}:\n')
```

```
    for i in range(start, end+1):
        wr(f'{text[i]}')
    wr('\n')


text = [x for x in 'OABAABAOBO']
findLPS()
```

11.7) Algoritmo de Aho-Corasick

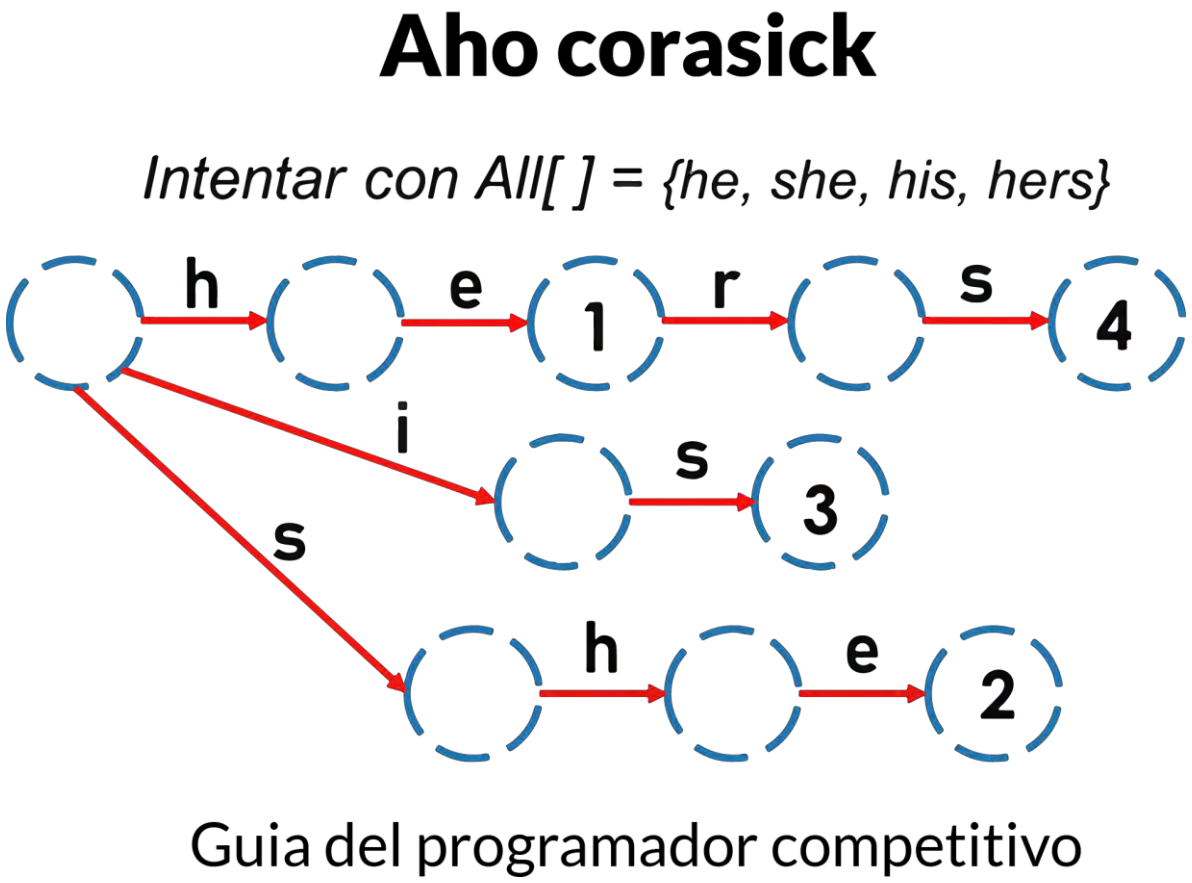
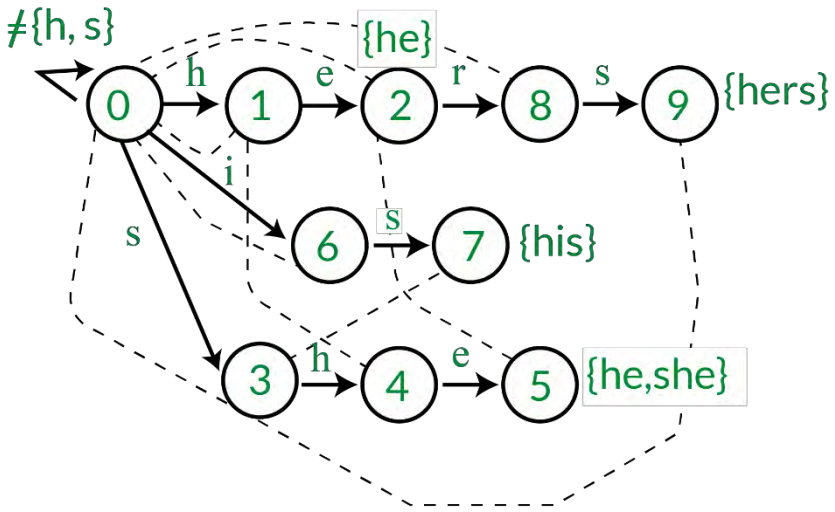


Ilustración 11-7 Ejemplo de búsqueda de patrones Aho-Corasick

Dada una entrada de texto y un array de k palabras, arr[], encontrar todas las ocurrencias de todas las palabras en el texto de entrada. Sea n la longitud del texto y m sea el número total de caracteres en todas las palabras, por ejemplo: m = length (arr [0]) + length (arr [1]) +… + length (arr [k-1]).

Si usamos un algoritmo de tiempo linear como KMP, entonces necesitaremos una búsqueda uno por uno de todas las palabras en text[]. Esto nos da un total de complejidad de tiempo de O(n + length(word[0]) + O(n + length(word[1]) + O(n + length(word[2]) + … O(n + length(word[k-1]). Esta complejidad de tiempo puede ser escrita como O(n*k + m).
Aho-corasick encuentra todas las palabras en tiempo O(n+m+z) donde z es el número total de ocurrencias de las palabras en el texto, este algoritmo forma las bases del comando original de Unix fgrep.

Aho-Corasick



Flechas punteadas son transacciones fallidas.

Flechas completas son transacciones validas.

Guia del programador competitivo

Ilustración 11-8 Automatón construido a partir de las palablas a buscar

Este algoritmo tiene dos pasos principales:

- Preprocesado: Construye un automaton de todas las palabras en arr[].
- Cotejamiento: Atraviesa el texto dado sobre el automaton formado para encontrar las palabras a cotejar. Por un estado s, los índices de todas las palabras terminados

en s son almacenadas. Estos índices son almacenados en un map de BitWise (Haciendo OR a lo valores. Esto también computa usando búsqueda primera en anchura con fallo.

Construimos un autómata para este conjunto de cadenas. Ahora procesaremos el texto letra por letra, haciendo la transición durante los diferentes estados. Inicialmente estamos en la raíz del trie. Si estamos en cualquier momento en el estado v, y la siguiente letra es c, entonces hacemos la transición al siguiente estado con go (v, c), aumentando así la longitud de la subcadena de coincidencia actual en 1, o disminuyéndola siguiendo Un enlace de sufijo.

¿Cómo podemos encontrar un estado v, si hay coincidencias con cadenas para el conjunto? Primero, está claro que si nos paramos en un vértice de la hoja, entonces la cadena correspondiente al vértice termina en esta posición en el texto. Sin embargo, este no es el único caso posible de lograr una coincidencia: si podemos alcanzar uno o más vértices de hojas moviéndonos a lo largo de los enlaces de sufijo, entonces también habrá una coincidencia correspondiente a cada vértice de hoja encontrado. Un ejemplo simple que demuestra esta situación puede ser crear usando el conjunto de cadenas {dabce, abc, bc} y el texto dabc.

Por lo tanto, si almacenamos en cada vértice de la hoja el índice de la cadena correspondiente (o la lista de índices si aparecen cadenas duplicadas en el conjunto), entonces podemos encontrar en O (n) tiempo los índices de todas las cadenas que coinciden con la actual estado, simplemente siguiendo los enlaces de sufijo desde el vértice actual hasta la raíz. Sin embargo, esta no es la solución más eficiente, ya que nos da una complejidad O (n len) en total. Sin embargo, esto se puede optimizar calculando y almacenando el vértice de la hoja más cercano al que se puede acceder mediante enlaces de sufijo (esto a veces se denomina enlace de salida). Este valor lo podemos calcular perezosamente en tiempo lineal. Por lo tanto, para cada vértice podemos avanzar en tiempo O (1) al siguiente vértice marcado en la ruta de enlace del sufijo, es decir, a la

siguiente coincidencia. Por lo tanto, para cada partido pasamos O (1) tiempo, y por lo tanto alcanzamos la complejidad O (len + ans).

Si solo desea contar las ocurrencias y no encontrar los índices, puede calcular el número de vértices marcados en la ruta de enlace del sufijo para cada vértice v. Esto se puede calcular en O (n) en total. Por lo tanto, podemos resumir todas las coincidencias en O (len).

Utilidades de este algoritmo:

1) **Encontrar la cadena lexicográfica más pequeña de una longitud dada que no coincide con ninguna cadena dada**

Se da un conjunto de cuerdas y una longitud L. Tenemos que encontrar una cadena de longitud L, que no contiene ninguna de las cadenas, y derivar la cadena lexicográfica más pequeña de tales cadenas.

Podemos construir el autómata para el conjunto de cadenas. Recordemos que los vértices desde los cuales podemos alcanzar el vértice de una hoja son los estados, en los cuales tenemos una coincidencia con una cadena del conjunto. Como en esta tarea debemos evitar las coincidencias, no se nos permite ingresar a dichos estados. Por otro lado, podemos ingresar todos los demás vértices. Por lo tanto, eliminamos todos los vértices "malos" de la máquina, y en el gráfico restante del autómata encontramos la ruta lexicográfica más pequeña de longitud L. Esta tarea se puede resolver en O (L), por ejemplo, mediante la búsqueda en profundidad.

2) **Encontrar la cadena más corta que contiene todas las cadenas dadas**

Aquí usamos las mismas ideas. Para cada vértice almacenamos una máscara que denota las cadenas que coinciden en este estado. Entonces el problema puede reformularse de la siguiente manera: inicialmente estando en el estado (v = raíz, máscara = 0), queremos llegar al estado (v, máscara = 2n − 1), donde n es el número de cadenas en el conjunto . Cuando hacemos la transición de un estado a otro usando una letra, actualizamos la máscara en

consecuencia. Al ejecutar una búsqueda de respiración primero podemos encontrar una ruta al estado (v, máscara = 2n − 1) con la longitud más pequeña.

3) **Encontrar la cadena lexicográfica más pequeña de longitud L que contiene k cadenas**

Como en el problema anterior, calculamos para cada vértice el número de coincidencias que le corresponden (es decir, el número de vértices marcados a los que se puede acceder mediante enlaces de sufijo). Reformulamos el problema: el estado actual está determinado por un triple de números (v, len, cnt), y queremos llegar desde el estado (raíz, 0, 0) al estado (v, L, k), donde v puede ser cualquier vértice Por lo tanto, podemos encontrar dicha ruta utilizando la búsqueda en profundidad primero (y si la búsqueda mira los bordes en su orden natural, entonces la ruta encontrada será automáticamente la más pequeña lexicográfica).

Complejidad de tiempo

Mejor caso : O(n + m + ocurrencias) **Peor caso :** O(n + m + ocurrencias)

Promedio: O(n + m + ocurrencias)

JAVA

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;

//Programa JAVA implementando Aho Corasick
//para matching de strings
public class Main {

    //Maximo numero de estados en la maquina de cotejamiento
    //debe ser igual a la suma del tamaño de todas las palablas clave
    static final int MAXS = 500;
    //numero maximo de caracteres en el alfabeto de entrada
    static final int MAXC = 26;
    //La funcion Output esta implementada usando out[]
    // bit i de esta mascara es uno si la palabra en el
    // indice i aparece cuando la maquina entra en este estado
    static int out[] = new int[MAXS];
```

```java
//función de fallo
static int f[] = new int[MAXS];
//implementación de las tries
static int g[][] = new int[MAXS][MAXC];

// Construye la maquina de cotejado
/* "out[state] & (1 << i)" es > 0 si encontramos la
palabra de word[i] en el texto
arr-> vector de palabras, el incide de cada palabla clave
es importante */
// retorna el numero de estados que la maquina tiene construidos
// los estados estan numerados desde 0 hasta el valor de retorno - 1
static int buildMatchingMachine(String arr[], int k) {
    //Inicializar todos los valores de out en 0.
    Arrays.fill(out, 0);
    // Inicializar todos los valores de g en -1.
    for (int i = 0; i < MAXS; i++) {
        Arrays.fill(g[i], -1);
    }
    // Inicialmente tenemos 0 estados
    int states = 1;
    //Esto es lo mismo que construir un Trie para arr[]
    for (int i = 0; i < k; ++i) {
        String word = arr[i];
        int currentState = 0;
        // Insertar todos los caracteres de la palabra actual
        // en arr[]
        for (int j = 0; j < word.length(); ++j) {
            int ch = word.charAt(j) - 'a';
            // crea un nuevo nodo (estado) si un nodo
            // para ch no existe
            if (g[currentState][ch] == -1) {
                g[currentState][ch] = states++;
            }
            currentState = g[currentState][ch];
        }
        // agrega la palabra actual a la función de salida
        out[currentState] |= (1 << i);
    }
    // para todos los caracteres que no tengan un camino desde
    // la raiz (El estado 0) en el trie, agregar un camino
    // hacia el estado 0 mismo
    for (int ch = 0; ch < MAXC; ++ch) {
        if (g[0][ch] == -1) {
            g[0][ch] = 0;
        }
    }
    //Inicializamos los valores en la funcion de fallo
    Arrays.fill(f, -1);
    // Se usa BFS y una cola para calcular los fallos
    Queue<Integer> q = new LinkedList<>();
    // Se itera por lo que se encuentre
    for (int ch = 0; ch < MAXC; ++ch) {
        // Todos los nodos de profundidad 1 tienen una funcion de fallo
```

```
            // como 0.
            if (g[0][ch] != 0) {
                f[g[0][ch]] = 0;
                q.offer((g[0][ch]));
            }
        }
        while (!q.isEmpty()) {
            // Tomamos el nodo o estado del frente de la cola
            int state = q.poll();
            // Se busca la funcion de fallo de todos los caracteres del estado
            // removido para cuales funcion g no esta definido
            for (int ch = 0; ch < MAXC; ++ch) {
                if (g[state][ch] != -1) {
                    //Encontrar el valor de la funcion de fallo
                    int failure = f[state];
                    // encuentra el nodo mas profundo con el sufijo
                    // apropiado del string desde el nodo raiz al estado actual
                    while (g[failure][ch] == -1) {
                        failure = f[failure];
                    }
                    failure = g[failure][ch];
                    f[g[state][ch]] = failure;
                    //Junta los valores de salida
                    out[g[state][ch]] |= out[failure];
                    //Inserta el nodo del siguiente nivel del trie
                    q.offer(g[state][ch]);
                }
            }
        }

        return states;
    }
    // currentState - El estado actual de la maquina
    //                entre 0 y el total de estados -1
    // nextInput - El siguiente caracter que entra en la maquina.

    static int findNextState(int currentState, char nextInput) {
        int answer = currentState;
        int ch = nextInput - 'a';
        // Si g no esta definido, use la funcion de fallo
        while (g[answer][ch] == -1) {
            answer = f[answer];
        }
        return g[answer][ch];
    }

    //Función que busca las ocurrencias en el texto
    static void searchWords(String arr[], int k, String text) {
        // Preprocesar patrones
        // Construir los tries como una maquina de estado finito
        buildMatchingMachine(arr, k);
        // Inicializar los estados
        int currentState = 0;
        //Atravesamos el texto buscando las ocurrencias
```

```
        for (int i = 0; i < text.length(); ++i) {
            currentState = findNextState(currentState, text.charAt(i));
            // Si no se encuentra cotejado, pasa al siguiente estado
            if (out[currentState] == 0) {
                continue;
            }
            // cotejado encontrado, imprimir todas las palabras de
            // arr[] que se encontraron
            for (int j = 0; j < k; ++j) {
                int aux = (out[currentState] & (1 << j));
                if (aux > 0) {
                    System.out.println("La palabra " + arr[j] + " aparece de "
                            + (i - arr[j].length() + 1) + " a " + i);
                }
            }
        }
    }

    public static void main(String[] args) {
        String arr[] = {"he", "she", "hers", "his"};
        String text = "ahishers";
        int k = arr.length;
        searchWords(arr, k, text);
    }
}
```

C++

```
using namespace std;
#include <bits/stdc++.h>

const int MAXS = 500;
const int MAXC = 26;
int out[MAXS];
int f[MAXS];
int g[MAXS][MAXC];

int buildMatchingMachine(string arr[], int k) {
    memset(out, 0, sizeof out);
    memset(g, -1, sizeof g);
    int states = 1;
    for (int i = 0; i < k; ++i) {
        const string &word = arr[i];
        int currentState = 0;
        for (int j = 0; j < word.size(); ++j) {
            int ch = word[j] - 'a';
            if (g[currentState][ch] == -1)
                g[currentState][ch] = states++;
            currentState = g[currentState][ch];
        }
        out[currentState] |= (1 << i);
    }
    for (int ch = 0; ch < MAXC; ++ch)
        if (g[0][ch] == -1)
            g[0][ch] = 0;
```

```
    memset(f, -1, sizeof f);
    queue<int> q;
    for (int ch = 0; ch < MAXC; ++ch) {
        if (g[0][ch] != 0) {
            f[g[0][ch]] = 0;
            q.push(g[0][ch]);
        }
    }

    while (q.size()) {
        int state = q.front();
        q.pop();
        for (int ch = 0; ch <= MAXC; ++ch) {
            if (g[state][ch] != -1) {
                int failure = f[state];
                while (g[failure][ch] == -1)
                    failure = f[failure];
                failure = g[failure][ch];
                f[g[state][ch]] = failure;
                out[g[state][ch]] |= out[failure];
                q.push(g[state][ch]);
            }
        }
    }
    return states;
}

int findNextState(int currentState, char nextInput) {
    int answer = currentState;
    int ch = nextInput - 'a';
    while (g[answer][ch] == -1)
        answer = f[answer];
    return g[answer][ch];
}

void searchWords(string arr[], int k, string text) {
    buildMatchingMachine(arr, k);
    int currentState = 0;
    for (int i = 0; i < text.size(); ++i) {
        currentState = findNextState(currentState, text[i]);
        if (out[currentState] == 0)
            continue;
        for (int j = 0; j < k; ++j) {
            int aux = out[currentState] & (1 << j);
            if (aux) {
                cout << "Word " << arr[j] << " appears from "
                        << i - arr[j].size() + 1 << " to " << i << endl;
            }
        }
    }
}

int main() {
    string arr[] = {"he", "she", "hers", "his"};
```

```
    string text = "ahishers";
    int k = sizeof (arr) / sizeof (arr[0]);

    searchWords(arr, k, text);

    return 0;
}
```

PYTHON

```
from collections import deque
MAXS = 500
MAXC = 26
out = []
f = []
g = []

def buildMatchingMachine(arr, k):
    global MAXS
    global MAXC
    global out
    global g
    global f
    out = [0 for x in range(MAXS)]
    g = [[-1 for y in range(MAXC)] for x in range(MAXS)]
    states = 1
    for i in range(k):
        word = arr[i]
        currentState = 0
        for j in range(len(word)):
            ch = ord(word[j]) - ord('a')
            if g[currentState][ch] == -1:
                states = states + 1
                g[currentState][ch] = states
            currentState = g[currentState][ch]
        out[currentState] |= (1 << i)
    for ch in range(MAXC):
        if g[0][ch] == -1:
            g[0][ch] = 0
    f = [-1 for x in range(MAXS)]
    q = deque()
    for ch in range(MAXC):
        if g[0][ch] != 0:
            f[g[0][ch]] = 0
            q.append((g[0][ch]))
    while not len(q) == 0:
        state = q.pop()
        for ch in range(MAXC):
            if g[state][ch] != -1:
                failure = f[state]
                while g[failure][ch] == -1:
                    failure = f[failure]
                failure = g[failure][ch]
                f[g[state][ch]] = failure
                out[g[state][ch]] |= out[failure]
```

```
                q.append(g[state][ch])
    return states

def findNextState(currentState, nextInput):
    global MAXS
    global MAXC
    global out
    global g
    global f
    answer = currentState
    ch = int(ord(nextInput) - ord('a'))
    while g[answer][ch] == -1:
        answer = f[answer]
    return g[answer][ch]

def searchWords(arr, k, text):
    global MAXS
    global MAXC
    global out
    global g
    global f
    buildMatchingMachine(arr, k)
    currentState = 0
    for i in range(len(text)):
        currentState = findNextState(currentState, text[i])
        if out[currentState] == 0:
            continue
        for j in range(k):
            aux = (out[currentState] & (1 << j))
            if aux > 0:
                print("la palabra ", arr[j], " aparece de ", ((i - len(arr[j]))
+ 1), " a ", i)


arr = ["he", "she", "hers", "his"]
text = "ahishers"
k = len(arr)
searchWords(arr, k, text)
```

11.8) Finite Automata

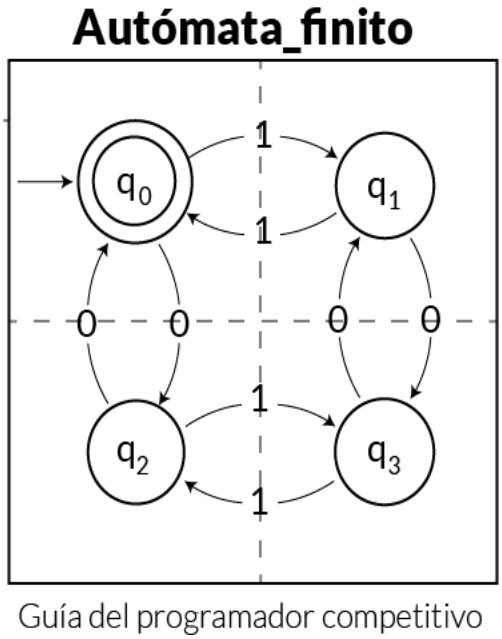


Ilustración 11-9 Ejemplo abstracto de un autómata finito

Dado un texto txt[0...n-1] y un patrón pat[0...m-1], busque todas las ocurrencias de pat en txt e imprímalas, asuma que n es mayor que m.

En el algoritmo basado en Finite autómata, nosotros preprocesamos el patrón y construimos un array de dos dimensiones que represente un autómata finito. Construcción del FA es la parte complicada de este algoritmo. Una vez el FA este construido, la búsqueda es simple, En la búsqueda nosotros simplemente necesitamos iniciar desde el primer estado del autómata y el primer carácter del texto. En cada paso, consideramos el siguiente carácter del texto, miramos por el siguiente estado en el FA construido y nos movemos a un nuevo estado. Si alcanzamos el estado final, entonces el patrón fue encontrado en el texto.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
// Programa java para el algoritmo
// Finite Automata de busqueda de patrones

public class FiniteAutomata {
```

```
static int NO_OF_CHARS = 256;

static int getNextState(char[] pat, int M,
        int state, int x) {
    // Si el caracter c es el mismo como el
    // siguiente en patrón, simplemente incrementa
    // state
    if (state < M && x == pat[state]) {
        return state + 1;
    }
    // ns almacena el resultado el cual es
    // el siguiente state
    int ns, i;
    /* ns finalmente contiene el prefijo más largo
    el cual tambien es sufijo en "pat[0..state-1]c"
    Empieza desde el valor más largo posible
    y se detiene cuando se encuentra un prefijo el cual
    es tambien sufijo*/
    for (ns = state; ns > 0; ns--) {
        if (pat[ns - 1] == x) {
            for (i = 0; i < ns - 1; i++) {
                if (pat[i] != pat[state - ns + 1 + i]) {
                    break;
                }
            }
            if (i == ns - 1) {
                return ns;
            }
        }
    }
    return 0;
}

/*Esta función construye la tabla TF la cual
representa Finite Automata del patrón dado*/
static void computeTF(char[] pat, int M, int TF[][]) {
    int state, x;
    for (state = 0; state <= M; ++state) {
        for (x = 0; x < NO_OF_CHARS; ++x) {
            TF[state][x] = getNextState(pat, M, state, x);
        }
    }
}

/* Imprime todas las ocurrencuas de pat en txt*/
static void search(char[] pat, char[] txt) {
    int M = pat.length;
    int N = txt.length;
    int[][] TF = new int[M + 1][NO_OF_CHARS];
    computeTF(pat, M, TF);
    // Procesa txt sobre FA.
    int i, state = 0;
    for (i = 0; i < N; i++) {
```

```
            state = TF[state][txt[i]];
            if (state == M) {
                System.out.println("Patrón encontrado "
                        + "en indice " + (i - M + 1));
            }
        }
    }

    public static void main(String[] args) {
        char[] txt = "AABAACAADAABAAABAA".toCharArray();
        char[] pat = "AABA".toCharArray();
        search(pat, txt);
    }
}
```

C++

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;
#define NO_OF_CHARS 256

int getNextState(char pat[], int M, int state, int x) {
    if (state < M && x == pat[state]) {
        return state + 1;
    }
    int ns, i;
    for (ns = state; ns > 0; ns--) {
        if (pat[ns - 1] == x) {
            for (i = 0; i < ns - 1; i++) {
                if (pat[i] != pat[state - ns + 1 + i]) {
                    break;
                }
            }
            if (i == ns - 1) {
                return ns;
            }
        }
    }
    return 0;
}

void computeTF(char pat[], int M, int TF[NO_OF_CHARS][NO_OF_CHARS]) {
    int state, x;
    for (state = 0; state <= M; state++) {
        for (x = 0; x < NO_OF_CHARS; x++) {
            TF[state][x] = getNextState(pat, M, state, x);
        }
    }
}

void finiteAutomata(char pat[], char txt[]) {
    int M = strlen(pat);
    int N = strlen(txt);
    int TF[M + 1][NO_OF_CHARS];
```

```
    computeTF(pat, M, TF);
    int i, state = 0;
    for (i = 0; i < N; i++) {
        state = TF[state][txt[i]];
        if (state == M) {
            cout << "Patron encontrado en indice " << i - M + 1 << " - " << i <<
endl;
        }
    }
}

int main(int argc, char const *argv[]) {
    char pat[] = {"looloolo"};
    char txt[] = {"olo"};
    finiteAutomata(txt, pat);
    return 0;
}
```

PYTHON

```
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write

NO_OF_CHARS = 256


def getNextState(pat, M, state, x):

    if state < M and x == ord(pat[state]):
        return state + 1

    i = 0
    for ns in range(state, 0, -1):
        if ord(pat[ns-1]) == x:
            for i in range(ns-1):
                if pat[i] != pat[state-ns + 1 + i]:
                    break
                i += 1
            if i == ns - 1:
                return ns

    return 0


def computeTF(pat, M, TF):
    for state in range(M + 1):
        for x in range(NO_OF_CHARS):
            TF[state][x] = getNextState(pat, M, state, x)


def finiteAutomataSearch(pat, txt):
```

```
    M = len(pat)
    N = len(txt)
    TF = [[0 for x in range(NO_OF_CHARS)] for x in range(M + 1)]
    computeTF(pat, M, TF)
    state = 0
    for i in range(N):
        state = TF[state][ord(txt[i])]
        if state == M:
            wr(f'"{"".join(pat)}" encontrado en indices ({i-M+1} - {i})\n')

txt = 'AABAACAADAABAABA'
pat = 'AABA'
finiteAutomataSearch(pat, txt)
```

11.9) Problemas de repaso

Ejercicios en Online Judge

454-Anagrams	148-Anagram checker
271-Simply Syntax	642- Word Amalgamation
401-Palindromes	10343-Base64 Decoding
455-Periodic Strings	10058- Jimmi’s Riddles

Ejercicios en CodeChef

CHEFSTLT	CHRL2
STRPALIN	PLAYST

Capitulo 12. Matemática, algebra y teoría de números

La matemática dentro de las ciencias de la computación abarca una enorme cantidad de algoritmos y teoremas que permiten el manejo de números y sus posibles utilidades dentro de la realización de una solución a un problema de programación competitiva, se reúne dentro de esta los métodos numéricos, los métodos estocásticos, el álgebra, la criptografía, la teoría de números, la estadística, la teoría de juegos, entre otros. Se busca con la utilización de todo esto la optimización de recursos dentro de las soluciones, y la aproximación a una solución que permita la completar el problema con mayor sencillez.

12.1) GCD/LCM

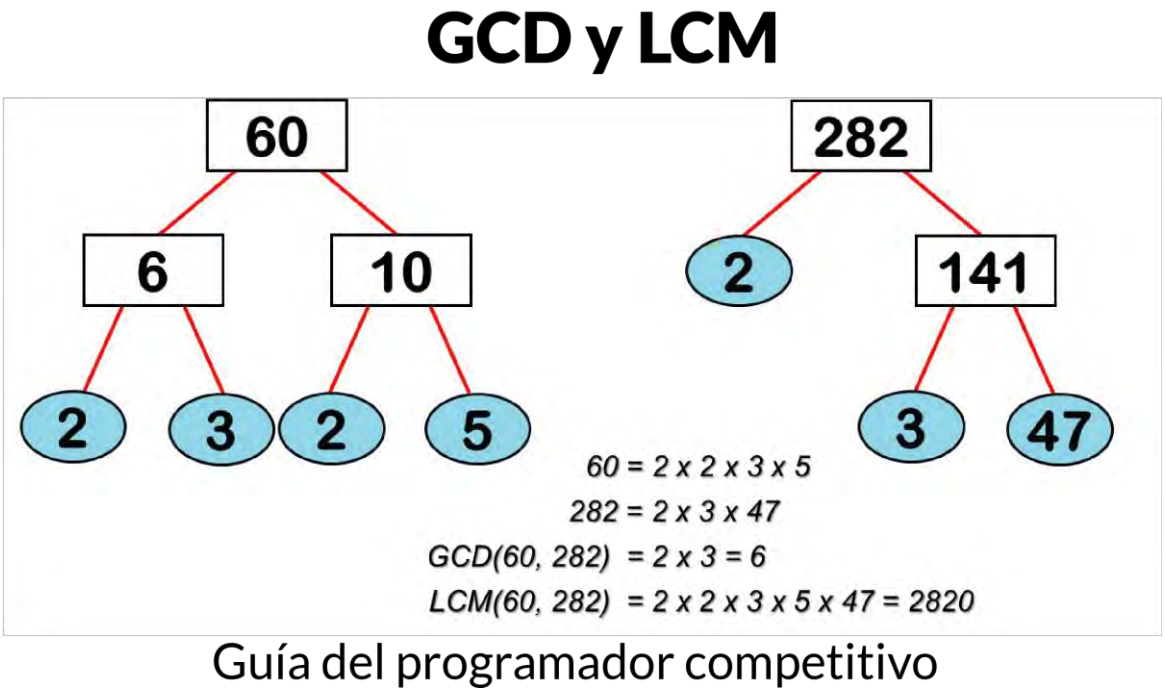


Ilustración 12-1 Ejemplo de GCD y LCM

El máximo común divisor (MCD o GCD) de dos o más números naturales o enteros (no números con decimales) es el número más grande que les divide.

El mínimo común múltiplo (MCM o LCM) de dos números a y b es el número más pequeño que es múltiplo de a y múltiplo de b.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```java
//Programa java que realiza
//Maximo comun divisor y minimo comun multiplo

public class GCDLCM {

    public static void main(String[] args) {
        int a = 8, b = 12;
        System.out.println("GCD de a y b es :" + gcd(a, b));
        System.out.println("LCM de a y b es :" + lcm(a, b));
    }
    //Maximo Comun Divisor
    public static int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }
    //Minimo Comun Multiplo
    public static int lcm(int a, int b) {
        return a * (b / gcd(a, b));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

int GCD(int a, int b) {
    return b == 0 ? a : GCD(b, a % b);
}

int LCM(int a, int b) {
    return a * (b / GCD(a, b));
}

int main() {
    int a = 8;
    int b = 12;
    cout << "GCD de a y b es: " << GCD(a, b) << endl;
    cout << "LCM de a y b es: " << LCM(a, b) << endl;
}
```

PYTHON

```python
from sys import stdout

def GCD(a, b):
    return a if b == 0 else GCD(b, a % b)

def LCM(a, b):
    return a * (b // GCD(a, b))

def main():
    a = int(8)
    b = int(12)
    stdout.write(f"GCD de a y b es : {GCD(a,b)}\n")
    stdout.write(f"LCM de a y b es : {LCM(a,b)}\n")

main()
```

12.2) Multiple GCD

Ilustración 12-2 GCD multiple da como resultado 3 como el divisor común mas grande

Dado un array de números, encontrar GCD del array de elementos.

El GCD de tres o más números es igual que el producto de los factores primos comunes de todos los números, pero puede también ser calculado tomando repetidamente el GCD de los pares de números.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```
// Programa java que encuentra
// el maximo comun divisor de 2 o más
//números

public class GCDMultiple {
    // Funcion que retorna gcd de a y b
    static int gcd(int a, int b) {
        if (a == 0) {
            return b;
```

```
        }
        return gcd(b % a, a);
    }
    // Función que encuentra gcd de un arreglo de números
    static int findGCD(int arr[], int n) {
        int result = arr[0];
        for (int i = 1; i < n; i++) {
            result = gcd(arr[i], result);
        }

        return result;
    }
    public static void main(String[] args) {
        int arr[] = {2, 4, 6, 8, 16};
        int n = arr.length;
        System.out.println(findGCD(arr, n));
    }
}
```

12.3) LCM en un vector

Dado un array de n números, encontrar el LCM de ellos.

La idea es extender nuestra relación de más de dos números, se tiene un array arr[] que contiene n elementos de los cuales se necesita calcular su LCM.

Los pasos principales del algoritmo son:

1) Inicializa ans = arr[0]
2) Iterar sobre todos los elementos del array, por ejemplo desde i=1 a i= n-1, en la iesima iteración ans= LCM(arr[0],arr[1],……., arr[i-1]). Esto puede ser fácilmente hecho como LCM(arr[0], arr[1], …., arr[i]) = LCM(ans, arr[i]). Entonces en la iesima iteración tenemos que hacer ans = LCM(ans, arr[i]) = ans x arr[i] / gcd(ans, arr[i])

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```
// Programa java que calcula el minimo común multiplo
// de n elementos
```

```java
public class Lcmofarrayelements {

    public static long lcm_of_array_elements(int[] element_array) {
        long lcm_of_array_elements = 1;
        int divisor = 2;
        while (true) {
            int counter = 0;
            boolean divisible = false;
            for (int i = 0; i < element_array.length; i++) {

                // lcm_of_array_elements (n1, n2, ... 0) = 0.
                // Para cada número negativo lo convertimos
                // En positivo y calculamos lcm_of_array_elements.
                if (element_array[i] == 0) {
                    return 0;
                } else if (element_array[i] < 0) {
                    element_array[i] = element_array[i] * (-1);
                }
                if (element_array[i] == 1) {
                    counter++;
                }
                /* Divide element_array por diviser si completa
                división */
                if (element_array[i] % divisor == 0) {
                    divisible = true;
                    element_array[i] = element_array[i] / divisor;
                }
            }
            /*Si el divisor es capaz de dividir completamente cualquier número.
            de la matriz multiplicar con lcm_of_array_elements
            y almacenar en lcm_of_array_elements y continuar
            al mismo divisor para encontrar el siguiente factor.
            si no incrementar divisor*/

            if (divisible) {
                lcm_of_array_elements = lcm_of_array_elements * divisor;
            } else {
                divisor++;
            }
            //Verifica si todo element_Array es 1 indicando
            // encontramos todos los factores y terminamos el ciclo
            if (counter == element_array.length) {
                return lcm_of_array_elements;
            }
        }
    }

    public static void main(String[] args) {
        int[] element_array = {2, 7, 3, 9, 4};
        System.out.println(lcm_of_array_elements(element_array));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

long lcm_array(int arr[], int len) {
    long lcm = 1;
    int divisor = 2;
    while (true) {
        int counter = 0;
        bool divisible = false;
        for (int i = 0; i < len; i++) {
            if (arr[i] == 0) {
                return 0;
            } else if (arr[i] < 0) {
                arr[i] = arr[i]*(-1);
            }
            if (arr[i] == 1) {
                counter++;
            }
            if (arr[i] % divisor == 0) {
                divisible = true;
                arr[i] = arr[i] / divisor;
            }
        }
        if (divisible) {
            lcm = lcm*divisor;
        } else {
            divisor++;
        }
        if (counter == len) {
            return lcm;
        }
    }
}

int main() {
    int arr[] = {2, 7, 3, 9, 4};
    int len = sizeof arr / sizeof arr[0];
    cout << lcm_array(arr, len);
}
```

12.4) GCD de números flotantes

Una aproximación simple para realizar GCD a números flotantes es:

- a=1.20

- b=22.5

Expresando cada uno de los números sin decimales como el producto de los primos obtenemos:

- 120=2^3*3*5
- 2250=2*3^2*5^3

GCD de 120 y 2250 = 2*3*5=30

Por lo tanto, el GCD de 1.20 y 22.5=0.30 (Tomando 2 dígitos decimales).

Podemos hacer esto usando el algoritmo de Euclides, Este algoritmo indica si el número más pequeño es restado del número más largo, el GCD de dos números no cambia.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```java
// Programa java que encuentra el GCD de dos
// números flotantes

public class FloatGCD {
    // Funcion recursiva que retorna gcd de a y b
    static double gcd(double a, double b) {
        if (a < b) {
            return gcd(b, a);
        }
        // Caso base
        if (Math.abs(b) < 0.001) {
            return a;
        } else {
            return (gcd(b, a - Math.floor(a / b) * b));
        }
    }

    public static void main(String args[]) {
        double a = 1.20, b = 22.5;
        System.out.printf("%.1f", gcd(a, b));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
```

```cpp
using namespace std;

double GCD(double a, double b) {
    if (a < b) {
        return GCD(b, a);
    }
    if (std::fabs(b) < 0.001) {
        return a;
    } else {
        return (GCD(b, a - (std::floor(a / b) * b)));
    }
}

int main() {
    double a = 1.20, b = 22.5;
    printf("%.2f\n", (GCD(a, b)));
}
```

12.5) Test de primalidad

Introducción al Test de Primalidad

- Meta: Dado un entero n>1, determina si es primo o no.
- Los primos mas conocidos son:
  - 2, 3, 5, 7, 11, 13, 17, 19, 23, ...

Datos importantes:

- $38{,}476?$ No, porque es par
- $4{,}359?$ No, porque la suma de sus dígitos es 21, múltiplo de 3
- $127?$ Si, porque no tiene ningun factor $<\sqrt{127}\approx 11.27$
- $2^{57{,}885{,}161}-1?$

Este tiene más de 17 millones de dígitos, se necesitan test más optimos...

Guía del programador competitivo

Ilustración 12-3 Información importante sobre los números primos

Un número primo es un número natural mayor que 1 que tiene únicamente dos divisores distintos: él mismo y el 1. Por el contrario, los números compuestos son los números

naturales que tienen algún divisor natural aparte de sí mismos y del 1, y, por lo tanto, pueden factorizarse. El número 1, por convenio, no se considera ni primo ni compuesto.

La propiedad de ser número primo se denomina primalidad. El estudio de los números primos es una parte importante de la teoría de números, rama de las matemáticas que trata las propiedades, básicamente aritméticas, de los números enteros.

El teorema fundamental de la aritmética establece que todo número natural tiene una representación única como producto de factores primos, salvo el orden. Un mismo factor primo puede aparecer varias veces. El 1 se representa entonces como un producto vacío.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```java
//Programa java que verifica si un número
// es primo o no

public class PrimalityTest {

    public static void main(String[] args) {
        boolean isprime = false;
        int n = 5;
        if (isPrime(n)) {
            System.out.println("Es primo");
        } else {
            System.out.println("No es primo");
        }
    }

    //Prueba de primalidad
    public static boolean isPrime(int x) {
        if (x < 2) {
            return false;
        }
        if (x == 2) {
            return true;
        }
        if (x % 2 == 0) {
            return false;
        }
        for (int i = 2; i * i <= x; i++) {
            if (x % i == 0) {
                return false;
            }
        }
```

```
        return true;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

bool isPrime(int x) {
    if (x < 2) {
        return false;
    }
    if (x == 2) {
        return true;
    }
    for (int i = 2; i * i <= x; i++) {
        if (x % i == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    int n = 25;
    if (isPrime(n)) {
        cout << n << " es primo" << endl;
    } else {
        cout << n << " no es primo" << endl;
    }
}
```

PYTHON

```
from sys import stdout

def isPrime(x):
    if (x % 2 == 0 and x > 2) or x < 2:
        return False
    if x == 2:
        return True
    i = 3
    while i * i <= x:
        if x % i == 0:
            return False
        i += 1
    return True

def main():
    n = int(13)
    if isPrime(n):
```

```
        stdout.write("es primo")
    else:
        stdout.write("no es primo")

main()
```

12.6) Factores primos

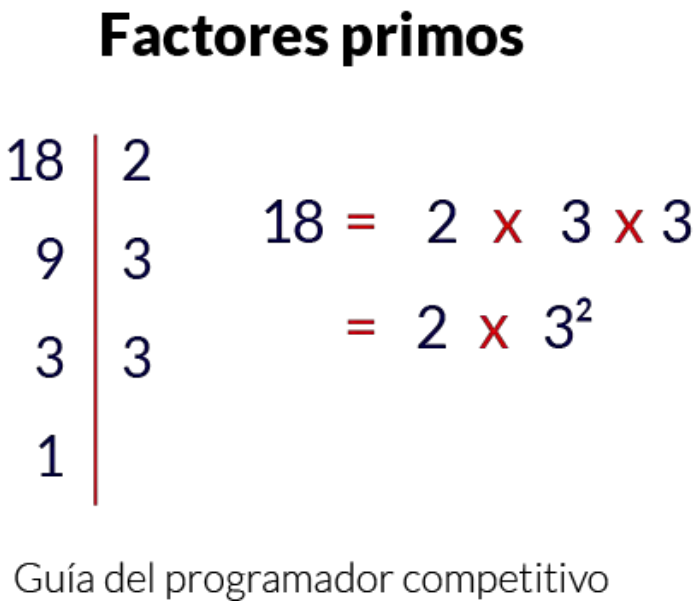


Ilustración 12-4 Factores primos de 18

Los factores primos de un número entero son los números primos divisores exactos de ese número entero. El proceso de búsqueda de esos divisores se denomina factorización de enteros, o factorización en números primos.

Determinar el número de factores primos de un número es un ejemplo de problema matemático frecuentemente empleado para asegurar la seguridad de los sistemas criptográficos: se cree que este problema requiere un tiempo superior al tiempo polinómico en el número de dígitos implicados; de hecho, es relativamente sencillo construir un problema que precisaría más tiempo que la Edad del Universo si se intentase calcular con los ordenadores actuales utilizando algoritmos actuales.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```java
//Programa que descompone un número n
// en sus factores primos

public class PrimeFactors {

    public static void main(String[] args) {
        primeFactors(12);
    }
    //Descomposicion en factores primos
    public static void primeFactors(int N) {
        for (long p = 2; p * p <= N; ++p) {
            while (N % p == 0) {
                System.out.println(p);
                N /= p;
            }
        }
        if (N > 1) {
            System.out.println(N);
        }
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
typedef long long int ll;

void primeFactors(ll n) {
    for (ll p = 2; p * p <= n; p++) {
        while (n % p == 0) {
            cout << p << " ";
            n /= p;
        }
    }
    if (n > 1) {
        cout << n << endl;
    }
}

int main() {
    ll n = 20;
    primeFactors(n);
}
```

PYTHON

```
from sys import stdout

def PrimeFactors(n):
    i = int(2)
    while i**2 <= n:
        while n % i == 0:
            stdout.write(str(i)+"\n")
            n //= i
        i+=1
    if n>1:
        stdout.write(str(n))

def main():
    PrimeFactors(20)

main()
```

12.7) Divisibilidad de un número

Ilustración 12-5 Divisibilidad de un número entro otros números

Los criterios de divisibilidad son reglas que sirven para saber si un número es divisible por otro sin necesidad de realizar la división, a continuación se presentan los criterios de divisibilidad para los números mas conocidos.

Número	Criterio	Ejemplo
2	El número termina en 0 o en una cifra par (2, 4, 6, 8).	378: porque la última cifra (8) es par.
3	La suma de sus cifras es un múltiplo de 3.	480: porque 4+ 8+ 0 = 12 es múltiplo de 3.
4	El número formado por las dos últimas cifras es un múltiplo de 4 o cuando termina en doble cero. O bien, si el resultado de sumar el doble del penúltimo dígito y el último da un número divisible entre 4.	7324: porque 24 es múltiplo de 4. 8200: porque termina en 00. 5232: porque 3*2+2=8 y 8 es múltiplo de 4.
5	La última cifra es 0 o 5.	485: porque termina en 5.
6	El número es divisible entre 2 y entre 3 a la vez.	18: es múltiplo común 2 y de 3.
7	Un número es divisible entre 7 cuando, al separar la última cifra de la derecha, multiplicarla por 2 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 7.	34349: separamos el 9,y lo duplicamos (18), entonces 3434-18=3416. Repetimos el proceso separando el 6 (341'6) y duplicándolo (12), entonces 341-12=329, y de nuevo, 32'9, 9*2=18, entonces 32-18=14; por lo tanto, 34349 es divisible entre 7 porque 14 es múltiplo de 7.
8	Un número es divisible por 8 si el número formado por las tres últimas cifras es un múltiplo de 8 o termina en tres ceros.	27896: porque 896 es múltiplo de 8. De otro modo considero las tres últimas cifras divido entre 2, veo si el cociente es múltiplo de 4, viendo las dos últimas; la mitad de 896 es 448, como termina en 48, múltiplo de 4.

9	Un número es divisible por 9 si la suma de sus cifras es múltiplo de 9.	3744: porque 3+7+4+4= 18 es múltiplo de 9.
10	Un número es divisible por 10 si su última cifra es 0.	470: termina en cifra 0.
11	Sumando las cifras (del número) en posición impar por un lado y las de posición par por otro. Luego se resta el resultado de ambas sumas obtenidas. Si el resultado es cero (0 o un múltiplo de 11, el número es divisible por este. Si el número tiene sólo dos cifras y estas son iguales será múltiplo de 11.	42702: 4+7+2=13 · 2+0=2 · 13-2=11 → 42702 es múltiplo de 11 66: porque las dos cifras son iguales. Entonces 66 es Múltiplo de 11
12	Un número es divisible por 12 cuando al hacer la división, siendo el cociente 3 y 4, sus restos dan como resultado 0.	420: es múltiplo de 3 ya que 4+2+0=6 y de 4 puesto que 20 también lo es. Por tanto es múltiplo de 12.
13	Un número es divisible entre 13 cuando, al separar la última cifra de la derecha, multiplicarla por 9 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 13	3822: separamos el último dos (382'2) y lo multiplicamos por 9, 2*9=18, entonces 382-18=364. Repetimos el proceso separando el 4 (36'4) y multiplicándolo por 9, 4*9=36, entonces 36-36=0; por lo tanto, 3822 es divisible entre 13
14	Un número es divisible entre 14 cuando es par y divisible entre 7	546: separamos el último seis (54'6) y lo doblamos, 6*2=12, entonces 54-12=42. 42 es múltiplo de 7 y 546 es par; por lo tanto, 546 es divisible entre 14
15	Un número es divisible entre 15 cuando es divisible entre 3 y 5	225: termina en 5 y la suma de sus cifras es múltiplo de 3; por lo tanto, 225 es divisible entre 15
17	Un número es divisible entre 17 cuando, al separar la última cifra de la derecha, multiplicarla por 5 y restarla de las cifras restantes la diferencia es igual a 0 o es un múltiplo de 17	2142: porque 214'2, 2*5=10, entonces 214-10=204, de nuevo, 20'4, 4*5=20, entonces 20-20=0; por lo tanto, 2142 es divisible entre 17.
18	Un número es divisible por 18 si es par y divisible por 9 (Si es par y además la suma de sus cifras es múltiplo de 9)	9702: Es par y la suma de sus cifras: 9+7+0+2=18 que también es divisible entre 9. Y efectivamente, si hacemos la división entre 18, obtendremos que el resto es 0 y el cociente 539.
19	Un número es divisible por 19 si al separar la cifra de las unidades, multiplicarla por 2 y sumar a las cifras restantes el resultado es múltiplo de 19.	3401: separamos el 1,lo doblamos (2) y sumamos 340+2= 342, ahora separamos el 2, lo doblamos (4) y sumamos 34+4=38 que es múltiplo de 19, luego 3401 también lo es.

20	Un número es divisible entre 20 si sus dos últimas cifras son ceros o múltiplos de 20. Cualquier número par que tenga uno o más ceros a la derecha, es múltiplo de 20.	57860: Sus 2 últimas cifras son 60 (Que es divisible entre 20), por lo tanto 57860 es divisible entre 20.
25	Un número es divisible entre 25 si sus dos últimas cifras son 00, o en múltiplo de 25 (25,50,75,...)	650: Es múltiplo de 25 por lo cual es divisible. 400 también será divisible entre 25.
27	Un número es divisible entre 27, si al dividirlo entre 3 da un cociente exacto que es múltiplo de 9.	: 11745. Entre 3, cociente =3915; cuyas cifras suman 18, luego 11745 en divisible por 27.
29	Un número es divisible por 29 si al separar la cifra de las unidades, multiplicarla por 3 y sumar a las cifras restantes el resultado es múltiplo de 29.	2262: separamos el último 2, lo triplicamos (6) y sumamos, 226+6= 232, ahora separamos el último 2, lo triplicamos (6) y sumamos 23+6=29 que es múltiplo de 29, luego 2262 también lo es.
31	Un número es divisible por 31 si al separar la cifra de las unidades, multiplicarla por 3 y restar a las cifras restantes el resultado es múltiplo de 31.	8618: separamos el 8, lo triplicamos (24) y restamos 861-24=837, ahora separamos el 7, lo triplicamos (21) y restamos, 83-21=62 que es múltiplo de 31, luego 8618 también lo es.
50	Un número es múltiplo de 50 cuando sus dos últimas cifras son 00 o 50	
100	Un número será divisible por 100 si dicho número termina en 00.	1000: Este número será divisible por cien ya que sus dos últimas cifras son 00, independientemente de las demás.
125	Un número será divisible por 125 si sus tres últimas cifras son 000 o múltiplo de 125.	3000: Sería divisible por 125 ya que sus tres últimas cifras son 000. 4250: Este número también sería divisible por 125 ya que sus tres últimas cifras son múltiplo de 125.

Tabla 12-1 Criterios de divisibilidad más conocidos

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
//Programa que verifica la divisibilidad de un
// número entre los números 1 a 12
```

```java
public class DivisibilityBySmallNumbers {

    public static void main(String[] args) {
        boolean[] res = divisility(12);
        for (int i = 0; i < res.length; i++) {
            System.out.println(i + " : " + res[i]);
        }
    }

    static public boolean[] divisility(int num) {
        String M = String.valueOf(num);
        int n = M.length();
        boolean[] isMultipleOf = new boolean[13];
        int sum = 0;
        for (char c : M.toCharArray()) {
            sum += c - '0';
        }
        int lastDigit = M.charAt(n - 1) - '0';
        //Divisibilidad entre 1
        isMultipleOf[1] = true;
        //Divisibilidad entre 2
        isMultipleOf[2] = lastDigit % 2 == 0;
        //Divisibilidad entre 3
        isMultipleOf[3] = sum % 3 == 0;
        //Divisibilidad entre 4
        if (n > 1) {
            isMultipleOf[4] = Integer.parseInt(M.substring(n - 2, n)) % 4 == 0;
        } else {
            isMultipleOf[4] = lastDigit % 4 == 0;
        }
        //Divisibilidad entre 5
        isMultipleOf[5] = lastDigit == 0 || lastDigit == 5;
        //Divisibilidad entre 6
        isMultipleOf[6] = isMultipleOf[2] && isMultipleOf[3];
        int altSum = 0;
        int[] pattern = {1, 3, 2, -1, -3, -2};
        int j = 0;
        for (int i = n - 1; i >= 0; i--) {
            altSum += pattern[j] * (M.charAt(i) - '0');
            j = (j + 1) % 6;
        }
        //Divisibilidad entre 7
        isMultipleOf[7] = Math.abs(altSum) % 7 == 0;
        //Divisibilidad entre 8
        if (n > 2) {
            isMultipleOf[8] = Integer.parseInt(M.substring(n - 3, n)) % 8 == 0;
        } else {
            isMultipleOf[8] = Integer.parseInt(M) % 8 == 0;
        }
        //Divisibilidad entre 9
        isMultipleOf[9] = sum % 9 == 0;
        //Divisibilidad entre 10
        isMultipleOf[10] = lastDigit == 0;
        altSum = 0;
```

```
        int s = 1;
        for (int i = n - 1; i >= 0; i--) {
            altSum += s * (M.charAt(i) - '0');
            s = -s;
        }
        //Divisibilidad entre 11
        isMultipleOf[11] = Math.abs(altSum) % 11 == 0;
        //Divisibilidad entre 12
        isMultipleOf[12] = isMultipleOf[3] && isMultipleOf[4];
        return isMultipleOf;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

void print(bool arr[]) {
    for (int i = 0; i < 13; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}

void divisibility(int num) {
    stringstream ss;
    ss << num;
    string M;
    ss>>M;
    int n = M.size();
    bool isMultipleof[13];
    memset(isMultipleof, false, sizeof isMultipleof);
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += (M[i] - '0');
    }
    int lastDigit = M[n - 1] - '0';
    isMultipleof[0] = false;
    isMultipleof[1] = true;
    isMultipleof[2] = lastDigit % 2 == 0;
    isMultipleof[3] = sum % 3 == 0;
    if (n > 1) {
        isMultipleof[4] = stoi(M.substr(n - 2, n)) % 4 == 0;
    } else {
        isMultipleof[4] = lastDigit % 4 == 0;
    }
    isMultipleof[5] = lastDigit == 0 || lastDigit == 5;
    isMultipleof[6] = isMultipleof[2] && isMultipleof[3];
    int altsum = 0;
    int pattern[] = {1, 3, 2, -1, -3, -2};
    int j = 0;
    for (int i = n - 1; i>-0; i--) {
        altsum += pattern[j]*(M[i] - '0');
```

```
        j = (j + 1) % 6;
    }
    isMultipleof[7] = std::abs(altsum) % 7 == 0;
    if (n > 2) {
        isMultipleof[8] = stoi(M.substr(n - 3, n)) % 8 == 0;
    } else {
        isMultipleof[8] = stoi(M) % 8 == 0;
    }
    isMultipleof[9] = sum % 8 == 0;
    isMultipleof[10] - lastDigit == 0;
    altsum = 0;
    int s = 1;
    for (int i = n - 1; i >= 0; i--) {
        altsum += s * (M[i] - '0');
        s = -s;
    }
    isMultipleof[11] = std::abs(altsum) % 11 == 0;
    isMultipleof[12] = isMultipleof[2] && isMultipleof[4];
    print(isMultipleof);
}

int main() {
    divisibility(33);
}
```

PYTHON

```
from sys import stdout


def divisibility(num):
    M = str(num)
    n = len(M)
    isMultipleOf = [False for x in range(13)]
    suma = 0
    for c in M:
        suma += ord(c) - ord("0")

    lastDigit = ord(M[n - 1]) - ord("0")
    isMultipleOf[1] = True
    isMultipleOf[2] = lastDigit % 2 == 0
    isMultipleOf[3] = suma % 3 == 0

    if n > 1:
        isMultipleOf[4] = int(M[n - 2:n]) % 4 == 0
    else:
        isMultipleOf[4] = lastDigit % 4 == 0

    isMultipleOf[5] = lastDigit == 0 or lastDigit == 5
    isMultipleOf[6] = isMultipleOf[2] and isMultipleOf[3]

    altsum = 0
    pattern = [1, 3, 2, -1, -3, -2]
    j = 0
```

```
    for i in range(n - 1, -1, -1):
        altsum = pattern[j] * (ord(M[i]) - ord("0"))
        j = (j + 1) % 6
    isMultipleOf[7] = abs(altsum) % 7 == 0
    if n > 2:
        isMultipleOf[8] = int(M[n - 3:n]) % 8 == 0
    else:
        isMultipleOf[8] = int(M) % 8 == 0

    isMultipleOf[9] = suma % 8 == 0
    isMultipleOf[10] = lastDigit == 0
    altsum = 0
    s = 1
    for i in range(n-1, -1, -1):
        altsum += s * (ord(M[i]) - ord("0"))
        s = -s

    isMultipleOf[11] = abs(altsum) % 11 == 0
    isMultipleOf[12] = isMultipleOf[4] and isMultipleOf[13]
    return isMultipleOf


res = divisibility(123)
for i in range(len(res)):
    stdout.write(f"{i} : {res[i]}\n")
```

12.8) Numero de divisores

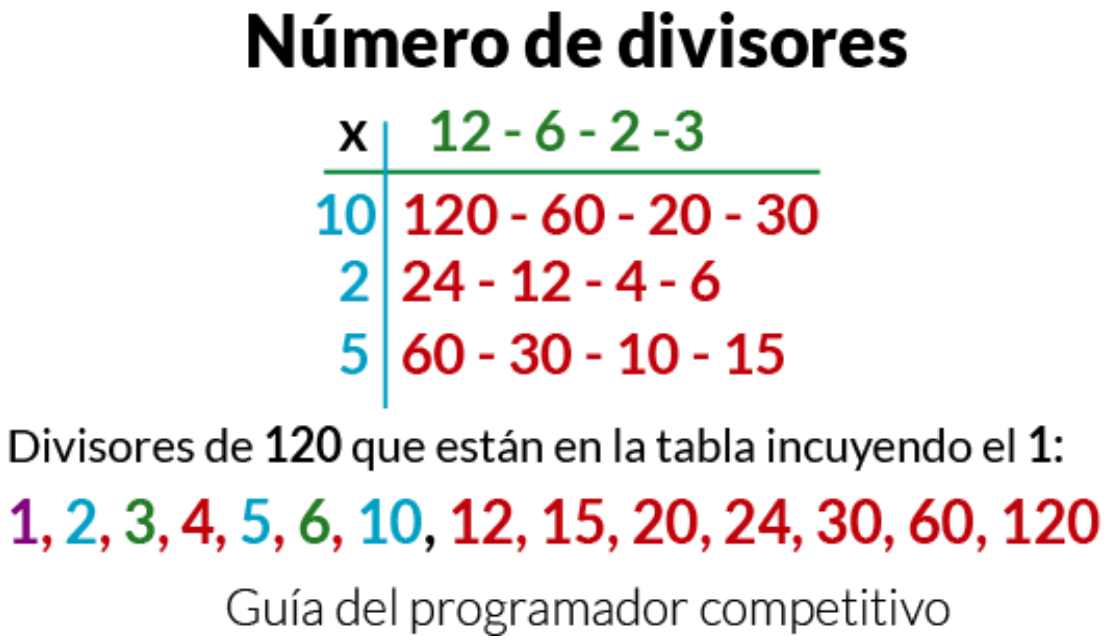


Ilustración 12-6 Los divisores de 120

Los divisores de un número son aquellos valores que dividen al número en partes exactas. Así, dado un número a, si la división a/b es exacta (el resto es cero), entonces se dice que b es divisor de a. También se puede decir que a es divisible por b o que a es un múltiplo de b. Esto resulta útil, por ejemplo, a la hora de agrupar una cantidad de objetos en partes iguales sin que nos sobre ninguno.

Lógicamente, el 1 siempre es divisor de cualquier número, porque siempre podemos hacer paquetes individuales y no nos sobrará ninguno. De igual forma, todo número es divisible por sí mismo, lo que equivaldría a hacer un único paquete.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```
//Programa java que cuenta el número
// de divisores de x

public class NumberOfDivisors {
```

```
    public static void main(String[] args) {
        System.out.println(divisors(56));
    }

    //Todos los divisores de un número
    public static int divisors(int x) {
        int nDiv = 1;
        for (int p = 2; p * p <= x; ++p) {
            int cnt = 0;
            while (x % p == 0) {
                ++cnt;
                x /= p;
            }
            nDiv *= cnt + 1;
        }
        if (x > 1) {
            nDiv *= 2;
        }
        return nDiv;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

int divisors(int x) {
    int nDiv = 1;
    for (int i = 2; i * i <= x; i++) {
        int cnt = 0;
        while (x % i == 0) {
            ++cnt;
            x /= i;
        }
        nDiv *= cnt + 1;
    }
    if (x > 1) {
        nDiv *= 2;
    }
    return nDiv;
}

int main() {
    cout << divisors(10);
}
```

PYTHON

```
from sys import stdout
```

```
def divisors(x):
    nDiv = 1
    i = 2
    while i ** 2 <= x:
        cnt = 0
        while x % i == 0:
            cnt += 1
            x //= i
        nDiv *= cnt + 1
        i += 1
    if x > 1:
        nDiv *= 2
    return nDiv

stdout.write(str(divisors(10)))
```

12.9) Criba de Eratóstenes

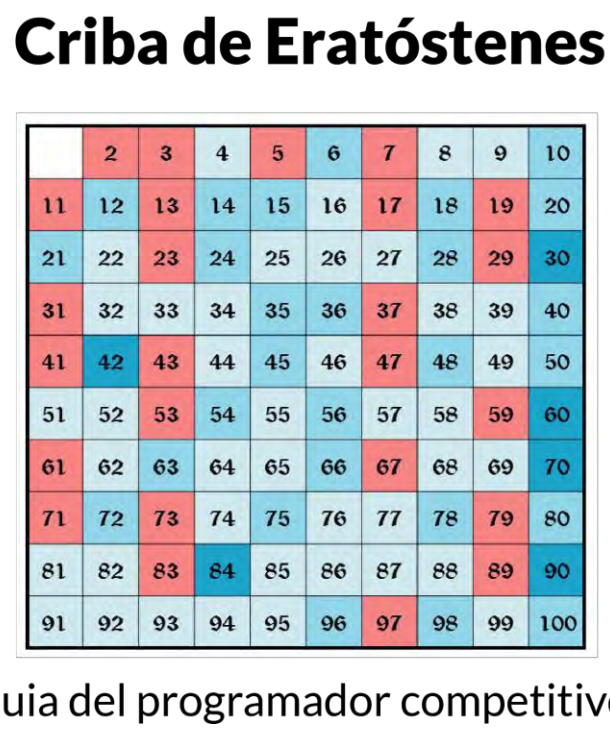


Ilustración 12-7 Vista en matriz de búsqueda de primos por medio de la criba de Eratóstenes

Criba de Eratóstenes es un algoritmo que permite hallar todos los números primos menores que un número natural dado n. Se forma una tabla con todos los números naturales

comprendidos entre 2 y n, y se van tachando los números que no son primos de la siguiente manera:

Comenzando por el 2, se tachan todos sus múltiplos; comenzando de nuevo, cuando se encuentra un número entero que no ha sido tachado, ese número es declarado primo, y se procede a tachar todos sus múltiplos, así sucesivamente. El proceso termina cuando el cuadrado del siguiente número confirmado como primo es mayor que n.

Complejidad de tiempo

Mejor caso : O(nlog(log(n))) **Peor caso :** O(nlog(log(n))) **Promedio:** O(nlog(log(n)))

JAVA

```java
//Programa java que encuentra todos los números
//primos bajo N

import java.util.Arrays;

public class PrimeNumbers {

    public static void main(String[] args) {
        boolean[] res = sieveEratostenes(100);
        for (int i = 0; i < res.length; i++) {
            System.out.println(i + " : " + res[i]);
        }
    }

    //Criba de Eratostenes (Todos los primos bajo un número)
    public static boolean[] sieveEratostenes(int N) {
        boolean[] prime = new boolean[N + 1];
        Arrays.fill(prime, true);
        prime[0] = prime[1] = false;
        for (int p = 2; p * p <= N; p++) {
            if (prime[p]) {
                for (int i = p * p; i <= N; i += p) {
                    prime[i] = false;
                }
            }
        }
        return prime;
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
```

```cpp
#define FAST ios_base::sync_with_stdio(false);cin.tie(NULL);
using namespace std;

void print(bool arr[], int n) {
    for (int i = 0; i <= n; i++) {
        cout << i << ": " << arr[i] << endl;
    }
}

void sieveErathostenes(int N) {
    bool prime[N + 1];
    memset(prime, true, sizeof prime);
    prime[0] = prime[1] = false;
    for (int p = 2; p * p <= N; p++) {
        if (prime[p]) {
            for (int i = p * p; i <= N; i += p) {
                prime[i] = false;
            }
        }
    }
    print(prime, N);
}

int main() {
    FAST;
    sieveErathostenes(1000);
    return 0;
}
```

PYTHON

```python
from sys import stdin
from sys import stdout


def sieveErathostenes(n):
    prime = [True for x in range(n + 1)]
    prime[0] = False
    prime[1] = False
    p = 2
    while p ** 2 <= n:
        if prime[p]:
            i = p ** 2
            while i <= n:
                prime[i] = False
                i += p
        p += 1
    return prime

res = sieveErathostenes(100)

for i in range(len(res)):
    stdout.write(str(i) + " " + str(res[i])+ "\n")
```

12.10) Criba de Eratóstenes O(n)

La clásica criba de Eratóstenes toma O(Nlog (logN)) para encontrar todos los números primos menores a N, este código es una versión modificada de esta criba que tiene una complejidad de tiempo de O(N).

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
/*Programa java que genera todos los números primos
menores a N en O(N) Eratostenes Optimizado*/
import java.util.ArrayList;

public class SieveofEratosthenesOptimizated {

    static final int MAX_SIZE = 1000001;
    // isPrime[] : isPrime[i] es true si el número es primo
    // prime[] : Almacena todos los números primos menores a N
    // SPF[] Almacena los factores primos más pequeños de un número
    // [Por ejemplo : factor primo más pequeño que '8' y '16' es
    //  '2' entonces nosotros ponemos SPF[8]=2 , SPF[16]=2 ]
    static ArrayList<Boolean> isprime = new ArrayList<>(MAX_SIZE);
    static ArrayList<Integer> prime = new ArrayList<>();
    static ArrayList<Integer> SPF = new ArrayList<>(MAX_SIZE);
    // Metodo que genera todos los factores primos menores de N

    static void manipulated_seive(int N) {   // 0 y 1 no son primos
        isprime.set(0, false);
        isprime.set(1, false);

        // llena el resto de las entradas
        for (int i = 2; i < N; i++) {
            // si isPrime[i] == True entonces i es
            // número primo
            if (isprime.get(i)) {
                // pone i dentro de prime[]
                prime.add(i);
                // un número primo es su propio factor primo
                // más pequeño
                SPF.set(i, i);
            }
            /*Eliminar todos los múltiplos de i * prime [j] que son
            no primos haciendo isPrime [i * prime [j]] = false
            y ponga el factor primo más pequeño de i * Prime [j] como prime [j]
```

```
            [Por ejemplo: dejemos i = 5, j = 0, prime[j] = 2 [i * prime [j] =
10]
            por lo que el factor primo más pequeño de '10' es '2' que es prime
[j]]
            este bucle se ejecuta solo una vez para el número que no es primo*/

            for (int j = 0;
                    j < prime.size()
                    && i * prime.get(j) < N && prime.get(j) <= SPF.get(i);
                    j++) {
                isprime.set(i * prime.get(j), false);
                // Pone el factor primo más pequeño de i*prime[j]
                SPF.set(i * prime.get(j), prime.get(j));
            }
        }
    }

    public static void main(String args[]) {
        int N = 13; //Debe ser menor que  MAX_SIZE
        // inicializando isprime y SPF
        for (int i = 0; i < MAX_SIZE; i++) {
            isprime.add(true);
            SPF.add(2);
        }
        manipulated_seive(N);
        //Imprima todos los números primos menores que n
        for (int i = 0; i < prime.size() && prime.get(i) <= N; i++) {
            System.out.print(prime.get(i) + " ");
        }
    }
}
```

C++

```
#include <bits/stdc++.h>
#define FAST ios_base::sync_with_stdio(false);cin.tie(NULL);
#define MAX_SIZE 1000001
using namespace std;
typedef long long int ll;
vector<int>prime;

void manipule(ll N) {
    bool arr[N + 1];
    ll SPF[N + 1];
    memset(arr, true, sizeof arr);
    memset(SPF, 2, sizeof SPF);
    arr[0] = false;
    arr[1] = false;
    for (ll i = 2; i < N; i++) {
        if (arr[i]) {
            prime.push_back(i);
            SPF[i] = i;
        }
        for (ll j = 0; j < prime.size() && i * prime[j] < N && prime[j] <=
SPF[i]; j++) {
```

```
            arr[i * prime[j]] = false;
            SPF[i * prime[j]] = prime[j];
        }
    }
    for (ll i = 0; i < prime.size(); i++) {
        cout << prime[i] << " ";
    }
}

int main() {
    manipule(1000);
}
```

PYTHON

```
from sys import stdin
from sys import stdout

wr = stdout.write

MAX_SIZE = 1000001

isprime = [True for x in range(MAX_SIZE)]

prime = []

SPF = [int for x in range(MAX_SIZE)]

def manipuledSieve(N):
    global isprime
    global prime
    global SPF
    isprime[0] = False
    isprime[1] = False
    for i in range(2, N):
        if isprime[i]:
            prime.append(i)
            SPF[i] = i
        j = 0
        while j < len(prime) and i * prime[j] < N and prime[j] <= SPF[i]:
            isprime[i * prime[j]] = False
            SPF[i * prime[j]] = prime[j]
            j += 1

N = 100
for i in range(MAX_SIZE):
    # isprime.append(True)
    SPF.append(2)
manipuledSieve(N)
for i in range(len(prime)):
    wr(f"{prime[i]}  ")

wr("\n")
```

```
for i in range(2, N):
    wr(f"{i} : {SPF[i]} \n")
```

12.11) Ultimo digito de un Fibonacci

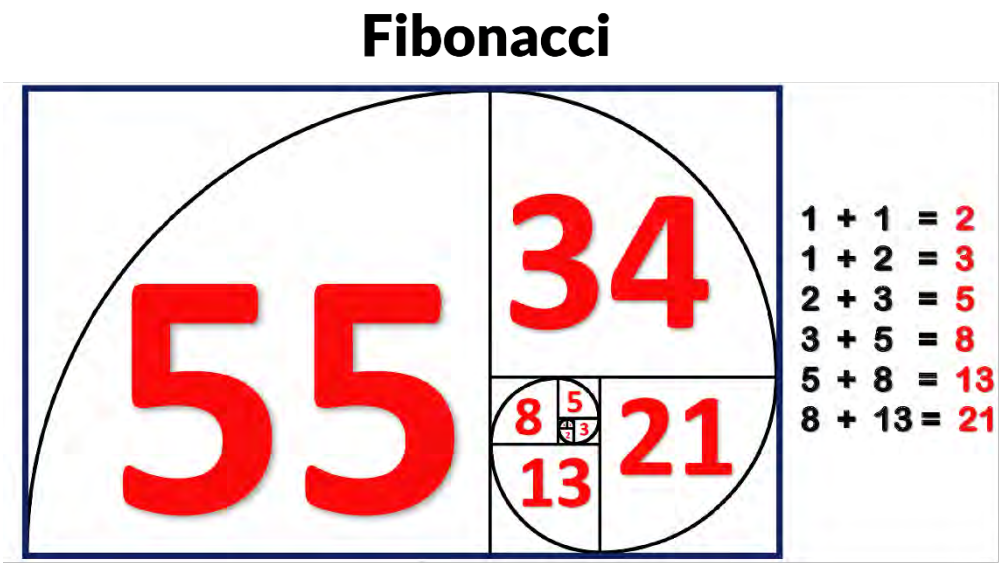


Ilustración 12-8 Sucesión de Fibonacci

La sucesión o serie de Fibonacci es una sucesión infinita de números naturales: 0 1 1 2 3 58 21 34 55 89....

La sucesión comienza con los números 0 y 1, y a partir de estos, cada término es la suma de los dos anteriores.

Los números de esta sucesión pueden llegar a ser muy grandes y calcular su último digito se hace complicado en máquinas.

Usando fib, multiply y power se puede calcular de forma optimizada un numero de Fibonacci, pero hay que tener en cuenta de no desbordar la variable o el resultado podría no ser correcto.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
// Programa java que encuentra el ultimo digito
// de un nesimo número fibonacci

public class FibonacciLastDigit {
    //Funcion que retona el nesimo
    // Número de fibonacci
    static long fib(long n) {
        long F[][] = new long[][]{{1, 1}, {1, 0}};
        if (n == 0) {
            return 0;
        }
        power(F, n - 1);
        return F[0][0];
    }
    //Función que multiplica dos
    // matrices y almacera el resultado en la primera
    static void multiply(long F[][], long M[][]) {
        long x = F[0][0] * M[0][0]
                + F[0][1] * M[1][0];
        long y = F[0][0] * M[0][1]
                + F[0][1] * M[1][1];
        long z = F[1][0] * M[0][0]
                + F[1][1] * M[1][0];
        long w = F[1][0] * M[0][1]
                + F[1][1] * M[1][1];
        F[0][0] = x;
        F[0][1] = y;
        F[1][0] = z;
        F[1][1] = w;
    }

    static void power(long F[][], long n) {
        if (n == 0 || n == 1) {
            return;
        }
        long M[][] = new long[][]{{1, 1}, {1, 0}};
        power(F, n / 2);
        multiply(F, F);
        if (n % 2 != 0) {
            multiply(F, M);
        }
    }
    // Retorna el ultimo digito
    // nesimo número fibonacci
    public static long findLastDigit(long n) {
        return (fib(n) % 10);
    }
```

```
    public static void main(String[] args) {
        int n;
        n = 1;
        System.out.println(findLastDigit(n));
        n = 61;
        System.out.println(findLastDigit(n));
        n = 7;
        System.out.println(findLastDigit(n));
        n = 67;
        System.out.println(findLastDigit(n));
    }
}
```

C++

```
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;

void multiply(ll F[][2], ll M[][2]) {
    ll x = F[0][0] * M[0][0] + F[0][1] * M[1][0];
    ll y = F[0][0] * M[0][1] + F[0][1] * M[1][1];
    ll z = F[1][0] * M[0][0] + F[1][1] * M[1][0];
    ll w = F[1][0] * M[0][1] + F[1][1] * M[1][1];
    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}

void power(ll F[][2], ll n) {
    if (n == 0 || n == 1) {
        return;
    }
    ll M[][2] = {
        {1, 1},
        {1, 0}};
    power(F, n / 2);
    multiply(F, F);
    if (n % 2 != 0) {
        multiply(F, M);
    }
}

ll fib(ll n) {
    ll F [][2] = {
        {1, 1},
        {1, 0}};
    if (n == 0) {
        return 0;
    }
    power(F, n - 1);
    return F[0][0];
}
```

```cpp
ll findLastDigit(ll n) {
    return (fib(n) % 10);
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout << fib(83) << " : " << findLastDigit(83);
    return 0;
}
```

PYTHON

```python
from sys import stdout
wr = stdout.write


def multiply(F, M):
    x = F[0][0] * M[0][0] + F[0][1] * M[1][0]
    y = F[0][0] * M[0][1] + F[0][1] * M[1][1]
    z = F[1][0] * M[0][0] + F[1][1] * M[1][0]
    w = F[1][0] * M[0][1] + F[1][1] * M[1][1]
    F[0][0] = x
    F[0][1] = y
    F[1][0] = z
    F[1][1] = w


def power(F, n):
    if n == 0 or n == 1:
        return
    M = [[1, 1], [1, 0]]
    power(F, n // 2)
    multiply(F, F)
    if n % 2 != 0:
        multiply(F, M)

def fib(n):
    F = [[1, 1], [1, 0]]
    if n == 0:
        return 0
    power(F, n - 1)
    return F[0][0]

def findLastDigit(n):
    return fib(n) % 10

wr(f'{fib(20)} : {findLastDigit(20)}')
```

12.12) Fibonacci largos

Ciertos ejercicios demandar realizar el cálculo de números de la secuencia de Fibonacci los cuales son demasiado grandes, para ciertos lenguajes el tamaño de las variables es un impedimento para realizar esta acción por lo que se debe usar librerías que permitan realizar operaciones con números increíblemente grandes, pero hay que tener en cuenta que entre más grande el numero, el tiempo de ejecución va a ser mayor.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Porgrama java que busca el nesimo número
// de fibonacci cuando n puede ser muy largo

import java.math.*;

public class LongFibonacci {

    static BigInteger fib(int n) {
        BigInteger a = BigInteger.valueOf(0);
        BigInteger b = BigInteger.valueOf(1);
        BigInteger c = BigInteger.valueOf(1);
        for (int j = 2; j <= n; j++) {
            c = a.add(b);
            a = b;
            b = c;
        }
        return (a);
    }

    public static void main(String[] args) {
        int n = 1000;
        System.out.println("Fibonacci de  " + n
                + " termino" + " " + "es" + " " + fib(n));
    }
}
```

12.13) Test de numero de Fibonacci

Teniendo cualquier número entero n, se necesita verificar si este hace parte de la secuencia de los números de Fibonacci de forma optimizada.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```java
// Programa que verifica si x es número fibonacci

public class IsFiboNumber {
    // Función que retorna si x es cuadrado perfecto
    static boolean isPerfectSquare(int x) {
        int s = (int) Math.sqrt(x);
        return (s * s == x);
    }
    //Función que verifica si es número fibonacci
    static boolean isFibonacci(int n) {
        // N es fibinacci si uno de 5*n*n+4 o or 5*n*n - 4
        // o ambos son cuadrados perfectos
        return isPerfectSquare(5 * n * n + 4)
                || isPerfectSquare(5 * n * n - 4);
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(isFibonacci(i) ? i + " Es un número de fibonacci"
                    : i + " No es un número de fibonacci");
        }
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

bool isPerfectSquare(int x) {
    int s = (int) sqrt(x);
    return (s * s == x);
}

bool isFibonacci(int n) {
    return isPerfectSquare(5 * n * n + 4) || isPerfectSquare(5 * n * n - 4);
}

int main() {
    for (int i = 0; i < 10; i++) {
        isFibonacci(i) ? cout << i << "Es Fibonacci" << endl : cout << i << "No
es fibonacci" << endl;
    }
}
```

12.14) Fibonacci Golden Ratio

Existen diferentes métodos para encontrar el enésimo número de Fibonacci, una simple manera de encontrarlo es usando el ratio dorado.

Golden ratio (Ratio dorado):

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887 \ldots.$$

Golden ratio nos puede dar una respuesta incorrecta.

Podemos obtener la respuesta correcta si redondeamos hacia arriba el resultado de cada punto.

Este método puede calcular los primeros 34 números de Fibonacci correctamente, luego de esto puede haber diferencia con el valor correcto.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```java
// Programa java que encuentra el
// nesimo número fibonacci

public class FiboGoldenRatio {   //Valor aproximado del golden ratio
    // Approximate value of golden ratio
    static double PHI = 1.6180339;
    // Números fibonacci hasta 5
    static int f[] = {0, 1, 1, 2, 3, 5};
    // Función que encuentra nesimo
    // número fibonacci
    static int fib(int n) {   // Números fibonacci menores a 6
        if (n < 6) {
            return f[n];
        }
        // Si no comience conteo desde el quinto
        int t = 5;
        int fn = 5;
        while (t < n) {
            fn = (int) Math.round(fn * PHI);
            t++;
        }
        return fn;
```

```
    }

    public static void main(String[] args) {
        int n = 9;
        System.out.println(n + " número fibonacci= "
                + fib(n));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
const double phi = 1.6180339;
int f[] = {0, 1, 1, 2, 3, 5};

int fib(int n) {
    if (n < 6) {
        return f[n];
    }
    int t = 5;
    int fn = 5;
    while (t < n) {
        fn = (int) round(fn * phi);
        t++;
    }
    return fn;
}

int main() {
    for (int i = 0; i < 10; i++) {
        cout << fib(i) << endl;
    }
}
```

12.15) Permutaciones

Permutaciones

$$P(n, r) = \frac{n!}{(n-r)!}$$

P es el número de permutaciones
n es el número de objetos en el conjunto
r es el número de objetos elegidos del conjunto

Ilustración 12-9 Ecuación de la permutación

Una permutación es la variación del orden o posición de los elementos de un conjunto ordenado o una tupla, hay dos tipos de permutaciones:

Se permite repetir: una cerradura de combinación de tres numeros, podría ser "333".
Sin repetición: Los tres primeros puestos en una carrera. No puedes quedar primero y segundo a la vez.

1. **Permutaciones con repetición**

Son las más fáciles de calcular. Si tiene n cosas para elegir y elige r de ellas, las permutaciones posibles son:

- n × n × ... (r veces) = nr

(Porque hay n posibilidades para la primera elección, luego hay n posibilidades para la segunda elección, y así.)
Por ejemplo en un candado de combinacion de 3 numeros, hay 10 números para elegir (0,1,...,9) y se elige 3 de ellos:

- 10 × 10 × ... (3 veces) = 10^3 = 1000 permutaciones

Así que la fórmula es simplemente:

- $n * r$

donde n es el número de cosas que puede elegir, y elige r de ellas, dentro de estas permutaciones **se puede repetir y el orden importa**.

2. Permutaciones sin repetición

En este caso, se reduce el número de opciones en cada paso.

¿cómo se puede ordenar 16 bolas de billar?

Después de elegir por ejemplo la "14" no puede elegirla otra vez, así que la primera elección tiene 16 posibilidades, y la siguiente elección tiene 15 posibilidades, después 14, 13, hasta 1 Y el total de permutaciones sería:

- 16 × 15 × 14 × 13 ... = 20.922.789.888.000

Pero a lo mejor no quiere elegirlas todas, sólo 3 de ellas, así que sería solamente:

- 16 × 15 × 14 = 3360

Es decir, hay 3.360 maneras diferentes de elegir 3 bolas de billar de entre 16.

La función factorial "!" significa que se multiplican números descendentes, varios ejemplos de factoriales son:

- 4! = 4 × 3 × 2 × 1 = 24
- 7! = 7 × 6 × 5 × 4 × 3 × 2 × 1 = 5040
- 1! = 1
- 16! = 20.922.789.888.000

Pero si sólo quiere elegir 3, tiene que dejar de multiplicar después de 14.

¿Cómo se puede escribir esto? Se divide entre 13!...

- 16x15x14x13….. /13x12x11….
- 16! / 13! = 16 × 15 × 14

La fórmula se escribe:

- $\frac{n!}{(n-r)!}$

donde n es el número de cosas que puede elegir, y elige r de ellas, dentro de este tipo e permutaciones **no se puede repetir y el orden importa**.

Complejidad de tiempo

Mejor caso : O(ecuación aplicable) **Peor caso :** O(ecuación aplicable)

Promedio: O(ecuación aplicable)

JAVA

```
//Programa java que imprime todas las permutaciones
// con o sin repetición, r modifica el tamaño de las permutaciones
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class Permutations {

    static Set<String> permutations;
    static Set<String> result = new HashSet<>();
    static int cont = 0;

    static void permWithRepUtil(String str, char[] data,
            int last, int index) {
        int length = str.length();
        for (int i = 0; i < length; i++) {
            data[index] = str.charAt(i);
            if (index == last) {
                System.out.println(new String(data));
                cont++;
            } else {
                permWithRepUtil(str, data, last,
                        index + 1);
            }
        }
    }

    static void permWithRep(String str) {
        int length = str.length();
        char[] data = new char[length + 1];
        char[] temp = str.toCharArray();
        Arrays.sort(temp);
        str = new String(temp);
        permWithRepUtil(str, data, length - 1, 0);
    }

    static void shuffle(char c) {
        if (permutations.isEmpty()) {
            permutations.add(String.valueOf(c));
        } else {
            Iterator<String> it = permutations.iterator();
```

```
            for (int i = 0; i < permutations.size(); i++) {
                String temp1;
                while (it.hasNext()) {
                    temp1 = it.next();
                    for (int k = 0; k < temp1.length() + 1; k++) {
                        StringBuilder sb = new StringBuilder(temp1);
                        sb.insert(k, c);
                        result.add(sb.toString());
                    }
                }
            }
            permutations = result;
            result = new HashSet<>();
        }
    }

    static Set<String> permutation(String string) {
        permutations = new HashSet<>();
        int n = string.length();
        for (int i = n - 1; i >= 0; i--) {
            shuffle(string.charAt(i));
        }
        Set<String> aux = new HashSet<>();
        Iterator<String> it = permutations.iterator();
        while (it.hasNext()) {
            String aux2 = it.next();
            //Entre mas aumente r, mas pequeñas seran las permutations
            int r = 0;
            aux.add(aux2.substring(0, aux2.length() - 0));
        }
        return aux;

    }

    public static void main(String[] args) {
        String entrada = "1234";
        Set<String> res = permutation(entrada);
        System.out.println("Hay en total " + res.size() + " permutations sin
repetición de " + entrada);
        Iterator<String> it = res.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
        String entrada2 = "ABC";
        permWithRep(entrada2);
        System.out.println("Hay en total " + cont + " permutations con
repetición de " + entrada2);

    }
}
```

C++

```
#include <iostream>
#include <set>
```

```
#include <string.h>
#include <algorithm>

using namespace std;

int cont1 = 0;
int cont2 = 0;

void permWithOutRepetition(string str) {
    char ayuda[str.size()];
    strcpy(ayuda, str.c_str());
    do {
        ++cont1;
        cout << ayuda << endl;
    } while (next_permutation(ayuda, ayuda + str.size()));
}

void permWithRepUtil(string str, char data [], int last, int index) {
    int length = str.size();
    for (int i = 0; i < length; i++) {
        data[index] = str[i];
        if (index == last) {
            cont2++;
            string ayuda = data;
            cout << ayuda << endl;
        } else {
            permWithRepUtil(str, data, last, index + 1);
        }
    }
}

void permWithRep(string str) {
    int length = str.size();
    char data[length + 1];
    char temp[length];
    strcpy(temp, str.c_str());
    sort(temp, temp + length);
    str = temp;
    permWithRepUtil(str, data, length - 1, 0);
}

int main() {
    ios_base::sync_with_stdio(false);
    cout.tie(NULL);
    cin.tie(NULL);
    string entrada = "Terry";
    permWithOutRepetition(entrada);
    cout << "Hay en total " << cont1 << " permutaciones sin repeticion de " <<
entrada << endl;
    cout << "\n\n" << endl;
    permWithRep(entrada);
    cout << "Hay en total " << cont2 << " permutaciones con repeticion de " <<
entrada << endl;
}
```

12.16) Combinatorias

Combinaciones

$$C(n, r) = \frac{n!}{r! \times (n - r)!}$$

C es el número de combinaciones
n es el número de objetos en el conjunto
r es el número de objetos elegidos del conjunto.

Guia del programador competitivo

Ilustración 12-10 Ecuación de la combinación

La Combinatoria es la parte de las Matemáticas que estudia las diversas formas de realizar agrupaciones con los elementos de un conjunto, formándolas y calculando su número.

Existen distintas formas de realizar estas agrupaciones, según se repitan los elementos o no, según se puedan tomar todos los elementos de que disponemos o no y si influye o no el orden de colocación de los elementos.

El orden NO es importante. La notación para las combinaciones es C(n,r) que es la cantidad de combinaciones de “n” elementos seleccionados, “r” a la vez. Es igual a la cantidad de permutaciones de “n” elementos tomados “r” a la vez dividido por “r” factorial.

- P(n,r)/r!.

Ejemplo: Si se seleccionan cinco cartas de un grupo de nueve, ¿cuantas combinaciones de cinco cartas habría?

La cantidad de combinaciones posibles sería:

- P(9,5)/5! = (9*8*7*6*5)/(5*4*3*2*1) = 126 combinaciones posibles.

Complejidad de tiempo

Mejor caso : O(ecuación aplicable) **Peor caso :** O(ecuación aplicable)

Promedio: O(ecuación aplicable)

JAVA

```
//Programa java que imprime todas las combinaciones con o sin repetición
// de tamaño r en un arreglo de tamaño n

public class Combinations {

    static void combinationUtil(int arr[], int data[], int start,
            int end, int index, int r) {
        if (index == r) {
            for (int j = 0; j < r; j++) {
                System.out.print(data[j] + " ");
            }
            System.out.println("");
            return;
        }

        for (int i = start; i <= end && end - i + 1 >= r - index; i++) {
            data[index] = arr[i];
            combinationUtil(arr, data, i + 1, end, index + 1, r);
        }
    }

    static void combination(int arr[], int n, int r) {
        int data[] = new int[r];
        combinationUtil(arr, data, 0, n - 1, 0, r);
    }

    static void CombinationRepetitionUtil(int chosen[], int arr[],
            int index, int r, int start, int end) {
        if (index == r) {
            for (int i = 0; i < r; i++) {
                System.out.printf("%d ", arr[chosen[i]]);
            }
            System.out.printf("\n");
            return;
        }
        for (int i = start; i < end; i++) {
            chosen[index] = i;
```

```
            CombinationRepetitionUtil(chosen, arr, index + 1,
                    r, i, end);
        }

    }

    static void CombinationRepetition(int arr[], int n, int r) {
        int chosen[] = new int[r + 1];
        CombinationRepetitionUtil(chosen, arr, 0, r, 0, n - 1);
    }

    public static void main(String[] args) {
        //Sin repetición
        int arr[] = {1, 2, 3, 4, 5};
        int r = 3;
        int n = arr.length;
        combination(arr, n, r);
        //Con repetición
        int arr2[] = {1, 2, 3, 4};
        int n2 = arr.length;
        int r2 = 2;
        CombinationRepetition(arr2, n2, r2);

    }
}
```

C++

```
#include <cstdlib>
#include <bits/stdc++.h>
using namespace std;

void combinationUtil(int arr[], int data[], int start, int end, int index, int
r) {
    if (index == r) {
        for (int j = 0; j < r; j++) {
            cout << data[j] << " ";
        }
        cout << endl;
        return;
    }
    for (int i = start; i <= end && end - i + 1 >= r - index; i++) {
        data[index] = arr[i];
        combinationUtil(arr, data, i + 1, end, index + 1, r);
    }
}

void combination(int arr[], int n, int r) {
    int data[r];
    combinationUtil(arr, data, 0, n - 1, 0, r);
}

void combinationRepetitionUtil(int chosen[], int arr[], int index, int r, int
start, int end) {
    if (index == r) {
```

```
        for (int i = 0; i < r; i++) {
            cout << arr[chosen[i]] << " ";
        }
        cout << endl;
        return;
    }
    for (int i = start; i < end; i++) {
        chosen[index] = i;
        combinationRepetitionUtil(chosen, arr, index + 1, r, i, end);
    }
    return;
}

void combinationRepetition(int arr[], int n, int r) {
    int chosen[r + 1];
    combinationRepetitionUtil(chosen, arr, 0, r, 0, n - 1);
}

int main(int argc, char const *argv[]) {
    cout << "Sin Repeticion" << endl;
    int arr[] = {1, 2, 3, 4, 5};
    int r = 3;
    int n = sizeof (arr) / sizeof (arr[0]);
    combination(arr, n, r);
    cout << "Con Repeticion" << endl;
    int arr2[] = {1, 2, 3, 4};
    int n2 = sizeof (arr2) / sizeof (arr2[0]);
    int r2 = 2;
    combinationRepetition(arr2, n2, r2);
    return 0;
}
```

PYTHON

```
from sys import stdout
wr = stdout.write

cont1 = 0
cont2 = 0

def combUtil(arr, data, start, end, index, r):
    global cont1
    if index == r:
        for j in range(r):
            wr(f'{data[j]} ')
        wr('\n')
        cont1 += 1
        return
    i = start
    while i <= end and end - i + 1 >= r - index:
        data[index] = arr[i]
        combUtil(arr, data, i+1, end, index+1, r)
        i += 1
```

```
def combinationRepetitionUtil(chosen, arr, index, r, start, end):
    global cont2
    if index == r:
        for i in range(r):
            wr(f'{arr[chosen[i]]} ')
        wr('\n')
        cont2 += 1
        return
    for i in range(start, end):
        chosen[index] = i
        combinationRepetitionUtil(chosen, arr, index+1, r, i, end)
    return

def printComb(arr, n, r):
    data = [0 for x in range(r)]
    combUtil(arr, data, 0, n-1, 0, r)

def combinationRepetition(arr, n, r):
    chosen = [0 for x in range(r+1)]
    combinationRepetitionUtil(chosen, arr, 0, r, 0, n-1)

arrint1 = [1, 2, 3, 4, 5]
r1 = 3
n1 = len(arrint1)
printComb(arrint1, n1, r1)
wr(f'Hay {str(cont1)} Combinaciones Sin Repetición\n')

arrint2 = [1, 2, 3, 4, 5]
r2 = 2
n2 = len(arrint2)
combinationRepetition(arrint2, n2, r2)
wr(f'Hay {str(cont2)} Combinaciones Con Repetición')
```

12.17) Combinatorias compuestas

Teniendo la teoría anterior clara, se puede modificar el algoritmo de tal forma que calcule las composiciones compuestas tal y como deseemos.

Complejidad de tiempo

Mejor caso : O(ecuación aplicable) **Peor caso :** O(ecuación aplicable)

Promedio: O(ecuación aplicable)

JAVA

```java
//Programa java que imprime todas
//las combinaciones que pueden componer un número dado

public class CombinationsCompose {
    //Función imprime todas las combinaciones de números 1, 2, ...MAX_POINT
    // que su suma resulte  n
    // i es usado en recursion para mantener revisión del indice
    // en arr[] donde el siguiente elemento sera añadido
    // Valor inicial de i debe ser pasado como 0

    static void printCompositions(int arr[], int n, int i) {
        int MAX_POINT = 3;
        if (n == 0) {
            printArray(arr, i);
        } else if (n > 0) {
            for (int k = 1; k <= MAX_POINT; k++) {
                arr[i] = k;
                printCompositions(arr, n - k, i + 1);
            }
        }
    }

    // Imprime array
    static void printArray(int arr[], int m) {
        for (int i = 0; i < m; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int n = 5;
        int size = 100;
        int[] arr = new int[size];
        System.out.println("Diferentes composiciones formadas por "
                + "1, 2 y 3 de " + n + " son ");
        printCompositions(arr, n, 0);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

void printArr(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
```

```
}

void printCompositions(int arr[], int n, int i) {
    int MAX_POINT = 3;
    if (n == 0) {
        printArr(arr, i);
    } else if (n > 0) {
        for (int k = 1; k <= MAX_POINT; k++) {
            arr[i] = k;
            printCompositions(arr, n - k, i + 1);
        }
    }
}

int main() {
    int n = 5;
    int len = 100;
    int arr[len];
    printCompositions(arr, n, 0);
    return 0;
}
```

12.18) Subsets de un Set (Conjuntos)

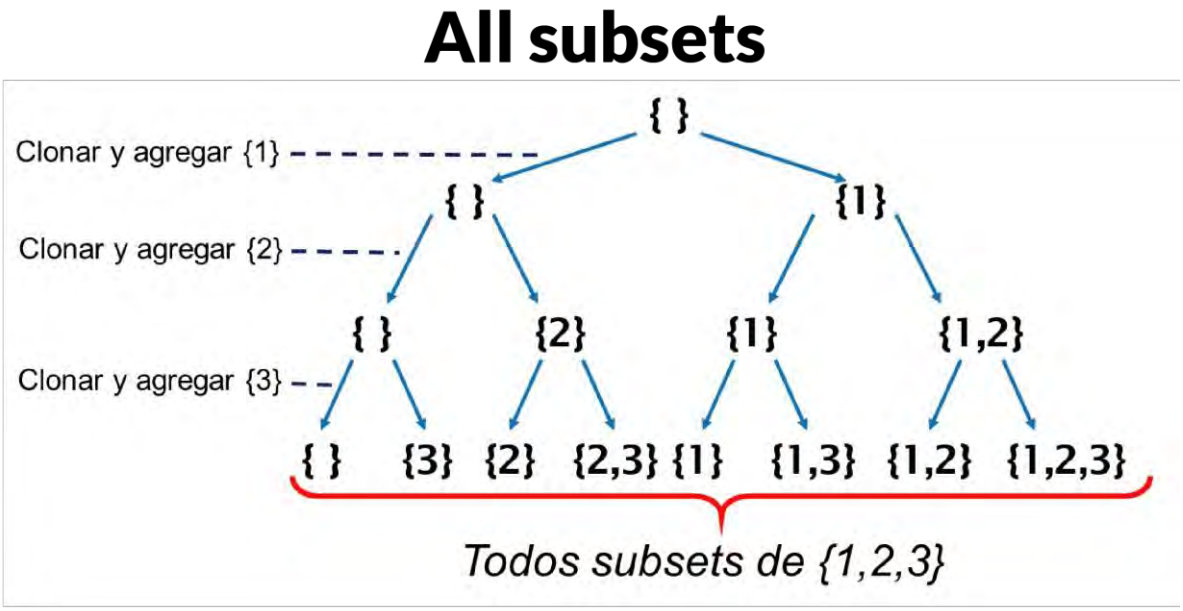


Ilustración 12-11 Búsqueda de todos los subconjuntos de un conjunto (Set)

Un conjunto (set) es una colección de elementos con características similares considerada en sí misma como un objeto. Los elementos de un conjunto, pueden ser las siguientes: personas, números, colores, letras, figuras, etc.

Se dice que un elemento (o miembro) pertenece al conjunto si está definido como incluido de algún modo dentro de él.

Complejidad de tiempo

Mejor caso : $O(n2^n)$ **Peor caso :** $O(n2^n)$ **Promedio:** $O(n2^n)$

JAVA

```java
//Programa java que imprime todos los subconjuntos de
//un conjunto

public class Allsets {

    static void printSubsets(char set[]) {
        int n = set.length;
        // Ejecuta un ciclo imprimiendo todos
        //  los subconjuntos 2^n uno por uno
        for (int i = 0; i < (1 << n); i++) {
            System.out.print("{ ");
            //Imprime el subconjunto actual
            for (int j = 0; j < n; j++) // (1<<j) es un número con jesimo bit 1
            {
                if ((i & (1 << j)) > 0) {
                    System.out.print(set[j] + " ");
                }
            }
            System.out.println("}");
        }
    }

    public static void main(String[] args) {
        char set[] = {'a', 'b', 'c'};
        printSubsets(set);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

void printSubset(char sets[], int n) {
    for (int i = 0; i < (1 << n); i++) {
        cout << "{";
        for (int j = 0; j < n; j++) {
```

```
            if ((i & (1 << j))) {
                cout << sets[j] << " ";
            }
        }
        cout << "}" << endl;
    }
}

int main() {
    char conjunto[] = {'a', 'b', 'c'};
    int n = sizeof conjunto / sizeof conjunto[0];
    printSubset(conjunto, n);
}
```

PYTHON

```
from sys import stdout
wr = stdout.write


def printSubsets(sset):
    n = len(sset)
    for i in range(1 << n):
        wr('{ ')
        for j in range(n):
            if (i & (1 << j)) > 0:
                wr(f'{sset[j]} ')
        wr('}\n')


conjunto = ['a', 'b', 'c']
printSubsets(conjunto)
```

12.19) Coeficientes binomiales

Coeficientes binomiales

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \frac{(n-1)!}{(k-1)! \cdot (n-k)!} + \frac{(n-1)!}{k! \cdot (n-k-1)!}$$
$$= \frac{(n-1)! \cdot k}{k \cdot (k-1)! \cdot (n-k)!} + \frac{(n-1)! \cdot (n-k)}{k! \cdot (n-k) \cdot (n-k-1)!}$$
$$= \frac{(n-1)! \cdot (k+n-k)}{k! \cdot (n-k)!}$$
$$= \frac{n!}{k! \cdot (n-k)!} = \binom{n}{k}.$$

Guía del programador competitivo

Ilustración 12-12 Ecuación completa de un coeficiente binomial

Los coeficientes binomiales, números combinatorios o combinaciones son números estudiados en combinatoria que corresponden al número de formas en que se puede extraer subconjuntos a partir de un conjunto dado.

Complejidad de tiempo

Mejor caso : O(n+k) **Peor caso :** O(n+k) **Promedio:** O(n+k)

JAVA

```
// Programa java que  calcula el valor
// de coeficientes binomiales

public class BinomialCoefficients {
    // Retorna el valor del coeficiente binomial
    // C(n, k)
    static int binomialCoeff(int n, int k) {
        // Casos base
        if (k == 0 || k == n) {
            return 1;
        }
        // Recursión
        return binomialCoeff(n - 1, k - 1)
```

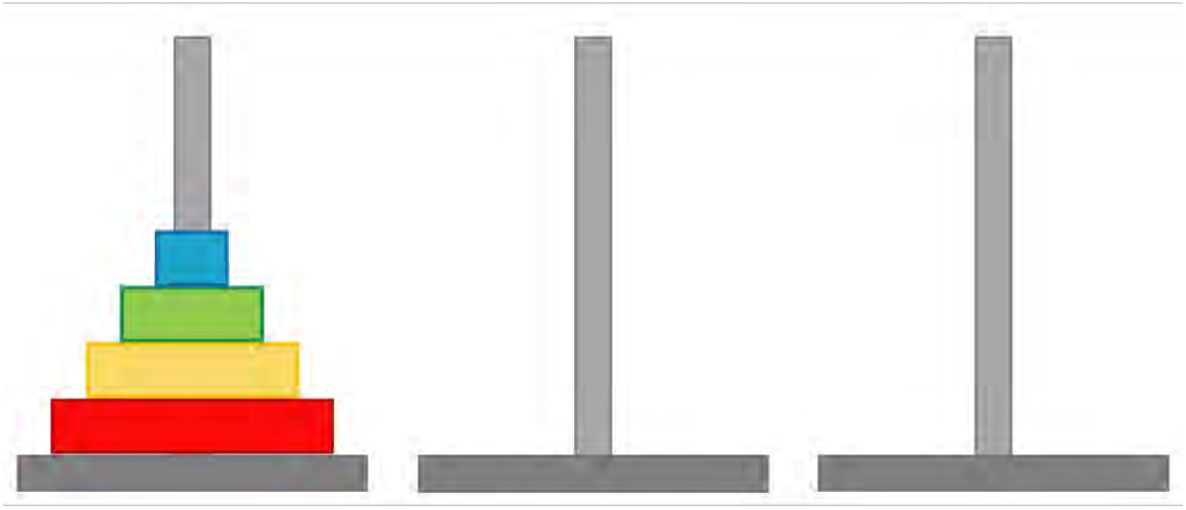
```
                + binomialCoeff(n - 1, k);
    }

    public static void main(String[] args) {
        int n = 5, k = 2;
        System.out.printf("Valor de C(%d, %d) is %d ",
                n, k, binomialCoeff(n, k));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int BinomialCoeff(int n, int k) {
    if (k == 0 || k == n) {
        return 1;
    }
    return BinomialCoeff(n - 1, k - 1) + BinomialCoeff(n - 1, k);
}

int main() {
    int n = 5, k = 2;
    printf("valor de C(%d, %d) is %d", n, k, BinomialCoeff(n, k));
}
```

12.20) Torres de Hanoi

Torre de Hanoi



Guía del programador competitivo

Ilustración 12-13 Juego "Las Torres de Hanoi"

Las Torres de Hanói son un rompecabezas o juego matemático inventado en 1883 por el matemático francés Édouard Lucas. Este juego de mesa individual consiste en un número de discos perforados de radio creciente que se apilan insertándose en uno de los tres postes fijados a un tablero. El objetivo del juego es trasladar la pila a otro de los postes siguiendo ciertas reglas, como que no se puede colocar un disco más grande encima de un disco más pequeño. El problema es muy conocido en la ciencia de la computación y aparece en muchos libros de texto como introducción a la teoría de algoritmos.

La fórmula para encontrar el número de movimientos necesarios para transferir n discos desde un poste a otro es: 2n – 1.

Complejidad de tiempo

Mejor caso : O(2n) **Peor caso :** O(2n) **Promedio:** O(2n)

JAVA

```
//Programa java que calcula los movimientos
```

```
// necesarios para completar las torres de Hanoi
// Sin importar el número de discos n

public class HanoiTowels {

    public static void main(String[] args) {
        //Número de discos
        int n;
        n = 8;
        hanoi(n, "Primera torre", "Segunda torre", "Tercera torre");
    }
    static int paso = 1;
    //Función recursiva de busqueda
    static void hanoi(int n, String from, String temp, String to) {
        if (n == 0) {
            return;
        }
        hanoi(n - 1, from, to, temp);
        System.out.println(paso + ": Mover disco " + n +
                " de " + from + " a " + to);
        paso++;
        hanoi(n - 1, temp, from, to);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int paso = 1;

void hanoi(int n, string from, string temp, string to) {
    if (n == 0) {
        return;
    }
    hanoi(n - 1, from, to, temp);
    cout << paso << " mover disco " << n << " de " << from << " a " << to <<
endl;
    paso++;
    hanoi(n - 1, temp, from, to);
}

int main() {
    int n = 8;
    hanoi(n, "Primer Torre", "Segunda Torre", "Tercer Torre");
    return 0;
}
```

12.21) AX+BY=N

axbyn

$$a\,x + b\,y = c$$

Si despejamos la incognita y:

$$2\,x - 3\,y = 4$$

$$2\,x - 3\,y - 2\,x = 4 - 2\,x$$

$$-3\,y = 4 - 2\,x$$

$$y = \frac{4 - 2\,x}{-3}$$

Guía del programador competitivo

Ilustración 12-14 Busqueda de incognitas en AX+BY=C

Dado a, b y n. Encuentre x y y que satisfaga ax+by=n, imprima cualquiera de los x y y que cumplan la ecuación.

Podemos verificar si alguna solución existe o no usando ecuaciones lineales de Diofanes, pero ahí se necesita encontrar para esta ecuación, entonces se puede simplemente iterar por todos los posibles valores de 0 a n sin exceder n para esta ecuación. Entonces resolviendo esta ecuación con lápiz y papel obtenemos y=(n-ax)/b y similarmente obtenemos el otro número con x=(n-by)/a, si ninguno de los valores satisface la ecuación, al final imprime “Sin solución”.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
//Programa que calcula la solución de
```

```java
// ax + by = n

public class AXplusBYequalsN {

    static void solution(int a, int b, int n) {
        //A traves de todos los posibles valores
        for (int i = 0; i * a <= n; i++) {
            //Verifica si se satisface la ecuación
            if ((n - (i * a)) % b == 0) {
                System.out.println("x = " + i
                        + ", y = "
                        + (n - (i * a)) / b);
                return;
            }
        }
        System.out.println("Sin solución");
    }

    public static void main(String[] args) {
        int a = 2, b = 3, n = 7;
        solution(a, b, n);
    }
}
```

C++

```cpp
#include <iostream>

using namespace std;

void solution(int a, int b, int n) {
    for (int i = 0; i * a <= n; i++) {
        if ((n - (i * a)) % b == 0) {
            cout << "x: " << i << ", y: " << ((n - (i * a)) / b) << endl;
            return;
        }
    }
    cout << "No hay solucion" << endl;
}

int main() {
    int a = 2, b = 3, n = 16;
    solution(a, b, n);
    return 0;
}
```

PYTHON

```python
from sys import stdout

def solution(a, b, n):
    i = 0
    while i * a <= n:
        if (n - (i * a)) % b == 0:
```

```
            stdout.write(f"x = {i} y = {(n-(i*a))/b}")
            return
        i += 1
    stdout.write("No tiene soluciÃ³n \n")

a, b, n = 2, 3, 7
solution(a, b, n)
```

12.22) A%X=B

Dados dos números a y b, encontrar todos los x que permitan a%x=b.

Existen tres casos:

- Si a es menor que b entonces no habrá respuesta.
- Si a es igual que b entonces todos los números más grandes que a, habrá infinitas soluciones.
- Si a es mayor que b, supone que x es una respuesta a nuestra ecuación. Entonces x divide (a-b) también desde a%x=b entonces b es menor que x

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
/* Programa java que encuentra x tal que
a % x es igual  b.*/
public class AmodXequalB {

    static void modularEquation(int a, int b) {
        // Si a es menos que b, entonces no hay solución
        if (a < b) {
            System.out.println("No solution possible ");
            return;
        }
        /*Si a es igual a b, entonces cada número
        más grande que a sera la solución, entonces
        es infinito*/
        if (a == b) {
            System.out.println("Infinite Solution possible ");
            return;
        }
```

```
        /*todo el número resultante debe ser mayor
        que b y (a-b) deben ser divisibles
        por número resultante
        variable count almacena el número de
        valores posibles*/
        int count = 0;
        int n = a - b;
        int y = (int) Math.sqrt(a - b);
        for (int i = 1; i <= y; ++i) {
            if (n % i == 0) {
                /*Revisando por ambos divisor y
                cociente cual divide (a-b) completamente
                y mayor que b*/
                if (n / i > b) {
                    count++;
                }
                if (i > b) {
                    count++;
                }
            }
        }
        /* Aqui y es añadido dos veces en la
        ultima iteracón entonces y deberia ser decrementado
        para obtener la solución correcta*/
        if (y * y == n && y > b) {
            count--;
        }
        System.out.println(count);
    }

    public static void main(String[] args) {
        int a = 21, b = 5;
        modularEquation(a, b);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

void modEquation(int a, int b) {
    if (a < b) {
        cout << "no hay solucion" << endl;
        return;
    }
    if (a == b) {
        cout << "infinitas soluciones" << endl;
        return;
    }
    int cont = 0;
    int n = a - b;
    int y = std::sqrt(a - b);
```

```
    for (int i = 1; i <= y; i++) {
        if (n % i == 0) {
            if (n / i > b) {
                cont++;
                cout << (n / i) << " ";
            }
            if (i > b) {
                cont++;
                cout << i << " ";
            }
        }
    }
    cout << endl;
    if (y * y == n && y > b) {
        cont--;
    }
    cout << cont << endl;
}

int main() {
    //a%x=b;
    int a = 21, b = 5;
    modEquation(a, b);
}
```

PYTHON

```
import math
from sys import stdout
wr = stdout.write


def modEquation(a, b):
    if a < b:
        wr(f'No hay Solucion')
        return
    if a == b:
        wr(f'Infinitas Soluciones')
    cont = 0
    n = a - b
    y = int(math.sqrt(a-b))
    for i in range(1, y + 1):
        if n % i == 0:
            if n // i > b:
                cont += 1
                wr(f'{n//i}\n')
            if i > b:
                cont += 1
                wr(f'{i}\n')
    if y * y == n and y > b:
        cont -= 1
    wr(f'{cont}')
```

```
a = 21
b = 5
modEquation(a, b)
```

12.23) E pow X

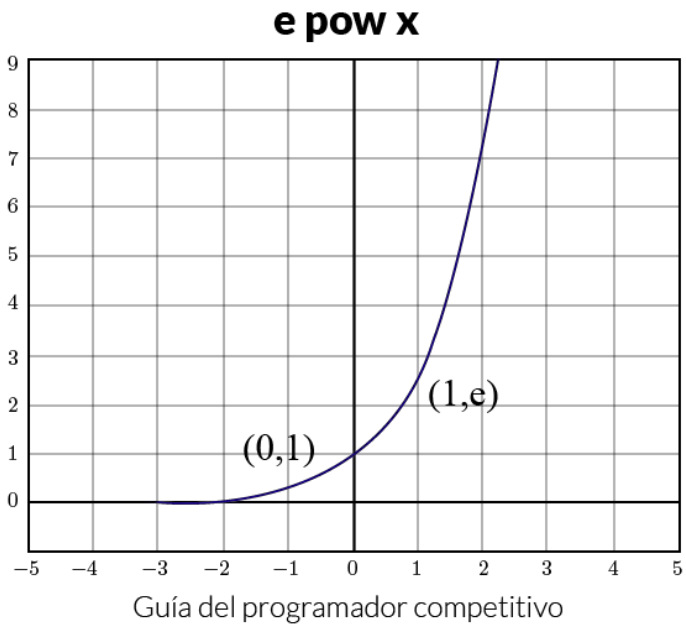


Ilustración 12-15 Función exponencial en un mapa cartesiano

El valor de la función exponencial e^x puede ser expresado usando la siguiente serie de Taylor:

- e^x = 1 + x/1! + x^2/2! + x^3/3! + ...

¿Cómo calcular eficientemente la suma de la serie de arriba? Puede ser escrita de la siguiente forma:

- e^x = 1 + (x/1) (1 + (x/2) (1 + (x/3) (.....)))

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa eficiente que calcula e elevado x

public class EpowX {
    // Función que retorna aproximado de e^x
    // Usando suma de los primeros n terminos
    // de la serie de Taylor
    static float exponential(int n, float x) {
        float sum = 1;
        for (int i = n - 1; i > 0; --i) {
            sum = 1 + x * sum / i;
        }
        return sum;
    }

    public static void main(String[] args) {
        int n = 10;
        float x = 1;
        System.out.println("e^x = " + exponential(n, x));
    }
}
```

C++

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

double exp(int n, double x) {
    double sum = 1;
    for (int i = n - 1; i > 0; i--) {
        sum = 1 + x * sum / i;
    }
    return sum;
}

int main() {
    ios_base::sync_with_stdio(false);
    cout.tie(NULL);
    int n = 10;
    double x = 2;
    cout << "exp: " << setprecision(25) << exp(n, x) << endl;
    cout << "exp: " << setprecision(25) << exp(n + 10, x) << endl;
    return 0;
}
```

PYTHON

```python
from sys import stdout
wr = stdout.write
```

```python
def exp(n, x):
    suma = 1
    for i in range(n-1, 0, -1):
        suma = 1 + x * suma / i
    return suma

n = 10
x = 2
wr(f'exp = {exp(n,x)}\n')
wr(f'exp = {exp(n+10,x)}')
```

12.24) **Factorial**

Factorial

$$1! = 1$$

$$2! = 2(1) = 2$$

$$3! = 3(2)(1) = 6$$

$$4! = 4(3)(2)(1) = 24$$

$$5! = 5(4)(3)(2)(1) = 120$$

Guía del programador competitivo

Ilustración 12-16 Composición de un número factorial

El factorial de un entero positivo n, el factorial de n o n factorial se define en principio como el producto de todos los números enteros positivos desde 1 (es decir, los números naturales) hasta n.

La operación de factorial aparece en muchas áreas de las matemáticas, particularmente en combinatoria y análisis matemático. De manera fundamental la factorial de n representa el número de formas distintas de ordenar n objetos distintos (elementos sin repetición).

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa java que encuentra el
// factorial de un número

public class Factorial {

    static int factorial(int n) {
        if (n == 0) {
            return 1;
        }
        return n * factorial(n - 1);
    }

    public static void main(String[] args) {
        int num = 5;
        System.out.println("Factorial de " + num + " es " + factorial(5));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
typedef long long int ll;
ll factorial(ll n){
    if(n==0){
        return 1;
    }
    return n*(factorial(n-1));
}
int main() {
    cout<<factorial(10)<<endl;
}
```

PYTHON

```
from sys import stdout
wr = stdout.write

def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

res = 5
wr(f'{factorial(res)}')
```

12.25) Factorial largo

Al igual que con otras secuencias numéricas, el cálculo de números grandes puede ser un problema debido a que pueden no caber dentro de una variable en específico, o ser muy lentas al calcularlo, el siguiente código presenta una forma optimizada de calcular factoriales grandes sin sacrificar tanto tiempo de ejecución, aunque para números demasiado grandes puede seguir siendo demasiado lento.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
// Programa java que calcula factoriales
// de números enormes

public class BigFactorial {

    static void factorial(int n) {
        int res[] = new int[500];
        res[0] = 1;
        int res_size = 1;
        // Aplicamos la formula normal de facotorial
        // n! = 1 * 2 * 3 * 4...*n
        for (int x = 2; x <= n; x++) {
            res_size = multiply(x, res, res_size);
        }

        System.out.println("Factorial del numero dado es ");
        for (int i = res_size - 1; i >= 0; i--) {
            System.out.print(res[i]);
        }
```

```
    }

    static int multiply(int x, int res[], int res_size) {
        int carry = 0; // inicializar carry
        // Uno por uno multiplicamos n con
        //los digitos individuales de res[]
        for (int i = 0; i < res_size; i++) {
            int prod = res[i] * x + carry;
            res[i] = prod % 10; //Almacenar ultimo digito de
            // 'prod' en res[]
            carry = prod / 10; // Poner el resto de carry
        }
        //Pone el carry en res e incrementa el tamaño del resultadolt size
        while (carry != 0) {
            res[res_size] = carry % 10;
            carry = carry / 10;
            res_size++;
        }
        return res_size;
    }

    public static void main(String args[]) {
        factorial(100);
    }
}
```

C++

```
#include <iostream>

using namespace std;

int multiply(int x, int res[], int res_size) {
    int carry = 0;
    for (int i = 0; i < res_size; i++) {
        int prod = res[i] * x + carry;
        res[i] = prod % 10;
        carry = prod / 10;
    }
    while (carry != 0) {
        res[res_size] = carry % 10;
        carry /= 10;
        res_size++;
    }
    return res_size;
}

void factorial(int n) {
    int res[10000];
    res[0] = 1;
    int res_size = 1;
    for (int i = 2; i <= n; i++) {
        res_size = multiply(i, res, res_size);
    }
    cout << "Factorial del numero dado es:" << endl;
```

```
    for (int i = res_size - 1; i >= 0; i--) {
        cout << res[i];
    }
    cout << endl;
}

int main() {
    factorial(100);
    return 0;
}
```

PYTHON

```
from sys import stdout
wr = stdout.write

def multilply(x, res, res_size):
    carry = 0
    for i in range(res_size):
        prod = res[i] * x + carry
        res[i] = prod % 10
        carry = prod // 10
    while carry != 0:
        res[res_size] = carry % 10
        carry //= 10
        res_size += 1
    return res_size

def factorial(n):
    res = [0 for x in range(500)]  # Maximo digitos de n
    res[0] = 1
    res_size = 1
    for i in range(2, n + 1):
        res_size = multilply(i, res, res_size)
    wr(f'Factorial de {n} es: \n')
    for i in range(res_size-1, -1, -1):
        wr(f'{res[i]}')

factorial(100)
```

12.26) Numero de dígitos de un factorial

Número de dígitos de un factorial

$$\int_1^8 log\ x\ dx <$$

$$< \log 2 + \log 3 + \cdots + log\ 8 <$$

$$< \int_1^9 log\ x\ dx$$

Guía del programador competitivo

Ilustración 12-17 Se puede averiguar el numero de digitos de un número usandoo logaritmos

Dado un entero n, encuentre el número de dígitos que aparecen en este factorial, donde factorial es definido como, factorial(n) =1*2*3*4……..*n y factorial (0) = 1……

Una solución ingenua puede ser calcular n! primero y luego calcular el número de dígitos presentes en el, sin embargo el valor de n! poder ser muy largo. Se vuelve algo complicado de almacenar esta variable (A menos que estés trabajando en Python).

Una mejor solución puede ser usar las útiles propiedades de los logaritmos para calcular la respuesta.

Sabemos que:

- log(a*b) = log(a) + log(b)

Por lo tanto:

- log (n!) = log (1*2*3……. * n) = log (1) + log (2) +…….. +log(n)

Ahora, observamos que el valor piso del logaritmo base 10 incrementado 1 de cualquier número da el número de dígitos presentes en ese número.

Entonces la salida puede ser: floor (log(n!)) + 1.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa java que encuentra número de
// digitos en un factorial

public class FactorialNumDigits {   // Retorna el número de digtos
    // en n!

    static int findDigits(int n) {   // Factorial existe solo para n>=0
        if (n < 0) {
            return 0;
        }
        // Caso base
        if (n <= 1) {
            return 1;
        }
        // si no itera atravez de n y calcula el valor
        double digits = 0;
        for (int i = 2; i <= n; i++) {
            digits += Math.log10(i);
        }
        return (int) (Math.floor(digits)) + 1;
    }

    public static void main(String[] args) {
        System.out.println(findDigits(1));
        System.out.println(findDigits(5));
        System.out.println(findDigits(10));
        System.out.println(findDigits(120));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;

int FindDigits(int n) {
    if (n < 0) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }
    double digits = 0;
    for (int i = 2; i <= n; i++) {
```

```
        digits += (std::log10(i));
    }
    return (int) (std::floor(digits)) + 1;
}

int main() {
    cout << FindDigits(1000) << endl;
    cout << FindDigits(5) << endl;
    cout << FindDigits(20) << endl;
    cout << FindDigits(100) << endl;
}
```

PYTHON

```
import math
from sys import stdout
wr = stdout.write

def findDigits(n):
    if n < 0:
        return 0
    if n <= 1:
        return 1
    digits = 0
    for i in range(2, n+1):
        digits += math.log10(i)
    return int(math.floor(digits)+1)

wr(f'{findDigits(1000000)}')
```

12.27) Numero de dígitos de un factorial optimizado

Si la solución anterior no es lo suficientemente rápida, podemos usar la fórmula de Kamenetsky para obtener la respuesta.

Se aproxima al número de dígitos en una factorial con:

- f(x) = log10(((n/e)^n) * sqrt(2*pi*n))

Además podemos fácilmente usar las propiedades de los logaritmos para obtener:

- f(x) = n* log10((n/ e)) + log10(2*pi*n)/2

Esta solución puede manejar número muy grandes de entrada, que pueden caber en un entero de 32 bits, e incluso más que esto.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
// Programa java que encuentra el número de digitos
// en un factorial

public class FactorialNumDigitsOP {

    public static double M_E = 2.71828182845904523536;
    public static double M_PI = 3.141592654;

    /* Funcion que retorna el número de digitos presente en
    n! desde que el resultado sea muy largo*/
    static long findDigits(int n) {

        if (n < 0) {
            return 0;
        }

        // caso base
        if (n <= 1) {
            return 1;
        }
        // Usamos la formula de Kamenestsky para
        // calcular el número de digitos
        double x = (n * Math.log10(n / M_E)
                + Math.log10(2 * M_PI * n)
                / 2.0);

        return (long) Math.floor(x) + 1;
    }

    public static void main(String[] args) {
        System.out.println(findDigits(1));
        System.out.println(findDigits(50000000));
        System.out.println(findDigits(1000000000));
        System.out.println(findDigits(120));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
const double PI = 3.14159265358979323846;
```

```cpp
const double E = 2.71828182845904523536;

long FindDigitsKame(int n) {
    if (n < 0) {
        return 0;
    }
    if (n <= 1) {
        return 1;
    }
    double x = (n * log10(n / E)) + std::log10(2 * PI * n) / 2.0;
    return (long) (std::floor(x)) + 1;
}

int main() {
    cout << FindDigitsKame(1000000) << endl;
}
```

PYTHON

```python
import math
from sys import stdout
wr = stdout.write

def findDigits(n):
    if n < 0:
        return 0
    if n <= 1:
        return 1
    x = (n * math.log10(n / math.e) + math.log10(2 * math.pi * n) / 2)
    return int(math.floor(x) + 1)

wr(f'{findDigits(10000)}')
```

12.28) Teorema de Euclides-Euler

Euler

Demuestra el recíproco del teorema de Euclides sobre números perfectos.
Desde entonces se conoce como Teorema de Euclides - Euler.

Si N es un número perfecto y par,
entonces $N = 2^{k-1}(2^k - 1)$,
donde 2^{k-1} es un número primo.

Guía del programador competitivo

Ilustración 12-18 Ejemplo del teorema de Euclides-Euler

De acuerdo con el teorema de Euclides-Euler, un número perfecto el cual es par, puede ser representado de la forma (2^n - 1)*(2^n / 2))) donde n es un número primo y 2^n – 1 es un número primo de Mersenne. Este es un producto de la potencia de 2 con un primo Mersenne, este teorema establece una conexión entre un número primo de Mersenne y un número par primo perfecto. Un número perfecto es aquél que es igual a la suma de sus divisores, exceptuando él mismo.

Algunos ejemplos de números perfectos los cuales satisfacen este teorema son:

- 6, 28, 496, 8128, 33550336, 8589869056, 137438691328

Explicación:

- 6 es un número perfecto par.

Entonces puede ser escrito de la forma

- (2^2 - 1) * (2^ (2 - 1)) = 6

Donde n =2 es un número primo y 2^n -1=3 es un número primo de Mersenne

Toma cada número primo y forma un primo de Mersenne con él. El primo de Mersenne = 2^n – 1 donde n es primo. Ahora se forma el número (2^n – 1)*(2^(n – 1)) y verificamos si es par y perfecto.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
//Programa que verifica el teorema de Euclides Euler

import java.util.ArrayList;

public class EuclidEulerTheorem {
    static ArrayList<Long> power2 = new ArrayList<Long>();

    public static void main(String[] args) {
        //Almacenando potencias de 2 para acceder
        // en tiempo O(1)
        for (int i = 0; i < 62; i++) {
            power2.add(0L);
        }

        for (int i = 0; i <= 60; i++) {
            power2.set(i, (1L << i));
        }
        System.out.println("Generando los primeros números que satisfacen "
                + "el teorema de Euclid Euler\n");
        for (long i = 2; i <= 25; i++) {
            long no = ((power2.get((int) i) - 1L) * (power2.get((int) (i -
1))));
            if (isperfect(no) && (no % 2 == 0)) {
                System.out.println("(2^" + i + " - 1) * (2^(" + i + " - 1)) = "
                        + no + "\n");
            }
        }
    }

    static boolean isperfect(long n) // Números perfectos
    {
        /* Verifica si n es suma perfecta de dividores
        excepto por el número en si mismo*/
        long s = -n;
        for (long i = 1; i * i <= n; i++) {
            // es i un divisor de n
            if (n % i == 0) {
                long factor1 = i, factor2 = n / i;
                s += factor1 + factor2;
                // aqui i*i == n
                if (factor1 == factor2) {
```

```
                    s -= i;
                }
            }
        }
        return (n == s);
    }

    boolean isprime(long n) {
        // Verifica cual número es primo o no
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return false;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
typedef long long int ll;
ll power2[1000];

bool isPerfect(ll n) {
    ll s = -n;
    for (ll i = 1; i * i <= n; i++) {
        if (n % i == 0) {
            ll factor1 = i, factor2 = n / i;
            s += factor1 + factor2;
            if (factor1 == factor2) {
                s -= i;
            }
        }
    }
    return (n == s);
}

bool isPrime(ll n) {
    for (ll i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return false;
}

int main() {
    memset(power2, 0L, sizeof power2);
    for (int i = 0; i < 62; i++) {
        power2[i] = 0L;
    }
```

```
    for (int i = 0; i <= 60; i++) {
        power2[i] = (1L << i);
    }
    cout << "Generando los primeros numeros que satiscafacen el teorema" <<
endl;
    for (ll i = 2; i <= 25; i++) {
        ll no = ((power2[(int) i] - 1L) * power2[(int) (i - 1)]);
        if (isPerfect(no) && (no % 2 == 0)) {
            cout << "(2^" << i << " -1 * (2^(" << i << " -1)) = " << no << endl;
        }
    }
}
```

12.29) Algoritmo Euclidiano

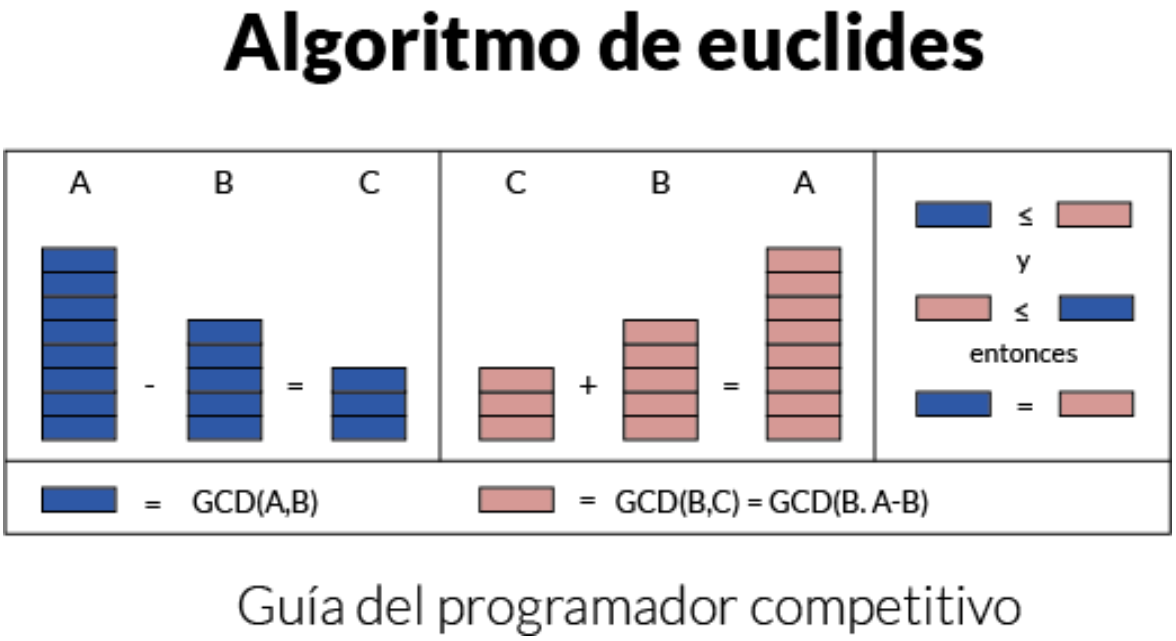


Ilustración 12-19 Ejemplo del algoritmo de Euclides

El MCD de dos números es el número más grande que divide ambos. Una forma simple de encontrar este número es factorizar ambos números y multiplicar los factores comunes.

El algoritmo se basa en lo siguiente:

- Si resta el número más pequeño del más grande, MCD (GCD) no cambia, entonces si sigue restando repetidamente el más grande dos, termina con MCD.
- Ahora en vez de restar, si divide el número más pequeño, el algoritmo termina cuando se encuentra residuo 0.

Complejidad de tiempo

Mejor caso : O(Log min(a, b)) **Peor caso :** O(Log min(a, b)) **Promedio:** O(Log min(a, b))

JAVA

```java
//Programa java que demuestra el algoritmo de Euclides

public class EuclideanAlgorithm {
    // Algoritmo de euclides extendido
    public static int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }

    public static void main(String[] args) {
        int a = 10, b = 15, g;
        g = gcd(a, b);
        System.out.println("GCD(" + a + " , " + b + ") = " + g);
        a = 35;
        b = 10;
        g = gcd(a, b);
        System.out.println("GCD(" + a + " , " + b + ") = " + g);
        a = 31;
        b = 2;
        g = gcd(a, b);
        System.out.println("GCD(" + a + " , " + b + ") = " + g);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

int GCD(int a, int b) {
    if (a == 0) {
        return b;
    }
    return GCD(b % a, a);
}
```

```
int main() {
    int a = 10;
    int b = 15;
    cout << "GCD" << "(" << a << "," << b << ")= " << GCD(a, b) << endl;
}
```

12.30) Euler Totient

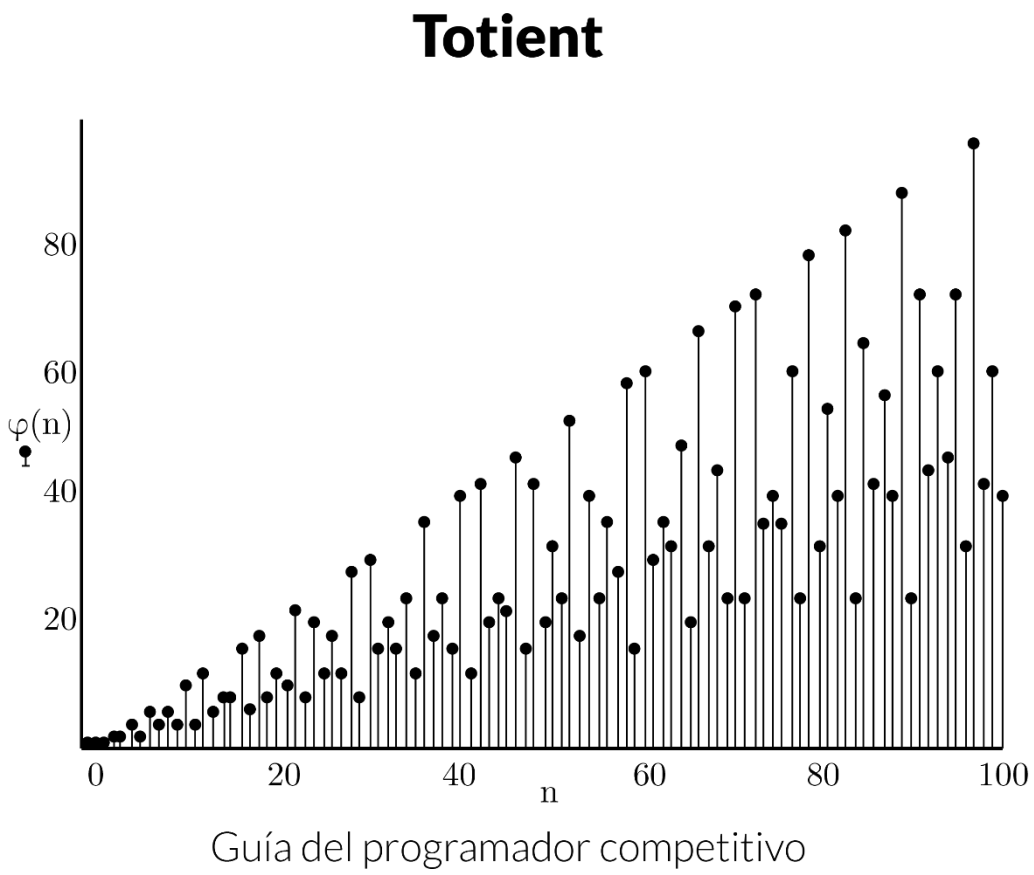


Ilustración 12-20 Grafica de los numeros resultado de la función totient

La función totient de Euler para una entrada n es un conteo de números desde 1 hasta n que son primos relativos con n, por ejemplo, los números cuyo GCD (Máximo común divisor)

con n es 1. Se llama primos relativos (coprimos) a cualquier par de números enteros que no tienen ningún divisor en común, excepto el 1.

El criptosistema RSA es basado en este teorema.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa simple que calcula el valor
// de la función  totient Euler

public class EulerTotient {
    // Función que retorna gcd de a y b
    static int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }
        // Función que evalua función totient de euler
    static int phi(int n) {
        int result = 1;
        for (int i = 2; i < n; i++) {
            if (gcd(i, n) == 1) {
                result++;
            }
        }
        return result;
    }

    public static void main(String[] args) {
        int n;
        for (n = 1; n <= 10; n++) {
            System.out.println("phi(" + n + ") = " + phi(n));
        }
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int gcd(int a,int b){
    if(a==0){
        return b;
    }
    return gcd(b%a,a);
}
```

```
int phi(int n){
    int result=1;
    for(int i=2;i<n;i++ ){
        if(gcd(i,n)==1){
            result++;
        }
    }
    return result;
}
int main() {
    for(int i=1;i<=10;i++){
        printf("el phi (%d) = %d\n",i,phi(i));
    }
}
```

PYTHON

```
from sys import stdout
wr = stdout.write

def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)

def phi(n):
    result = 1
    for i in range(2, n):
        if gcd(i, n) == 1:
            result += 1
    return result

for i in range(1, 11):
    wr(f'Phi ({i}) = {phi(i)}\n')
```

12.31) El pequeño teorema de Fermat

La función de Euler y el teorema de Fermat

La generalización del pequeño teorema de Fermat es:

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

Donde a elevada a la función de Euler $\varphi(n)$ es consiguiente con 1 módulo n.

Guía del programador competitivo

Ilustración 12-21 Explicación del pequeño teorema de Fermat

El pequeño teorema de Fermat dice que si p es un número primo, entonces para cualquier entero a, el número de p-a es un entero múltiplo de p.

Aquí p es un número primo:

- a^p ≡ a (mod p).

Caso especial: si a no es divisible por p, el pequeño teorema de Fermat es equivalente a la sentencia que un p-1-1 es un entero múltiplo de p.

- a^p-1 ≡ 1 (mod p)

O

- a^p-1 % p = 1 Aquí a no es divisible por p.

Usos del pequeño teorema de Fermat

Si sabemos que m es primo, entones se puede también usar el pequeño teorema de Fermat para buscar la inversa:

- a^m-1 ≡ 1 (mod m)

Si nosotros multiplicamos ambos saldos con a-1, obtenemos:

- a^-1 ≡ a m-2 (mod m)

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
/*Programa de Java para encontrar modular.
 inverso de un módulo bajo m
 utilizando el pequeño teorema de Fermat.
 Este programa funciona solo si m es primo.*/
public class FermatLittleTheorem {

    static int __gcd(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return __gcd(b, a % b);
        }
    }
    // Computa x^y bajo modulo m

    static int power(int x, int y, int m) {
        if (y == 0) {
            return 1;
        }
        int p = power(x, y / 2, m) % m;
        p = (p * p) % m;
        return (y % 2 == 0) ? p : (x * p) % m;
    }
    // Función para encontrar modular
    // inverso bajo un modulo m
    // Asumimos m es primo

    static void modInverse(int a, int m) {
        if (__gcd(a, m) != 1) {
            System.out.print("No existe inverso");
        } else {
            // Si a y m son primos relativos, entonces
            // modulo inverso es a^(m-2) mod m
            System.out.print("Multiplicación modular inversa es "
                    + power(a, m - 2, m));
        }
    }
```

```
    public static void main(String[] args) {
        int a = 3, m = 11;
        modInverse(a, m);
    }
}
```

12.32) Producto de fracciones

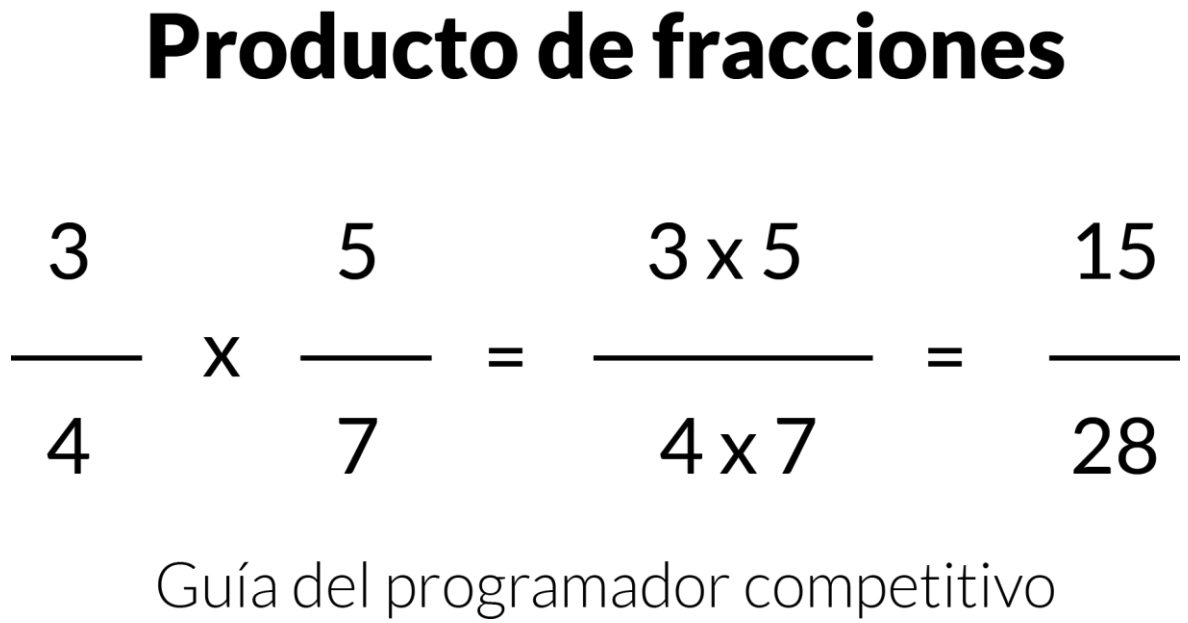


Ilustración 12-22 Ejemplo del producto de fracciones

Dados el numerador y el denominador de N fracciones, la tarea es encontrar el producto de N fracciones e imprimir la respuesta en forma reducida.

La idea es encontrar el producto de numeradores en una variable, como new_num, ahora encontrar el producto de los denominadores en otra variable como new_den.

Ahora para encontrar la respuesta en forma reducida, encuentre el GCD de new_num y new_den y dividir el new_num y new_den por el GCD calculado.

La solución causa desbordamiento para números grandes, podemos evadir esto si encontramos los factores primos de todos los numeradores y denominadores, una vez hayamos encontrado los factores, podemos cancelar los factores primos comunes.

Cuando se solicita representar la respuesta de la forma {P \veces {Q} ^ {-1}}. Primero convierta el numerador y el denominador en forma reducible de P/Q. luego busque el multiplicativo inverso de Q con respecto a un número primo m (Generalmente 10^9 + 7) el cual es dado como pregunta, luego de encontrar el multiplicativo inverso de Q, multiplicarlo con P y tomar el modulo con el número primo m, el cual nos da nuestra salida requerida.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Programa java que encuentra el producto
// de N fracciones en forma reducida

public class Fractionsproduct {
    // Función que retorna el gcd de a y b
    static int gcd(int a, int b) {
        if (a == 0) {
            return b;
        }
        return gcd(b % a, a);
    }

    static void productReduce(int n, int num[],
            int den[]) {
        int new_num = 1, new_den = 1;
        //Encontrando el producto de todos los N
        // numeradores y denominadores
        for (int i = 0; i < n; i++) {
            new_num *= num[i];
            new_den *= den[i];
        }
        // Encontrando GCD de nuevo numerados y denominador
        int GCD = gcd(new_num, new_den);
        // Convirtiendo en forma reducids
        new_num /= GCD;
        new_den /= GCD;
        System.out.println(new_num + "/" + new_den);
    }

    public static void main(String[] args) {
        int n = 3;
        int num[] = {1, 2, 5};
```

```
        int den[] = {2, 1, 6};
        productReduce(n, num, den);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int GCD(int a,int b){
    if(a==0){
        return b;
    }
    return GCD(b%a,a);
}
void productReduce(int n,int num[],int den[]){
    int new_num=1,new_den=1;
    for(int i=0;i<n;i++){
        new_num*=num[i];
        new_den*=den[i];
    }
    int gcd = GCD(new_num,new_den);
    new_num/=gcd;
    new_den/=gcd;
    cout<<new_num<<"/"<<new_den<<endl;
}
int main() {
    int n=3;
    int num[]={1,2,5};
    int den[]={2,1,6};
    productReduce(n,num,den);
}
```

12.33) Josephus

Josephus

Guía del programador competitivo

Ilustración 12-23 Flavio Josefo y sus compañeros soldados en el circulo

En las ciencias de la computación y las matemáticas, el problema Josephus (O permutación Josephus) es un problema teorético el cual su enunciado es el siguiente.

Hay n personas sentadas en círculo, esperando a ser ejecutadas. El conteo empieza en el mismo punto en el círculo y procede alrededor del círculo en dirección horaria, en cada paso un cierto número de personas son saltadas y la siguiente persona es ejecutada. La eliminación se realiza alrededor del circulo (el comienza a ser más pequeño y pequeño mientras la gente es ejecutada y removida), hasta que solo quede una persona, a quien se le dará la libertad. Dado el total de personas n y un número k que indica que k-1 personas serán saltadas y la kesima persona es asesinada en el círculo, la tarea es escoger el lugar en el círculo inicial de tal forma que sea la posición de la última persona que sobreviva.

Por ejemplo, si n =5 y k = 2, entonces la posición segura es 3. (1,2,3,4,5 y la posición inicial es 5). Primeramente la persona en la posición 2 es asesinada, luego la persona en la posición 4 es asesinada, luego la persona en la posición 1 es asesinada, finalmente la persona en la

posición 5 es asesinada dejando a la persona en la 4 posición viva y con capacidad de disfrutar su libertad.

Si n=7 y k=3, entonces la posición segura es 4, las personas en las posiciones 3,6,2,7,5,1 son asesinadas en ese orden y la 4 sobrevive.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Implementación java de dos Algoritmos que realizan la
// busqueda josephus sabiendo número personas y tamaño
// de salto

public class Josephus {

    public static void main(String[] args) {
        System.out.println(josephus(6, 2));
        System.out.println(josephusModular(6, 2));
    }
    //Busqueda por algoritmo voraz
    static int josephus(int n, int k) {
        int d[] = new int[n + 1];
        d[0] = -1;
        for (int i = 1; i <= n; i++) {
            d[i] = i;
        }
        int i = n;
        int a = 0;
        while (i != 1) {
            i--;
            a = next(a, k, d);
            d[a] = -1;
            a++;
        }
        for (i = 0; d[i] == -1; i++);
        return d[i];
    }
    //Calcule la posición siguiente a caer
    static int next(int a, int k, int[] d) {
        int j = a - 1;
        for (int i = 0; i < k;) {
            j = (j + 1) % d.length;
            if (d[j] != -1) {
                i++;
            }
        }
        return j;
    }
```

```
    //busqueda por matematica modular
    static int josephusModular(int n, int k) {
        int f = 0;
        for (int i = 1; i <= n; i++) {
            f = (f + k) % i;

        }
        return f + 1;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int next(int a, int k, int d[], int n){
    int j = a - 1;
    for(int i = 0; i < k;){
        j = (j + 1) % n;
        if(d[j] != -1){
            i++;
        }
    }
    cout<<"next: "<<j<<endl;
    return j;
}
int josephus(int n, int k){
    int d[n + 1];
    d[0] = -1;
    for(int i = 1; i <= n; i++){
        d[i] = i;
    }
    int indice = n;
    int a = 0;
    while(indice != 1){
        indice--;
        a = next(a, k, d, n + 1);
        d[a] = -1;
        a++;
    }
    for(indice = 0; d[indice] == -1; indice++);
        return d[indice];
}
int main(){
    cout<<josephus(10, 3)<<endl;
    return 0;
}
```

PYTHON

```
from sys import stdout
wr = stdout.write
```

```
def next_(a, k, d):
    j = a - 1
    i = 0
    while i < k:
        j = (j+1) % len(d)
        if d[j] != -1:
            i += 1
    wr(f'Next = {j}\n')
    return j

def josephus(n, k):
    d = [-1 for x in range(n+1)]
    d[0] = -1
    for i in range(1, n+1):
        d[i] = i
    i = n
    a = 0
    while i != 1:
        i -= 1
        a = next_(a, k, d)
        d[a] = -1
        a += 1
    i = 0
    while d[i] == -1:
        i += 1
    return d[i]

def josephusModular(n, k):
    f = 0
    for i in range(1, n+1):
        f = (f+k) % i
    return f + 1

wr(f'{josephusModular(6,2)}\n')
wr(f'{josephus(6,2)}')
```

12.34) Números de la forma Cardinal/Ordinal

Ordinal cardinal

Número Abreviado #	Letra	Español	#	Letra	Español
1 st	First	Primero	21st	Twenty - first	Vigésimo primero
2nd	Second	Segundo	22nd	Twenty - second	Vigésimo segundo
3rd	Third	Tercero	23rd	Twenty - third	Vigésimo tercer
4th	Fourth	Cuarto	30th	Thirtieth	Trigésimo
5th	Fifth	Quinto	31th	Thirty - first	Trigésimo primero
6th	Sixth	Sexto	4oth	Fortieth	Cuadragésimo
7th	Seventh	Séptimo	41th	Forty - first	Cuadragésimo primero
8th	Eight	Octavo	50th	Fiftieth	Quincuagésimo
9th	Nineth	Noveno	51th	Fifty - first	Quincuagésimo primero
10th	Tenth	Décimo	60th	Sixtieth	Sexagésimo
11th	Eleventh	Undécimo	61th	Sixty - first	Sexagésimo primero
12th	Twelfth	Duodécimo	70th	Seventieth	Septuagésimo
13th	Thirteenth	Décimo tercero	71th	Seventy - first	Septuagésimo primero
14th	Fourteenth	Décimo cuarto	80th	Eightieth	Octogésimo
15th	Fifteenth	Décimo quinto	81th	Eighty - first	Octogésimo primero
16th	Sixteenth	Décimo sexto	90th	Ninetieth	Nonagésimo
17th	Seventeenth	Décimo séptimo	91th	Ninety - first	Nonagésimo primero
18th	Eighteenth	Décimo octavo	100th	Hundredth	Centésimo
19th	Nineteenth	Décimo noveno	101th	Hundred and first	Centésimo primero
20th	Twentieth	Vigésimo	200th	Two hundredth	Dos centésimas

Ilustración 12-24 Números cardinales en ingles

Cuando usamos los números naturales para contar los elementos de un determinado conjunto los llamamos números cardinales.

En muchas ocasiones es necesario dar un orden a las cosas: las posiciones finales de una carrera o los pisos de un edificio son algunos ejemplos. Cuando se usan los números naturales para este ordenar los llamamos ordinales.

Para representar los números ordinales se usan los números naturales acompañados por una pequeña letra así: 1°a, 2°o etc. Cuando acompañamos el número por la letra a es para femenino, y con la letra o es para masculino. Así, si queremos decir que Anita es la número uno de la clase decimos que es la primera: 1°a; y si queremos decir que Pablo ocupó el lugar número uno en la carrera decimos que fue el primero: 1°o

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Programa java que convierte números ordinales
// en números cardinales en ingles

public class CardinalNumbers {

    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i + cardinalNumber(i));
        }
    }

    static String cardinalNumber(int n) {
        if (n % 10 == 1 && n % 100 != 11) {
            return "st";
        }
        if (n % 10 == 2 && n % 100 != 12) {
            return "nd";
        }
        if (n % 10 == 3 && n % 100 != 13) {
            return "rd";
        }
        return "th";
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
string cardinalNumber(int n){
    if(n%10==1 && n%100!=11){
        return "st";
    }if(n%10==2 && n%100!=12){
        return "nt";
    }if(n%10==3 && n%100!=13){
        return "st";
    }
    return "th";
}
int main(){
    for(int i=1;i<=10;i++){
        cout<<i<<" : "<<cardinalNumber(i)<<endl;
    }
}
```

12.35) ## Números Romanos

Números romanos

1 = I	20 = XX	300 = CCC	4000 = $M\bar{V}$
2 = II	30 = XXX	400 = CD	5000 = $\bar{V}$
3 = III	40 = XL	500 = D	6000 = $\overline{VI}$
4 = IV	50 = L	600 = DC	7000 = $\overline{VII}$
5 = V	60 = LX	700 = DCC	8000 = $\overline{VIII}$
6 = VI	70 = LXX	800 = DCCC	9000 = $\overline{IX}$
7 = VII	80 = LXXX	900 = CM	10000 = $\bar{X}$
8 = VIII	90 = LC	1000 = M	
9 = IX	100 = C	2000 = MM	
10 = X	200 = CC	3000 = MMM	

Guía del programador competitivo

Ilustración 12-25 Principales componentes de los números Romanos

La numeración romana es un sistema de numeración que se desarrolló en la Antigua Roma y se utilizó en todo el Imperio romano, manteniéndose con posterioridad a su desaparición y todavía utilizado en algunos ámbitos.

Este sistema emplea algunas letras mayúsculas como símbolos para representar ciertos valores. Los números se escriben como combinaciones de letras. Por ejemplo, el año 2019 se escribe como MMXIX, donde cada M representa 1000 unidades, la X representa 10 unidades más y IX representa 9 unidades más (al ser X, que representa el 10, precedido por I, que representa el 1).

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Implementación java que convierte números decimales
// a números romanos y viceversa usando matematica modular

public class RomanNumbers {
    // Arrays constantes de letras romanas
    static String unit[] = {"", "I", "II", "III", "IV", "V", "VI", "VII",
"VIII", "IX"};
    static String ten[] = {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX",
"XC"};
    static String hnd[] = {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC",
"CM"};

    public static void main(String[] args) {
        System.out.println(toRoman(122));
        System.out.println(toDecimal("CXXII"));
    }
    //Función que convierte a romano

    static String toRoman(int n) {
        //Los números romanos no tienen letras
        //más alla de los miles
        int a;
        StringBuilder sb = new StringBuilder();
        if (n >= 1000) {
            a = n / 1000;
            for (int i = 1; i <= a; i++) {
                sb.append("M");
            }
            n %= 1000;
        }
        //Letras de las centenas
        sb.append(hnd[n / 100]);
        n = n % 100;
        //Letras de las decenas
        sb.append(ten[n / 10]);
        //Letras de las unidades
        sb.append(unit[n % 10]);
        return sb.toString();
    }
    //Devuelve el valor de cada letra

    static int valor(char ch) {
        switch (ch) {
            case 'I':
                return 1;
            case 'V':
                return 5;
            case 'X':
                return 10;
            case 'L':
                return 50;
            case 'C':
                return 100;
            case 'D':
```

```
                return 500;
            case 'M':
                return 1000;
        }
        return 0;
    }
    //Función que convierte a decimal

    static int toDecimal(String num) {
        int sum = 0;
        int last = 0;
        int next;
        //Toma cada letra y verifica si esta antes o despues
        // suma si esta despues, resta si esta antes
        for (int i = num.length() - 1; i >= 0; i--) {
            next = valor(num.charAt(i));
            if (last <= next) {
                sum += next;
            } else {
                sum -= next;
            }
            last = next;
        }
        return sum;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
string unit[]={"","I","II","III","IV","V","VI","VII","VIII","IX"};
string ten[]={"","X","XX","XXX","XL","L","LX","LXX","LXXX","XC"};
string hnd[]={"","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"};
int valor(char ch){
    switch(ch){
    case 'I':
        return 1;
    case 'V':
        return 5;
    case 'X':
        return 10;
    case 'L':
        return 50;
    case 'C':
        return 100;
    case 'D':
        return 500;
    case 'M':
        return 1000;
    }
    return 0;
}
```

```
string toRoman(int n){
    int a;
    string sb="";
    if(n>=1000){
        a = n/1000;
        for(int i=1;i<=a;i++){
            sb+="M";
        }
        n%=1000;
    }
    sb+=(hnd[n/100]);
    n%=100;
    sb+=(ten[n/10]);
    sb+=(unit[n%10]);
    return sb;
}
int toDecimal(string num){
    int sum=0;
    int last=0;
    int next;
    for(int i=num.size()-1;i>=0;i--){
        next=valor(num[i]);
        if(last<=next){
            sum+=next;
        }else{
            sum-=next;
        }
        last=next;
    }
    return sum;
}
int main(){
    cout<<toRoman(122)<<endl;
    cout<<toRoman(1)<<endl;
    cout<<toRoman(2323)<<endl;
    cout<<toRoman(343)<<endl;

    cout<<toDecimal("CXXII")<<endl;
    cout<<toDecimal("I")<<endl;
    cout<<toDecimal("MMCCCXXIII")<<endl;
    cout<<toDecimal("CCCXLIII")<<endl;
}
```

12.36) Teorema de Hardy-Ramanujan

Hardy - Ramanujan

1.729 es el número más pequeño que se puede representar de dos maneras diferentes como la suma de dos cubos.

$$1.729 = 1^3 + 12^3$$
$$= 9^3 + 10^3$$

También es por cierto el producto de 3 números primos:

$$1.729 = 7 * 13 * 19$$

El número similar más grande conocido es:

$$885623890831 = 7511^3 + 7730^3$$
$$= 8759^3 + 5978^3$$
$$= 3943 * 14737 * 15241$$

Ilustración 12-26 Teorema de Hardy-Ramanujan

El teorema de Hardy Ramanujan propone que el número de factores de n debe ser aproximadamente log(log(n)) para la mayoría de números naturales n.

- 5192 tiene 2 factores primos distintos y log(log(5192)) = 2.1615
- 51242183 tiene 3 factores primos distintos y log(log(51242183)) = 2.8765

Este teorema es principalmente usado en algoritmos de aproximación y es prueba líder para conceptos más grandes de teoría de la probabilidad

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```
// Programa java que cuenta todos los
// factores primos

public class HardyRamanujanTheorem {
    // Una función que cuenta factores primos de
    //un número n
    static int exactPrimeFactorCount(int n) {
        int count = 0;
```

```
        if (n % 2 == 0) {
            count++;
            while (n % 2 == 0) {
                n = n / 2;
            }
        }
        // n debe ser impar en este punto, asi
        //podemos saltar un elemento (i=i+2)
        for (int i = 3; i <= Math.sqrt(n); i = i + 2) {
            if (n % i == 0) {
                count++;
                while (n % i == 0) {
                    n = n / i;
                }
            }
        }
        // Esta condición es para controlar el caso
        // cuando n es un factor primo más grande que 2
        if (n > 2) {
            count++;
        }
        return count;
    }

    public static void main(String[] args) {
        int n = 51242183;
        System.out.println("El número de diferentes "
                + " factores primos es "
                + exactPrimeFactorCount(n));
        System.out.println("El valor de (log(n))"
                + " es " + Math.log(Math.log(n)));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int exactPrimeFactorCount(int n){
    int cont=0;
    if(n%2==0){
        cont++;
        while(n%2==0){
            n/=2;
        }
    }
    for(int i=3;i<=(sqrt(n));i+=2){
        if(n%i==0){
            cont++;
            while(n%i==0){
                n/=i;
            }
        }
```

```
    }
    if(n>2){
        cont++;
    }
    return cont;
}
int main(){
    int n=51242183;
    cout<<"El numero de diferentes factores primos es
"<<exactPrimeFactorCount(n)<<endl;
    cout<<"EL valor de (log(n)) es "<<log(log(n))<<endl;
}
```

12.37) Hoax Number

Dado un número n, verificar si es un número falso o no.

Un número falso es definido como un número compuesto, cuya suma de dígitos es igual a la suma de dígitos de sus distintos factores primos, 1 no es considerado un número primo, entonces no es incluido en la suma de dígitos de los distintos factores primos.

La definición de un número falso está cerca de la de la definición de un número de Smith, algunos de los números falsos son también números de Smith, es aparente que estos números falsos no tienen factores repetidos en su descomposición de primos.

Implementación

1) Primero se generan todos los distintos factores primos del número n.
2) Si el n no es un número primo, encuentre la suma de dígitos de los factores obtenidos en el paso 1
3) Encuentre la suma de dígitos de n
4) Verifique si la suma obtenida en 2 y 3 son iguales o no.
5) Si las sumas son iguales, entonces n es un número falso.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```java
// Programa java que verifica si un número es
// falso o no

import java.util.*;

public class HoaxNumber {

    /*Función que encuentra distintos factores primos
    dado un número n*/
    static List<Integer> primeFactors(int n) {
        List<Integer> res = new ArrayList<>();
        if (n % 2 == 0) {
            while (n % 2 == 0) {
                n = n / 2;
            }
            res.add(2);
        }
        /*N es impar en este punto
        desde que ya no sea divisible por 2
        entonces podemos probar solamente
        por números impares, mientras sean
        factores de n*/
        for (int i = 3; i <= Math.sqrt(n);
                i = i + 2) {
            //Verifica si i es factor primo
            if (n % i == 0) {
                while (n % i == 0) {
                    n = n / i;
                }
                res.add(i);
            }
        }
        /*Esra condición es para controlar
        el caso cuando n es un número primo
        más grande que 2*/
        if (n > 2) {
            res.add(n);
        }
        return res;
    }

    /*Función que calcula suma de digitos de distintos
    factores primos de dado n y la suma de digitos
    de número n. compara las sumás obtenidas*/
    static boolean isHoax(int n) {
        /*Distintos factores primos de n seran almacenados
        en vector pf*/
        List<Integer> pf = primeFactors(n);
        /* Si n es un número primo
        no puede ser un número falso*/
        if (pf.get(0) == n) {
            return false;
        }
        /*Encontrando suma de digitos de
```

```
        distintos factores primos de n*/
        int all_pf_sum = 0;
        for (int i = 0; i < pf.size(); i++) {
            // Encontramos la suma de digitos de
            // el actual factor primo pf[i]
            int pf_sum;
            for (pf_sum = 0; pf.get(i) > 0;
                    pf_sum += pf.get(i) % 10, pf.set(i, pf.get(i) / 10));

            all_pf_sum += pf_sum;
        }
        // Encontrando suma de digitos de n
        int sum_n;
        for (sum_n = 0; n > 0; sum_n += n % 10, n /= 10);
        // Comparando las dos sumás calculadas
        return sum_n == all_pf_sum;
    }

    public static void main(String args[]) {
        int n = 84;
        if (isHoax(n)) {
            System.out.print("Un número falso\n");
        } else {
            System.out.print("No es número falso\n");
        }
    }
}
```

12.38) Potencia más grande que divide un factorial

Dados dos números, fact y n, encuentre la potencia más grande de n que divide fact! (Factorial de fact).

La idea es basada en la fórmula de Legendre la cual encuentra la potencia más grande de un número primo que divide fact!. Encontramos todos los factores primos de n. para cada factor primo encontramos la potencia más grande que divide fact! y finalmente retornamos el mínimo de todas las potencias encontradas.

Si hay múltiples potencias de un factor primo presentes en n, entonces se divide el conteo para obtener el valor de la máxima potencia de este factor.

Complejidad de tiempo

Mejor caso : O(√n) **Peor caso :** O(√n) **Promedio:** O(√n)

JAVA

```java
/* Programa java que encuentra la potencia más larga de
un número (La cual puede ser compuesta) que divide
factorial*/
public class LargestPowDividesFactorial {

    /* For que encuentra la maxima potencia de número primo
    p que puede dividir un número facorial*/
    static int findPowerPrime(int fact, int p) {
        int res = 0;
        while (fact > 0) {
            res += fact / p;
            fact /= p;
        }

        return res;
    }

    // Retorna la suma de todos los factores de n
    static int findPowerComposite(int fact, int n) {
        // Para almacernar el resultado ( Potencia minima de un
        // factor primo que divide fact)
        int res = Integer.MAX_VALUE;
        // Atravesar atraves de todos los factores
        // primos de n
        for (int i = 2; i <= Math.sqrt(n); i++) {
            // contador para contar la potencia del número primo
            int count = 0;
            if (n % i == 0) {
                count++;
                n = n / i;
            }
            if (count > 0) {
                // Maxima potencia de i que divide
                // fact, dividimos por count para
                // manejar multiples ocurrencias de
                // un factor primo
                int curr_pow = findPowerPrime(fact, i) / count;
                res = Math.min(res, curr_pow);
            }
        }
        // Esta condición es para manejar
        // el caso cuando n es un número primo mayor
        // que 2
        if (n >= 2) {
            int curr_pow = findPowerPrime(fact, n);
            res = Math.min(res, curr_pow);
        }
        return res;
    }
```

```
    public static void main(String[] args) {
        int fact = 146, n = 5;
        System.out.println(findPowerComposite(fact, n));
    }
}
```

12.39) Exponenciación modular

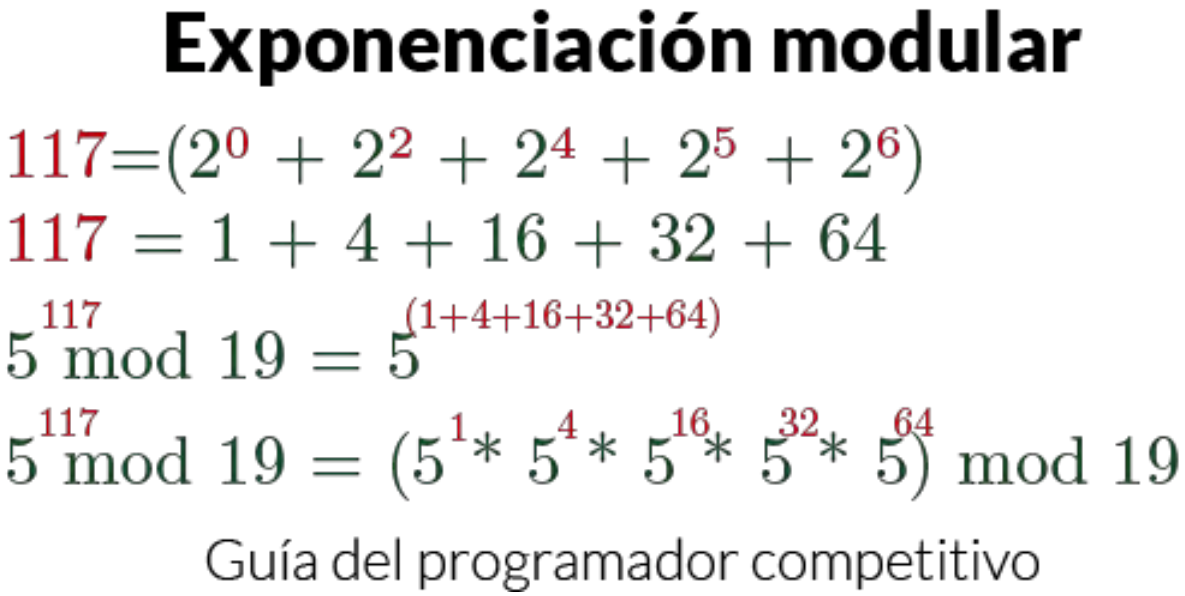


Ilustración 12-27 Ejemplos de exponenciación modular

Dados tres números x, y y p, calcule (x^y) % p.

Bajo esta propiedad fundamental modular que es usada para computación eficiente, calcular la potencia usando matemática modular.

- (ab) mod p = ((a mod p) (b mod p)) mod p

Por ejemplo a = 50, b = 100, p = 13

- 50 mod 13 = 11
- 100 mod 13 = 9
- (50 * 100) mod 13 = ((50 mod 13) * (100 mod 13)) mod 13

- or (5000) mod 13 = (11 * 9) mod 13
- or 8 = 8

Complejidad de tiempo

Mejor caso : O(log n) **Peor caso :** O(log n) **Promedio:** O(log n)

JAVA

```
// Programa iteratico que calcula
// potencia modular

public class ModularExponentiation {

    /* Función iterativa que calcula
       (x^y)%p in O(log y) */
    static int power(int x, int y, int p) {
        // Inicializar resultado
        int res = 1;
        // Actualiza x si es más que
        // o igual a p
        x = x % p;
        while (y > 0) {   //Si y es impar, multiplica x con res
            if ((y & 1) == 1) {
                res = (res * x) % p;
            }
            // y debe ser par ahora
            // y = y / 2
            y = y >> 1;
            x = (x * x) % p;
        }
        return res;
    }

    public static void main(String args[]) {
        int x = 2;
        int y = 5;
        int p = 13;
        System.out.println("La potencia es " + power(x, y, p));
    }
}
```

12.40) Multiple Euler Totient

Función Totient de Euler de una entrada n es el conteo de números en {1,2,3,..., n} que sea primo relativo a n, por ejemplo, los números los cuales su GCD con n es 1.

En problemas donde tenemos que llamar a la función totient muchas veces como 10^5 veces, una solución simple puede retornar un TLE (time limit exceded). La idea es usar la criba de Eratóstenes.

Encuentre todos los factores primos con limite en 10^5 usando la criba de Eratóstenes.

Para realizar este Φ(n), se hace lo siguiente.

1) Inicializa el resultado como n.
2) Itera a través de todos los primos más pequeños o iguales que la raíz cuadrada de n. Dejamos que el actual número primo sea p, revisamos si p divide n, si lo hace, removemos todas las ocurrencias de p de n dividiéndolo repetidamente por n, también reducimos nuestro resultado por n/p.
3) Finalmente retornamos nuestro resultado.

Complejidad de tiempo

Mejor caso : $O(n^2)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```
// Programa java que eficientemente calcula valores de
// la formula totient de euler para multiples entradas

import java.util.*;

public class MultipleEulerTotient {

    static int MAX = 100001;
    // Almacena números primos arriba hasta MAX -1
    static ArrayList<Integer> p = new ArrayList<Integer>();
    // Encurntra los números primos hasta MAX-1 Y
    // los almacena en p

    static void sieve() {
        int[] isPrime = new int[MAX + 1];
        for (int i = 2; i <= MAX; i++) {
            // Si prime[i] no es marcado antes
            if (isPrime[i] == 0) {
                // Llena el vector para cada nuevo
                // primo encontrado
                p.add(i);
                for (int j = 2; i * j <= MAX; j++) {
                    isPrime[i * j] = 1;
                }
            }
```

```
        }
    }
    // Función que encuentra totient de n

    static int phi(int n) {
        int res = n;
        // Este ciclo corre sqrt(n / ln(n)) veces
        for (int i = 0; p.get(i) * p.get(i) <= n; i++) {
            if (n % p.get(i) == 0) {
                // resta multiplos de p[i] de r
                res -= (res / p.get(i));
                // Remueve todas las ocurrencias de p[i] en n
                while (n % p.get(i) == 0) {
                    n /= p.get(i);
                }
            }
        }
        // cuando n es un factor primo mayor
        // que sqrt(n)
        if (n > 1) {
            res -= (res / n);
        }
        return res;
    }

    public static void main(String[] args) {
        //Preprocesa todos los primos hasta 10 ^ 5
        sieve();
        System.out.println(phi(11));
        System.out.println(phi(21));
        System.out.println(phi(31));
        System.out.println(phi(41));
        System.out.println(phi(51));
        System.out.println(phi(61));
        System.out.println(phi(91));
        System.out.println(phi(101));
    }
}
```

12.41) Sumatoria de naturales coprimos

Ilustración 12-28 Ejemplo de números coprimos

Dado N y M, la tarea es encontrar cuales números de 1 a n pueden ser divididos en dos conjuntos los cuales su diferencia absoluta entre la suma de los dos sets es M y el GCD de la suma de los dos sets es 1.

Desde que tenemos 1 a N números, sabemos que la suma de todos los números es N*(N+1)/2. Dejamos S1 y S2 de esta manera:

1) sum(S1) + sum(S2) = N * (N + 1) / 2
2) sum(S1) – sum(S2) = M

Resolviendo estas dos ecuaciones podemos dar la suma de ambos conjuntos. Si sum(S1) y sum(S2) son enteros y ellos son coprimos (Su GCD es 1), entonces ahí existe una forma de separar el número en dos sets. De otra forma no hay forma de separar esos números N.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```
/* Código de Java para determinar si los números
1 a N se puede dividir en dos conjuntos
```

```
tal que la diferencia absoluta entre
la suma de estos dos conjuntos es M y estos
dos sumás son co-primos*/
public class NaturalCoprimeSum {

    static int GCD(int a, int b) {
        return b == 0 ? a : GCD(b, a % b);
    }

    /*función que devuelve valor booleano
     sobre la base de si es posible
     dividir 1 a N números en dos conjuntos
     Que satisfacen las condiciones dadas.*/
    static boolean isSplittable(int n, int m) {
        // Inicializando suma total de 1
        //a n
        int total_sum = (n * (n + 1)) / 2;
        /*desde (1) total_sum = sum_s1 + sum_s2
        y (2) m = sum_s1 - sum_s2 asumiendo
        sum_s1> sum_s2. resolviendo estas 2
        ecuaciones para obtener sum_s1 y sum_s2*/
        int sum_s1 = (total_sum + m) / 2;
        // total_sum = sum_s1 + sum_s2
        // y asi
        int sum_s2 = total_sum - sum_s1;
        /*Si la suma total es menor que la
        diferencia absoluta, no hay forma
        de que podamos dividir n números
        en dos conjuntos, así que devuelva falso*/
        if (total_sum < m) {
            return false;
        }
        /*Compruebe si estas dos sumás son
        enteros y se suman a
        suma total y también si su
        La diferencia absoluta es m.*/
        if (sum_s1 + sum_s2 == total_sum
                && sum_s1 - sum_s2 == m) // Ahora si las dos sumás son coprimos
        // Entonces retorna true, si no false
        {
            return (GCD(sum_s1, sum_s2) == 1);
        }
        /*si dos sumás no suman la suma total
          o si su diferencia absoluta.
         no es m, entonces no hay manera de
         dividir n números, por lo tanto retorna false*/
        return false;
    }

    public static void main(String args[]) {
        int n = 5, m = 7;
        if (isSplittable(n, m)) {
            System.out.println("Si");
        } else {
```

```
            System.out.println("No");
        }

    }
}
```

12.42) Secuencias y sucesiones matemáticas más conocidas

Una secuencia o una sucesión es un grupo de números o de otros elementos matemáticos que forman un conjunto ordenado, es una serie de elementos que se suceden unos a otros y guardan relación entre sí. A continuación presentamos las secuencias y sucesiones mas conocidas, con descripción, ejemplo y función generadora de la misma (Si existe).

Nombre	Primeros elementos	Descripción	Formula o ejemplo
Secuencia de Kolakoski	{1, 2, 2, 1, 1, 2, 1, 2, 2, 1, ...}	El enésimo termino describe el tamaño de la enésima búsqueda	a(1)=1 a(2)=2 a(a(1)+a(2)+...+a(k))= (3 +(-1)^k)/2 a(a(1)+a(2)+...+a(k)+1)= (3 -(-1)^k)/2.
Function totient de Euler ϕ(n)	{1, 1, 2, 2, 4, 2, 6, 4, 6, 4, ...}	ϕ(n) es el numero de los enteros positivos no mas grandes que n que son primos de n;	$\varphi(n) = n\prod_{p\mid n}\left(1-\frac{1}{p}\right)$ Phi(6) = 6 * (1-1/2) * (1 – 1/3) = 2.
Numeros de Lucas L(n)	{2, 1, 3, 4, 7, 11, 18, 29, 47, 76, ...}	L(n) = L(n − 1) + L(n − 2) Para n ≥ 2, con L(0) = 2 y L(1) = 1.	L1=1 L2=3 Ln=Ln−2+Ln−1

Numeros primos pn	{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...}	Los n numeros primos mayores que 1.	n2+n+41 ->para n=0,…,39. n2−n+41 -> n=0,…,40. n2−15n+97 -> n=0,…,47. n2−61n+971 -> n=0,…,70. n2−79n+1601 -> n=0,…,79. n2−81n+1681 -> n=1,…,80.
Partición de numeros Pn	{1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42, ...}	El n numero de particiones cuyos términos al sumarse su resultado es el numero inicial;	Particiones del número 5 5 4 + 1 3 + 2 3 + 1 + 1 2 + 2 + 1 2 + 1 + 1 + 1 1 + 1 + 1 + 1 + 1
Numeros de fibonacci F(n)	{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...}	F(n) = F(n − 1) + F(n − 2) Para n ≥ 2, con F(0) = 0 y F(1) = 1.	$f_n = f_{n-1} + f_{n-2}$
Secuencia de Sylvester	{2, 3, 7, 43, 1807, 3263443, 10650056950807, 113423713055421844361000443, ...}	Sucesión de números enteros en la cual cada término es el producto de todos los anteriores, más uno. a(n + 1) = a(n)·a(n − 1)· ⋯ a(0) + 1 = a(n)2 − a(n) + 1 para n ≥ 1, con a(0) = 2.	$S_n = n\prod_{i=0}^{n-1} S_I$
Numeros de Tribonacci	{0, 1, 1, 2, 4, 7, 13, 24, 44, 81, ...}	T(n) = T(n − 1) + T(n − 2) + T(n − 3) Para n ≥ 3, con T(0) = 0 y T(1) = T(2) = 1.	$T(n) = \left\lceil 3b \frac{\left(\frac{1}{3}(a_+ + a_- + 1)\right)^n}{b^2 - 2b + 4} \right\rceil$
Poliominó	{1, 1, 1, 2, 5, 12, 35, 108, 369, ...}	Es un objeto geométrico obtenido al unir varios cuadrados o celdas del mismo tamaño de forma que cada par de celdas vecinas compartan un lado.	n(0)=1 ¿De cuantas formas se puede hacer un poliominó de dos celdas libres? 2 formas 
Números de catalan Cn	{1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, ...}	Los números de Catalan forman una secuencia de números naturales que aparece en varios problemas	Con n>=0

		de conteo que habitualmente son recursivos	$C_n = \frac{(2n)!}{(n+1)!\,n!}$
Numeros de Bell Bn	{1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, ...}	Bn es el número de particiones de un set con n elementos.	$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$
Nuemros de Euler zigzag En	{1, 1, 1, 2, 5, 16, 61, 272, 1385, 7936, ...}	En es el numero de extensiones lineares de una permutación con alternación.	$E(n,k) = \{0\ E(n,k-1) + E(n-1, n-k)$
Secuencia del cartero perezoso	{1, 2, 4, 7, 11, 16, 22, 29, 37, 46, ...}	El máximo número de piezas que se forman cuando se corta un pancake con n cortes.	$p = \frac{n^2+n+2}{2}$
Numeros de Pell Pn	{0, 1, 2, 5, 12, 29, 70, 169, 408, 985, ...}	a(n) = 2a(n − 1) + a(n − 2) Para n ≥ 2, con a(0) = 0, a(1) = 1.	$P_n = \frac{(1+\sqrt{2})^2 - (1-\sqrt{2})^2}{2\sqrt{2}}$
Factoriales n!	{1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...}	n! := 1·2·3·4· ⋯ ·n para n ≥ 1 con 0! = 1 (Producto vacio).	$n! = \prod_{k=1}^{n} k$
Desarreglos	{1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, ...}	Número de posibles desarreglos (permutación donde ninguno de sus elementos aparece en la posición original) de un conjunto con n elementos.	$n = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}$
Función divisor σ(n)	{1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, ...}	σ(n) := σ1(n) Es la suma de divisores de un entero positivo n.	$\sigma_x(n) = \sum_{\frac{d}{n}} d^x$
Números de Fermat Fn	{3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, 340282366920938463463374607431768211457, ...}	Los números de Fermat son números de la forma F_n=2^{2^n}+1, desde n = 0 en adelante.	$F_n = 2^{2^n} + 1$

Poliarboles	{1, 1, 3, 8, 27, 91, 350, 1376, 5743, 24635, 108968, ...}	Número de árboles orientados con n nodos.	Cualquier numero (Natural)
Números perfectos	{6, 28, 496, 8128, 33550336, 8589869056, 137438691328, 2305843008139952128, ...}	n es igual a la suma s(n) = σ(n) − n de los divisors propios de n.	$T_{2^P-1} = 1 + \frac{(2^P-2)*(2^P+1)}{2}$
Función de Landaw	{1, 1, 2, 3, 4, 6, 6, 12, 15, 20, ...}	El orden más largo de permutaciones de n elementos	$\frac{lnln\,(g(n))}{\sqrt{nlnln\,(n)}}$ =1 $g(n) \leq e^{\frac{n}{e}}$
Las vacas de Narayana	{1, 1, 1, 2, 3, 4, 6, 9, 13, 19, ...}	El número de vacas cada año si cada vaca tiene una vaca al año comenzando su cuarto año.	$A(n) = A(n-1) + A(n-3)$
Secuencia de Padovan	{1, 1, 1, 2, 2, 3, 4, 5, 7, 9, ...}	P(n) = P(n − 2) + P(n − 3) Para n ≥ 3, con P(0) = P(1) = P(2) = 1.	$P(n) = P(n-2) + P(n-3)$
Secuencia de Euclid–Mullin	{2, 3, 7, 43, 13, 53, 5, 6221671, 38709183810571, 139, ...}	a(1) = 2; a(n + 1) es el factor primo mas pequeño de a(1) a(2) ⋯ a(n) + 1.	$\left(\prod_{i<n} a_i\right) + 1$
Lucky numbers	{1, 3, 7, 9, 13, 15, 21, 25, 31, 33, ...}	Un número de la suerte es un número natural en un conjunto que se genera utilizando un sistema de criba de Josefo Flavio en el cual se van eliminando números dados unos saltos, los números restantes son los números de la secuencia	N/A
Potencias primas	{1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 16, 17, 19, ...}	Potencias enteras positivas de números primos Las potencias primas son los enteros positivos divisibles por exactamente un número primo.	p^n

Coeficientes binomiales centrales	{1, 2, 6, 20, 70, 252, 924, ...}	Números en el centro de las filas pares del triángulo de Pascal	$\binom{2n}{n} = \frac{(2n)!}{(n!)^2}$ for all $n \geq 0$.
Números de Motzkin	{1, 1, 2, 4, 9, 21, 51, 127, 323, 835, ...}	Para un cierto número n es la cantidad de maneras distintas de dibujar cuerdas que no se intersecan en un círculo entre n puntos.	$M_n = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{2k} C_k.$ C es un numero catalan $M_n = M_{n-1} + \sum_{i=0}^{n-2} M_i M_{n-2-i} = \frac{2n+1}{n+2} M_{n-1} + \frac{3n-3}{n+2} M_{n-2}.$
Números de Jacobsthal	{0, 1, 1, 3, 5, 11, 21, 43, 85, 171, 341, ...}	a(n) = a(n − 1) + 2a(n − 2) Para n ≥ 2 con a(0) = 0, a(1) = 1.	$J_{n+1} = 2J_n + (-1)^n$
Suma de divisores propios (n)	{0, 1, 1, 3, 1, 6, 1, 7, 4, 8, ...}	s(n) = σ(n) − n es la suma de los divisores propios del entero positivo n.	$s(n) = \sigma(n) - n,$
Numeros Wedderburn–Etherington	{0, 1, 1, 1, 2, 3, 6, 11, 23, 46, ...}	El número de árboles binarios enraizados (cada nodo tiene un grado externo 0 o 2) con n puntos finales (y 2n - 1 nodos en total).	$a_{2n-1} = \sum_{i=1}^{n-1} a_i a_{2n-i-1}$
Secuencia de Gould	{1, 2, 2, 4, 2, 4, 4, 8, 2, 4, 4, 8, 4, 8, 8, ...}	Número de entradas impares en la fila n del triángulo de Pascal.	a(n) = 2^(numero de 1 de la forma binaria de(n-1)).
Semiprimos	{4, 6, 9, 10, 14, 15, 21, 22, 25, 26, ...}	Producto de dos primos, no necesariamente distintos	$\varphi(n) = n + 1 - (p + q).$
Numeros de Perrin Pn	{3, 0, 2, 3, 2, 5, 5, 7, 10, 12, ...}	P(n) = P(n−2) + P(n−3) Para n ≥ 3, con P(0) = 3, P(1) = 0, P(2) = 2.	$G(P(n); x) = \frac{3 - x^2}{1 - x^2 - x^3}$
Sorting number	{0, 1, 3, 5, 8, 11, 14, 17, 21, 25, 29, 33, 37, 41, 45, 49 ...}	Utilizado en el análisis de tipos de comparación.	$A(n) = A(\lfloor n/2 \rfloor) + A(\lceil n/2 \rceil) + n - 1.$
Numeros de Cullen Cn	{1, 3, 9, 25, 65, 161, 385, 897, 2049, 4609, 10241, 22529, 49153, 106497, ...}	Cn = n·2n + 1, con n ≥ 0.	$n \cdot 2^n + 1$

Primordiales **pn#**	{1, 2, 6, 30, 210, 2310, 30030, 510510, 9699690, 223092870, ...}	pn#, el producto de los primeros n primos.	$p_n\# \equiv \prod_{k=1}^{n} p_k$
Numeros altamente compuestos	{1, 2, 4, 6, 12, 24, 36, 48, 60, 120, ...}	Un entero positivo con más divisores que cualquier entero positivo más pequeño.	$N = 2^{a_2}\ 3^{a_3} \cdots p^{a_p}$
Numeros altamente compuestos superiores	{2, 6, 12, 60, 120, 360, 2520, 5040, 55440, 720720, ...}	Un entero positivo n para el cual hay un e > 0 tal que d(n)/ne ≥ d(k)/ke para todos los k > 1.	$\frac{d\,(n)}{n^e} \geq \frac{d\,(k)}{k^e}$
Numeros Pronicos	{0, 2, 6, 12, 20, 30, 42, 56, 72, 90, ...}	2t(n) = n (n + 1), con n ≥ 0.	$\lfloor\sqrt{n}\rfloor \cdot \lceil\sqrt{n}\rceil = n$
Numeros de Markov	{1, 2, 5, 13, 29, 34, 89, 169, 194, ...}	Enteros positivos que sean solucion de x2 + y2 + z2 = 3xyz.	$m_n = \frac{1}{3}e^{C\sqrt{n}+o(1)}$ with $C = 2.3523414972$
Numeros compuestos	{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, ...}	Los numeros n de la forma xy para x > 1 y y > 1.	Cualquier numero (No primo != 1)
Numero de Ulam	{1, 2, 3, 4, 6, 8, 11, 13, 16, 18, ...}	a(1) = 1; a(2) = 2; para n > 2, a(n) es el número mínimo> a (n - 1) que es una suma única de dos términos anteriores distintos; semiperfecto	$\{a_i\} = (u,\ v)$
Nudos primos	{0, 0, 1, 1, 2, 3, 7, 21, 49, 165, 552, 2176, 9988, ...}	El número de nudos primos con n cruces.	$n = 1, 2, \ldots$ crossings are 0, 0, 1, 1, 2, 3, 7,
Numeros de Carmichael	{561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, ...}	Números compuestos n tales que a^n − 1 ≡ 1 (mod n) si es un primo a n.	$a^{n-1} \equiv 1 \pmod{n}$
Numeros de Woodall	{1, 7, 23, 63, 159, 383, 895, 2047, 4607, ...}	n·2n − 1, con n ≥ 1.	$W_n = n \cdot 2^n - 1$
Números aritméticos.	{1, 3, 5, 6, 7, 11, 13, 14, 15,	Un número entero para el que el promedio de sus divisores	Ejemplo

	17, 19, 20, 21, 22, 23, 27, ...}	positivos también es un número entero.	$\frac{1+2+3+6}{4} = 3$
Numeros colosamente abundantes	{2, 6, 12, 60, 120, 360, 2520, 5040, 55440, 720720, ...}	Un número n es colosalmente abundante si hay un ε> 0 tal que para todo k> 1, Donde σ denota la función de suma de divisores.	$\frac{\sigma(n)}{n^{1+\varepsilon}} \geq \frac{\sigma(k)}{k^{1+\varepsilon}}$
Secuencia de Alcuin	{0, 0, 0, 1, 0, 1, 1, 2, 1, 3, 2, 4, 3, 5, 4, 7, 5, 8, 7, 10, 8, 12, 10, 14, ...}	Number of triangles with integer sides and perimeter n.	$x^3 + x^5 + x^6 + 2x^7 + x^8 + 3x^9 + \cdots$
Numeros deficientes	{1, 2, 3, 4, 5, 7, 8, 9, 10, 11, ...}	Enteros positivos n tal que σ(n) < 2n.	1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17,
Numeros abundantes	{12, 18, 20, 24, 30, 36, 40, 42, 48, 54, ...}	N enteros positivos tal que σ(n) > 2n.	12, 18, 20, 24, 30, 36, 40, 42, 48, 54, 56,
Numeros intocables	{2, 5, 52, 88, 96, 120, 124, 146, 162, 188, ...}	No se puede expresar como la suma de todos los divisores propios de cualquier número entero positivo.	2, 5, 52, 88, 96,
Secuencia de Recaman	{0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, 22, 10, 23, 9, 24, 8, 25, 43, 62, ...}	" reste si es posible, de lo contrario agregue ": a(0) = 0; Para n > 0, a(n) = a(n - 1) - n si ese número es positivo y no está ya en la secuencia, de lo contrario a (n) = a (n - 1) + n, ya sea que ese número ya esté o no en la secuencia.	$a_n = \begin{cases} 0 & \text{if } n = 0 \\ a_{n-1} - n & \text{if } a_{n-1} - n > 0 \\ a_{n-1} + n & \text{otherwise} \end{cases}$
Secuencia de mirar y decir	{1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, 31131211131221, 13211311123113112211, ...}	A = 'frecuencia' seguida de 'dígito' -indicación.	1, 11, 21, 1211, 111221, 312211
Números prácticos	{1, 2, 4, 6, 8, 12, 16, 18, 20, 24, 28, 30, 32, 36, 40...}	Todos los enteros positivos más pequeños se pueden representar como sumas de factores distintos del número.	1, 2, 4, 6, 8, 12, 16, 18, 20, 24,

Factorial alternado.	{1, 1, 5, 19, 101, 619, 4421, 35899, 326981, 3301819, 36614981, 442386619, 5784634181, 81393657019, ...}	n! - (n-1)! + (n-2)! - ... 1!.	$af(n) = \sum_{i=1}^{n} (-1)^{n-i} i!$
Numeros afortunados	{3, 5, 7, 13, 23, 17, 19, 23, 37, 61, ...}	El entero más pequeño m> 1, de modo que pn # + m es un número primo, donde el pn # primitivo es el producto de los primeros n números primos.	3, 5, 7, 13, 17, 19, 23, 37,
Numeros semi perfectos	{6, 12, 18, 20, 24, 28, 30, 36, 40, 42, ...}	Un número natural n que es igual a la suma de todos o algunos de sus divisores propios.	$2^m(2^{m+1} - 1)$
Constantes magicas	{15, 34, 65, 111, 175, 260, ...}	Suma de números en cualquier fila, columna o diagonal de un cuadrado mágico de orden n = 3, 4, 5, 6, 7, 8, ...	$M = n \cdot \frac{n^2+1}{2}$ where n is the side length
Numeros raros	{70, 836, 4030, 5830, 7192, 7912, 9272, 10430, 10570, 10792, ...}	Un número natural que es abundante pero no semiperfecto.	$n = 2^{56} \cdot (2^{61} - 1) \cdot 153722867280912929 \approx 2 \cdot 10^{52}$.
Numeros de kaprekar	{1, 9, 45, 55, 99, 297, 703, 999, 2223, 2728, ...}	X2 = Abn + B Donde 0< B < bn y X = A + B.	Sea X un entero no negativo. X es un número de Kaprekar para la base b si existen n números enteros no negativos, A y B, que satisfagan las siguientes condiciones: 0 < B < bn X^2 = Abn + B X = A + B
Numeros esfenicos	{30, 42, 66, 70, 78, 102, 105, 110, 114, 130, ...}	Productos de 3 primos distintos.	N= p x q x r Donde p,q y ,r son primos distintos y no repetidos. 230 = 2×5×23 y 231 = 3×7×11

Radical de un entero	{1, 2, 3, 2, 5, 6, 7, 2, 3, 10, ...}	En teoría de números, el radical de un entero positivo n, es el producto de los números primos que dividen n.	$504 = 2^3 \cdot 3^2 \cdot 7 \longrightarrow \operatorname{rad}(504) = 2 \cdot 3 \cdot 7 = 42$ $143 = 11 \cdot 13 \longrightarrow \operatorname{rad}(143) = 11 \cdot 13 = 143$
Secuencia de Thue–Morse	{0, 1, 1, 0, 1, 0, 0, 1, 1, 0, ...}	En matemáticas, la constante de Prouhet–Thue–Morse, es el número cuya expansión binaria 01101001100101101001011001101001... está dada por la Sucesión de Thue–Morse.	$\tau(x) = \prod_{n=0}^{\infty}(1 - x^{2^n}).$
Enteros de Blum	{21, 33, 57, 69, 77, 93, 129, 133, 141, 161, 177, ...}	Un número natural n es un número entero Blum si n = p × q es un semiprimo para el que p y q son distintos números primos congruentes a 3 mod 4. Es decir, p y q deben ser de la forma 4 t + 3 , para algún entero t .	N/A
Números mágicos	{2, 8, 20, 28, 50, 82, 126, ...}	Una cantidad de nucleones (ya sea protones o neutrones) de modo que estén dispuestos en capas completas dentro del núcleo atómico.	a(n) = n*(n^2 + 5)/3. G.f.: 2*x*(1 - x + x^2)/(1-x)^4. a(n) = 4*a(n-1) - 6*a(n-2) + 4*a(n-3) - a(n-4).
Números superperfectos	{2, 4, 16, 64, 4096, 65536, 262144, 1073741824, 1152921504606846976, 3094850098213450687247 81056, ...}	Enteros positivos n para los cuales σ2(n) = σ(σ(n)) = 2n.	N/A
Números de Bernoulli Bn	{1, -1, 1, 0, -1, 0, 1, 0, -1, 0, 5, 0, -691, 0, 7, 0, -3617, 0, 43867, 0, ...}	Los números de Bernoulli, son números muy importantes relacionados con distintos temas matemáticos. Aparecen principalmente relacionados con la función zeta de Riemann, dando solución a los casos pares.	$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = \frac{x}{e^x - 1}$ $B_m = 1 - \frac{1}{m+1} \sum_{k=0}^{m-1} \binom{m+1}{k} B_k$
Números hiperperfectos	{6, 21, 28, 301, 325, 496, 697, ...}	Un número perfecto es un número natural n para el que se cumple la igualdad n = 1 + k (σ (n) - n - 1), donde σ (n) es la función divisor (es decir, la suma de todos divisores positivos de n). Un número hiperperfecto es un número k	n = 1 + k(σ(n) − n − 1)

		-perfecto perfecto para algún entero k .	
Números de Aquiles	{72, 108, 200, 288, 392, 432, 500, 648, 675, 800, ...}	Enteros positivos que son poderosos pero imperfectos. Un número n = p 1 a 1 p 2 a 2 ... p k a k es poderoso si min (a 1, a 2, ..., a k) ≥ 2. Si además de mcd (a 1, a 2,..., a k) = 1.	$784 = 2^4 \cdot 7^2 = (2^2)^2 \cdot 7^2 = (2^2 \cdot 7)^2 = 28^2.$
Números pseudoperfectos primarios	{2, 6, 42, 1806, 47058, 2214502422, 52495396602, ...}	N es un número primario semiperfecto si satisface la ecuación de fracción egipcia.	$\frac{1}{N} + \sum_{p\mid N} \frac{1}{p} = 1,$
Números de Erdős-Woods	{16, 22, 34, 36, 46, 56, 64, 66, 70, 76, 78, 86, 88, ...}	k es un número de Erdős-Woods si existe un entero positivo tal que para cada entero i entre 0 y k, al menos uno de los máximos comunes divisores MCD (a, a + i) y MCD (a + i, a + k) es mayor que 1.	N/A
Números de Sierpinski	{78557, 271129, 271577, 322523, 327739, 482719, 575041, 603713, 903983, 934909, ...}	Impar k para el cual {k·2n + 1: n ∈ ℕ} consiste solo en números compuestos.	N/A
Números de riesel	{509203, 762701, 777149, 790841, 992077, ...}	Impar k para cual { k·2n −1 : n ∈ ℕ } consiste solo en números compuestos.	N/A
Secuencia Baum-Sweet	{1, 1, 0, 1, 1, 0, 0, 1, 0, 1, ...}	a (n) = 1 si la representación binaria de n no contiene ningún bloque de ceros consecutivos de longitud impar; de lo contrario a (n) = 0.	N/A
Secuencia de Gijswijt	{1, 1, 2, 1, 1, 2, 2, 2, 3, 1, ...}	El enésimo término cuenta el número máximo de bloques repetidos al final de la subsecuencia de 1 a n-1	N/A
Numeros de Carol	{−1, 7, 47, 223, 959, 3967,	Un número de Carol es un entero de la forma 4n - 2 (n +	a(n) = (2^n-1)^2 - 2.

	16127, 65023, 261119, 1046527, ...}	1) - 1. Una fórmula equivalente es (2n-1) 2 - 2. Para n> 2, la representación binaria del enésimo número de Carol es n-2 consecutivo, un cero simple en el medio y n + 1 más consecutivo. Ejemplo, n = 4 el número de villancicos es 223 y el binario de 223 es 11011111, aquí n-2 = 4-2 = 2 uno consecutivo en el inicio y luego único 0 en el medio y luego n + 1 = 4 + 1 = 5 consecutivo después de él .	o a(n) = 6*a(n-1) - 7*a(n-2) - 6*a(n-3) + 8*a(n-4).
Secuencia del malabarista **(Juggler)**	{0, 1, 1, 5, 2, 11, 2, 18, 2, 27, ...}	El nombre de esta sucesión viene de la subida y bajada de los valores de los números que conforman las secuencias como las bolas que lanza un malabarista.	$a_{n+1} = \text{int}(\sqrt{a_n})$ si a_n es par $a_{n+1} = \text{int}(\sqrt{a_n^3})$ si a_n es impar , con n ≥ 0 entero
Números altamente totient	{1, 2, 4, 8, 12, 24, 48, 72, 144, 240, ...}	Un número muy totient k es un número entero que tiene más soluciones a la ecuación ϕ (x) = k , donde ϕ es función totient de Euler , que cualquier número entero debajo de ella.	$\phi(x) = \prod_i (p_i - 1) p_i^{e_i - 1}.$
Números de Euler	{1, 0, −1, 0, 5, 0, −61, 0, 1385, 0, ...}	Los números de Euler aparecen en las expansiones de la serie Taylor de las funciones secante e secante hiperbólica. Esta última es la función en la definición. También se producen en combinatoria, específicamente al contar el número de permutaciones alternas de un conjunto con un número par de elementos.	$\frac{1}{\cosh t} = \frac{2}{e^t + e^{-t}} = \sum_{n=0}^{\infty} \frac{E_n}{n!} \cdot t^n,$
Números corteses **(Polite)**	{3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, ...}	Un entero positivo que se puede escribir como la suma de dos o más enteros positivos consecutivos.	$f(n) = n + \lfloor \log_2(n + \log_2 n) \rfloor.$
Números de Erdős-Nicolas	{24, 2016, 8190, 42336, 45864, 392448, 714240, 1571328, ...}	En teoría de números, un número de Erdős-Nicolas es un número que no es perfecto, pero que equivale a una de las sumas parciales de sus divisores.	$\sum_{d \mid n,\ d \le m} d = n.$[1]

Números triangulares t (n)	{0, 1, 3, 6, 10, 15, 21, 28, 36, 45, ...}	Aquel que puede recomponerse en la forma de un triángulo equilátero.	t(n) = C(n + 1, 2) = n (n + 1)/2 = 1 + 2 + ⋯ + n n ≥ 1, con t(0) = 0 (suma vacia).
Números cuadrados n2	{0, 1, 4, 9, 16, 25, 36, 49, 64, 81, ...}	Numeros resultado de elevar n a su segunda potencia.	n^2 = n × n
Números tetraédricos T (n)	{0, 1, 4, 10, 20, 35, 56, 84, 120, 165, ...}	T(n) es la suma de los primeros n números triangulares, con T(0) = 0 (suma vacia).	$T_n = \frac{1}{6}n(n+1)(n+2).$
Números piramidales cuadrados	{0, 1, 5, 14, 30, 55, 91, 140, 204, 285, ...}	El número de esferas apiladas en una pirámide con una base cuadrada.	n (n + 1)(2n + 1)/6
Números de cubos n^3	{0, 1, 8, 27, 64, 125, 216, 343, 512, 729, ...}	Numeros resultado de elevar n a su tercera potencia.	n^3 = n × n × n
Potencias 5 de n	{0, 1, 32, 243, 1024, 3125, 7776, 16807, 32768, 59049, 100000, ...}	Numeros resultado de elevar n a su quinta potencia.	N^5
Número de estrella	{1, 13, 37, 73, 121, 181, 253, 337, 433, 541, 661, 793, 937, ...}	Un número de estrella es un número figurado centrado, un hexagrama centrado (estrella de seis puntas), como el que se juega con las fichas chinas.	Sn = 6n(n − 1) + 1.
Números de Stella octangula	{0, 1, 14, 51, 124, 245, 426, 679, 1016, 1449, 1990, 2651, 3444, 4381, ...}	Los números de stella octangula son números figurados que cuentan el número de bolas que se pueden organizar en la forma de un octaedro estrellado	n (2n^2 − 1), con n ≥ 0.
Número primo de Mersenne	{2, 3, 5, 7, 13, 17, 19, 31, 61, 89, ...}	Número entero positivo m que es una unidad menor que una potencia entera positiva de 2:	P=2^p − 1 = resultado primo
Primos de Mersenne	{3, 7, 31, 127, 8191, 131071, 524287, 2147483647, 23058430092 13693951, 61897001964 26901374495 62111, ...}	Número entero positivo m que es una unidad menor que una potencia entera positiva de 2:	P=2^p − 1 = resultado primo, con P siendo primo.

Primos de Wagstaff	{3, 11, 43, 683, 2731, 43691, ...}	Los primos de Wagstaff llevan el nombre del matemático Samuel S. Wagstaff Jr .; Las páginas principales le dan crédito a François Morain por nombrarlas en una conferencia en la conferencia Eurocrypt de 1990. Los primos de Wagstaff aparecen en la conjetura de New Mersenne y tienen aplicaciones en criptografía.	$p = \frac{2^q + 1}{3}$ $3 = \frac{2^3 + 1}{3},$ $11 = \frac{2^5 + 1}{3},$ $43 = \frac{2^7 + 1}{3}.$
Numeros felices	{1, 7, 10, 13, 19, 23, 28, 31, 32, 44, ...}	Los números felices se definen por el siguiente procedimiento: empezando con cualquier número entero positivo, se reemplaza el número por la suma de los cuadrados de sus dígitos, y se repite el proceso hasta que el número es igual a 1 o hasta que se entra en un bucle que no incluye el 1.	7 es un número feliz, ya que: 72 = 49 42 + 92 = 97 92 + 72 = 130 12 + 32 + 02 = 10 12 + 02 = 1. Si n no es feliz la suma de los cuadrados entrará en un bucle (de periodo 8): 4, 16, 37, 58, 89, 145, 42, 20, 4,...

Tabla 12-2 Secuencias y sucesiones mas conocidas

12.43) Números de Leonardo

Leonardo Numbers

- Cuántos nodos hay en Lt(n)?
- Pregunta: El enésimo número de Leonardo:
  - L(0)=1
  - L(1)=1
  - L(n+2)=1+L(n)+L(n+1)
- Los primeros términos son 1, 1, 3, 5, 9, 15, 25, 41, ...

Guía del programador competitivo

Ilustración 12-29 Números de Leonardo

Los números de Leonardo son una secuencia de números con la recurrencia:

- L(0)=0
- L(1)=1
- L(n)=L(n-1)+l(n-2)+1 si n>1

Los primeros números de Leonardo son 1, 1, 3, 5, 9, 15, 25, 41, 67, 109, 177, 287, 465, 753, 1219, 1973, 3193, 5167, 8361, ⋯

Complejidad de tiempo: Exponencial

JAVA

```java
//Pograma java qque busca el nesimo número
// de Leonardo

public class LeonardoNumber {

    static int Leonardo(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return (Leonardo(n - 1) + Leonardo(n - 2) + 1);
    }
    public static void main(String args[]) {
        System.out.println(Leonardo(3));
    }
}
```

12.44) Teorema de Zeckendorf

Ilustración 12-30 Ejemplo del teorema de Zekendorf

El teorema de Zeckendorf indica que cada posible entero puede ser escrito como la suma de distintos números de Fibonacci no vecinos. Dos números Fibonacci son vecinos si uno viene luego del otro en la secuencia (0, 1, 1, 2, 3, 5, ..). Por ejemplo 3 y 5 son vecinos pero 2 y 5 no lo son.

Dado un número, encontrar la representación del número como la suma de números de Fibonacci no consecutivos.

1) Sea n el número de entrada
2) Mientras n >= 0

a) Encontrar el Fibonacci más grande que sea menor que n. Dejar que este número sea ‘f’, e imprimir f.

b) n = n – f

JAVA

```
/*Programa java para el Teorema de Zeckendorf,
encuentra la representación de n como suma de
números de fibonacci no vecinos*/
public class ZeckendorfTheorem {

    public static int nearestSmallerEqFib(int n) {
        // Casos base
        if (n == 0 || n == 1) {
            return n;
        }
        //Encuentra el mayor número fibonacci menor que n
        int f1 = 0, f2 = 1, f3 = 1;
        while (f3 <= n) {
            f1 = f2;
            f2 = f3;
            f3 = f1 + f2;
        }
        return f2;
    }
    // Imprime representación de fibonacci

    public static void printFibRepresntation(int n) {
        while (n > 0) {
            // Encuentra el mayor número fibonacci menor
            // o igual que n
            int f = nearestSmallerEqFib(n);
            // Imprime el número fibonacci encontrado
            System.out.print(f + " ");
            // Reduce n
            n = n - f;
        }
    }

    public static void main(String[] args) {
        int n = 30;
        System.out.println("Representación de fibonacci no vecinos "
                + " de  " + n + " es");
        printFibRepresntation(n);
    }
}
```

12.45) **Teorema de Rosser**

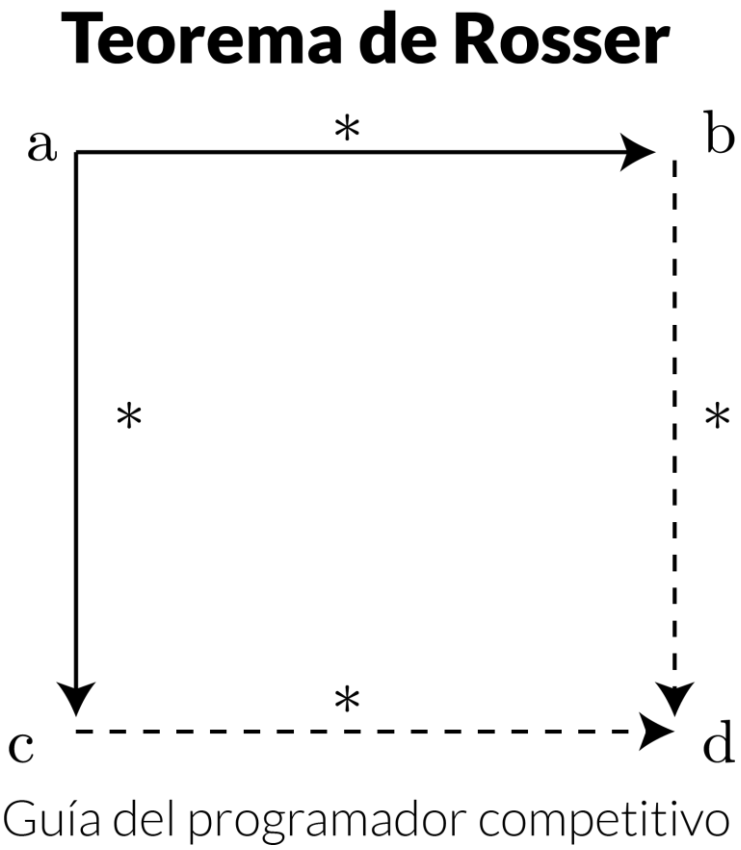


Ilustración 12-31 Teorema de Rosser

El teorema de Rosser dicta que el nesimo número es mayor que el producto de n y el logaritmo natural de n para todos los n mayores a 1.

Matemáticamente:

Para n >= 1, si pn es el nesimo número primo, entonces then:

- pn > n * (ln n)

Para n = 1, enésimo número primo = 2

- 2 > 1 * ln(1)

Para n = 2, enésimo número primo = 3

- 3 > 2 * ln(2)

Para n = 3, enésimo número primo = 5

- 5 > 3 * ln(3)

Para n = 4, enésimo número primo = 7

- 7 > 4 * ln(4)

Para n = 5, enésimo número primo = 11

- 11 > 5 * ln(5)

Para n = 6, enésimo número primo = 13

- 13 > 6 * ln(6)

JAVA

```java
// Programa java que verifica el Teorema de Rosser

import java.util.*;

public class RosserTheorem {

    static ArrayList<Integer> prime = new ArrayList<Integer>();
    // Criba de Eratostenes
    static void sieve() {
        int n = 10000;
        boolean[] isprime = new boolean[n + 2];
        for (int i = 0; i < n; i++) {
            isprime[i] = true;
        }
        isprime[0] = false;
        isprime[1] = false;
        for (int i = 2; i <= n; i++) {
            if (isprime[i]) {
                for (int j = i * i; j <= n; j += i) {
                    isprime[j] = false;
                }
            }
        }
        //Almacena primos en prime[]
        for (int i = 0; i <= n; i++) {
            if (isprime[i]) {
                prime.add(i);
            }
        }
    }

    // Verifica el TEOREMA DE  ROSSER para todos los números
    // Menores a n
    static void verifyRosser(int n) {
```

```
        System.out.println("TEOREMA DE ROSSER: nesimo número primo > n * (ln
n)");
        for (int i = 0; i < n; i++) {
            if (prime.get(i) > (i + 1) * Math.log(i + 1)) {
                System.out.println("para n = " + (i + 1)
                        + ", nesimo número primo = "
                        + prime.get(i) + "\n\t"
                        + prime.get(i) + " > " + (i + 1)
                        + " * ln(" + (i + 1) + ")");
            }
        }
    }

    public static void main(String[] args) {
        sieve();
        verifyRosser(20);
    }
}
```

12.46) Números de Smith

Ilustración 12-32 Números de Smith conocidos

Dado un número n, la tarea encontrar si un número es número de Smith o no, un número de Smith es un número compuesto cuya suma de dígitos es igual a la suma de los dígitos en su factorización prima.

- n = 4
- Factorización prima = 2, 2 and 2 + 2 = 4
- Por lo tanto, 4 es un número de Smith

La idea es primero encontrar todos los factores primos por debajo de un límite usando la criba de Sundaram. (Esto es útil para buscar y verificar varios números de Smith). Ahora por cada entrada que será verificada como número de Smith, atravesamos por todos los factores primos en él, y encontramos la suma de los dígitos en cada factor primo. También buscamos la suma de los dígitos en el número dado. Finalmente comparamos las dos sumas, si son lo mismo, retornamos true.

JAVA

```java
// Programa java que verifica si un número es
// número de Smith o no

import java.util.ArrayList;

public class SmithNumber {

    static int MAX = 10000;
    //Array que almacena todos los primos menores o iguales
    // a 10^6
    static ArrayList<Integer> primes = new ArrayList<>();
    //Función de la criba de Sundaram

    static void sieveSundaram() {
        /*En general criva de Sundaram, produce primos más pequeños.
        que (2 * x + 2) para un número dado el número x. Ya que
         Queremos primos más pequeños que MAX, reducimos MAX a la mitad
         Esta matriz se usa para separar números del forma
         i + j + 2ij de otros donde 1 <= i <= j*/
        boolean marked[] = new boolean[MAX / 2 + 100];
        //Lógica principal de Sundaram, marcar todos los números
        // loc cuales no generan número primo haciendo 2*i+1
        for (int i = 1; i <= (Math.sqrt(MAX) - 1) / 2; i++) {
            for (int j = (i * (i + 1)) << 1; j <= MAX / 2; j = j + 2 * i + 1) {
                marked[j] = true;
            }
        }

        // 2s es número primo
        primes.add(2);
        // Imprime los otros primos, primos restantes son de la
        //forma 2*i+1 de tal manera que marked[i] es falso
        for (int i = 1; i <= MAX / 2; i++) {
            if (marked[i] == false) {
                primes.add(2 * i + 1);
            }
        }
    }
    // Retorna true si n es un número de Smith, si no falso

    static boolean isSmith(int n) {
        int original_no = n;
        // Encuentra la suma de los digitos de los factores
        //  primos de n
        int pDigitSum = 0;
        for (int i = 0; primes.get(i) <= n / 2; i++) {
            while (n % primes.get(i) == 0) {   // Si primes[i] es un factor
primo
                // agrega sus digitos a pDigitSum
                int p = primes.get(i);
                n = n / p;
                while (p > 0) {
```

```
                    pDigitSum += (p % 10);
                    p = p / 10;
                }
            }
        }
        /*Si n!=! entonces un primo sigue para ser sumado*/
        if (n != 1 && n != original_no) {
            while (n > 0) {
                pDigitSum = pDigitSum + n % 10;
                n = n / 10;
            }
        }
        // Todos los factores primos sumados
        // Ahora suma los digitos del número original
        int sumDigits = 0;
        while (original_no > 0) {
            sumDigits = sumDigits + original_no % 10;
            original_no = original_no / 10;
        }
        // Si la suma de los digistos en factores primos
        // y la suma de digitos en el número original son los mismos
        // entonces true, si no false
        return (pDigitSum == sumDigits);
    }

    public static void main(String[] args) {   //Encuentra todos los números
primos antes del limite
        // estos números son usados para encontrar factores primos
        sieveSundaram();
        System.out.println("Imprimiento primeros números de Smith"
                + " usando isSmith()");
        for (int i = 1; i < 500; i++) {
            if (isSmith(i)) {
                System.out.print(i + " ");
            }
        }
    }
}
```

12.47) Números esfénicos

Un número escénico es un entero positivo el cual es el producto de exactamente tres primos distintos, los primeros números esfénicos son 30, 42, 66, 70, 78, 102, 105, 110, 114,…

Dado un número n, determine si es un número esfenico o no.

Un número esfenico puede ser verificado generando los últimos factores primos de los números hasta n.

Luego podemos simplemente dividir el número por sus factores primos y luego ese número por sus factores primos, y así en Adelante, y luego verificar si el número tiene exactamente 3 factores primos distintos.

Complejidad de tiempo: O (nlog(n))

JAVA

```java
// Programa JAVA que verifica si un número
// es esfenico o no

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

public class SphenicNumber {

    static final int MAX = 1000;
    //Crea un vector donde almacenar primos
    // inicializa todas las entradas en 0
    static ArrayList<Integer> least_pf = new ArrayList<>(MAX);

    /* Esta función llena valores en least_pf[]
    de tal modod que el valor de least_pf[] almacene
    el factor primo más pequeño de i
    Se encuentra basado en la criba
    de Eratostenes*/
    public static void main(String[] args) {
        for (int i = 0; i < 1001; i++) {
            least_pf.add(0);
        }
        Collections.fill(least_pf, 0);
        leastPrimeFactor(MAX);
        for (int i = 1; i < 100; i++) {
            if (isSphenic(i)) {
                System.out.println(i + " ");
            }
        }

    }
    /*Función que verifica si un número es esfenico*/
    static boolean isSphenic(int n) {
        /*Almacena tres factores primos de n
        tenemos al menos 3 elementos en s*/
        Set<Integer> s = new HashSet<>();
        // Siga encontrando factores primos hasta que n sea 1
        while (n > 1) {
            // Encuenta al menos un factor del actual valor de n
            int lpf = least_pf.get(n);
            // Almacenamos actual tamaño de s para verificar si
            //algun factor primo se repite
```

```
            int init_size = s.size();
            // Inserta un valor primo al actual valor de n
            s.add(lpf);
            // si lpf repite un número o se pasa de 3, retorna falso.
            if (s.size() == init_size || s.size() > 3) // El mismo primo divide
            // un número más de una vez
            {
                return false;
            }
            // dividir n en lpf
            n /= lpf;
        }
        // True si el tamaño es 3
        return (s.size() == 3);
    }

    static void leastPrimeFactor(int n) {
        // un factor primo de 1 es 1
        least_pf.set(1, 1);
        // Almacena números primos para todos
        // los otros números
        for (int i = 2; i <= n; i++) {
            // least_pf[i] == 0 significa que i es primo
            if (least_pf.get(i) == 0) {
                least_pf.set(i, i);
                for (int j = 2 * i; j <= n; j += i) {
                    if (least_pf.get(j) == 0) {
                        least_pf.set(j, i);
                    }
                }

            }
        }
    }
}
```

12.48) Identidad de Cassini

Secuencias

Algunas de las muchas identidades conocidas que cumplen sus términos:

Identidad de Catalán	$F_n^2 - F_{n+r}F_{n-r} = (-1)^{n-r}F^2$
Identidad de d'Ocagne:	$F_m F_{n+1} - F_n F_{m+1} = (-1)^n F_{m-n}$
Identidad de Gelin - Cesàro:	$F_n^4 - F_{n-2}F_{n-1}F_{n+1}F_{n+2} = 1$
Identidad de Cassini:	$F_{n-1}F_{n+1} - F_n^2 = (-1)^n$
Identidad de Honsberger:	$F_{k+n} = F_{k-1}F_m + F_k F_{m+1}$

Guía del programador competitivo

Ilustración 12-33 Ecuaciones de secuencias semejantes a la identidad de Cassini

La identidad de Cassini y la identidad de Catalan son relaciones matemáticas ligadas con los números de la sucesión de Fibonacci, afirma que para cada número n-ésimo de la sucesión de Fibonacci, se cumple que:

- Fn-1 x Fn+1-F^2n=(-1)^n

JAVA

```
//Programa java que demuestra
// la identidad de Cassini

public class CassiniIdentity {
    // Retorna (-1)^n
    static int cassini(int n) {
        return (n & 1) != 0 ? -1 : 1;
    }

    public static void main(String args[]) {
        int n = 5;
        System.out.println(cassini(n));
    }
}
```

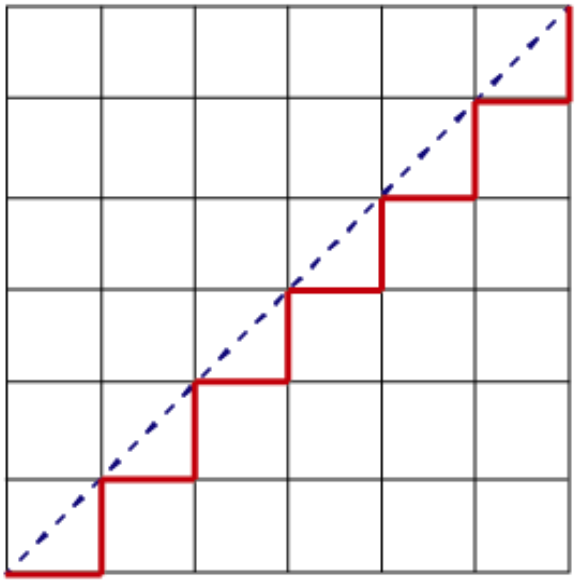
C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
int cassini(int n){
    return (n&1)!=0? -1 : 1;
}
int main(){
    int n=5;
    printf("%d\n",cassini(n));
}
```

12.49) Números de Catalan

Números de catalán

$$Cn = \frac{(2n)!}{(n+1)!n!}$$



Guía del programador competitivo

Ilustración 12-34 Formula de los números de Catalán

En combinatoria, los números de Catalan forman una secuencia de números naturales que aparece en varios problemas de conteo que habitualmente son recursivos. Obtienen su nombre del matemático belga Eugène Charles Catalan (1814–1894).

El n-ésimo número de Catalan se obtiene, aplicando coeficientes binomiales, a partir de la siguiente fórmula:

$$Cn = \frac{(2n)!}{(n+1)!\,n!}$$

JAVA

```
//Programa Java que recursivamente encuentra el
// nesimo número Catalan

public class CatalanNumber {
```

```
    public static int catalan(int n) {
        int res = 0;
        // Caso base
        if (n <= 1) {
            return 1;
        }
        for (int i = 0; i < n; i++) {
            res += catalan(i) * catalan(n - i - 1);
        }
        return res;
    }

    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            System.out.print(catalan(i) + " ");
        }
    }
}
```

12.50) Números de Charmichael

Un número n es un número de Carmichael si satisface la siguiente condición aritmética modular:

- pow(b, n-1) MOD n = 1,

Para todos los b en rango de 1 a n tal que b y n son relativos primos gcd(b, n) = 1 .

Dado un entero positivo n, encontrar si es un número de Carmichael, estos números tienen importancia en el método de Fermat para el test de primalidad.

JAVA

```
//Programa java que verifica si un número
// es un número de carmichael

public class CarmichaelNumbers {

//función que busca el GCD de dos números
    static int gcd(int a, int b) {
        if (a < b) {
            return gcd(b, a);
        }
        if (a % b == 0) {
            return b;
```

```
        }
        return gcd(b, a % b);
    }
    //Función que busca el pow(x,y)
    // bajo un modulo mod
    static int power(int x, int y, int mod) {
        if (y == 0) {
            return 1;
        }
        int temp = power(x, y / 2, mod) % mod;
        temp = (temp * temp) % mod;
        if (y % 2 == 1) {
            temp = (temp * x) % mod;
        }
        return temp;
    }

//Función que verifica si un número es número de charmichael
    static int isCarmichaelNumber(int n) {
        for (int b = 2; b < n; b++) {
            // Si 'b' es primo relativo de n
            if (gcd(b, n) == 1) // y pow(b, n-1)%n no es 1,
            // retorne falso
            {
                if (power(b, n - 1, n) != 1) {
                    return 0;
                }
            }
        }
        return 1;
    }

    public static void main(String args[]) {
        System.out.println(isCarmichaelNumber(500));
        System.out.println(isCarmichaelNumber(561));
        System.out.println(isCarmichaelNumber(1105));
    }
}
```

12.51) Secuencia Stern-Brocot

La secuencia de Stern Brocot es similar la secuencia de Fibonacci, pero es diferente en la forma en que la secuencia es generada.

1) Primero y segundo elemento de la secuencia es 1 y 1.

2) Considere el segundo miembro de la secuencia, luego sume el miembro considerado de la secuencia y su predecesor, por ejemplo (1+1=2) ahora 2 es el siguiente elemento de nuestra serie, la secuencia será [1,1,2].
3) Después de este elemento, nuestro siguiente elemento en la secuencia será considerado el elemento en nuestro segundo paso, ahora la secuencia seria [1,1,2,1].
4) De nuevo realizamos el paso 2, pero ahora consideramos el elemento 2 (Tercer elemento), entonces nuestro siguiente número de la secuencia será la suma de los números considerados, y su predecesor(2+1=3), la secuencia ahora será [1,1,2,1,3]
5) Como en el paso 3, el siguiente elemento será considerado, por ejemplo 2, la secuencia será [1,1,2,1,3,2]
6) El proceso continua, nuestro elemento considerado será 1(Cuarto elemento).

JAVA

```java
// Programa java que imprime
// La secuencia de Stern Brocot

import java.util.*;

public class SternBrocotSequence {

    static void SternSequenceFunc(ArrayList<Integer> BrocotSequence, int n) {
        // Ciclo que crea la secuencia
        for (int i = 1; BrocotSequence.size() < n; i++) {
            int considered_element = BrocotSequence.get(i);
            int precedent = BrocotSequence.get(i - 1);
            //Agregando la suma de los elementos considerados
            // y son precedentes
            BrocotSequence.add(considered_element + precedent);
            //  Agregando siguiente elemento considerado
            BrocotSequence.add(considered_element);
        }
        // Imprimiendo secuencia
        for (int i = 0; i < 15; ++i) {
            System.out.print(BrocotSequence.get(i) + " ");
        }
    }

    public static void main(String[] args) {
        int n = 15;
        ArrayList<Integer> BrocotSequence = new ArrayList<Integer>();
        // Agregando primer y segundo valor
        // A la secuencia
```

```
        BrocotSequence.add(1);
        BrocotSequence.add(1);
        SternSequenceFunc(BrocotSequence, n);
    }
}
```

12.52) Secuencia Sylvester

El en sistema numérico, La secuencia de Sylvester es una secuencia de enteros la cual cada miembro es un producto de los números previos, más uno, dado un entero positivo N, imprima los primeros N miembros de la secuencia.

Los números pueden ser muy largos, se usa %10^9 + 7.

La idea es correr un ciclo y tomar dos variables, inicializarlos como 1 y 2, uno para almacenar el producto hasta ahora, y el otro para almacenar el número actual el cual es el primer número +1 y por cada paso, multiplicar ambos usando aritmética modular, por ejemplo (a + b)%N = (a%N + b%N)%N donde N es un número modular.

JAVA

```java
//Implementación java de la secuencia Sylvester

public class SylvesterSequence {

    public static void printSequence(int n) {
        int a = 1; // Para almacenar el producto
        int ans = 2; // Para almacenar el número actual
        int N = 1000000007;
        //Ciclo hasta n
        for (int i = 1; i <= n; i++) {
            System.out.print(ans + " ");
            ans = ((a % N) * (ans % N)) % N;
            a = ans;
            ans = (ans + 1) % N;
        }
    }

    public static void main(String[] args) {
        int n = 6;
        printSequence(n);

    }
}
```

12.53) Secuencia Aliquot

Dado un número n, la tarea es imprimir su secuencia de Aliquot. La secuencia de Aliquot de un número empieza con el mismo, términos restantes de la secuencia son la suma de los divisores propios del término inmediatamente anterior.

Por ejemplo, la secuencia de Aliquot de 10, es 10,8,7,1,0. La secuencia puede repetir. Por ejemplo para 6, tenemos una secuencia infinita de todos los 6, En esos casos imprimimos el número repetido y paramos.

- Números los cuales tienen una secuencia de Aliquot repetitiva de longitud 1 son llamados números perfectos. Por ejemplo 6, suma de sus divisores propios es 6.
- Números que tienen una secuencia de Aliquot repetitiva de tamaño 2 son llamados números de Amicable. Por ejemplo 220 es un número de Amicable.
- Números que tienen secuencia de Aliquot repetitiva de tamaño 3 son llamados números sociales

Podemos generar la secuencia primero imprimiendo el número n y luego calculando los siguientes términos usando la suma de los divisores propios. Cuando computemos el siguiente termino, verificamos si nosotros ya habíamos visto ese término o no, si el término aparece de nuevo tenemos una secuencia repetida, imprimimos el mismo y rompemos el ciclo.

JAVA

```
//Implementación java de una apoximación
// de la secuencia Aliquot

import java.util.*;

public class AliquotSequence {
    // Función que calcula suma de
    // los divisores adecuados
```

```
    static int getSum(int n) {
        int sum = 0; // 1 es un divisor adecuado
        for (int i = 1; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {   // Si divisores son iguales, solo toma uno
                if (n / i == i) {
                    sum = sum + i;
                } else // Si no tome ambos
                {
                    sum = sum + i;
                    sum = sum + (n / i);
                }
            }
        }
        // Calcula la suma de los divisores adecuados
        return sum - n;
    }

    static void printAliquot(int n) {
        // Imprime el primer termino
        System.out.printf("%d ", n);
        TreeSet<Integer> s = new TreeSet<>();
        s.add(n);

        int next = 0;
        while (n > 0) {   // Calcula siguiente termino desde el anterior
            n = getSum(n);
            if (s.contains(n) && n != s.last()) {
                System.out.print("\nRepeats with " + n);
                break;
            }
            //Imprimir siguiente termino
            System.out.print(n + " ");
            s.add(n);
        }
    }

    public static void main(String[] args) {
        printAliquot(12);
    }
}
```

12.54) Secuencia Juggler

La secuencia de Juggles es una serie de números enteros la cual su primer término comienza con un entero positivo y los términos restantes son generados de los anteriores inmediatos números.

La secuencia Juggler comenzando con el número 3: 5, 11, 36, 6, 2, 1

La secuencia Juggler comenzando desde el número 9: 9, 27, 140, 11, 36, 6, 2, 1

Dado un número n tenemos que imprimir la secuencia Juggler de este número como el primer número de la secuencia

- Los términos en la secuencia de Juggler primero crecen hasta un valor pico, y luego empiezan a decrecer.
- El último término de la secuencia de Juggler es siempre 1.

JAVA

```
// Implementación java de la secuenca de Juggler

public class JugglerSequence {

    static void printJuggler(int n) {
        int a = n;
        // Imprime el primer termino
        System.out.print(a + " ");
        // Calcula terminos hasta que el ultimo no sea 1
        while (a != 1) {
            int b = 0;
            // Verifica si los previos terminos son pares o impares
            if (a % 2 == 0) // calcular siguiente termino
            {
                b = (int) Math.floor(Math.sqrt(a));
            } else // para impar anterior, calcular
            // Siguiente termino
            {
                b = (int) Math.floor(Math.sqrt(a)
                        * Math.sqrt(a) * Math.sqrt(a));
            }
            System.out.print(b + " ");
            a = b;
        }
    }

    public static void main(String[] args) {
        printJuggler(3);
        System.out.println();
        printJuggler(9);
    }
}
```

12.55) Secuencia Moser de Brujin

Dado un entero n, imprima los primeros n términos de la secuencia de Moser de Brujin.

La secuencia de Moser de Brujin es una secuencia obtenida adicionando las distintas potencias del número 4, por ejemplo 1,4,16,6..

Debe notarse aquí que cualquier número el cual es la suma de potencias de 4 no distintas no son parte de la secuencia, por ejemplo 8 no es parte de la secuencia debido a que es formado de la suma de no distintas potencias de 4, que son 4 y 4.

Por lo tanto cualquier número el cual no sea una potencia de 4 y está presente en la secuencia debe ser la suma de distintas potencias de 4.

Por ejemplo, 21 es parte de la secuencia, incluso a través de que no es una potencia de 4 porque es la suma de distintas potencias de 4, que son 1,4 y 16.

JAVA

```java
// Codigo java que genera los primeros n terminos
// de la secuencia de  Moser-de Bruijn

public class MoserdeBruijnSequence {

    public static int gen(int n) {
        // S(0) = 0
        if (n == 0) {
            return 0;
        } // S(1) = 1
        else if (n == 1) {
            return 1;
        } // S(2 * n) = 4 * S(n)
        else if (n % 2 == 0) {
            return 4 * gen(n / 2);
        } // S(2 * n + 1) = 4 * S(n) + 1
        else if (n % 2 == 1) {
            return 4 * gen(n / 2) + 1;
        }
        return 0;
    }

    public static void moserDeBruijn(int n) {
        for (int i = 0; i < n; i++) {
            System.out.print(gen(i) + " ");
        }
        System.out.println();
    }

    public static void main(String args[]) {
        int n = 15;
        System.out.println("Primeros " + n
```

```
                + " terminos de la secuencia de  "
                + "Moser-de Bruijn  : ");
        moserDeBruijn(n);
    }
}
```

12.56) Secuencia Newman-Conway

La secuencia de Newman-Conway es aquella que genera la siguiente secuencia de enteros:

1 1 2 2 3 4 4 4 5 6 7 7…

En términos matemáticos, la secuencia P(n) de Newman-Conway es definida por la siguiente relación de recurrencia:

- P(n) = P(P(n - 1)) + P(n - P(n - 1)) with seed values P(1) = 1 and P(2) = 1

Dado un número n, imprima el enésimo número de la secuencia Newman-Conway

Complejidad de tiempo: O(n)

JAVA

```
// Programa java que encuentra el nesimo
// elemento de la secuencia de  Newman-Conway

public class NewmanConwaySequence {

    static int sequence(int n) {
        if (n == 1 || n == 2) {
            return 1;
        } else {
            return sequence(sequence(n - 1))
                    + sequence(n - sequence(n - 1));
        }
    }

    static int sequenceDP(int n) {
        int f[] = new int[n + 1];
        int i;
        f[0] = 0;
        f[1] = 1;
        f[2] = 1;

        for (i = 3; i <= n; i++) {
            f[i] = f[f[i - 1]] + f[i - f[i - 1]];
        }
```

```
        return f[n];
    }

    public static void main(String args[]) {
        int n = 10;
        System.out.println(sequence(n));
    }
}
```

12.57) Secuencia Padovan

La secuencia de Padovan es similar a la secuencia de Fibonacci, con una secuencia recursiva similar cuya fórmula es:

- P(n) = P(n-2) + P(n-3)
- P(0) = P(1) = P(2) = 1

JAVA

```
// Programa JAVA que encuentra el enesimo termino
// de la secuencia de Padovan
// usando programación dinamica

public class PadovanSequence {

    /* Función que calcula el número de Padovan*/
    static int pad(int n) {   //0,1 y 2 número de la serie es 1
        int pPrevPrev = 1, pPrev = 1,
                pCurr = 1, pNext = 1;
        for (int i = 3; i <= n; i++) {
            pNext = pPrevPrev + pPrev;
            pPrevPrev = pPrev;
            pPrev = pCurr;
            pCurr = pNext;
        }
        return pNext;
    }

    public static void main(String args[]) {
        int n = 12;
        System.out.println(pad(n));
    }
}
```

12.58) Secuencia Recaman

Dado un entero n, imprima los primeros n elementos de la secuencia de Recaman

Es básicamente una función con dominio y co dominio como números naturales y 0, su recursividad se define de la siguiente manera.

Específicamente, dejamos a(n) denotar el (n+1)simo termino (0 ya está ahí).

La regla dice:

- a(0) = 0,
- Si n > 0 y el número no está incluido en la secuencia
- a(n) = a(n - 1) - n
- si no
- a(n) = a(n-1) + n.

Complejidad de tiempo: O(n^2)

JAVA

```java
// Programa java que imprime el nesimo número
// de la secuencia de Recaman

public class RecamanSequence {

    static void recaman(int n) {
        //Crea un array que almacenara los terminos
        int arr[] = new int[n];
        // Primer termino de la secuencia es siempre 0
        arr[0] = 0;
        System.out.print(arr[0] + " ,");
        // Llena terminos restantes usando formula recursiva
        for (int i = 1; i < n; i++) {
            int curr = arr[i - 1] - i;
            int j;
            for (j = 0; j < i; j++) {
   // si arr[i-1] - i es negativo or ya existe
                if ((arr[j] == curr) || curr < 0) {
                    curr = arr[i - 1] + i;
                    break;
                }
            }
            arr[i] = curr;
            System.out.print(arr[i] + ", ");
```

```
        }
    }

    public static void main(String[] args) {
        int n = 17;
        recaman(n);
    }
}
```

12.59) **Problemas de repaso**

Ejercicios en Online Judge

440-Eeny Meeny Moo

355-The Bases Are Loaded

498-Polly the Polynomial

550- Multiplying by Rotation

919- Cutting Polyominones

10539- Almost Prime Numbers

686-Goldbach's Conjeture (II)

847-A Multiplication Game

10299-Relatives

10407-Simple division

Ejercicios en CodeChef

FCTRLZ

RPD

Capitulo 13. Geometría y trigonometría

La geometría es la rama de las matemáticas que ocupa el estudio de las propiedades de las figuras en planos y en espacios, esta área permite el cálculo de diversas medidas necesarias para la obtención de información acerca de un fenómeno u objeto, y con esta información realizar diversas soluciones a problemas de programación.

13.1) Longitud de arco

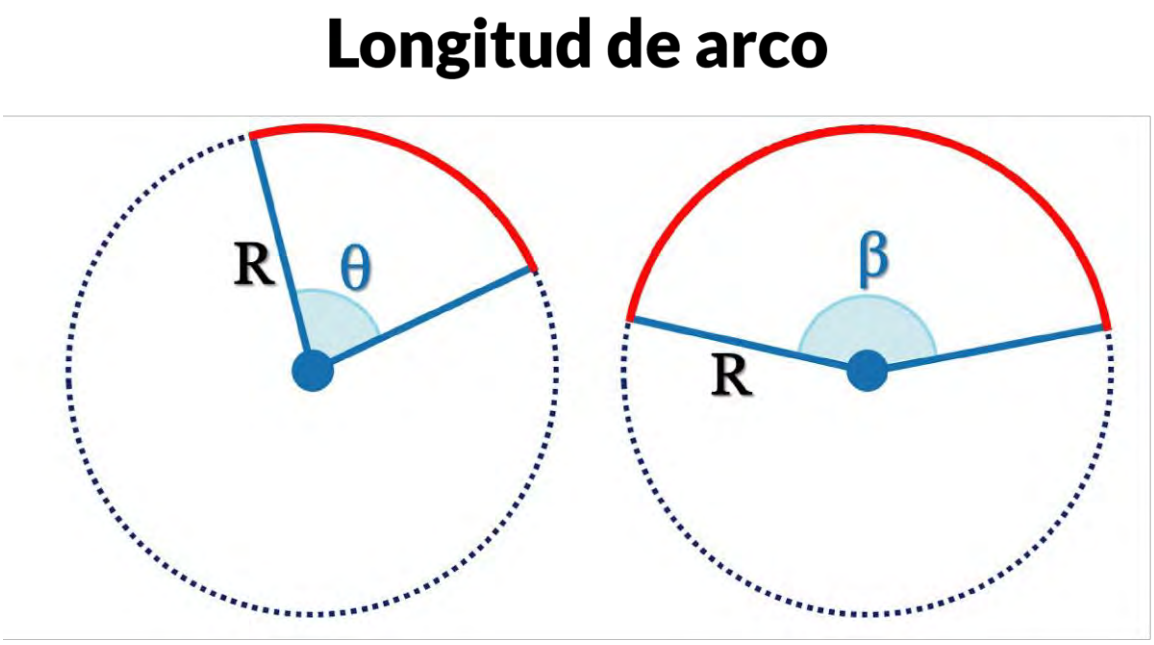


Ilustración 13-1 Arco dentro de un circulo de radio R

Un ángulo se forma cuando dos rayos se encuentran en un punto en un plano, esos rayos forman los lados del ángulo, y el punto de encuentro es referido como el vértice del ángulo. Hay que tener en cuenta que el plano que forma un ángulo no tiene que ser un plano Euclidiano, ahora en un círculo el largo de un arco es una porción de la circunferencia.

Dado un ángulo y el diámetro de un círculo, podemos calcular el largo de un arco usando la fórmula:

- ArcLength = (2 * pi * radio) * (ángulo / 360)
- Donde pi = 22/7,
- Diámetro = 2 * radio,
- El ángulo está en grados.

Si el ángulo es mayor o igual a 360 grados, entonces el largo del arco no puede ser calculado desde que ningún ángulo es posible.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
//Programa java que calcula
//la longitud de un arco

public class ArcLength {

    static double arcLength(double diameter,
            double angle) {
        double pi = 22.0 / 7.0;
        double arc;
        if (angle >= 360) {
            System.out.println("Angulo no puede ser fromado");
            return 0;
        } else {
            arc = (pi * diameter) * (angle / 360.0);
            return arc;
        }
    }

    public static void main(String args[]) {
        double diameter = 25.0;
        double angle = 45.0;
        double arc_len = arcLength(diameter, angle);
        System.out.println(arc_len);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
#define PI 22/7
//---------------//
using namespace std;
typedef long double ld;
ld arcLen(ld diametro,ld angulo){
```

```
    ld arc;
    if(angulo>=360){
        cout<<"el angulo no puede ser formado"<<endl;
        return 0;
    }
    else{
        arc=(PI*diametro)*(angulo/360);
        return arc;
    }
}
ld DegToRad(ld d){
    return ((d * PI )/ 180);
}
ld RagToDeg(ld r){
    return ((r * 180 )/ PI);
}
int main() {
    ld d=5000;
    ld angle=140.5;
    printf("%.5llf",arcLen(d,angle));
}
```

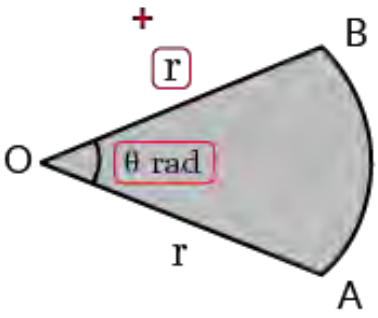
PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def arclen(diametro, angulo):
    pi = 22 / 7
    arc = 0
    if angulo >= 360:
        wr(f'Angulo no puede ser formado')
        return 0
    else:
        arc = (pi * diametro) * (angulo / 360)
        return arc

d = 5000
angle = 140.5
wr(f'{arclen(d,angle)}')
```

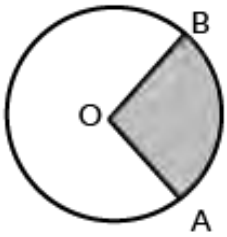
13.2) Area de un sector circular

Área de un sector circular

Él área de un Sector Ciruclar es igual a la mitad del cuadrado del valor de su radio multiplicado por radianes de su ángulo central.



$$S = \frac{r^2}{2}\theta$$



Notación: $S_{\sphericalangle AOB}$ = Área del Sector Circular AOB

Guía del programador competitivo

Ilustración 13-2 Elementos de un sector circular

Un sector circular o un sector círculo, es la porción de un disco encerrado por dos radios y un arco, donde el área más pequeña es conocida como el sector menor y el grande como el sector mayor.

- Sector =(pi*r^2)*(Angulo/360)

El área de un sector es similar al cálculo del área de un círculo, solo se le multiplica el área de un círculo con el ángulo del sector.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Programa java que encuentra el area de un sector circular

public class AreaCircularSector {

    static void SectorArea(double radius, double angle) {
        if (angle >= 360) {
            System.out.println("Angle not possible");
        } // Calculando el area
```

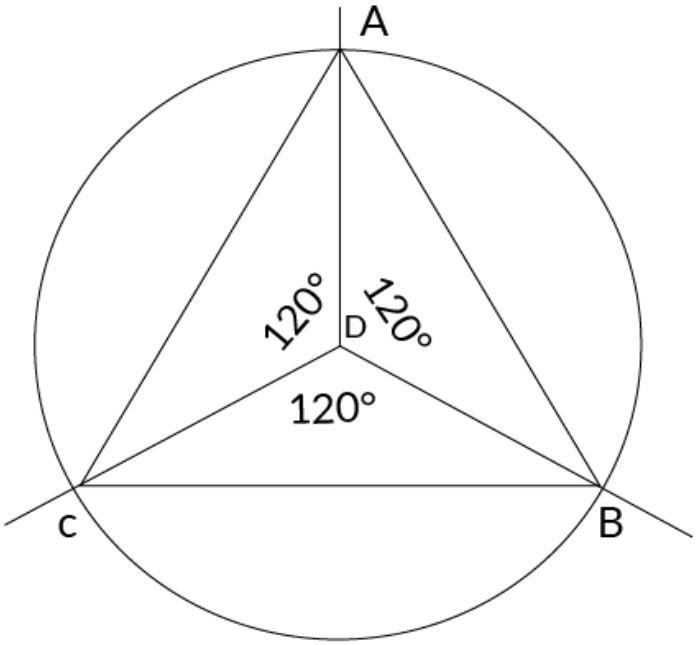
```
        else {
            double sector = ((22 * radius * radius) / 7)
                    * (angle / 360);
            System.out.println(sector);
        }
    }

    public static void main(String[] args) {
        double radius = 9;
        double angle = 60;
        SectorArea(radius, angle);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define PI 22/7
//---------------//
using namespace std;
typedef long double ld;
void setCir(ld radio,ld angle){
    if(angle>=360){
        cout<<"el angulo no es posible"<<endl;
    }else{
        ld sector=((22*radio*radio)/7)*(angle/360);
        printf("%.8llf\n",sector);
    }
}
int main() {
    ld radio=6;
    ld angle=22.7;
    setCir(radio,angle);
}
```

13.3) Circulo circunscrito en un triángulo equilátero

Círculo en base a un triángulo



Guía del programador competitivo

Ilustración 13-3 Circulo dibujado a partir de un triángulo

Dado el largo de los lados de un triángulo equilátero, necesitamos encontrar el área de un circuncirculo del triángulo dado. Todos los lados del triángulo equilátero son de igual largo, y todos los ángulos interiores son de 60 grados.

Las propiedades de un circuncirculo son las siguientes:

- El centro del circuncirculo es el punto donde las medianas del triángulo equilátero se intersectan.
- El circulo circuncirculo de un triángulo equilátero es hecho a través de los tres vértices de un triángulo equilátero.
- El radio de un circuncirculo de un triángulo equilátero es igual a (a/√3), donde 'a' es el largo de los lados del triángulo equilátero

La fórmula usada para calcular el área de un círculo circuncirculo es:

- (π*a^2)/3

Donde a es el largo del lado del triángulo dado.

Sabemos que el área de un circulo es π*r^2, donde r es el radio del circulo dado.

También sabemos que el radio de un circuncirculo de un triángulo equilátero = (Lado del triángulo/√3.

Por lo tanto, área = π*r^2 = π*a^2/3.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
// código Java para encontrar el área de
// un círculo curcunscripto a un triángulo equilátero

public class CircumscribedCircleOfEquilateral {

    static double PI = 3.14159265;
    //Función que encuentra el area
    // del circulo circumscrito
    public static double area_cicumscribed(double a) {
        return (a * a * (PI / 3));
    }

    public static void main(String[] args) {
        double a = 6.0;
        System.out.println("Area of circumscribed circle is :"
                + area_cicumscribed(a));
    }
}
```

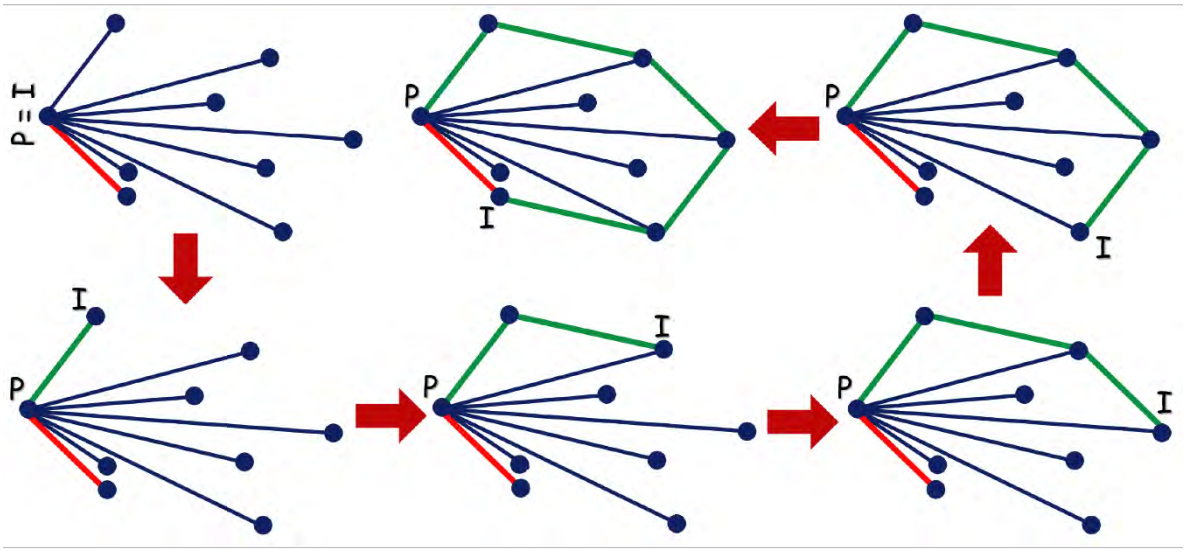
C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
#define PI 22/7
//---------------//
using namespace std;
typedef long double ld;
ld areaEquilatero(ld lado){
    return (lado*lado*(PI/3));
}
int main() {
    ld a=6;
    printf("%.8llf",areaEquilatero(a));
}
```

PYTHON

```python
from sys import stdin, stdout
import math
rl = stdin.readline
wr = stdout.write

def area_circumscribed(a):
    return (a * a * (math.pi / 3))

a = int(rl())
wr(f'Area de circulo circunscrito: {area_circumscribed(a)}')
```

13.4) Convex Hull (Envoltorio convexo)



Guia del programador competitivo

Ilustración 13-4 Búsqueda del casco convexo

Se define como la envolvente convexa, envoltura convexa o cápsula convexa de un conjunto de puntos X de dimensión n como la intersección de todos los conjuntos convexos que contienen a X.

En el caso particular de puntos en un plano, si no todos los puntos están alineados, entonces su envolvente convexa corresponde a un polígono convexo cuyos vértices son algunos de los puntos del conjunto inicial de puntos.

Una forma intuitiva de ver la envolvente convexa de un conjunto de puntos en el plano, es imaginar una banda elástica estirada que los encierra a todos. Cuando se libere la banda elástica tomará la forma de la envolvente convexa.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(n^2) **Promedio:** O(n*puntostomados)

JAVA

```java
// Programa java que halla el casco convexo de un set de puntos

import java.util.*;

public class ConvexHull {

    static class Point {
        int x, y;
        Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
    // Para encontrar la orientación de la tripleta
    //la función retorna lo siguiente
    // 0 --> p, q y r son colineares
    // 1 --> Manecillas de reloj
    // 2 --> en contra de las manecillas
    static int orientation(Point p, Point q, Point r) {
        int val = (q.y - p.y) * (r.x - q.x)
                - (q.x - p.x) * (r.y - q.y);
        if (val == 0) {
            return 0;  // colinear
        }
        return (val > 0) ? 1 : 2; // Manecillas
    }
    //Imprime el casco convexo del set de puntos

    static void convexHull(Point points[], int n) {
        //Debe haber al menos 3 puntos
        if (n < 3) {
            return;
        }
        // Inicializar resultado
        ArrayList<Point> hull = new ArrayList<>();
```

```
        // Encuentra el punto de más a la izquierda
        int l = 0;
        for (int i = 1; i < n; i++) {
            if (points[i].x < points[l].x) {
                l = i;
            }
        }
        // Comienza desde el punto más a la izquierda, sigue moviéndose.
        // en sentido antihorario hasta llegar al punto de inicio
        // otra vez. Este ciclo corre O (h) veces donde h es
        // número de puntos en resultado o salida
        int p = l, q;
        do {
            // Agrega el punto actual al resultado
            hull.add(points[p]);
            // Buscar un punto 'q' tal que
            // la orientación (p, x, q) es antihorario
            // para todos los puntos 'x'. La idea es mantener
            // pista de los últimos visitados más contra reloj
            // punto en q. Si algún punto 'i' es más
            // en sentido contrario a las agujas del reloj que q, luego
//actualiza q.
            q = (p + 1) % n;
            for (int i = 0; i < n; i++) {
                //Si i es más antihorario que el actual q, actualice q
                if (orientation(points[p], points[i], points[q])
                        == 2) {
                    q = i;
                }
            }
            //Ahora q es el más antihoriario con respecto
            // a p, ubica p como q para la siguiente iteración
            // asi q es agregado al casco resultado
            p = q;
        } while (p != l);
        // Mientras no vengamos del primer punto
        // Imprima resultado
        for (Point temp : hull) {
            System.out.println("(" + temp.x + ", "
                    + temp.y + ")");
        }
    }

    public static void main(String[] args) {
        Point points[] = new Point[7];
        points[0] = new Point(0, 3);
        points[1] = new Point(2, 3);
        points[2] = new Point(1, 1);
        points[3] = new Point(2, 1);
        points[4] = new Point(3, 0);
        points[5] = new Point(0, 0);
        points[6] = new Point(3, 3);
        int n = points.length;
        convexHull(points, n);
```

```
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define x first
#define y second
//---------------//
using namespace std;
typedef pair<int, int> point;

int orientation(point p, point q, point r) {
    int val = (q.y - p.y)*(r.x - q.x)-(q.x - p.x)*(r.y - q.y);
    if (val == 0) {
        return 0;
    }
    return (val > 0 ? 1 : 2);
}

void convexHull(point po[], int n) {
    if (n < 3) {
        return;
    }
    vector<point>hull;
    int l = 0;
    for (int i = 1; i < n; i++) {
        if (po[i].x < po[l].x) {
            l = i;
        }
    }
    int p = l, q;
    do {
        hull.push_back(po[p]);
        q = (p + 1) % n;
        for (int i = 0; i < n; i++) {
            if (orientation(po[p], po[i], po[q]) == 2) {
                q = i;
            }
        }
        p = q;
    } while (p != l);
    for (point punto : hull) {
        printf("{%d,%d}\n", punto.x, punto.y);
    }
}

int main() {
    point po[7];
    po[0] = make_pair(0, 3);
    po[1] = make_pair(2, 3);
    po[2] = make_pair(1, 1);
    po[3] = make_pair(2, 1);
    po[4] = make_pair(3, 0);
```

```
    po[5] = make_pair(0, 0);
    po[6] = make_pair(3, 3);
    convexHull(po, 7);
}
```

13.5) Sumatoria de cortes

Dado el número de cortes, encuentre el máximo número de posibles piezas.

Este problema no es más que el problema del cartero flojo, y tiene la siguiente formula.

Máximo número de piezas = 1 + n*(n+1)/2

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
//Programa que calcula el maximo número
// de piezas de pizza dados el número de cortes

public class CutsSumatory {

    static int findMaximumPieces(int n) {
        return 1 + n * (n + 1) / 2;
    }

    public static void main(String arg[]) {

System.out.print(findMaximumPieces(3));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int findMaxSum(int n){
    return 1 + n* (n+1) / 2;
}
int main() {
    int a=5;
    cout<<findMaxSum(a)<<endl;
}
```

13.6) Formula de Heron

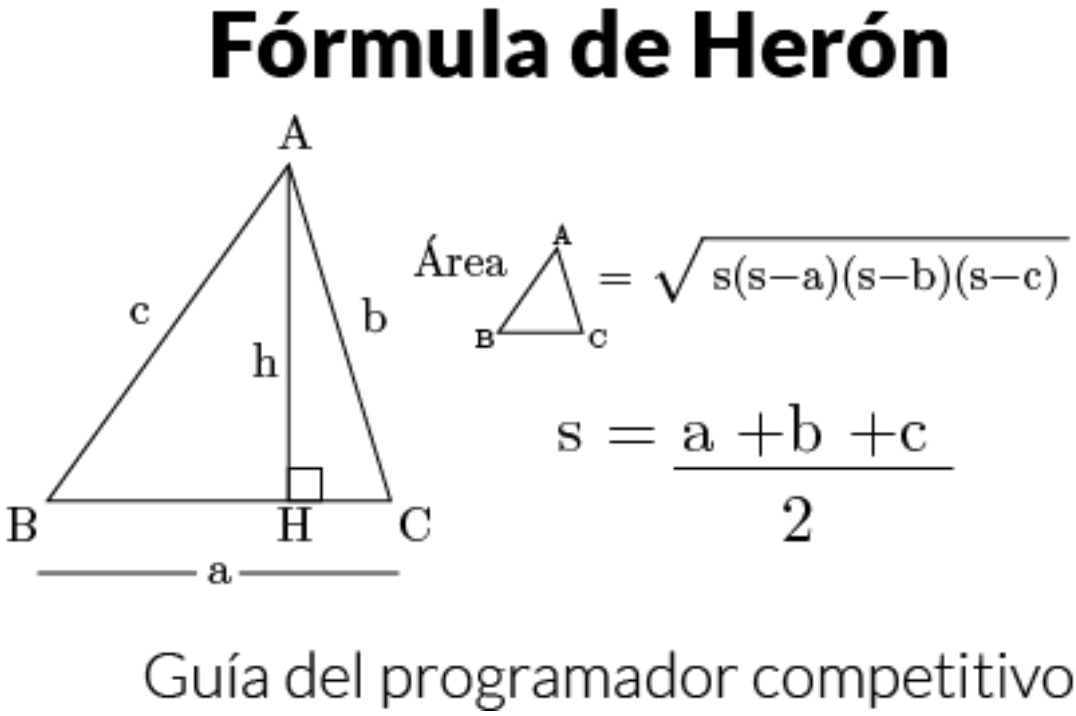


Ilustración 13-5 Área de un triángulo por medio de la formula de Herón

En geometría plana elemental la fórmula de Herón, cuya invención se atribuye al matemático griego Herón de Alejandría, da el área de un triángulo conociendo las longitudes de sus tres lados a, b y c:

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
//Programa Java que calcula el area
// De un triángulo usando la formula de Herón

public class HeronFormula {

    public static void main(String[] args) {
        //Enviar las coordenadas de los vertices
        System.out.println(heron(1, 0, -1, 0, 0, 2));
    }
    //función que usa la formula de Herón
    static double heron(double x1, double y1, double x2,
            double y2, double x3, double y3) {
        double a = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
        double b = Math.sqrt((x1 - x3) * (x1 - x3) + (y1 - y3) * (y1 - y3));
        double c = Math.sqrt((x3 - x2) * (x3 - x2) + (y3 - y2) * (y3 - y2));
        double s = (a + b + c) / 2.0;
        double A = Math.sqrt(s * (s - a) * (s - b) * (s - c));
        return A;
    }

}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
#define point pair<double,double>
#define x first
#define y second
//---------------//
using namespace std;

double heron(double x1,double y1,double x2,double y2,double x3,double y3){
    double a =std::sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    double b =std::sqrt((x2-x3)*(x2-x3)+(y2-y3)*(y2-y3));
    double c =std::sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
    double s=(a+b+c)/2;
    return (std::sqrt(s*(s-a)*(s-b)*(s-c)));

}
int main() {
    cout<<heron(1,0,-1,0,0,2)<<endl;
}
```

13.7) Area de un hexágono

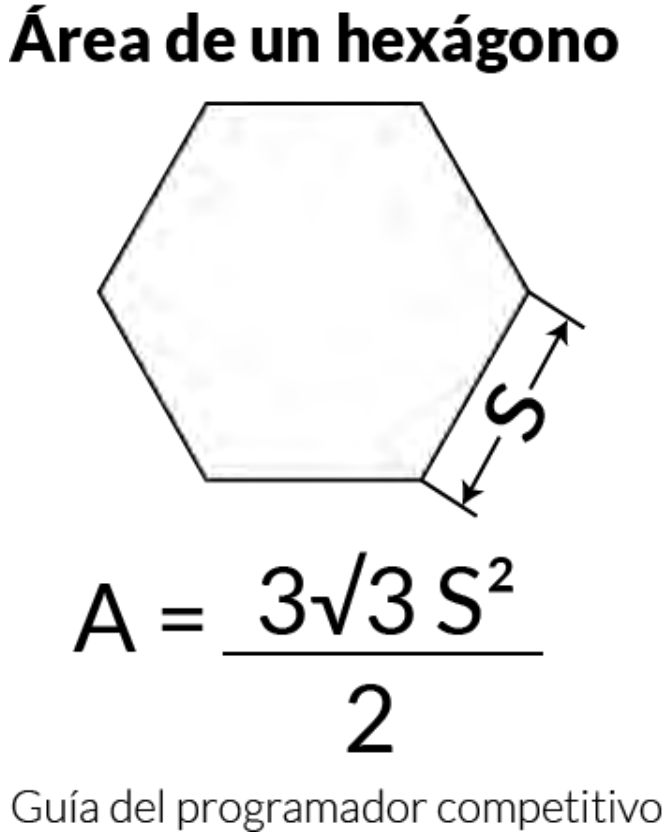


Ilustración 13-6 Elementos necesarios para calcular el área de un hexágono

Un hexágono es una figura geométrica de 6 lados, en dos dimensiones, el total de la suma de los ángulos internos de cualquier hexágono es 720°. Un hexágono regular tiene 6 simetrías rotacionales, y 6 simetrías reflectivas, todos los ángulos internos son de 120 grados.

Ahí hay principalmente 6 triángulos equiláteros de lado n y el área de un triángulo equilátero es sqrt (3)/4*n*n. Desde el hexágono, hay en total 6 triángulos equiláteros con lado n, el área del hexágono se convierte en (3*sqrt (3)/2) * n * n)

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

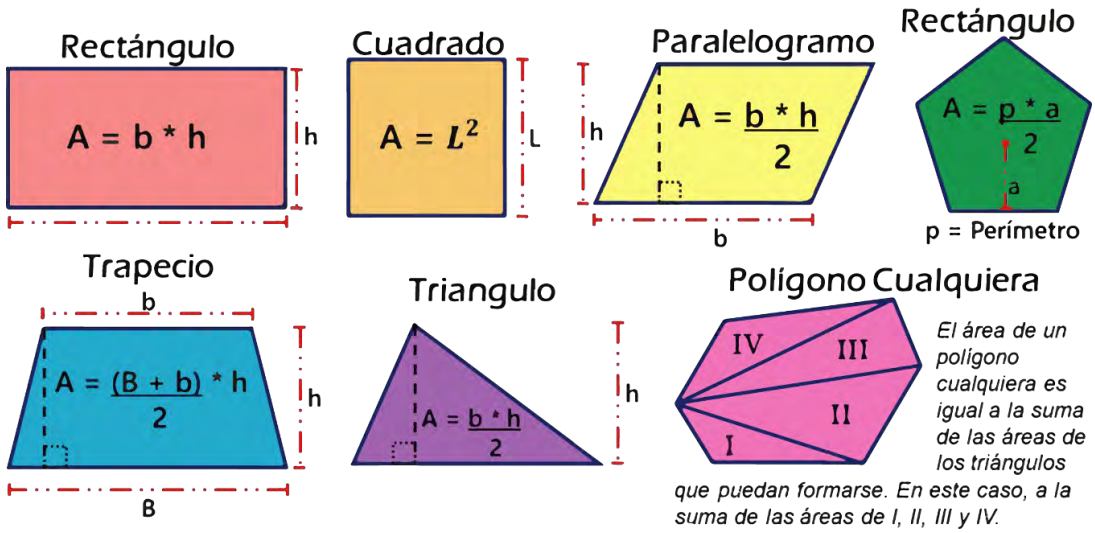
```
//Programa Java que calcula el area de un Hexagono

public class HexagonArea {
```

```
    public static double hexagonArea(double s) {
        return ((3 * Math.sqrt(3)
                * (s * s)) / 2);
    }

    public static void main(String[] args) {
        // Largo de un lado
        double s = 4;
        System.out.print("Area: "
                + hexagonArea(s));
        System.out.println("");
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
double hexArea(double s){
    return (3*sqrt(3)*(s*s))/2;
}
int main() {
    double s=6;
    printf("%.8f\n",hexArea(6));
}
```

13.8) Area de un polígono



Guia del programador competitivo

Ilustración 13-7 Áreas de diferentes polígonos

Dadas ordenadamente las coordenadas de un polígono, con n vértices, encontrar el área del polígono. Aquí ordenado significa que las coordenadas son dadas en sentido horario o anti horario, desde el primer vértice hasta el último.

Podemos dividir un polígono en triángulos, la fórmula del área es derivada de tomar cada camino AB, y calcular el área del triángulo ABO, con un vértice de origen O, tomando el producto cruz (El cual da el área de un paralelogramo) y dividiendo por 2. Mientras pasemos alrededor del polígono, estos triángulos con área positiva o negativa se sobreponen, y las ares en medio del origen y el polígono pueden ser canceladas y sumadas a 0, mientras solo haya área interna, el triángulo de referencia de mantiene.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```
//Programa Java que calcula el area de un poligono

import java.awt.Point;
```

```java
import java.awt.Polygon;
import java.util.Arrays;

public class AreaOfPolygon {

    public static void main(String[] args) {
        //Crea el poligono con los puntos dados
        Polygon p = new Polygon();
        p.addPoint(0, 2);
        p.addPoint(2, 2);
        p.addPoint(2, 0);
        p.addPoint(0, 0);
        System.out.println(area(p));
    }
    //Por medio de triángulos va calculando el area completa
    static int signedTriangleArea(Point a, Point b, Point c) {
        return a.x * b.y - a.y * b.x + a.y * c.x - a.x * c.y + b.x * c.y - c.x *
b.y;
    }
    static boolean ccw(Point a, Point b, Point c) {
        return signedTriangleArea(a, b, c) > 0;
    }
    // Verifica si los puntos son colineares
    static boolean collinear(Point a, Point b, Point c) {
        return signedTriangleArea(a, b, c) == 0;
    }
    //Calcula la distancia entre dos puntos
    static double distance(Point p1, Point p2) {
        double dx = p1.x - p2.x;
        double dy = p1.y - p2.y;
        return Math.sqrt(dx * dx + dy * dy);
    }
    //Función que calcula el area de cualquier poligono
    static double area(Polygon poly) {
        int N = poly.npoints;
        int[] x = poly.xpoints;
        int[] y = poly.ypoints;
        Point[] p = new Point[N];
        for (int i = 0; i < N; ++i) {
            p[i] = new Point(x[i], y[i]);
        }
        final Point first;
        int min = 0;
        for (int i = 1; i < N; i++) {
            if (p[i].y < p[min].y) {
                min = i;
            } else if (p[i].y == p[min].y) {
                if (p[i].x < p[min].x) {
                    min = i;
                }
            }
        }
        first = p[min];
        p[min] = p[0];
```

```
        p[0] = first;
        //Usar expresion lambda en vez de comparator
        Arrays.sort(p, 1, N, (Point p1, Point p2) -> {
            if (collinear(first, p1, p2)) {
                return Double.compare(distance(first, p1), distance(first, p2));
            }
            if (ccw(first, p1, p2)) {
                return -1;
            } else {
                return 1;
            }
        });
        double A = 0.0;
        for (int i = 0; i < N; i++) {
            int j = (i + 1) % N;
            A += p[i].x * p[j].y - p[j].x * p[i].y;
        }
        return A / 2.0;

    }
}
```

13.9) Intersección de líneas

Intersección de Líneas

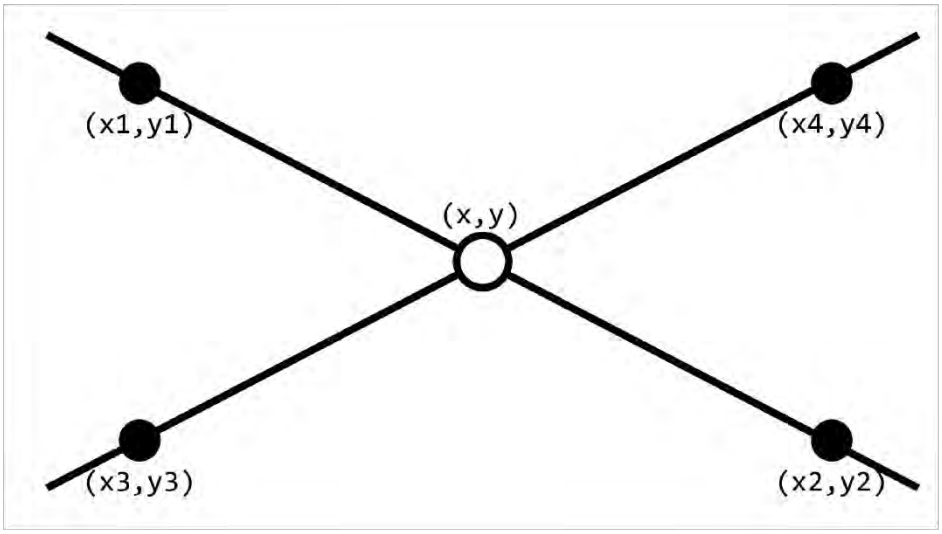


Ilustración 13-8 Intersección de dos líneas

Dados puntos A y B, correspondientes a la línea AB y los puntos P y Q, correspondientes a la línea PQ, encuentre el punto de intersección de estas líneas. Los puntos están dados en un plano 2D con sus coordenadas X y Y.

Primero que todo, asumimos que tenemos dos puntos (x1,y1,) y (x2,y2). Ahora encontramos la ecuación de la línea formada por esos puntos.

Dejaremos que las líneas dadas sean:

- a1x + b1y = c1
- a2x + b2y = c2

Tenemos que ahora resolver estas dos ecuaciones para encontrar el punto de intersección, para resolver esto, multiplicamos a1 por b2 y a2 por b1, esto nos da:

- a1b2x + b1b2y = c1b2
- a2b1x + b2b1y = c2b1

Restando esto obtenemos:

- (a1b2 – a2b1) x = c1b2 – c2b1

Esto nos da el valor de x. similarmente podemos encontrar el valor de y, (x,y) nos da el punto de intersección.

Esto nos da el punto de intersección de dos líneas, pero si nos dan segmentos de línea en vez de líneas, tenemos que revisar el punto que computado yace en ambos segmentos de líneas,

Si el segmento de línea es especificado por los puntos (x1,y2) y (x2,y2), entonces debemos verificar si (x,y) está en el segmento que tenemos de la siguiente manera:

- min (x1, x2) <= x <= max (x1, x2)
- min (y1, y2) <= y <= max (y1, y2)

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
//Implementación Java que encuentra el punto de
//interseccion de dos lineas

public class LineLineIntersection {
    //Clase usada para almacenar las coordenadas X y la Y
    // de un punto respectivo
    static class Point {
        double x, y;
        public Point(double x, double y) {
            this.x = x;
            this.y = y;
        }
        // Metodo usado para imprimir las cordenadas
        // X y Y de un punto
        static void displayPoint(Point p) {
            System.out.println("(" + p.x + ", " + p.y + ")");
        }
    }

    static Point lineLineIntersection(Point A, Point B, Point C, Point D) {
        // Linea AB representada como a1x + b1y=c1
        double a1 = B.y - A.y;
        double b1 = A.x - B.x;
        double c1 = a1 * (A.x) + b1 * (A.y);
        // Linea CD representada como a2x + b2y=c2
        double a2 = D.y - C.y;
```

```
        double b2 = C.x - D.x;
        double c2 = a2 * (C.x) + b2 * (C.y);
        double determinant = a1 * b2 - a2 * b1;
        if (determinant == 0) {
            //Las lineas son pararelas, esto es simplificado
            //Retornando un par de FLT_MAX
            return new Point(Double.MAX_VALUE, Double.MAX_VALUE);
        } else {
            double x = (b2 * c1 - b1 * c2) / determinant;
            double y = (a1 * c2 - a2 * c1) / determinant;
            return new Point(x, y);
        }
    }

    public static void main(String args[]) {
        Point A = new Point(1, 1);
        Point B = new Point(4, 4);
        Point C = new Point(1, 8);
        Point D = new Point(2, 4);
        Point intersection = lineLineIntersection(A, B, C, D);
        if (intersection.x == Double.MAX_VALUE
                && intersection.y == Double.MAX_VALUE) {
            System.out.println("la linea AB y CD son paralelas.");
        } else {
            System.out.print("La intersección de las lineas AB "+ "y CD es: ");
            Point.displayPoint(intersection);
        }
    }
}
```

13.10) Punto medio de una línea

Punto medio de una línea

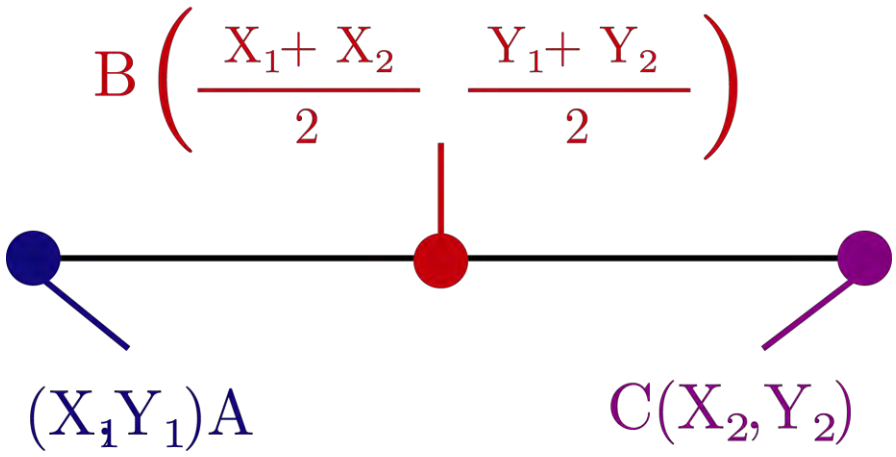


Ilustración 13-9 Punto medio de una línea

Dado dos coordenadas de una línea iniciando en (x1,y1) y terminando en (x2,y2) encontrar el punto medio de una línea.

El punto medio de dos puntos (x1,y1) y (x2,y2) es el punto M encontrado con la siguiente fórmula:

- M = ((x1+x2)/2 , (y1+y2)/2)

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
//Programa java para buscar
//el punto medio de una linea

public class LineMidPoint {

    static void midpoint(int x1, int x2,
            int y1, int y2) {
        System.out.print((x1 + x2) / 2
                + " , " + (y1 + y2) / 2);
    }
```

```
    public static void main(String[] args) {
        int x1 = -1, y1 = 2;
        int x2 = 3, y2 = -6;
        midpoint(x1, x2, y1, y2);

    }
}
```

C++

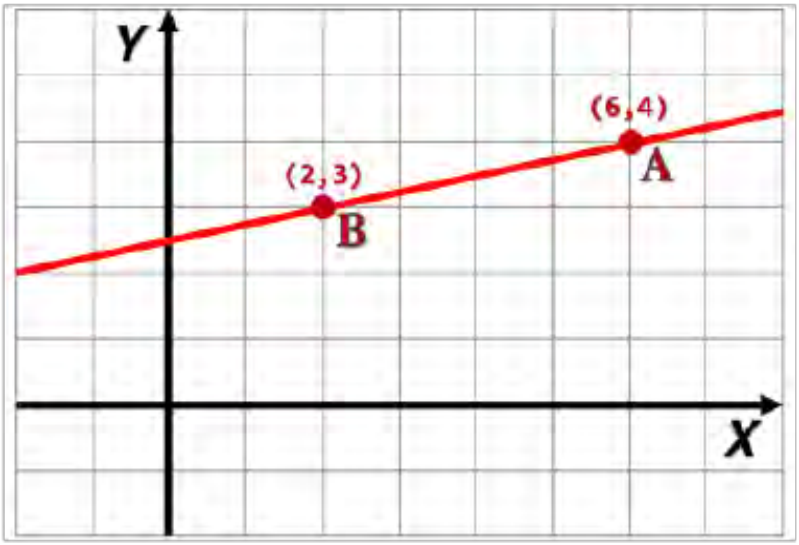
```
#include <iostream>

using namespace std;

string midPoint(int x1, int x2, int y1, int y2){
    string res1 = std::to_string((x1+x2)>>1);
    string res2 = std::to_string((y1+y2)>>1);
    return (res1+", "+res2);
}

int main()
{
    ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
    int x1, x2; cin>>x1>>x2;
    int y1, y2; cin>>y1>>y2;
    cout<<midPoint(x1, x2, y1, y2)<<endl;
    return 0;
}
```

PYTHON

```
from sys import stdin,stdout

# m = ((x1+x2)/2 , (y1+y2)/2)

def midpoint(x1,y1,x2,y2):
    res = str(((x1+x2)/2))+" , "+str(((y1+y2)/2))
    return res

x1,y1,x2,y2 = stdin.readline().strip().split()

x1,y1,x2,y2 = int(x1),int(y1),int(x2),int(y2)

medio = midpoint(x1, y1, x2, y2)

stdout.write(f"el punto medio es: {medio}")
```

13.11) Línea dados dos puntos

Línea dados dos puntos



Guía del programador competitivo

Ilustración 13-10 Línea dados dos puntos

Dados dos puntos P(x1,y1) y Q(x2,y2), encuentre la ecuación de la línea formada por esos dos puntos.

Cualquier línea puede ser representada como:

- ax + by = c

Dejamos dos puntos que satisfagan la línea dada, entonces tenemos:

- ax1 + by1 = c
- ax2 + by2 = c

Podemos cambiar los siguientes valores para que la ecuación de mantenga verdadera:

- a = y2 - y1
- b = x1 - x2
- c = ax1 + by1

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
//Implementación Java para encontrar la linea
//que pasa atraves de dos puntos

public class LineThroughTwoPoints {

    public static void main(String[] args) {
        Pair P = new Pair(3, 2);
        Pair Q = new Pair(2, 6);
        lineFromPoints(P, Q);
    }

    static void lineFromPoints(Pair P, Pair Q) {
        double a = Q.second - P.second;
        double b = P.first - Q.first;
        double c = a * (P.first) + b * (P.second);
        if (b < 0) {
            System.out.println("La linea que pasa a traves de "
                    + "los puntos P y Q es: "
                    + a + "x " + b + "y = " + c);
        } else {
            System.out.println("La linea que pasa a traves de l"
            + "os puntos P y Q es: "
                    + a + "x + " + b + "y = " + c);
        }
    }
    /* Este par  es usado para almacenar la X y Y
    de un punto respectivamente*/
    static class Pair {
        int first;
        int second;
        public Pair(int first, int second) {
            this.first = first;
            this.second = second;
        }
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
#define x first
#define y second
using namespace std;
typedef pair<int,int> point;
void lineFromPoints(point P,point Q){
    double a=Q.y-P.y;
    double b=P.x-Q.x;
    double c=a*(P.x)+b*(P.y);
    if(b < 0){
```

```
        cout<<"La linea que pasa a traves de los puntos P y Q es:"<<"("<<a<<"x)
* ("<<b<<"y) = "<<c<<endl;
    }else{
        cout << "La linea que pasa a traves de los punto P y Q es:"<<a<<"x + "
<< b << "y = " << c << endl;
    }
}
int main() {
    point p=make_pair(3,2);
    point q=make_pair(2,6);
    lineFromPoints(p,q);
}
```

13.12) Triángulo de monedas ordenadas

Tenemos N monedas las cuales necesitamos ordenar en forma de triángulo, por ejemplo la primera fila podrá tener una moneda, la segunda fila dos monedas y así en adelante, necesitamos saber la máxima altura que podemos obtener usando esas N monedas.

Este problema puede ser resuelto encontrando la relación entre la altura del triángulo y el número de monedas, dejamos como la altura máxima como H, luego la suma total de monedas debe ser menos de N.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Programa java que encuentra la maxima altura
// de un triángulo de monedas arregladas

public class ArrangedCoinTriangle {

    /*Retrorna el la raiz cuadrada de n
    Note como la función lo realiza*/
    static float squareRoot(float n) {
        /*Usamos n como aproximción inicial*/
        float x = n;
        float y = 1;
        // Se decide el nivel de precisión
        float e = 0.000001f;
```

```
        while (x - y > e) {
            x = (x + y) / 2;
            y = n / x;
        }
        return x;
    }
    //Metodo que encuentra la maxima altura
    //del arreglo de monedas
    static int findMaximumHeight(int N) {
        //Calculando la porción interna
        //de la raiz cuadrada
        int n = 1 + 8 * N;
        int maxH = (int) (-1 + squareRoot(n)) / 2;
        return maxH;
    }

    public static void main(String[] args) {
        int N = 12;
        System.out.print(findMaximumHeight(N));
    }
}
```

C++

```
#include <iostream>
using namespace std;

int findMaxH(int);
float squareRoot(float);

int main() {
    int N = 12;
    cout << findMaxH(N) << endl;
}

int findMaxH(int n) {
    int n1 = 1 + 8 * n;
    int maxh = (int) (-1 + squareRoot(n1) / 2);
    return maxh;
}

float squareRoot(float n) {
    float x = n;
    float y = 1;
    float e = 0.0000001f;
    while (x - y > e) {
        x = (x + y) / 2;
        y = n / x;
    }
    return x;
}
```

PYTHON

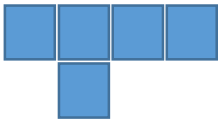
```python
from sys import stdin
from sys import stdout

def findMaxH(n):
    n1 = 1 + 8 * n
    maxh = int(-1 + squareRoot(n1) / 2)
    return maxh

def squareRoot(n):
    x = n
    y = 1
    e = 0.000001
    while x - y > e:
        x = (x + y) / 2
        y = n / x
    return x

N = 12
stdout.write(f"{findMaxH(N)}")
```

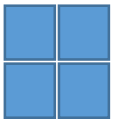
13.13) Perímetro usando bloques

Tenemos n bloques de tamaño 1x1, necesitamos encontrar el mínimo perímetro de una matriz hecha de estos bloques.

Tomemos un ejemplo para ver el patrón, tenemos 4 bloques, las siguientes son las diferentes posibilidades:

 Perímetro = 12

 Perímetro = 10

 Perímetro = 8

Si hacemos algunos ejemplos usando lápiz y papel, podemos notar que el perímetro se vuelve mínimo cuando la figura formada es más cercana a un cuadrado. La razón de esto

es, que queremos el máximo de lados de bloques que miren dentro de la figura, entonces el perímetro de la figura se vuelve mínimo.

Si el número de bloques es un cuadrado perfecto, entonces el perímetro puede ser simplemente 4*sqrt(n).

Pero si el número de bloques no es una raíz cuadrada perfecta, entonces nosotros calculamos el número de filas y columnas cercanas a la raíz cuadrada, luego de arreglar los bloques en un rectángulo, y tenemos bloques restantes, simplemente podemos agregar 2 al perímetro porque solo 2 lados extra faltarían.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
/*Codigo JAVA que permire encontrar el minimo
perimetro usando n bloques*/
public class PerimeterUsingBlocks {

    public static long minPerimeter(int n) {
        int l = (int) Math.sqrt(n);
        int sq = l * l;
        //si n es un cudrado perfeto
        if (sq == n) {
            return l * 4;
        } else {
            //Número de filas
            long row = n / l;
            //Perimetro de la matriz rectangular
            long perimeter
                    = 2 * (l + row);
            // Si hay bloques restantes
            if (n % l != 0) {
                perimeter += 2;
            }
            return perimeter;
        }
    }

    public static void main(String[] args) {
        int n = 10;
        System.out.println(minPerimeter(n));
    }
}
```

C++

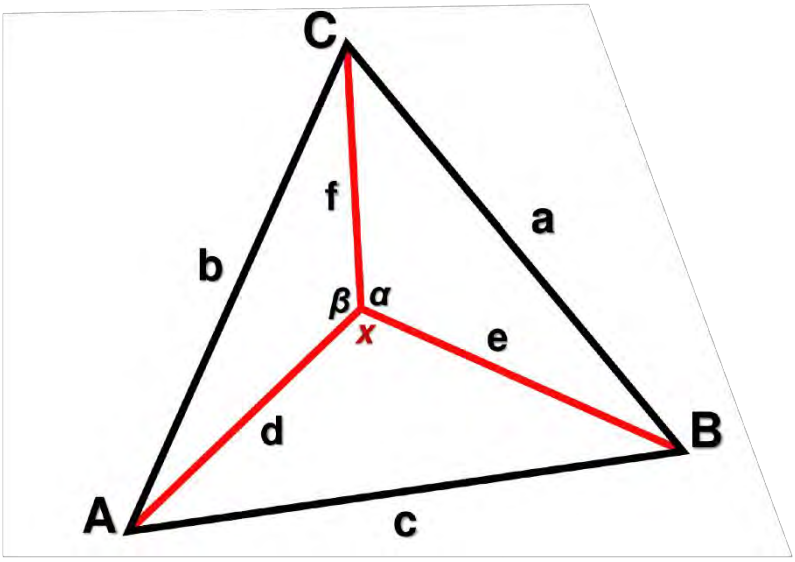
```cpp
#include <iostream>
#include <cmath>

using namespace std;

long minPerimeter(int n) {
    int x = (int) sqrt(n);
    int sq = x * x;
    if (sq == n) {
        return x << 2;
    } else {
        long row = n / x;
        long perimeter = (x + row) << 1;
        if (n % x != 0) {
            perimeter += 2;
        }
        return perimeter;
    }
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int n;
    cin>>n;
    cout << minPerimeter(n) << endl;
    return 0;
}
```

PYTHON

```python
import math
from collections import namedtuple
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write


def minPerimeter(n):
    l = int(math.sqrt(n))
    sq = l * l
    if sq == n:
        return l * 4
    else:
        row = n // l
        perimeter = 2 * (l + row)
        if n % l != 0:
            perimeter += 2
        return perimeter


n = int(rl())
wr(f'{minPerimeter(n)}')
```

13.14) Punto dentro de un triangulo

Punto dentro de un triángulo



Guía del programador competitivo

Ilustración 13-11 Búsqueda de un punto dentro de un triángulo

Dados tres puntos esquina de un triángulo, y un punto más P, verifique si P yace dentro del triángulo o no.

Dejaremos las coordenadas de las tres esquinas ser (x1, y1), (x2, y2) y (x3, y3), y las coordenadas de P ser (x, y).

1) Calcular área del triángulo dado, por ejemplo el área del triángulo ACB: Area A = [x1(y2 – y3) + x2(y3 – y1) + x3(y1-y2)]/2
2) Calcular el área del triángulo PAB. Podemos usar la misma fórmula para esto, dejamos esta área ser A1.
3) Calculamos el área del triángulo PBC, dejamos esta área ser A2.

4) Calcular el área del triángulo PAC, dejaremos esta área ser A3.

5) Si P yace dentro del triángulo, entonces A1+A2+A3 debe ser igual a A.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
//Codigo Java que verifica cuando un punto
//yace dentro de un triángulo o no

public class PointInsideTriangle {

    /* Una función de utilidad que calcula el
    area del triángulo formado por (x1, y1) (x2, y2) y (x3, y3)*/
    static double area(int x1, int y1, int x2, int y2,
            int x3, int y3) {
        return Math.abs((x1 * (y2 - y3) + x2 * (y3 - y1)
                + x3 * (y1 - y2)) / 2.0);
    }

    /* Una función que verifica cu cualquier punto P(x,y)
    yace dentro de un triángulo formado por (x1, y1),
    B(x2, y2) and C(x3, y3) */
    static boolean isInside(int x1, int y1, int x2,
            int y2, int x3, int y3, int x, int y) {
        /* Calcula el area del triángulo ABC */
        double A = area(x1, y1, x2, y2, x3, y3);
        /* Calcula el area del triángulo PCB */
        double A1 = area(x, y, x2, y2, x3, y3);
        /* Calcula el area del triángulo PAC */
        double A2 = area(x1, y1, x, y, x3, y3);
        /* Calcula el area del triángulo PAB */
        double A3 = area(x1, y1, x2, y2, x, y);
        /* verifica si la suma de A1,A2 y A3 es igual a A*/
        return (A == A1 + A2 + A3);
    }

    public static void main(String[] args) {
        /* Verificamos si el punto P(10,15)
        yace dentro del triángulo formado por
        A(0, 0), B(20, 0) and C(10, 30)*/
        if (isInside(0, 0, 20, 0, 10, 30, 10, 15)) {
            System.out.println("Adentro");
        } else {
            System.out.println("Por fuera");
        }
    }
}
```

C++

```cpp
#include <iostream>
#include <math.h>
using namespace std;


double area(int x1, int y1, int x2, int y2, int x3, int y3) {
    return fabs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0);
}

bool isinside(int x1, int y1, int x2, int y2, int x3, int y3, int x, int y) {
    double A = area(x1, y1, x2, y2, x3, y3);
    double A1 = area(x1, y1, x2, y2, x, y);
    double A2 = area(x1, y1, x3, y3, x, y);
    double A3 = area(x2, y2, x3, y3, x, y);
    return (A == A1 + A2 + A3);
}

int main() {
    int x1 = 0, x2 = 5, x3 = 10, y1 = 0, y2 = 5, y3 = 3, puntox = 5, puntoy = 3;
    if (isinside(x1, y1, x2, y2, x3, y3, puntox, puntoy)) {
        cout << "Esta dentro" << endl;
    } else cout << "Esta fuera" << endl;
}
```

PYTHON

```python
from sys import stdin
from sys import stdout

def Area(x1, y1, x2, y2, x3, y3):
    return abs((x1 * (y2-y3) + x2 * (y3-y1) + x3 * (y1-y2)) / 2)

def isInside(x1, y1, x2, y2, x3, y3, x, y):
    A = Area(x1, y1, x2, y2, x3, y3)
    A1 = Area(x1, y1, x2, y2, x, y)
    A2 = Area(x1, y1, x3, y3, x, y)
    A3 = Area(x2, y2, x3, y3, x, y)
    return (A == A1 + A2 + A3)

x1, y1, x2, y2, x3, y3, x, y = 0, 0, 5, 5, 10, 3, 5, 3
if isInside(x1, y1, x2, y2, x3, y3, x, y):
    stdout.write(f"el punto ({x},{y}) esta dentro del triangulo")
else:
    stdout.write(f"el punto ({x},{y}) no esta dentro del triangulo")
```

13.15) Línea dividida en ratios

Dividir una línea en diferentes ratios

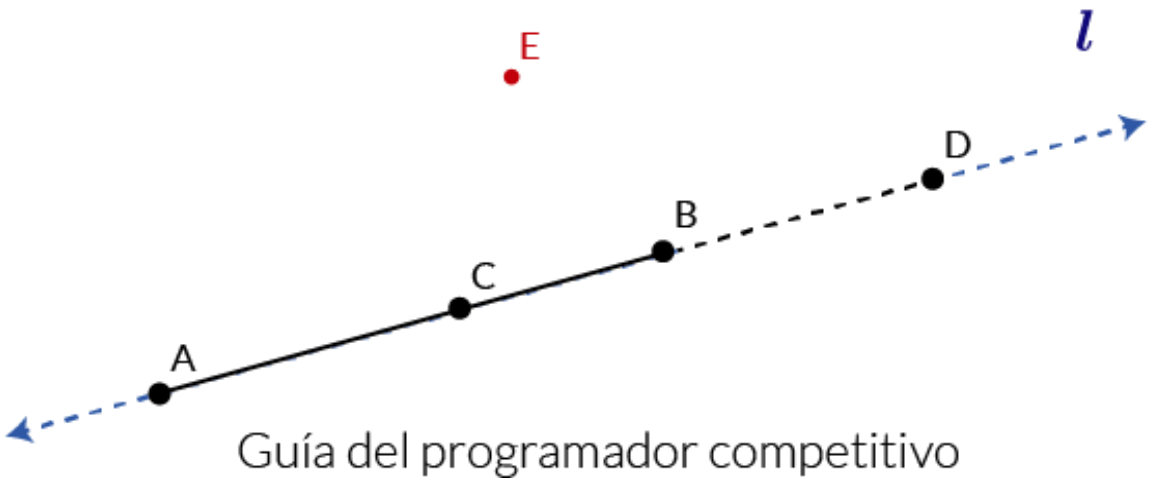


Ilustración 13-12 Una línea recta dividida en ratios

Dadas dos coordenadas (x1, y1) y (x2, y2), y m y n, encuentre las coordenadas que dividen la línea juntando (x1, y1) y (x2,y2) en el ratio m:n.

La fórmula de la sección nos dice las coordenadas del punto que divide una línea segmento dada en dos partes las cuales tendrán la longitud del ratio m:n.

- ((mx2+nx1/m+n),(my2+ny1/m+n))

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Programa java que encuentra el punto que divide una
//linea dada, dado un ratio

public class RatioDivideLine {
    static void section(double x1, double x2,
            double y1, double y2,
            double m, double n) {
        //Aplicando la formula de la sección
        double x = ((n * x1) + (m * x2))
                / (m + n);
        double y = ((n * y1) + (m * y2))
```

```
                / (m + n);
        // Imprimiendo resultado
        System.out.println("(" + x + ", " + y + ")");
    }

    public static void main(String[] args) {
        double x1 = 2, x2 = 4, y1 = 4,
                y2 = 6, m = 2, n = 3;
        section(x1, x2, y1, y2, m, n);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define x first
#define y second
using namespace std;
typedef pair<double,double> point;
void section(point a,point b,point n){
    double x1 = ((n.y*a.x)+(n.x*a.y))/(n.x+n.y);
    double y1 = ((n.y*b.x)+(n.x*b.y))/(n.x+n.y);
    cout<<"("<<x1<<" , "<<y1<<")"<<endl;
}
int main() {
    point a=make_pair(2,0);
    point b=make_pair(5,5);
    point n=make_pair(3,2);
    section(a,b,n);
}
```

PYTHON

```
from collections import namedtuple
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write


# Usando coordenadas independientes
def section(x1, y1, x2, y2, m, n):
    x = ((n * x1) + (m * x2)) / (m + n)
    y = ((n * y1) + (m * y2)) / (m + n)
    wr(f'({x} , {y})')


x1, y1, x2, y2, m, n = rl().strip().split()
x1, y1, x2, y2, m, n = int(x1), int(y1), int(x2), int(y2), int(m), int(n)
section(x1, y1, x2, y2, m, n)
```

```
# Usando NamedTuple
Puntos = namedtuple('Puntos', ['x', 'y'])


def sectionNT(P1, P2, m, n):
    x = ((n * P1.x) + (m * P2.x)) / (m + n)
    y = ((n * P1.y) + (m * P2.y)) / (m + n)
    wr(f'({x} , {y})')


P1 = Puntos(x1, y1)
P2 = Puntos(x2, y2)
sectionNT(P1, P2, m, n)
```

13.16) Rectángulos en NxM

Tenemos una matriz N*M, imprima el número de rectángulos en él.

- Si la matriz es 1x1, entonces habrá 1 rectángulo.
- Si la matriz es 2x1, entonces habrá 2 +1 = 3 rectángulos
- Si la matriz es 3x1, entonces habrá 3+2+1=6 rectángulo.

Podemos decir que para N*1 ahí habrá N+(N+1)+(N-2)… +1=(N)(N+1)/2 rectángulos.

Si nosotros añadimos una columna más a Nx1, primero tendríamos tantos rectángulos en la segunda columna como en la primera, y luego tendríamos el mismo número de 2xM rectángulos, entonces Nx2=3 (N)(N+1)/2, luego de deducir esto podemos decir que:

- Para N*M nosotros tendríamos (M)(M+1)/2 (N)(N+1)/2 = M(M+1)(N)(N+1)/4

Entonces la fórmula para el total de rectángulos es:

- M(M+1)(N)(N+1)/4

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Codigo Java que cuenta el número
// de rectangulos en una matriz N*M

public class RectanglesInNxM {
```

```
    public static long rectCount(int n, int m) {
        return (m * n * (n + 1) * (m + 1)) / 4;
    }

    public static void main(String[] args) {
        int n = 5, m = 4;
        System.out.println(rectCount(n, m));
    }
}
```

13.17) Cuadrados 2x2 en un triángulo

Cuál es el máximo número de cuadrados de tamaño 2x2 unidades que pueden caber en un triángulo isósceles de ángulos correctos dada la base en unidades.

Un lado del cuadrado debe ser paralelo a la base del triángulo.

Desde que el triángulo es isósceles, la base dada será también igual a la altura. Ahora en la parte diagonal, podríamos siempre necesitar un largo extra de dos unidades en la altura y la base del triángulo para acomodar un triángulo. En la longitud restante de la base, podemos construir largo/2 cuadrados. Desde que cada cuadrado es de dos unidades, lo mismo puede hacerse en el caso de la altura, ahí no hay necesidad de calcular eso de nuevo. Entonces, para cada nivel de la longitud dada, podemos construir "(largo-2)/2" cuadrados. Esto nos da una base de "(largo-2)" encima de él. Continuando con el proceso de obtener el número de cuadrados para toda la disponible "largo/2" altura, podemos calcular los cuadrados.

Para una forma más eficiente, podemos usar la fórmula de la suma de AP n*(n+1)/2, donde n= largo-2.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

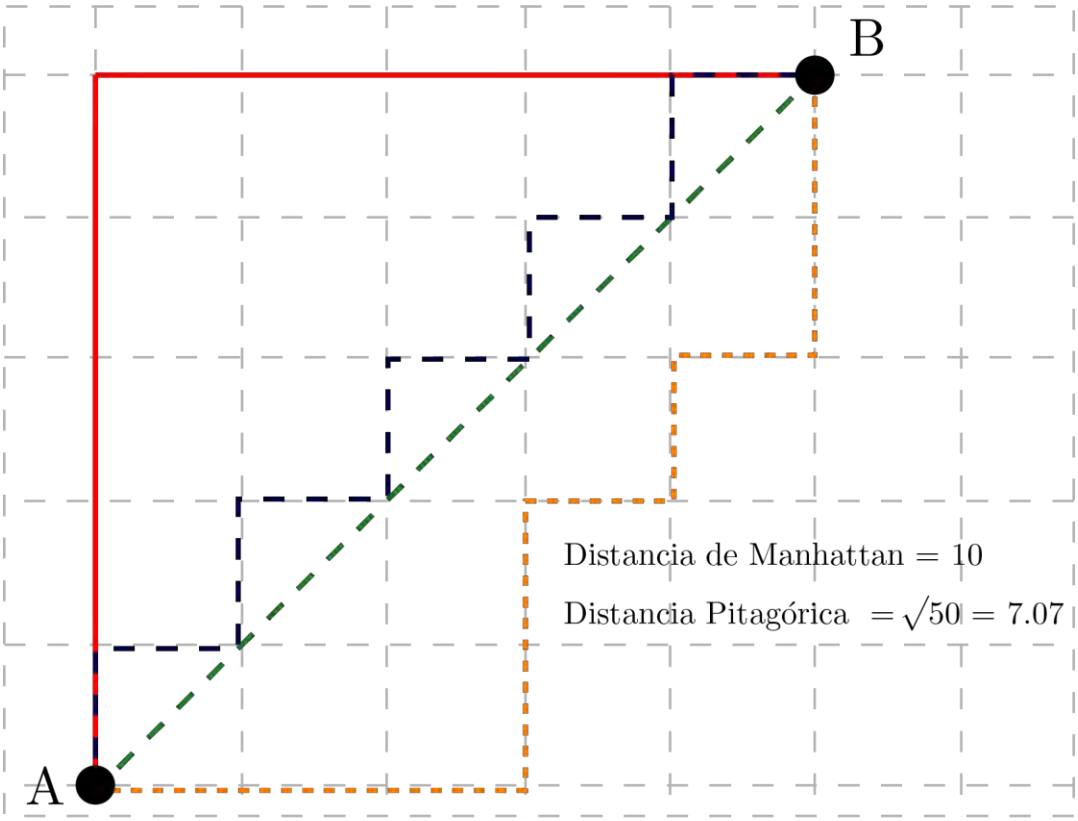
JAVA

```
// Programa Java que cuenta el número de cuadros2
// 2x2 que cabe en un triángulo isoceles
```

```java
public class SquaresInTriangle2x2 {

    public static int numberOfSquares(int base) {
        // Removiendo la parte extra
        // que podamos necesitar
        base = (base - 2);
        // Desde cada cuadrado que tenga
        // base de tamaño 2
        base = base / 2;
        return base * (base + 1) / 2;
    }

    public static void main(String args[]) {
        int base = 8;
        System.out.println(numberOfSquares(base));
    }
}
```

13.18) Suma de Manhattan

Suma de Manhattan



Guía del programador competitivo

Ilustración 13-13 Ejemplo de suma de Manhattan

Dados n coordenadas enteras, encuentre la suma de la distancia de Manhattan entre todos los pares de coordenadas.

La distancia de Manhattan entre dos puntos (x1,y1) y (x2,y2) es:

- |x1 – x2| + |y1 – y2|

La idea es recorrer dos ciclos anidados, por ejemplo cada punto, encontrar la distancia de todos los otros puntos con este.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```java
/*Programa Java para encontrar la suma de
las distancias de Manhattan entre todos
los pares de puntos dados */

public class SumOfManhattan {

    /*Retorna la suma de la distancia entre todos
    los pares de puntos*/
    static int distancesum(int x[], int y[], int n) {
        int sum = 0;
        /* Por cada punto, encuentra la distancia
        al resto de puntos */
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                sum += (Math.abs(x[i] - x[j])
                        + Math.abs(y[i] - y[j]));
            }
        }
        return sum;
    }

    public static void main(String[] args) {
        int x[] = {-1, 1, 3, 2};
        int y[] = {5, 6, 5, 3};
        int n = x.length;
        System.out.println(distancesum(x, y, n));
    }
}
```

13.19) Tres puntos colineales

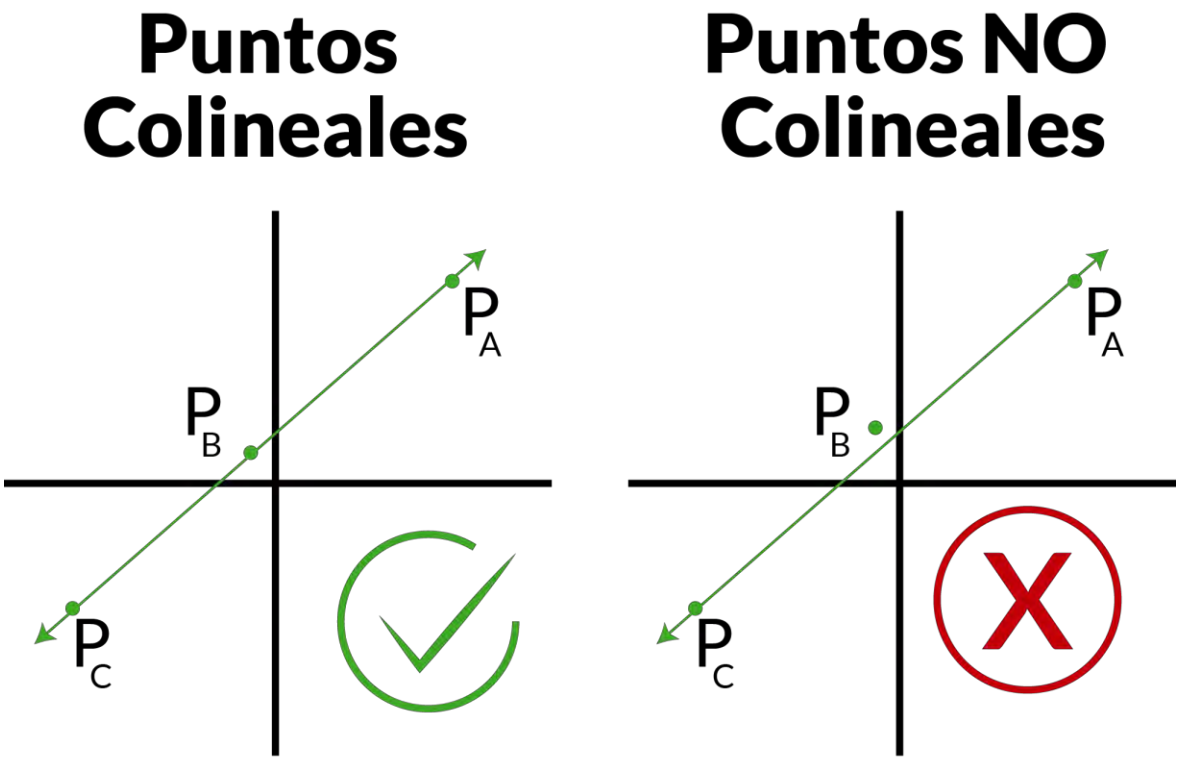


Ilustración 13-14 Puntos colineales en un plano

Dados tres puntos, verificar si estos puntos yacen en recta (colineales) o no.

Ejemplo: (1, 1), (1, 4), (1, 5)

Los tres puntos yacen en una línea recta.

Tres puntos yacen en una línea recta si el área formada por un triángulo de estos tres puntos es cero.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
/*Programa en java para verificar si
tres puntos son colineares
o no usando el area del triángulo*/
public class ThreePointsCollinear {
    //Función que verifica si un punto es colineal o no
    static void collinear(int x1, int y1, int x2,
```

```
            int y2, int x3, int y3) {
        /*Calcular el area del triángulo
        Debemos omitir multiplicaciones con 0.5
        para evitar calculos flotantes*/
        int a = x1 * (y2 - y3)
                + x2 * (y3 - y1)
                + x3 * (y1 - y2);
        if (a == 0) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }

    public static void main(String args[]) {
        int x1 = 1, x2 = 1, x3 = 1,
                y1 = 1, y2 = 4, y3 = 5;
        collinear(x1, y1, x2, y2, x3, y3);

    }
}
```

13.20) Ángulos de un triángulo

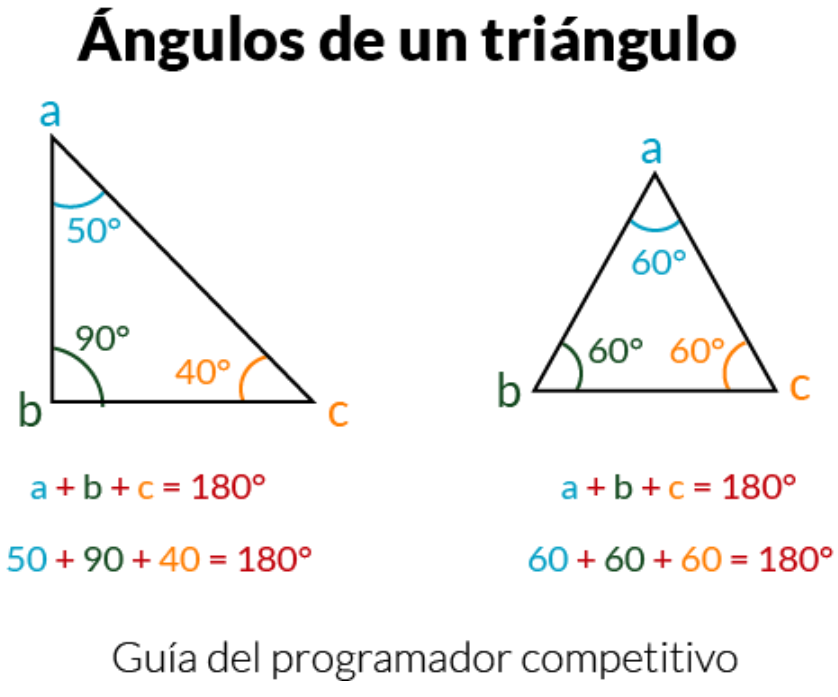


Ilustración 13-15 Ángulos de un triángulo

Dadas las coordenadas de tres vértices de un triángulo en un plano 2D, encuentre sus tres ángulos.

- c^2 = a^2 + b^2 - 2(a)(b)(cos beta)

Luego del despeje:

- beta = acos((a^2 + b^2 - c^2) / (2ab))

En trigonometría la ley del coseno cuenta que los largos de los lados de un triángulo con el coseno de uno de sus ángulos luego de un despeje nos da el ángulo.

Primero calculamos el largo de todos los lados, luego aplicamos la fórmula de arriba para obtener todos los ángulos en radianes, luego convertimos de radianes a grados.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
/* Codigo JAVA para encontrar todos los angulos
de un triángulos dadas las coordenadas
de los tres vertices*/
import java.awt.Point;
import static java.lang.Math.PI;
import static java.lang.Math.sqrt;
import static java.lang.Math.acos;

public class TriangleAngles {
    // Regresa el cuadrado de la distancia b/w de dos puntos
    static int lengthSquare(Point p1, Point p2) {
        int xDiff = p1.x - p2.x;
        int yDiff = p1.y - p2.y;
        return xDiff * xDiff + yDiff * yDiff;
    }

    static void printAngle(Point A, Point B,
            Point C) {
        //Cuadrado de los tamaños de a2, b2, c2
        int a2 = lengthSquare(B, C);
        int b2 = lengthSquare(A, C);
        int c2 = lengthSquare(A, B);
        // Longitus de los lados de a, b, c
        float a = (float) sqrt(a2);
        float b = (float) sqrt(b2);
        float c = (float) sqrt(c2);
        // De la ley del coseno
        float alfa = (float) acos((b2 + c2 - a2) / (2 * b * c));
        float beta = (float) acos((a2 + c2 - b2) / (2 * a * c));
        float gamma = (float) acos((a2 + b2 - c2) / (2 * a * b));
```

```
        //Conversión a grados
        alfa = (float) (alfa * 180 / PI);
        beta = (float) (beta * 180 / PI);
        gamma = (float) (gamma * 180 / PI);
        //Imprimiendo los angulos
        // printing all the angles
        System.out.println("alfa : " + alfa);
        System.out.println("beta : " + beta);
        System.out.println("gamma : " + gamma);
    }

    public static void main(String[] args) {
        Point A = new Point(0, 0);
        Point B = new Point(0, 1);
        Point C = new Point(1, 0);
        printAngle(A, B, C);
    }
}
```

13.21) Problemas de repaso

Ejercicios en Online Judge

184-Laser Lines

378-Intersecting Lines

477-Points in Figures: Rectangles and Circles

478-Points in Figures: Rectangles, Circles, Triangles

476-Points in Figures: Rectangles

681-Convex Hull Finding

10200-Prime Time

10283-The Kissing Circles

10573-Geometry Paradox

Ejercicios en CodeChef

TRICOIN

RECTAGL

Capitulo 14. Operaciones binarias BitWise

Las operaciones bit a bit o BitWise, opera números binarios a nivel de cada bit individual, las cuales son operaciones rápidas que se soportan directamente en los procesadores. Existen operaciones bit a bit como el uso de compuertas lógicas, de desplazamiento en las cuales los valores binarios se desplazan hacia la derecha y la izquierda una o más posiciones, y las operaciones de rotación en donde se mueven de un lado a otro los valores binarios.

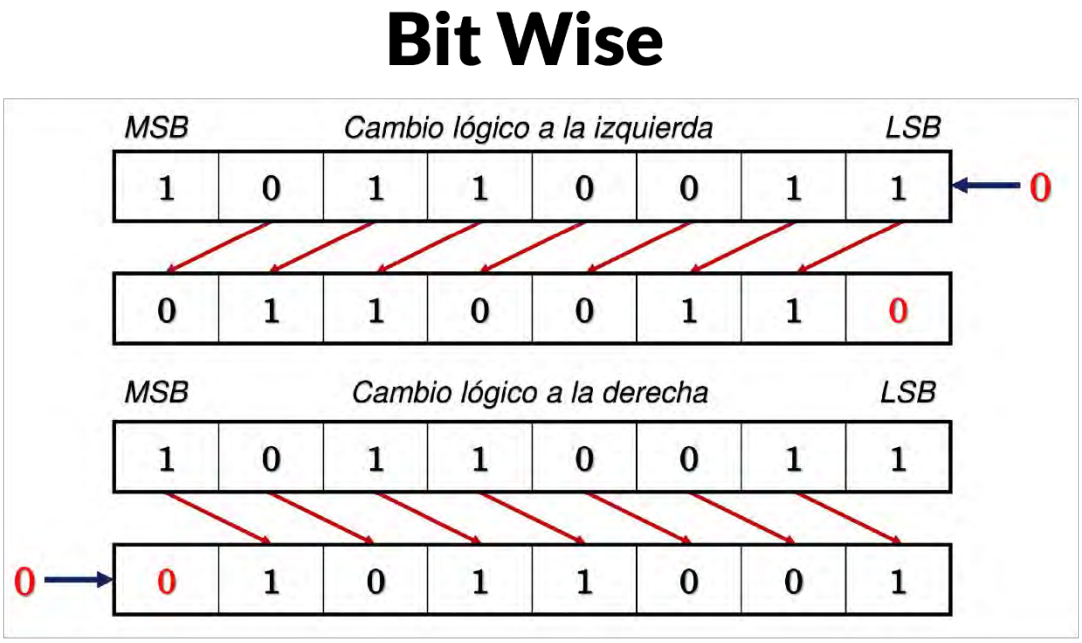


Ilustración 14-1 Bit Wise

14.1) BitWise básico

Operaciones a nivel de bits.

n & 1 -> Verifica si n es impar o no

n & (1<<k) -> Verifica si el k-esimo bit esta encendido o no

n | (1<<k) -> Enciende el k-esimo bit

n & ~(1<<k) -> Apaga el k-esimo bit

n ^ (1<<k) -> Invierte el k-esimo bit

~n -> Invierte todos los bits

n & -n -> Devuelve el bit encendido mas a la derecha

~n & (n+1) -> Devuelve el bit apagado mas a la derecha

n | (n+1) -> Enciende el bit apagado mas a la derecha

n & (n-1) -> Apaga el bit encendido mas a la derecha

Otras operaciones a nivel de bits.

1) Borrar todos los bits desde el bit menos significativo (LSB) hasta el bit i

- mask = ~((1 << i+1) - 1);
- x &= mask;

Para borrar todos los bits de LSB a i-ésimo bit, tenemos que AND X con la máscara que tiene LSB a i-ésimo bit 0. Para obtener dicha máscara, primero se desplaza a la izquierda 1 veces. Ahora, si sumamos 1 a partir de eso, todos los bits de 0 a i-1 se convierten en 1 y los bits restantes se convierten en 0. Ahora simplemente podemos tomar el complemento de máscara para obtener todos los primeros bits de i en 0 y permanecer en 1.

Ejemplo:

- x = 29 (00011101) y queremos borrar LSB al 3er bit, total 4 bits
- mask -> 1 << 4 -> 16(00010000)
- mask -> 16 – 1 -> 15(00001111)
- mask -> ~mask -> 11110000
- x & mask -> 16 (00010000)

2) Borrar todos los bits del bit más significativo (MSB) al bit i-ésimo

- mask = (1 << i) - 1;
- x &= mask;

Para borrar todos los bits de MSB a i-ésimo bit, tenemos que AND x con la máscara que tiene MSB a i-ésimo bit 0. Para obtener dicha máscara, primero se desplaza a la izquierda 1 veces. Ahora, si menos 1 de eso, todos los bits de 0 a i-1 se convierten en 1 y los bits restantes se convierten en 0.

Ejemplo:

- x = 215 (11010111) y queremos borrar MSB al 4to bit, total 4 bits
- mask -> 1 << 4 -> 16(00010000)
- mask -> 16 – 1 -> 15(00001111)
- x & mask -> 7(00000111)

3) Dividir por 2

- x >>= 1;

Cuando hacemos un desplazamiento aritmético a la derecha, cada bit se desplaza a la derecha y la posición en blanco se sustituye con un bit de signo de número, 0 en caso de número positivo y 1 en caso de número negativo. Como cada bit es una potencia de 2, con cada cambio estamos reduciendo el valor de cada bit por un factor de 2 que es equivalente a la división de x por 2.

Ejemplo:

- x = 18(00010010)
- x >> 1 = 9 (00001001)

4) Multiplicar por 2

- x <<= 1;

Cuando hacemos desplazamiento aritmético a la izquierda, cada bit se desplaza a la izquierda y la posición en blanco se sustituye por 0. Como cada bit es una potencia de 2,

con cada cambio aumentamos el valor de cada bit por un factor de 2 que es equivalente a la multiplicación de x por 2.

Ejemplo:

- x = 18(00010010)
- x << 1 = 36 (00100100)

5) Mayusculas a minusculas en el alfabeto ingles

- ch |= ' ';

La representación en bits de las letras en inglés en mayúsculas y minúsculas es:

A -> 01000001 a -> 01100001

B -> 01000010 b -> 01100010

C -> 01000011 c -> 01100011

. .

. .

Z -> 01011010 z -> 01111010

Como podemos ver si establecemos el quinto bit de caracteres en mayúsculas, se convertirá en caracteres en minúsculas. Tenemos que preparar una máscara que tenga el quinto bit 1 y otro 0 (00100000). Esta máscara es una representación en bits del carácter de espacio (‘‘). El carácter "ch" luego OR con la máscara.

Ejemplo:

- ch = ‘A’ (01000001)
- mask = ‘ ‘ (00100000)
- ch | mask = ‘a’ (01100001)

6) Minusculas a mayusculas en el alfabeto ingles

- ch &= '_’ ;

La representación en bits de las letras en inglés en mayúsculas y minúsculas es:

A -> 01000001 a -> 01100001

B -> 01000010 b -> 01100010

C -> 01000011 c -> 01100011

. .

. .

Z -> 01011010 z -> 01111010

Como podemos ver si borramos el quinto bit de caracteres en minúsculas, se convertirá en caracteres en mayúsculas. Tenemos que preparar una máscara que tenga el quinto bit 0 y otro 1 (10111111). Esta máscara es una representación en bits del carácter de subrayado (‘_‘). El carácter "ch" luego AND con la máscara.

Ejemplo:

ch = ‘a’ (01100001)

mask = ‘_ ‘ (11011111)

ch & mask = ‘A’ (01000001)

7) Cuenta los bits establecidos en un entero

```
int countSetBits(int x) {
    int count = 0;
    while (x) {
        x &= (x - 1);
        count++;
    }
    return count;
}
```

Se realiza esta operacion basado en el algoritmo de Brian Kernighan.

8) Encuentra la log base 2 de un entero de 32 bits

```
int log2(int x) {
    int res = 0;
    while (x >>= 1)
        res++;
    return res;
}
```

Desplazamos a la derecha x repetidamente hasta que se convierte en 0, mientras tanto, contamos con la operación de desplazamiento. Este valor de conteo es el log2 (x).

9) Comprobando si un entero de 32 bits es una potencia de 2

```
int isPowerof2(int x) {
    return (x && !(x & x - 1));
}
```

Toda la potencia de 2 tiene solo un bit establecido, p. 16 (00010000). Si menos 1 de esto, todos los bits de LSB para establecer el bit se alternan, es decir, 16-1 = 15 (00001111). Ahora si realizamos AND x con (x-1) y el resultado es 0, entonces podemos decir que x es una potencia de 2, de lo contrario no. Tenemos que tener mucho cuidado cuando x = 0.

Ejemplo:

- x = 16(000100000)
- x – 1 = 15(00001111)
- x & (x-1) = 0
- Entonces 16 es potencia de 2

10) Cómo establecer un bit en el número "num":

Si queremos establecer un bit en la enésima posición en el número "num", se puede hacer usando el operador "OR" (|).

- Primero dejamos el desplazamiento "1" a la posición n a través de (1 << n)
- Luego, use el operador "OR" para establecer el bit en esa posición. El operador "OR" se usa porque establecerá el bit incluso si el bit no se ha establecido previamente en la representación binaria del número "num".

C++

```
#include<iostream>
using namespace std;
```

```
//num es el numero y pos es la posición
//del bit que queremos activar

void set(int & num, int pos) {
    //Primer paso es correr '1',
    //Segundo paso es realizar OR
    num |= (1 << pos);
}

int main() {
    int num = 4, pos = 1;
    set(num, pos);
    cout << (int) (num) << endl;
    return 0;
}
```

Salida:

6

11) Cómo desestablecer un bit en el número "num":

Supongamos que queremos desarmar un poco en la enésima posición en el número "num", entonces tenemos que hacer esto con la ayuda del operador "AND" (&).

- Primero dejamos el desplazamiento ‘1’ a la posición n mediante (1 << n) que usamos el operador NOT bit a bit ‘~’ para desactivar este desplazamiento sh 1 ’.
- Ahora, después de despejar esta izquierda desplazada ‘1’, es decir, llegar a ‘0’, ‘AND '(&) con el número‘ num ’que desarmará el bit en la enésima posición.

C++

```
#include <iostream>
using namespace std;

void unset(int &num, int pos) {
    num &= (~(1 << pos));
}

int main() {
    int num = 7;
    int pos = 1;
    unset(num, pos);
    cout << num << endl;
    return 0;
}
```

Salida:

5

12) Alternar un bit en la enésima posición:

Alternar significa activar el bit 'on' (1) si estaba 'off' (0) y desactivar '(0) si estaba' on '(1) anteriormente. Usaremos el operador' XOR 'aquí que es esto '^' La razón detrás del operador "XOR" se debe a sus propiedades.

- Propiedades de operador XOR.
- 1^1 = 0
- 0^0 = 0
- 1^0 = 1
- 0^1 = 1
- Si dos bits son diferentes, el operador "XOR" devuelve un bit establecido (1) de lo contrario, devuelve un bit no establecido (0).

C++

```
#include <iostream>
using namespace std;

void toggle(int &num, int pos) {
    num ^= (1 << pos);
}

int main() {
    int num = 4;
    int pos = 1;
    toggle(num, pos);
    cout << num << endl;
    return 0;
}
```

Salida:

6

13) Comprobando si el bit en la enésima posición está activado o desactivado:

Es bastante fácil de hacer con el operador "AND".

- Desplazar a la izquierda '1' a la posición dada y luego 'Y' ('y').#include <iostream>
- using namespace std;

C++

```
bool at_position(int num, int pos) {
    bool bit = num & (1 << pos);
    return bit;
}

int main() {
    int num = 5;
    int pos = 0;
    bool bit = at_position(num, pos);
    cout << bit << endl;
    return 0;
}
```

Salida:

1

Observe que primero hemos dejado "1" desplazado y luego hemos usado el operador "Y" para obtener el bit en esa posición. Entonces, si hay '1' en la posición 'pos' en 'num', luego de 'AND' nuestra variable 'bit' almacenará '1' más si hay '0' en la posición 'pos' en el número 'num' que después de 'Y' nuestro bit variable almacenará '0'.

14) Invertir cada bit de un numero a un numero con complement a 1

Si queremos invertir cada bit de un número, es decir, cambiar el bit "0" a "1" y el bit "1" a "0". Podemos hacerlo con la ayuda del operador "~". Por ejemplo: si número es num = 00101100 (representación binaria), entonces '~ num' será '11010011'.

Este es también el "complemento de número 1".

C++

```
#include <iostream>
using namespace std;

int main() {
    int num = 4;
    // Invertir cada bit del numero
    cout << (~num);
```

```
    return 0;
}
```

Salida:

-5

15) Complemento a dos del número

Entonces, formalmente podemos tener el complemento de 2 al encontrar el complemento de 1 y agregar 1 al resultado, es decir (~ num + 1) o qué más podemos hacer es usar el operador '-'.

C++

```
#include <iostream>
using namespace std;

int main() {
    int num = 4;
    int twos_complement = -num;
    cout << "complemento a 2 " << twos_complement << endl;
    cout << "Tambien complemento a 2  " << (~num + 1) << endl;
    return 0;
}
```

Salida:

Complemento a 2 -4

Tambien complemento a 2 -4

16) Eliminando el bit activo más bajo:

En muchas situaciones, queremos quitar el bit establecido más bajo, por ejemplo, en la estructura de datos de árbol indexado binario, contando el número de bit establecido en un número.

Hacemos algo como esto:

X = X & (X-1)

Veamos esto tomando un ejemplo, sea X = 1100.

(X-1) invierte todos los bits hasta que encuentre el conjunto más bajo '1' y también invierte ese conjunto más bajo '1'.

X-1 se convierte en 1011. Después de realizar 'AND' X con X-1 obtenemos el bit de ajuste más bajo despojado.

C++

```
#include <iostream>
using namespace std;

void strip_last_set_bit(int &num) {
    num = num & (num - 1);
}

int main() {
    int num = 7;
    strip_last_set_bit(num);
    cout << num << endl;
    return 0;
}
```

Salida:

6

17) Obtener el bit activo mas bajo de un numero:

Esto se hace usando la expresión 'X & (- X)'. Veamos esto tomando un ejemplo: Sea X = 00101100. Entonces ~ X (complemento de 1) será '11010011' y el complemento de 2 será (~ X + 1 o -X), es decir, '11010100'. Entonces, si 'AND' el número original 'X' con su complemento de dos que es '-X', obtenemos el bit de ajuste más bajo.

00101100
& 11010100

00000100

C++

```
#include <iostream>
using namespace std;

int lowest_set_bit(int num) {
```

```
    int ret = num & (-num);
    return ret;
}

int main() {
    int num = 10;
    int ans = lowest_set_bit(num);
    cout << ans << endl;
    return 0;
}
```

Salida:

2

14.2) Suma uno

Sume uno a un número dado, el uso de los operadores como '+', '-','*','/','++','—'.... Entre otros no está permitido.

La respuesta se consigue con algo de magia de bits.

Para agregar 1 a un número x(como 0011000111), voltee todos los bits luego del cero de más a la derecha (bit 0) (obtenemos 0011000000) finalmente, voltee el cero de más a la derecha también (obtenemos 0011001000) para obtener la respuesta.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Programa Java en donde se agrega uno a
// un número dado

public class AddOne {
```

```
    static int addOne(int x) {
        return -(~x);
    }

    public static void main(String[] args) {
        System.out.println(addOne(13));
    }
}
```

C++

```
#include <iostream>
using namespace std;

string decToBin(int n) {
    if (n == 0)
        return "0";
    string bin = "";
    int ayuda = n;
    n = abs(n);
    while (n > 0) {
        bin = ((n & 1) == 0 ? '0' : '1') + bin;
        n >>= 1;
    }

    return (ayuda < 0 ? bin = '-' + bin : bin);
}

int addOne(int x) {
    return (-(~x));
}

int main() {
    int a = 13;
    cout << a << " --> " << decToBin(a) << endl;
    a = addOne(a);
    cout << a << " --> " << decToBin(a) << endl;
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def addOne(n):
    wr(f'{n} -> {bin(n)[2:]}\n')
    n = (-(~n))
    wr(f'{n} -> {bin(n)[2:]}\n')

n = int(rl())
addOne(n)
```

14.3) Comparación de signos

Dado dos enteros con signo, retorne true si los signos de los enteros dados son diferentes, de otra forma retorne falso, por ejemplo la función debe retornar true en -1 y +100, y debe retornar falso para -200 y -100. La función no debe usar ningún operador aritmético.
Dejaremos los enteros dados ser x y y, el bit de signo es 1 en números negativos, y 0 en números positivos, El XOR de x y y puede tener el bit de signo en 1 si ellos tiene signos opuestos, en otras palabras, XOR de x y y será un número negativo si x y y tienen signos opuestos.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
//Programa Java para detectar si dos enteros
//tienen signos opuestos

public class OppositeSigns {

    static boolean oppositeSigns(int x, int y) {
        return ((x ^ y) < 0);
    }

    public static void main(String[] args) {
        int x = 100, y = -100;
        if (oppositeSigns(x, y) == true) {
            System.out.println("Signos opuestos");
        } else {
            System.out.println("Signos no opuestos");
        }
        x = 100;
        y = 100;
        if (oppositeSigns(x, y) == true) {
            System.out.println("Signos son opuestos");
        } else {
            System.out.println("Signos no son opuestos");
        }
    }
}
```

C++

```cpp
#include <iostream>

using namespace std;

bool opposite(int a, int b) {
    return (a ^ b) < 0;
}

int main() {
    int a = 123;
    int b = -123;
    if (opposite(a, b)) {
        cout << "Distintos signos" << endl;
    } else {
        cout << "Iguales signos" << endl;
    }
    return 0;
}
```

PYTHON

```python
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def opposite(n, m):
    res = (n ^ m) < 0
    if res:
        wr(f'Distintos\n')
    else:
        wr(f'Iguales\n')

n = 30
m = -100
opposite(n, m)
```

14.4) Multiplicación básica

Podemos multiplicar un número por 7 usando operadores BitWise, primero hacemos corrimiento izquierdo del número de 3 bits (Se obtendrá 8n) luego se resta la forma original del número corrido y se retorna la diferencia (8n-n).

Complejidad de tiempo: O(1).

Funciona solo para enteros positivos.

El mismo concepto puede ser usado para multiplicación rápida por 9 u otros números cambiando la formula bitwise.

Por ejemplo:

- (n << 1) = x2
- ((n << 1) + n) = x3
- (n << 2) = x4
- ((n << 2) + n) = x5
- ((n << 3) - (n << 1)) = x6
- ((n << 3) - n) = x7
- (n << 3) = x8
- ((n << 3)+n) = x9
- (n << 3)+(n<<1) = x10

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
public class Multiply_2_to_10 {

    public static void main(String[] args) {
        int a = 18;
        System.out.println(multiplyByTwo(a));
        System.out.println(multiplyByThree(a));
        System.out.println(multiplyByFour(a));
        System.out.println(multiplyByFive(a));
        System.out.println(multiplyBySix(a));
        System.out.println(multiplyBySeven(a));
        System.out.println(multiplyByEight(a));
        System.out.println(multiplyByNine(a));
        System.out.println(multiplyByTen(a));
    }

    static int multiplyByTwo(int n) {
        return (n << 1);
    }

    static int multiplyByThree(int n) {
        return ((n << 1) + n);
    }
```

```
    static int multiplyByFour(int n) {
        return (n << 2);
    }

    static int multiplyByFive(int n) {
        return ((n << 2) + n);
    }

    static int multiplyBySix(int n) {
        return ((n << 3) - (n << 1));
    }

    static int multiplyBySeven(int n) {
        return ((n << 3) - n);
    }

    static int multiplyByEight(int n) {
        return (n << 3);
    }

    static int multiplyByNine(int n) {
        return ((n << 3) + n);
    }

    static int multiplyByTen(int n) {
        return (n << 3) + (n << 1);
    }
}
```

C++

```
#include <iostream>
//-------------//
#define FAST ios_base::sync_with_stdio(false);cout.tie(NULL);

using namespace std;

int multiplyByTwo(int n) {
    return (n << 1);
}

int multiplyByThree(int n) {
    return ((n << 1) + n);
}

int multiplyByFour(int n) {
    return (n << 2);
}

int multiplyByFive(int n) {
    return ((n << 2) + n);
}

int multiplyBySix(int n) {
    return ((n << 3) - (n << 1));
```

```
}

int multiplyBySeven(int n) {
    return ((n << 3) - n);
}

int multiplyByEight(int n) {
    return (n << 3);
}

int multiplyByNine(int n) {
    return ((n << 3) + n);
}

int multiplyByTen(int n) {
    return (n << 3) + (n << 1);
}

int main() {
    FAST;
    int a = 9;
    cout << multiplyByTwo(a) << endl;
    cout << multiplyByThree(a) << endl;
    cout << multiplyByFour(a) << endl;
    cout << multiplyByFive(a) << endl;
    cout << multiplySix(a) << endl;
    cout << multiplySeven(a) << endl;
    cout << multiplyByEight(a) << endl;
    cout << multiplyByNine(a) << endl;
    cout << multiplyByTen(a) << endl;
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def multiplyZero(n):
    res = (n << 0) - n
    wr(f'{n} x 0 = {res}\n')

def multiplyOne(n):
    res = (n << 1) - n
    wr(f'{n} x 1 = {res}\n')

def multiplyTwo(n):
    res = (n << 1)
    wr(f'{n} x 2 = {res}\n')

def multiplyThree(n):
    res = (n << 1)+n
    wr(f'{n} x 3 = {res}\n')
```

```
def multiplyFour(n):
    res = (n << 2)
    wr(f'{n} x 4 = {res}\n')

def multiplyFive(n):
    res = (n << 2)+n
    wr(f'{n} x 5 = {res}\n')

def multiplySix(n):
    res = (n << 3)-(n << 1)
    wr(f'{n} x 6 = {res}\n')

def multiplySeven(n):
    res = (n << 3)-n
    wr(f'{n} x 7 = {res}\n')

def multiplyEight(n):
    res = (n << 3)
    wr(f'{n} x 8 = {res}\n')

def multiplyNine(n):
    res = (n << 3)+n
    wr(f'{n} x 9 = {res}\n')

def multiplyTen(n):
    res = (n << 3)+(n << 1)
    wr(f'{n} x 10 = {res}\n')

n = int(rl())
multiplyZero(n)
multiplyOne(n)
multiplyTwo(n)
multiplyThree(n)
multiplyFour(n)
multiplyFive(n)
multiplySix(n)
multiplySeven(n)
multiplyEight(n)
multiplyNine(n)
multiplyTen(n)
```

14.5) Cuadrado de N sin usar pow

Dado un entero n, calcular el cuadrado de un número sin usar *,/ y pow().

Podemos hacerlo en tiempo O(Logn) usando operadores de BitWise, la idea está basada en el siguiente hecho:

- square(n) = 0 if n == 0

Si n es par

- square(n) = 4*square(n/2)

Si n es impar

- square(n) = 4*square(floor(n/2)) + 4*floor(n/2) + 1

Por ejemplo:

- square(6) = 4*square(3)
- square(3) = 4*(square(1)) + 4*1 + 1 = 9
- square(7) = 4*square(3) + 4*3 + 1 = 4*9 + 4*3 + 1 = 49

Si n es par, puede ser escrito como:

- n = 2*x
- n^2 = (2*x)2 = 4*x2

Si n es impar, puede ser escrito como:

- n = 2*x + 1
- n^2 = (2*x + 1)2 = 4*x2 + 4*x + 1

floor(n/2) puede ser calculado usando el operados de BitWise corrimiento derecho.

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
// Programa en java para calcular el cuadrado
// Sin usar * ni Pow()

public class NSquareNoPow {

    static int square(int n) {
        // Caso base
        if (n == 0) {
            return 0;
```

```
        }
        if (n < 0) {
            n = -n;
        }
        // Obtener floor(n/2) usando
        // right shift
        int x = n >> 1;
        // si n es par
        ;
        if (n % 2 != 0) {
            return ((square(x) << 2)
                    + (x << 2) + 1);
        } else// si n es impar
        {
            return (square(x) << 2);
        }
    }

    public static void main(String args[]) {
        for (int n = 1; n <= 5; n++) {
            System.out.println("n = " + n
                    + " n^2 = "
                    + square(n));
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define c(x) cout<<x<<endl;
#define cn(x) cout<<x;
#define l(c) cin>>t;
using namespace std;
int square(int n){
    if(n==0){
        return 0;
    }
    if(n<0){
        n=-n;
    }
    int x=n>>1;
    if((n&1)>0){
        return ((square(x)<<2)+(x<<2)+1);
    }else{
        return (square(x)<<2);
    }
}
int main() {
    for(int i=0;i<10;i++){
        c(square(i));
    }
}
```

PYTHON

```python
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write

def square(n):
    if n == 0:
        return 0
    if n < 0:
        n = -n
    x = n >> 1
    if n & 1:
        return (square(x) << 2) + (x << 2) + 1
    else:
        return (square(x) << 2)

for i in range(11):
    wr(f'{i} -> {square(i)}\n')
```

14.6) Palíndromo binario

Palíndromo binario

$$10101_{16} = 10000000100000001_{2}$$

$$333_{16} = 1100110011_{2}$$

$$52725_{16} = 1010010011100100101_{2}$$

$$7227_{16} = 111001000100111_{2}$$

$$90F09_{16} = 1010000111100001001_{2}$$

$$F9F_{16} = 111110011111_{2}$$

Guía del programador competitivo

Ilustración 14-2 Números cuya representación binaria es un palíndromo

Encuentre el número cuya representación binaria es un palíndromo. No se consideran los ceros iniciales, Mientras se considera la representación binaria, considere el primer número cuya representación binaria es palíndroma como 1, en vez de 0.

Una aproximación ingenua puede ser atravesar a través de todos los enteros desde 1 hasta 2^31-1 e incrementar el conteo palíndromo, si el número es palíndromo, cuando el conteo palíndromo alcanza el n requerido, rompe el ciclo y retorna el actual entero.

La complejidad de tiempo de esta solución es O(x) donde x es el número resultado. Note que el valor de x es generalmente más grande que n.

Complejidad de tiempo

Mejor caso : O(x) **Peor caso :** O(x) **Promedio:** O(x)

JAVA

```
/* Programa Java para buscar el Nesimo número el cual
su binario es un palíndromo*/
public class BinaryPalindromeN {

    static int INT_MAX = 2147483647;
    /*Busca si el kesimo bit esta
    puesto en la representación binaria*/
```

```
static int isKthBitSet(int x, int k) {
    return ((x & (1
            << (k - 1))) > 0) ? 1 : 0;
}

/*Retorna la posicion de más a la izquierda
el set de bits en la representacion binaria*/
static int leftmostSetBit(int x) {
    int count = 0;
    while (x > 0) {
        count++;
        x = x >> 1;
    }
    return count;
}

/*Encuentra cuales sean los enteros en binario
siendo plindromos o no*/
static int isBinPalindrome(int x) {
    int l = leftmostSetBit(x);
    int r = 1;
    //Uno a uno se comparan los bits
    while (l > r) {
    //Comprara bits de izquierda y derecha
    // y converge
        if (isKthBitSet(x, l)
                != isKthBitSet(x, r)) {
            return 0;
        }
        l--;
        r++;
    }
    return 1;
}

static int findNthPalindrome(int n) {
    int pal_count = 0;
    /*Comienza desde 1, atravieza por todos los
    enteros*/
    int i = 0;
    for (i = 1; i <= INT_MAX; i++) {
        if (isBinPalindrome(i) > 0) {
            pal_count++;
        }
        /*Si nosotros llegamos n
        rompe el ciclo*/
        if (pal_count == n) {
            break;
        }
    }
    return i;
}

public static void main(String[] args) {
```

```
        int n = 9;
        System.out.println(findNthPalindrome(n));
    }
}
```

PYTHON

```
from sys import maxsize
from sys import stdin
from sys import stdout
rl = stdin.readline
wr = stdout.write

def iskthBitSet(x, k):
    return 1 if (x & (1 << (k-1))) > 0 else 0

def leftMostSetBit(x):
    cont = 0
    while x > 0:
        cont += 1
        x = x >> 1
    return cont

def isBinPal(x):
    l = leftMostSetBit(x)
    r = 1
    while l > r:
        if iskthBitSet(x, 1) != iskthBitSet(x, r):
            return 0
        l -= 1
        r += 1
    return 1

def findNthPal(n):
    pal_cont = 0
    i = 0
    for i in range(maxsize):
        if isBinPal(i) > 0:
            pal_cont += 1
        if pal_cont == n:
            break
    return i

string = rl()
n = int(string, 16)
res = findNthPal(n)
wr(f'{res} -> {bin(res)[2:]}')
```

14.7) Número más cercano con los mismos bits

Dado un entero positivo n, imprimir el siguiente más pequeño y el previo más largo que tiene el mismo número de bits 1 en su representación binaria.

Aproximación por fuerza bruta:

Una simple aproximación es contar el número de 1 en n, y luego incrementar o decrementar hasta que encontremos un número con el mismo número de 1.

Aproximación optimizada:

Vamos a inicial con el código de getNext, y luego nos movemos a getPrev.

Aproximación de manipulación de bits para obtención del siguiente número:

Si pensamos acerca cual será el siguiente número, podemos observar lo siguiente, dado el número 13948, su representación binaria es:

1 1 0 1 1 0 0 1 1 1 1 1 0 0

13 12 11 10 9 8 7 6 5 4 3 2 1 0

Buscamos hacer este número más grande, pero no muy grande, también necesitamos mantener el mismo número de unos.

Nota: Dado un número N y dos localizaciones de bits i y j, suponga que volteamos bit i desde 1 a 0, y bit j desde 0 a 1, si i>j, entonces n podrá decrementarse, si i<j entonces n podrá incrementarse.

Sabemos lo siguiente:

- Si volteamos un cero a un uno, debemos voltear un uno a un cero.
- El número (Luego de dos volteretas) podría ser más grande si y solo si el bit cero a uno fue el izquierdo del bit uno a cero.

Nosotros queremos hacer el número más grande, pero no necesariamente más grande, por lo tanto necesitamos voltear el cero de más a la derecha el cual tiene unos en el derecho de él.

Para pone resto en una forma diferente, nosotros estamos volteando el cero no final de más a la derecho, esto es usando el ejemplo de abajo, los ceros finales son en la primera y

cero posición. El cero no final de más a la derecha es un bit 7, vamos a llamar esta posición p.

- p -> posición de más a la derecha que no sea cero final.

Paso 1: volteamos el cero más a la derecha no final.

- 1 1 0 1 1 0 1 1 1 1 1 1 0 0
- 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Con este cambio, hemos incrementado el número de unos en n, podemos encojer el número reordenando todos los bits de la derecha del bit p tales que los ceros están en la izquierda y los unos están a la derecha, cuando se hace esto, se busca reemplazar uno de los unos con cero.

Una forma relativamente fácil de hacer esto es contando cuantos unos están a la derecha de p, despejar todos los bits desde 0 hasta p, y luego agregar de nuevo en c1-1 unos. Dejar c1 ser el número de unos de la derecha de p y c0 el número de ceros de la derecha de p.

Vamos a verificar esto con un ejemplo:

- c1-> Número de unos de la derecho de p
- c0-> Número de ceros de la derecho de p
- p = c0 + c1

Paso 2: despejar los bits de la derecha de p, como antes c0 = 2. c1 = 5. p = 7.

- 1 1 0 1 1 0 1 0 0 0 0 0 0 0
- 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Para despejar estos bits, necesitamos crear una máscara que sea una secuencia de unos, seguido por p ceros, podemos hacer esto de la siguiente forma:

Todos los ceros excepto por un 1 en la posición p.

- a = 1 << p;

Todos los ceros, seguidos por p unos.

- b = a - 1;

Todos los unos, seguidos por p ceros.

- mask = ~b;

Despeja los p bits de más a la derecha.

- n = n & mask;

O más concisamente, hacemos:

- n &= ~ ((1 << p) - 1).

Paso 3: Agrega un c1=1 unos.

- 1 1 0 1 1 0 1 0 0 0 1 1 1 1
- 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Para insertar c1-1 unos a la derecha, se realiza:

Ceros con un uno en la posición c1– 1

- a = 1 << (c1 - 1);

Ceros con unos en posiciones cero a través de c1-1

- b = a - 1;

Inserta unos en las posiciones 0 a través de c1-1

- n = n | b;

O más concisamente:

- n | = (1 << (c1 - 1)) - 1;

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
/* Programa JAVA de busqueda del siguiente número con
la misma cantidad de 1 que algun anterior*/
public class ClosestNumbersSamebits {

    static int getNext(int n) {
        // Computa c0 y c1
        int c = n;
        int c0 = 0;
        int c1 = 0;
        while (((c & 1) == 0)
                && (c != 0)) {
            c0++;
            c >>= 1;
        }
        while ((c & 1) == 1) {
            c1++;
            c >>= 1;
        }
        /*Si no hay número más grande
        con el número de 1 buscado*/
        if (c0 + c1 == 31
                || c0 + c1 == 0) {
            return -1;
        }
        //Posicion de el más derecho cero
        int p = c0 + c1;
        //Voltea el zero más derecho
        n |= (1 << p);
        // Despeja todos los bits de la derecha de p
        n &= ~((1 << p) - 1);
        //Inserta los (c1-1) a la derecha
        n |= (1 << (c1 - 1)) - 1;
        return n;
    }

    static int getPrev(int n) {

        int temp = n;
        int c0 = 0;
        int c1 = 0;
        while ((temp & 1) == 1) {
            c1++;
            temp = temp >> 1;
        }
        if (temp == 0) {
            return -1;
        }
        while (((temp & 1) == 0)
                && (temp != 0)) {
            c0++;
            temp = temp >> 1;
        }
        // posicion de el cero no final de
        // mas a la derecha
```

```
        int p = c0 + c1;
        //limpia del bit c hacia adelante
        n = n & ((~0) << (p + 1));
        // Secuencia de (c1+1) unos
        int mask = (1 << (c1 + 1)) - 1;
        n = n | mask << (c0 - 1);
        return n;
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.println(n + "->" + Integer.toString(n, 2) + " " +
getNext(n));

        System.out.println(n + "->" + Integer.toString(n, 2) + " " +
getPrev(n));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int getNext(int n){
    int c=n;
    int c0=0;
    int c1=0;
    while((c&1)==0 && (c!=0)){
        c0++;
        c >>=1;
    }
    while((c&1)==1){
        c1++;
        c >>=1;
    }
    if(c0+c1 ==31 || c0+c1==0){
        return -1;
    }
    int p=c0+c1;
    n|=(1<<p);
    n&=~((1<<p)-1);
    n|=(1<<(c1-1))-1;
    return n;
}

int getPrev(int n){
    int temp=n;
    int c0=0;
    int c1=0;
    while((temp&1)==1){
        c1++;
        temp >>=1;
    }
```

```
    if(temp==0){
        return -1;
    }
    while(((temp&1)==0) && (temp!=0)){
        c0++;
        temp >>=1;
    }
    int p=c0+c1;
    n&=((~0)<<(p+1));
    int mask=(1<<(c1+1))-1;
    n|=mask<<(c0-1);
    return n;
}
int main() {
    int n=5;
    cout<<(getNext(n))<<endl;
    n=5;
    cout<<(getPrev(n))<<endl;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write


def getNext(n):
    c = n
    c0 = 0
    c1 = 0
    while (c & 1 == 0) and (c != 0):
        c0 += 1
        c >>= 1
    while (c & 1) == 1:
        c1 += 1
        c >>= 1
    if (c0 + c1 == 31) or (c0 + c1 == 0):
        return -1
    p = c0 + c1
    # Mascaras de Bits
    n |= (1 << p)
    n &= ~((1 << p)-1)
    n |= (1 << (c1-1))-1
    return n


def getPrev(n):
    temp = n
    c0 = 0
    c1 = 0
    while (temp & 1) == 1:
        c1 += 1
        temp = temp >> 1
```

```
    if temp == 0:
        return -1
    while (temp & 1 == 0) and (temp != 0):
        c0 += 1
        temp = temp >> 1
    p = c0 + c1
    n &= ((~0) << (p + 1))
    mask = (1 << (c1 + 1)) - 1
    n |= mask << (c0 - 1)
    return n


n = int(rl())
wr(f'{n} -> {bin(n)[2:]}\n')
wr(f'{getNext(n)} -> {bin(getNext(n))[2:]}\n')
wr(f'{getPrev(n)} -> {bin(getPrev(n))[2:]}')
```

14.8) Códigos de Gray a Binario e inversos

Código de gray

DECIMAL	BINARIO	BCD 8421	2421	Exce. a 3	Biquinario 5043210
0	0000	0000	0000	0011	0100001
1	0001	0001	0001	0100	0100010
2	0010	0010	0010	0101	0100100
3	0011	0011	0011	0110	0101000
4	0100	0100	0100	0111	0110000
5	0101	0101	1011	1000	1000001
6	0110	0110	1100	1001	1000010
7	0111	0111	1101	1010	1000100
8	1000	1000	1110	1011	1001000
9	1001	1001	1111	1100	1010000

Guía del programador competitivo

Ilustración 14-3 Códigos de Gray para un numero de 4 bits

Números binarios es la forma por defecto para almacenar números, pero en muchas aplicaciones los números binarios son difíciles de usar y una variación de los números binarios es necesaria, aquí es cuando los códigos de Gray son muy útiles.

El código de gray tiene una propiedad, dos números sucesivos difieren en un solo bit porque esta propiedad permite tener ciclo a través de varios estados con un esfuerzo mínimo y son usados en mapas k, también permiten corrección de errores, comunicación entre muchas cosas más.

¿Cómo generar códigos de Gray de n bits?

La siguiente es la secuencia de dos bits (n=2)

- 00 01 11 10

La siguiente es la secuencia de 3 bits (n = 3)

- 000 001 011 010 110 111 101 100

Y la siguiente es la secuencia de 4 bits (n = 4)

- 0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110 1010 1011 1001 1000

Los códigos de Gray de n bits pueden ser generados de la lista de (n-1) códigos de Gray con los siguientes pasos.

- Dejar la lista de (n-1)bits ser L1, crear otra lista L2 la cual es la reversa de L1
- Modificar la lista L1 usando prefijo ‘0’ en todos los códigos de L1
- Concatene L1 y L2. La lista concatenada es la lista requerida de los códigos de Gray de n bits

En ciencias de la computación muchas veces necesitamos convertir de código binario a código de Gray y viceversa, esta conversion puede ser realizada bajo las siguientes reglas:

Conversión de binario a Gray:

El bit más significante (MSB most significant bit) del código de gray es siempre igual a el MSB del código binario dado.

Otros bits de la salida del código de gray pueden ser obtenidos realizando XOR al bit del código binario en ese índice y en el índice anterior.

Conversión de Gray a binario:

El MSB del código binario es siempre igual al MSB del código de Gray.

Los otros bits de la salida del código binario pueden ser obtenidos verificando el bit del código Gray en ese incide, si el bit actual es 0, entonces copia el anterior bit del código binario, si no copia el inverso del anterior bit del código binario.

Complejidad de tiempo

Mejor caso : $O(n^2)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```
// Programa en java para coversion binario - Gray e inverso

import java.io.*;

public class CodeConversionGrayToBinary {

    public static void main(String args[]) throws IOException {

        String binary = "01001";
        System.out.println("Codigo gray de " + binary + " is " +
binarytoGray(binary));

        String gray = "01101";
        System.out.println("Codigo Binaruo de " + gray + " is " +
graytoBinary(gray));
    }

    static char xor_c(char a, char b) {
        return (a == b) ? '0' : '1';
    }
    //Funcion para voltear el bit

    static char flip(char c) {
        return (c == '0') ? '1' : '0';
    }
    //Funcion binario a gray

    static String binarytoGray(String binary) {
        String gray = "";
        gray += binary.charAt(0);
        // Coomputa bits restantes, siguiente bit es conmutado haciendole
        // XOR del previo con el actual en binario
        for (int i = 1; i < binary.length(); i++) {
```

```
            /*Comcatena XOR del bit anterior con el actual*/
            gray += xor_c(binary.charAt(i - 1),
                    binary.charAt(i));
        }
        return gray;
    }
    //Funcion gray a binario
    static String graytoBinary(String gray) {
        String binary = "";
        binary += gray.charAt(0);
        // Computa bits restantes,
        for (int i = 1; i < gray.length(); i++) {
            //Si el bit actual es 0, concatena el bit anterior
            if (gray.charAt(i) == '0') {
                binary += binary.charAt(i - 1);
            } //Sino, concatena invertidamente el bit anterior
            else {
                binary += flip(binary.charAt(i - 1));
            }
        }
        return binary;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
char xor_c(char a,char b){
    return (a==b)?'0':'1';
}
char flip(char c){
    return (c==0)?'1':'0';
}
string grayToBinary(string gray){
    string binary = "";
    int tam = gray.size();
    binary += gray[0];
    for(int i = 1; i < tam; i++){
        if(gray[i] == '0'){
            binary += binary[i - 1];
        }else{
            binary += flip(binary[i - 1]);
        }
    }
    return binary;
}
string binaryToGrey(string binary){
    string grey="";
    grey+=binary[0];
    for(int i=1;i<binary.size();i++){
        grey+=xor_c((binary[i-1]),(binary[i]));
    }
```

```
    return grey;
}
int main() {
    string n="01001";
    cout<<(binaryToGrey(n))<<endl;
    n="01001";
    cout<<(grayToBinary(n))<<endl;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def binarytoGray(binary):
    gray = ''
    gray += binary[0]
    for i in range(1, len(binary)):
        gray += xor_c(binary[i-1], binary[i])
    return gray

def xor_c(a, b):
    return '0' if a == b else '1'

def flip(c):
    return '1' if c == '0' else '0'

def graytoBinary(gray):
    binary = ''
    binary += gray[0]
    for i in range(1, len(gray)):
        if gray[i] == '0':
            binary += binary[i-1]
        else:
            binary += flip(binary[i-1])
    return binary

b1 = '01001'
res = binarytoGray(b1)
wr(f'{b1} -> {res}\n')
res2 = graytoBinary(res)
wr(f'{res} -> {res2}')
```

14.9) Códigos de Gray de N bits

Dado un número n, generar los patrones de bits de 0 a 2^n-1 de los cuales cada patrón sucesivo difiera en uno solo bit.

Los códigos de Gray de n bits pueden ser generados de la lista de (n-1) códigos de Gray con los siguientes pasos.

- Dejar la lista de (n-1)bits ser L1, crear otra lista L2 la cual es la reversa de L1
- Modificar la lista L1 usando prefijo ‘0’ en todos los códigos de L1
- Concatene L1 y L2. La lista concatenada es la lista requerida de los códigos de Gray de n bits

Por ejemplo, los siguientes son los pasos para generar el código de Gray de 3 bits desde la lista de códigos de Gray de 2 bits.

- L1 = {00, 01, 11, 10} (Lista de Gray de dos bits)
- L2 = {10, 11, 01, 00} (Reversa de L1)
- Agrega prefijo a todas las entradas de L1 con ‘0’, L1 se convierte en {000, 001, 011, 010}
- Agrega prefijo a todas las entradas de L2 con ‘1’, L2 se convierte en {110, 111, 101, 100}
- Concatena L1 y L2, obtenemos {000, 001, 011, 010, 110, 111, 101, 100}

Para generar los códigos de Gray de n bits, empezamos desde la lista de Grays de un solo bit, la cual es {0,1}, repetimos los pasos de arriba para general la lista de 2 bits a partir de la de 1 solo bit, luego generamos la de 3 bits a partir de la de 2 bits, y así hasta que el número de bits sea igual a n.

Complejidad de tiempo

Mejor caso : O(nlog(n)) **Peor caso :** O(nlog(n)) **Promedio:** O(nlog(n))

JAVA

```
//Programa Java para generar codigos de Gray Nesimos

import java.util.ArrayList;

public class GrayCodesOfN {

    static void generateGrayarr(int n) {
        // Caso base
```

```
        if (n <= 0) {
            return;
        }
        // 'arr' podria almacenar todos los codigos generados
        ArrayList<String> arr = new ArrayList<String>();
        // Comienza con un patron de un bit
        arr.add("0");
        arr.add("1");
        /* Cada iteracion de este ciclo genera 2*i codigos desde los
        i codigos generados previamente*/
        int i, j;
        for (i = 2; i < (1 << n); i = i << 1) {
            /*Entra los previamente generados codigos de nuevo a arr[]
                en orden reverso, arr[] tiene el doble de número de codigos*/
            for (j = i - 1; j >= 0; j--) {
                arr.add(arr.get(j));
            }
            // Concatena 0 a la primera mitad
            for (j = 0; j < i; j++) {
                arr.set(j, "0" + arr.get(j));
            }
            // Concatena 1 a la segunda mitad
            for (j = i; j < 2 * i; j++) {
                arr.set(j, "1" + arr.get(j));
            }
        }
        // Imprime el contenido de arr
        for (i = 0; i < arr.size(); i++) {
            System.out.println(arr.get(i));
        }
    }

    public static void main(String[] args) {
        generateGrayarr(3);
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
void generateGray(int n){
    if(n<=0){
        return;
    }
    vector<string>arr;
    arr.push_back("0");
    arr.push_back("1");
    for(int i=2;i<(1<<n);i <<=1){
        for(int j = i-1;j>=0;j--){
            arr.push_back(arr[j]);
        }
        for(int j=0;j<i;j++){
```

```
            arr[j]="0"+arr[j];
        }
        for(int j=i;j<2*i;j++){
            arr[j]="1"+arr[j];
        }
    }
    for(int i=0;i<arr.size();i++){
        cout<<arr[i]<<endl;
    }
}
int main(){
    generateGray(5);
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def generateGray(n):
    if n <= 0:
        return
    arr = []
    arr.append('0')
    arr.append('1')
    i = 2
    j = 0
    while i < (1 << n):
        for j in range(i-1, -1, -1):
            arr.append(arr[j])
        for j in range(0, i):
            arr[j] = '0' + arr[j]
        for j in range(i, 2*i):
            arr[j] = '1' + arr[j]
        i = i << 1
    for i in range(len(arr)):
        wr(f'{arr[i]}\n')

generateGray(5)
```

14.10) Conteo de bits a activar para que A sea B

Dados dos números ‘a’ y ‘b’, contar el número de bits necesarios a voltear para convertir ‘a’ en ‘b’.

1) Calcular el XOR de A y B: a_xor_b = A ^ B
2) Contar los bits encendidos de lo de arriba

Resultado:

countSetBits(a_xor_b)

XOR de dos números tendrá bits encendidos solo en esos lugares donde A difiera de B

Complejidad de tiempo

Mejor caso : O(bits) **Peor caso :** O(bits) **Promedio:** O(bits)

JAVA

```java
//Contar el número de bits que van a ser volteados para convertir A en B

public class CountAtoB {

    public static int countSetBits(int n) {
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>= 1;
        }
        return count;
    }

    public static int FlippedCount(int a, int b) {
        //Retorna el conteo del set de bits en a XOR b
        return countSetBits(a ^ b);
    }

    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        System.out.print(FlippedCount(a, b));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int countSetBits(int n){
    int cont=0;
    while(n!=0){
        cont+=n&1;
        n>>=1;
    }
```

```
    return cont;
}
int flippedCount(int a,int b){
    return countSetBits(a^b);
}
string toBinary(int n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    return r;
}
int main(){
    int a=5;
    int b=100;
    cout<<a<<"->"<<toBinary(a)<<endl;
    cout<<b<<"->"<<toBinary(b)<<endl;
    cout<<flippedCount(a,b)<<endl;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def countSetBits(n):
    count = 0
    while n != 0:
        count += n & 1
        n >>= 1
    return count

def flipperCount(a, b):
    return countSetBits(a ^ b)

a = 50
b = 100
wr(f'{a} -> {bin(a)[2:]}\n')
wr(f'{b} -> {bin(b)[2:]}\n')
wr(f'{flipperCount(a,b)}')
```

14.11) Conteo de bits activos

Dado un entero positivo n, cuente el número total de bits encendidos en representación binaria de todos los números de 1 a n.

Una solución simple es correr un ciclo desde 1 hasta n y sumar el conteo de todos los números de 1 a n.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Un programa simpre que cuenta los bits set (Encendidos)
//en todos los números de 1 a n.

public class CountSetBits {

    static int countSetBits(int n) {
        int bitCount = 0;
        for (int i = 1; i <= n; i++) {
            bitCount += countSetBitsUtil(i);
        }
        return bitCount;
    }

    static int countSetBitsUtil(int x) {
        if (x <= 0) {
            return 0;
        }
        return (x % 2 == 0 ? 0 : 1)
                + countSetBitsUtil(x / 2);
    }

    public static void main(String[] args) {
        int n = 4;
        System.out.print("Conteo total del set de bits es ");
        System.out.print(countSetBits(n));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
string toBinary(int n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    return r;
}
int countSetBitUtil(int x){
    if(x<=0){
        return 0;
    }
```

```
    return (x%2==0?0:1)+countSetBitUtil(x/2);
}
int countBitSetBits(int n){
    int bitCount=0;
    for(int i=1;i<=n;i++){
        cout<<i<<"->"<<toBinary(i)<<endl;
        bitCount+=countSetBitUtil(i);
    }
    return bitCount;
}
int main(){
    int n=4;
    cout<<countBitSetBits(n)<<endl;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def countSetBit(n):
    bitcount = 0
    for i in range(1, n+1):
        wr(f'{i} -> {bin(i)[2:]}\n')
        bitcount += countSetBitsUtil(i)
    return bitcount

def countSetBitsUtil(x):
    if x <= 0:
        return 0
    return (0 if (x & 1 == 0) else 1) + countSetBitsUtil(x//2)

n = 5
wr(f'{countSetBit(n)}')
```

14.12) Euclides sin modulo ni división

El algoritmo de Euclides es usado para encontrar el GCD de dos números.

La idea es usar operaciones de BitWise, podemos encontrar x/2 usando x>>1. Podemos verificar si x es par o impar usando x&1.

- gcd(a, b) = 2*gcd(a/2, b/2) si ambos a y b son pares.
- gcd(a, b) = gcd(a/2, b) si a es par y b es impar.
- gcd(a, b) = gcd(a, b/2) si a es impar y b es par.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```java
//Programa Java eficiente para realizar maximo comun divisor sin % y /

public class EuclidNoModAndDivide {

    public static void main(String[] args) {
        System.out.println(gcd(8, 9));
    }

    static int gcd(int a, int b) {
        // Casos base
        if (b == 0 || a == b) {
            return a;
        }
        if (a == 0) {
            return b;
        }
        /*Si ambos a y b son pares, divide ambos por 2
        y multiplica el resultado con 2*/
        if ((a & 1) == 0 && (b & 1) == 0) {
            return gcd(a >> 1, b >> 1) << 1;
        }
        //Si a es par, y b es impar, divide a por 2
        if ((a & 1) == 0 && (b & 1) != 0) {
            return gcd(a >> 1, b);
        }
        //Si a es impar y b es par, divide b por 2
        if ((a & 1) != 0 && (b & 1) == 0) {
            return gcd(a, b >> 1);
        }
        /*Si ambos son impares, entonces aplica el algoritmode
        resta normal, notese que el casp impar-impar siempre
        convierte casos impar-par  luego de una recursion*/
        return (a > b) ? gcd(a - b, b) : gcd(a, b - a);
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int gcd(int a,int b){
    if (b == 0 || a == 0) {
        return a;
    }
    if (a == 0) {
        return b;
```

```
    }
    if ((a & 1) == 0 && (b & 1) == 0) {
        return gcd(a >> 1, b >> 1) << 1;
    }
    if ((a & 1) == 0 && (b & 1) != 0) {
        return gcd(a >> 1, b);
    }
    if ((a & 1) != 0 && (b & 1) == 0) {
        return gcd(a, b >> 1);
    }
    return (a > b) ? gcd(a - b, b) : gcd(a, b - a);
}
int main(){
    cout<<gcd(50,30)<<endl;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def gcd(a, b):
    if b == 0 or a == 0:
        return a
    if a == 0:
        return b
    if (a & 1 == 0) and (b & 1 == 0):
        return gcd(a >> 1, b >> 1) << 1
    if (a & 1 == 0) and (b & 1 != 0):
        return gcd(a >> 1, b)
    if (a & 1 != 0) and (b & 1 == 0):
        return gcd(a, b >> 1)
    return gcd(a - b, b) if a > b else gcd(a, b - a)

wr(f'{gcd(50,10)}')
```

14.13) Buscar duplicados usando un arreglo de bits

Se tiene un arreglo de N números, donde n es al menos 32000, el array puede tener entradas duplicadas y no se sabe que N es.

Con solo 4 kilobytes de memoria disponible, ¿cómo podría imprimir todos los elementos duplicados en el array?

Tenemos 4 kilobytes de memoria lo cual significa que podemos direccionar hasta 8*4*210 bits, note que 32*210 es más grande que 32000, podemos crear un array de bits con 32000 bits, donde cada bit representa un entero.

Si se necesita crear un array de bit con más de 32000 bits entonces se puede crear fácilmente más y más de 32000.

Usando este vector de bits, podemos entonces iterar a través del arreglo, marcando cada elemento v poniendo el bit v en 1, cuando pasemos por un elemento duplicado, los imprimimos.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Programa Java para imprimir todos los duplicados en un arreglo

public class FindDuplicatesBitArray {

    static class BitArray {

        int[] arr;
        // Constructor
        public BitArray(int n) {
            /*Divide por 32, para almacenar n bits, nosotros necesitamos
            n/32 +1 enteros (Asumiendo int esta almacenado usando 32 bits*/
            arr = new int[(n >> 5) + 1];
        }
        // Obtener el valor de un bit en una posicion dada
        boolean get(int pos) {
            //Divide por 32 para encontrar la posicion del entero
            int index = (pos >> 5);
            //Ahora encuentra el bumero de bits en arr[index]
            int bitNo = (pos & 0x1F);
            //Encuentra el valor dado un número bit en arr[index]
            return (arr[index] & (1 << bitNo)) != 0;
        }
        // Acomoda un bit en una posicion dada
        void set(int pos) {
            // Encuentra indice de una posicion de un bit
            int index = (pos >> 5);
            //Acomoda un número bot es arr[index]
            int bitNo = (pos & 0x1F);
            arr[index] |= (1 << bitNo);
        }
        // Funcion de impresion de los duplicados
        static void checkDuplicates(int[] arr) {
```

```
            // Crea un bit con 320000 bits
            BitArray ba = new BitArray(320000);
            // Arreglo transverso de los elementos
            for (int i = 0; i < arr.length; i++) {
                // Indice de un arreglo de bits
                int num = arr[i] - 1;
                //Si número ya se encuentra presente en el arreglo de bits
                if (ba.get(num)) {
                    System.out.print((num + 1) + " ");
                } // Si no inserte el número
                else {
                    ba.set(num);
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {10, 10, 1, 1, 2, 2, 3, 3};
        BitArray.checkDuplicates(arr);
    }
}
```

C++

```
#include <iostream>
#include <vector>

using namespace std;

struct BitArray {
    vector <int> arr;

    BitArray(int n) {
        arr.resize((n >> 5) + 1);
    }

    bool get(int pos) {
        int index = (pos >> 5);
        int bitNo = (pos & 0x1F);
        return (arr[index] & (1 << bitNo)) != 0;
    }

    void setPos(int pos) {
        int index = (pos >> 5);
        int bitNo = (pos & 0x1F);
        arr[index] |= (1 << bitNo);
    }

    void checkDuplicates(vector <int>& arr, BitArray ba) {
        for (int i = 0; i < arr.size(); i++) {
            int num = arr[i] - 1;
            if (ba.get(num)) {
                cout << (num + 1) << " ";
            } else {
```

```
                ba.setPos(num);
            }
        }
    }
};

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    vector <int> arr{10, -10, -10, 8, 8, 5, 3, 4, 4, 3, 7, 9, -9, -8, -9};
    BitArray ba(32000);
    ba.checkDuplicates(arr, ba);
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

class BitArray:
    arr = []
    def __init__(self, n):
        self.arr = [0 for x in range((n >> 5)+1)]

    def get(self, pos):
        self.index = (pos >> 5)
        self.bitNo = (pos & 0x1f)
        return (self.arr[self.index] & (1 << self.bitNo)) != 0

    def set(self, pos):
        self.index = (pos >> 5)
        self.bitNo = (pos & 0x1f)
        self.arr[self.index] |= (1 << self.bitNo)

    def checkDuplicates(self, arr):
        for i in range(len(arr)):
            num = arr[i]-1
            if ba.get(num):
                wr(f'{num+1} ')
            else:
                ba.set(num)

arr = [10, 10, 10, 5, 58, 8, 1, 5, 3, 2, 16, 4, 4]
ba = BitArray(32000)
ba.checkDuplicates(arr)
```

14.14) Máximo de unos consecutivos

Dado un entero n, pudiendo voltear exactamente un bit, encuentre la longitud de la secuencia más larga de 1 que se pueda crear.

Una solución eficiente es andar a través de los bits en representación binaria del número dado, mantenemos rastreo de la longitud de la secuencia actual de unos, de la longitud de la secuencia de unos previa, cuando veamos un cero, actualizamos la longitud anterior:

- Si el siguiente bit es un 1, la longitud anterior deberá ser ahora la longitud actual.
- Si el siguiente bit es un 0, entonces no podemos unir esas secuencias juntas, entonces la longitud previa es 0.

Actualizamos la longitud máxima comparando las dos siguientes cosas:

- Valor actual de max_lenght
- Current-lenght+previos-length

Entonces

- result = return max-length+1 (// Agrega 1 para el Contador de bits volteados)

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
//Programa java para buscar el subset de 1 más largo

public class MaxConsecutiveOne {

    static int flipBit(int a) {
        /*Si todos los bits son 1, la representacion de 'a'
        tiene todos los unos*/
        if (~a == 0) {
            return 8 * sizeof();
        }
        int currLen = 0, prevLen = 0, maxLen = 0;
        while (a != 0) {
            /*Si el bit actual es un 1
            entonces incrementa currLen++*/
```

```
            if ((a & 1) == 1) {
                currLen++;
            } /*Si el bit actual es un 0
            revisa el siguiente bit de a*/ else if ((a & 1) == 0) {
                /*Actualiza prevLan a 0 (Si el siguiente bit es 0)
                o currLen (Si el siguiente bit es 1)*/
                prevLen = (a & 2) == 0 ? 0 : currLen;
                /*Si dos bits consecutivos son 0
                entonces currLen tambien sera 0*/
                currLen = 0;
            }
            // Actualiza maxLen si es requerido
            maxLen = Math.max(prevLen + currLen, maxLen);
            //Remueve el ultimo digito (Right shift)
            a >>= 1;
        }
        /*Nosotros siempre podremos tener un secuencia de
        al menos un 1, este es un bit volteado*/
        return maxLen + 1;
    }

    static byte sizeof() {
        byte sizeOfInteger = 8;
        return sizeOfInteger;
    }

    public static void main(String[] args) {
        System.out.println(flipBit(13));
        System.out.println(flipBit(1775));
        System.out.println(flipBit(15));
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int SIZE(){
    int sizeOfInteger=8;
    return sizeOfInteger;
}
int flipBit(int a){
    if(~a==0){
        return 8*SIZE();
    }
    int currentlen=0,prevlen=0,maxlen=0;
    while(a!=0){
        if((a&1)==1){
            currentlen++;
        }
        else if((a&1)==0){
            prevlen=(a&2==0)?0:currentlen;
            currentlen=0;
```

```
        }
        maxlen=std::max(prevlen+currentlen,maxlen);
        a>>=1;
    }
    return maxlen+1;
}
string toBinary(int n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    return r;
}
int main(){
    int a=654321;
    cout<<a<<"->"<<toBinary(a)<<" : "<<flipBit(a)<<endl;

}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def flipBit(a):
    if ~a == 0:
        return 8 * sizeof()
    currlen, prevlen, maxlen = 0, 0, 0
    while a != 0:
        if a & 1 == 1:  # Cambiar a 0
            currlen += 1
        elif a & 1 == 0:  # Cambiar a 1
            prevlen = 0 if ((a & 2) == 0) else currlen
            currlen = 0
        maxlen = max(prevlen + currlen, maxlen)
        a >>= 1
    return maxlen + 1

def sizeof():
    sizeOfInteger = 8
    return sizeOfInteger

a = int(rl())
wr(f'{a} -> {bin(a)[2:]}\n')
wr(f'{flipBit(a)}')
```

14.15) Máximo XOR Subarray

Dado un array de enteros, encontrar el valor máximo del Subarray XOR en el array dado. Una solución simple es usar dos ciclos para encontrar el XOR de todos los subarrays y retornar el máximo.

Complejidad de tiempo

Mejor caso : $O(n^2)$ **Peor caso :** $O(n^2)$ **Promedio:** $O(n^2)$

JAVA

```java
// Programa en JAVA para buscar el maximo subarreglo XOR

public class MaxSubarrayXOR {

    static int maxSubarrayXOR(int arr[], int n) {
        int ans = Integer.MIN_VALUE; // Inicializar resultado
        // Escogiendo puntos de inicio para los subarreglos
        for (int i = 0; i < n; i++) {
            // para guardar XOR del actual subarreglo
            int curr_xor = 0;
            // Escojiendo puntos finales de subarreglos empezando por i
            for (int j = i; j < n; j++) {
                curr_xor = curr_xor ^ arr[j];
                ans = Math.max(ans, curr_xor);
            }
        }
        return ans;
    }

    public static void main(String args[]) {
        int arr[] = {8, 1, 2, 12};
        int n = arr.length;
        System.out.println("Maximo subarray XOR es "
                + maxSubarrayXOR(arr, n));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int maxSubArrayXOR(int arr[],int n){
    int ans=INT_MIN;
    for(int i=0;i<n;i++){
        int curr_xor=0;
        for(int j=i;j<n;j++){
            curr_xor = curr_xor ^ arr[j];
            ans = std::max(ans, curr_xor);
        }
```

```
    }
    return ans;
}
int main(){
    int arr[] = {8, 1, 2, 12};
    int n = sizeof (arr)/sizeof (arr[0]);
    cout<<"Maximo subarray XOR es "<<maxSubArrayXOR(arr,n)<<endl;
}
```

14.16) N número mágico

Un número mágico es definido como un número el cual puede ser expresado como una potencia de 5 o la suma de potencias únicas de 5, algunos números mágicos son 5, 25, 30(5 + 25), 125, 130(125 + 5), ….

Si revisamos cuidadosamente los números mágicos, pueden ser representados como 001, 010, 011, 100, 101, 110, entre otros, donde 001 es 0*pow(5,3)+0*pow(5,2)+1*pow(5,1), entonces básicamente necesitamos agregar potencias de 5 por cada bit dado en el entero dado n.

Complejidad de tiempo

Mejor caso : O(log(n)) **Peor caso :** O(log(n)) **Promedio:** O(log(n))

JAVA

```
// Programa en java para buscar el Nsimo
// número magico, un número magico esta definido como un número el cual puede
ser expresado como
// una potencia de 5 o suma de potencias unicas de 5
// Algunos primeros número magicos son: 5, 25, 30(5 + 25), 125, 130(125 + 5), …
en adelante

public class NMagicNumber {

    static int nthMagicNo(int n) {
        int pow = 1, answer = 0;
        // Ir a traves de cada bit de n
        while (n != 0) {
```

```
            pow = pow * 5;
            // Si el untimo bit de n esta puesto
            if ((n & 1) == 1) {
                answer += pow;
            }
            //Proceder con el siguiente bit
            // 0 n= n/2
            n >>= 1;
        }
        return answer;
    }

    public static void main(String[] args) {
        int n = 5;
        System.out.println("Enesimo número"
                + " magico es " + nthMagicNo(n));
    }
}
```

14.17) Intercambio de pares e impares

Dado un entero sin signo, intercambia todos los bits pares con impares, por ejemplo si el número dado es 23 (00010111), puede ser convertido en 43 (00101011). Cada posición de bit par es intercambiada con el bit adyacente del lado de derecho, y cada posición impar es cambiada con el adyacente del lado izquierdo.

Si realizamos una revisión en el ejemplo, podemos observar que básicamente necesitamos el corrimiento derecho (>>) de todos los bits pares por 1, entonces se convierten en bits impares, y corrimiento izquierdo (<<) todos los bits impares por 1 entonces se convierten en pares, la siguiente solución está basada en estas ideas, se asume que el número de entrada está almacenado usando 32 bits.

Dejemos la entrada ser x.

1) Obtener todos los bits pares de x realizando BitWise, y or de x con 0xAAAAAAAA, el número 0xAAAAAAAA es un entero de 32 bit con todos los bits pares ubicados en 1, y todos los impares en 0.

2) Obtener todos los bits impares usando BitWise, y or de x con 0x55555555, el número 0x55555555 es un número entero de 32 bits con todos los bits impares en 1 y los pares en 0.
3) Corrimiento derecho de todos los bits pares
4) Corrimiento izquierdo de todos los bits impares
5) Combina los nuevos pares e impares y retorna

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```java
// Programa java para invertir bits pares
// e impares de un número dado

public class OddEvenSwap {

    static int swapBits(int x) {
        // Obtener todos los bits par de x
        int even_bits = x & 0xAAAAAAAA;
        //Obtener todos los bits impar de x
        int odd_bits = x & 0x55555555;
        //  Movimiento derecho de bits pares
        even_bits >>= 1;
        // Movimiento izquierdo de bits impares
        odd_bits <<= 1;
        // combinar pares e impares
        return (even_bits | odd_bits);
    }

    public static void main(String[] args) {
        int x = 23; // 00010111
        // La salida es 43 (00101011)
        System.out.println(swapBits(x));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
int swapBits(int x){
    int even_bits=x & 0xAAAAAAAA;
    int oddbits=x & 0x55555555;
    even_bits>>=1;
    oddbits<<=1;
```

```
    return (even_bits|oddbits);
}
string toBinary(int n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    return r;
}
int main(){
   int x=7;
   cout<<x<<"->"<<toBinary(x)<<endl;
   int y=swapBits(x);
   cout<<swapBits(x)<<"->"<<toBinary(y)<<endl;

}
```

14.18) Ocurrencia impar

Dado un array de enteros positivos, todos los números ocurren número par de veces excepto un número el cual ocurre impar veces, encuentre el número en O(n) tiempo y espacio constante.

Una solución simple es correr dos ciclos anidados, el ciclo externo toma todos los elementos uno por uno y el ciclo interno cuenta el número de las ocurrencias del elemento tomado en el ciclo externo, la complejidad de tiempo de esta solución es O(n^2).

La mejor solución es hacer XOR bit a bit de todos los elementos. Un XOR de todos los elementos nos muestra cuales elementos son impares. Tenga en cuenta que XOR de dos elementos es 0 si ambos elementos son iguales y XOR de un número x con 0 es x.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```
//Programa JAVA para buscar la ocurrencia de un elemento
// número impar de veces

public class OddOccurence {
```

```
    static int getOddOccurrence(int ar[], int ar_size) {
        int i;
        int res = 0;
        for (i = 0; i < ar_size; i++) {
            res = res ^ ar[i];
        }
        return res;
    }

    public static void main(String[] args) {
        int ar[] = new int[]{2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};
        int n = ar.length;
        System.out.println(getOddOccurrence(ar, n));
    }
}
```

14.19) Multiplicación Russian Peasant

Russia peasant multiply

37	6
18	12
9	24
4	48
2	96
1	192

37	6
~~18~~	~~12~~
9	24
~~4~~	~~48~~
~~2~~	~~96~~
1	192
	222

Guía del programador competitivo

Ilustración 14-4 Ejemplo de la multiplicación de la campesina rusa

Dados dos enteros, multiplicar sin usar el operador de multiplicación.

Una forma interesante es usar el algoritmo de la campesina rusa, la idea es duplicar el primer número y dividir en dos el segundo número repetidamente mientras el segundo número no se convierta en 1, en el proceso cuando el segundo número se vuelta impar, añadimos el primer número al resultado, el cual esta inicializado en 0.

El valor de a*b es el mismo que (a*2)*(b/2), si b es par, de otra forma el valor es el mismo de ((a*2)*(b/2) + a), en el ciclo while, seguimos multiplicando 'a' con dos y seguimos dividiendo 'b' por 2, si 'b' se convierte en impar en el ciclo, agregamos 'a' a 'res' cuando el valor de 'b' se convierta en 1, el valor de 'res' + 'a' nos da el resultado.

Note que cuando 'b' es una potencia de 2, el 'res' puede mantenerse en 0 y 'a' puede ser la multiplicación.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
// Programa en java para multiplicar usando el algoritmo de  Russian Peasant

public class RussianPeasantMultiply {
    // Funcion para multiplicar dos números
    static int russianPeasant(int a, int b) {
        // Inicializar resultado
        int res = 0;
        //Mientras el segundo no se convierta en 1
        while (b > 0) {
            // Si el segundo número es  impar,
            // añade el primer número al resultado
            if ((b & 1) != 0) {
                res = res + a;
            }
            // El doble del primer número
            // y la mitad del segundo número
            a = a << 1;
            b = b >> 1;
        }
        return res;
    }

    public static void main(String[] args) {
        System.out.println(russianPeasant(18, 1));
        System.out.println(russianPeasant(20, 12));
    }
}
```

C++

```cpp
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;

int RP(int a, int b) {
    int res = 0;
    while (b > 0) {
        if ((b & 1) != 0) {
            res += a;
        }
        a <<= 1;
        b >>= 1;
    }
    return res;
}

int main() {
    cout << (RP(2, 18)) << endl;
}
```

PYTHON

```python
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

def RP(a, b):
    res = 0
    while b > 0:
        if b & 1 != 0:
            res += a
        a <<= 1
        b >>= 1
    return res

a, b = int(rl()), int(rl())
wr(f'{RP(a, b)}')
```

14.20) Multiplicación de Karatsuba

Multiplicación de Karatsuba

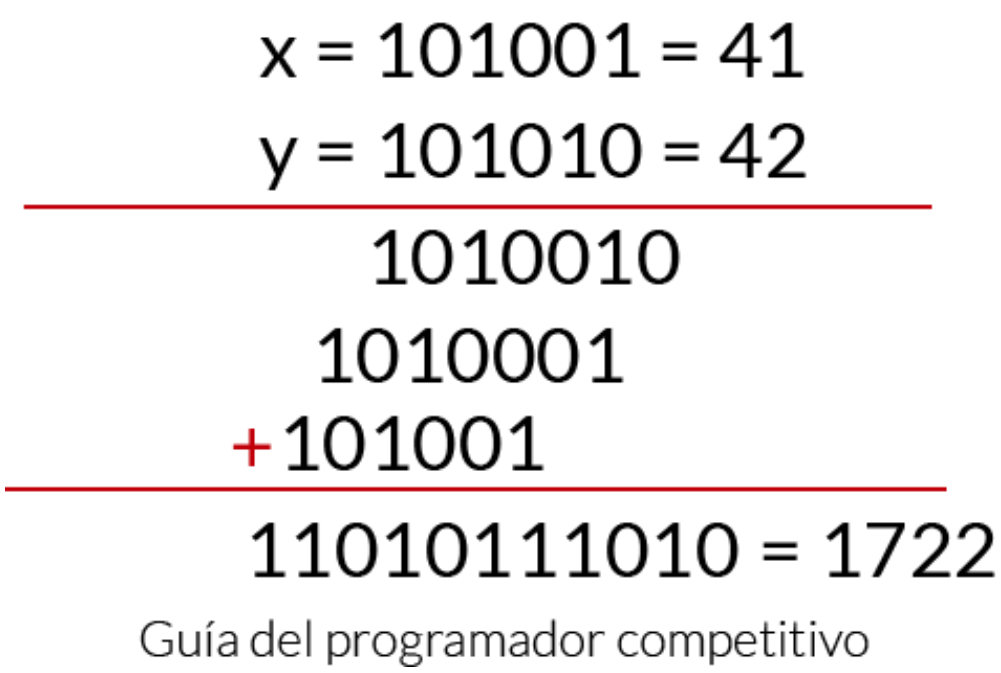


Ilustración 14-5 Ejemplo de la multiplicación de Karatsuba

Dados dos strings binarios (bigIntegers) que representan el valor de dos enteros, encuentre el producto de los dos strings, por ejemplo, si el primer string de bits es “1100” y el segundo string de bits es “1010” la salida debe ser 120.

Por simplicidad, dejaremos la longitud de ambos strings ser igual y ser n.

- x=5678 y=1234
- a=56,b=78
- c=12,d=34

Paso 0 = m = n/2 + n%2

Paso 1 = a*c

Paso 2 = b*d

Paso 3 = (a + b)*(c + d)

Paso 4 = 3) - 2) - 1)

Paso 5 = 1)*pow(10, m*2) + 2) + 4)*pow(10, m)

Complejidad de tiempo

Mejor caso : O(1) **Peor caso :** O(1) **Promedio:** O(1)

JAVA

```
//Programa JAVA que realiza multiplicaciones
//Por medio del algoritmo de Karatsuba

import java.math.BigInteger;
import java.util.Scanner;

public class KaratsubaMultiply {

    public static void main(String[] args) {
        BigInteger x, y;
        Scanner sc = new Scanner(System.in);
        x = sc.nextBigInteger();
        y = sc.nextBigInteger();

        BigInteger result = karatsuba(x, y);
        long result2 = karatsuba(x.longValue(), y.longValue());
        System.out.println(result);
        System.out.println(result2);
    }

    private static long karatsuba(long x, long y) {
        if (x < 10 && y < 10) {
            return x * y;
        }
        int n = Math.max(Long.valueOf(x).toString().length(),
                (Long.valueOf(y).toString().length()));
        int m = n / 2 + n % 2;
        long a = x / (long) Math.pow(10, m);
        long b = x % (long) Math.pow(10, m);
        long c = y / (long) Math.pow(10, m);
        long d = y % (long) Math.pow(10, m);
        long step1 = karatsuba(a, c);
        long step2 = karatsuba(b, d);
        long step3 = karatsuba(a + b, c + d);
        long step4 = step3 - step2 - step1;
        long step5 = step1 * (long) Math.pow(10, m * 2) + step2 + step4 *
                (long) Math.pow(10, m);
        return step5;
    }

    private static BigInteger karatsuba(BigInteger x, BigInteger y) {
        if (x.compareTo(BigInteger.valueOf(10)) < 0 &&
                y.compareTo(BigInteger.valueOf(10)) < 0) {
            return x.multiply(y);
        }
        int n = Math.max(x.toString().length(), y.toString().length());
        int m = n / 2 + n % 2;
```

```
        BigInteger[] a_b = x.divideAndRemainder(BigInteger.valueOf(10).pow(m));
        BigInteger a = a_b[0];
        BigInteger b = a_b[1];
        BigInteger[] c_d = y.divideAndRemainder(BigInteger.valueOf(10).pow(m));
        BigInteger c = c_d[0];
        BigInteger d = c_d[1];
        BigInteger step1 = karatsuba(a, c);
        BigInteger step2 = karatsuba(b, d);
        BigInteger step3 = karatsuba(a.add(b), c.add(d));
        BigInteger step4 = step3.subtract(step2).subtract(step1);
        BigInteger step5 =
                step1.multiply(BigInteger.valueOf(10).pow(m * 2)).add(step2)
                .add(step4.multiply(BigInteger.valueOf(10).pow(m)));
        return step5;
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
using namespace std;
typedef long long int ll;
ll makeEqualLength(string &str1, string &str2) {
    ll len1 = str1.size();
    ll len2 = str2.size();
    if (len1 < len2) {
        for (ll i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1;
        return len2;
    }
    else if (len1 > len2) {
        for (ll i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1;
}
string addBitStrings( string first, string second ) {
    string result;
    ll length = makeEqualLength(first, second);
    ll carry = 0;
    for (ll i = length-1 ; i >= 0 ; i--) {
        ll firstBit = first.at(i) - '0';
        ll secondBit = second.at(i) - '0';
        ll sum = (firstBit ^ secondBit ^ carry)+'0';
        result = (char)sum + result;
        carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
    }
    if (carry){
        result = '1' + result;
    }
    return result;
}
ll multiplyiSingleBit(string a, string b) {
```

```
     return (a[0] - '0')*(b[0] - '0');
}
ll karatsuba(string X, string Y) {
    ll n = makeEqualLength(X, Y);
    if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);
    ll fh = n/2;
    ll sh = (n-fh);
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);
    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);
    ll P1 = karatsuba(Xl, Yl);
    ll P2 = karatsuba(Xr, Yr);
    ll P3 = karatsuba(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));
    return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}
string toBinary(ll n){
    string r;
    while(n!=0) {r=(n%2==0 ?"0":"1")+r; n/=2;}
    return r;
}
int main(){
    ll a;
    ll b;
    cin>>a;
    cin>>b;
    string x=toBinary(a);
    string y=toBinary(b);
    cout<<karatsuba(x,y)<<endl;
}
```

14.21) Problemas de repaso

Ejercicios en Online Judge

574-Sum It Up

10446-The Marriage Interview : -)

10176-Ocean Deep! Make it shallow!!

10664-Luggage

10469-To Carry or not to Carry

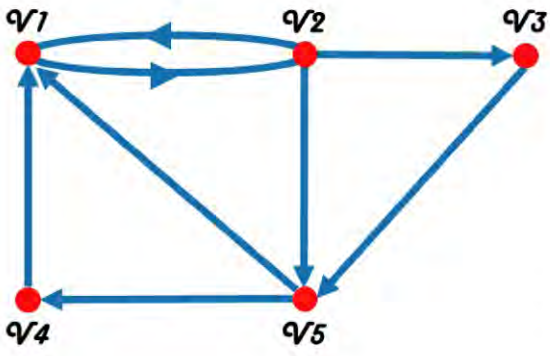
10911- Forming Quiz Teams

Capitulo 15. Grafos y arboles

Un grafo es un conjunto de nodos y aristas que permiten representar relaciones binarias, entre elementos de un conjunto, se representan abstractamente como un conjunto de puntos (Nodos o vértices) unidos por líneas (Aristas o caminos). Permiten estudiar las interrelaciones entre unidades que tienen interacción unas con otras, un ejemplo de esto son máquinas interactuando con otras por medio de redes. La historia de estos problemas inicia con el problema de los puentes de Konigsberg, en el cual se planteaba la pregunta ¿es posible dar un paseo comenzando desde cualquiera de las regiones, pasando por todos los puentes, recorriendo solo una vez cada uno y regresando al mismo punto de partida?.

15.1) Matriz de adyacencia

Matriz de adyacencia



	V1	V2	V3	V4	V5
V1	0	1	0	0	0
V2	1	0	1	0	1
V3	0	0	0	0	1
V4	1	0	0	0	0
V5	1	0	0	1	0

Guía del programador competitivo

Ilustración 15-1 Matriz de adyacencia de un grafo

La matriz de adyacencia es una matriz cuadrada que se utiliza como una forma de representar relaciones binarias, Se crea una matriz cero, cuyas columnas y filas representan los nodos del grafo.

1) Por cada arista que une a dos nodos, se suma 1 al valor que hay actualmente en la ubicación correspondiente de la matriz.
2) Si tal arista es un bucle y el grafo es no dirigido, entonces se suma 2 en vez de 1.
3) Finalmente, se obtiene una matriz que representa el número de aristas (relaciones) entre cada par de nodos (elementos).

Existe una matriz de adyacencia única para cada grafo (sin considerar las permutaciones de filas o columnas), y viceversa.

- Para un grafo no dirigido la matriz de adyacencia es simétrica.
- El número de caminos Ci,j(k), atravesando k aristas desde el nodo i al nodo j, viene dado por un elemento de la potencia k-ésima de la matriz de adyacencia: -
- Ci,j(k)=[A^K]ij

Existen otras formas de representar relaciones binarias, como por ejemplo los pares ordenados o los grafos.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Notacion java de una matriz de adyacencia

import java.util.Scanner;

public class AdyacencyMatrix {
    //matriz del grafo
    static int[][] G;
    static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        //Vertices y caminos
        int V = 4, E = 8;
        G = new int[V][V];
        for (int i = 0; i < V; i++) {
            //desde hasta peso
            int aux1, aux2, aux3;
            aux1 = sc.nextInt();
            aux2 = sc.nextInt();
            aux3 = sc.nextInt();
            // solo este si es dirigido
            G[aux1][aux2] = aux3;
```

```
            // inverso si es no dirigido
            G[aux2][aux1] = aux3;
        }
        print(V);
    }
    //Función que imprime la matriz de adyacencia
    static void print(int V) {
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < V; j++) {
                System.out.print(G[i][j] + " ");
            }
            System.out.println("");
        }
    }
}
```

C++

```
#include <bits/stdc++.h>
#define MAX_V 101
using namespace std;
static int G[MAX_V][MAX_V];

int main() {
    memset(G, 0, sizeof G);
    int V, E;
    cin >> V>>E;
    memset(G, NULL, MAX_V);
    for (int i = 0; i < E; ++i) {
        int desde, hasta, peso;
        cin >> desde>>hasta;
        cin>>peso;
        G[desde][hasta] = peso;
        G[hasta][desde] = peso;
    }
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            cout << G[i][j] << "\t";
        }
        cout << endl;
    }
    return 0;
}
```

PYTHON

```
from sys import stdin, stdout
rl = stdin.readline
wr = stdout.write

V, E = rl().strip().split()
V, E = int(V), int(E)

G = [[-1 for x in range(V)] for x in range(V)]
for i in range(V):
```

```
    G[i][i] = 0

for i in range(E):
    desde, hasta, peso = rl().strip().split()
    desde, hasta, peso = int(desde), int(hasta), int(peso)
    G[desde][hasta] = peso
    G[hasta][desde] = peso

for i in range(V):
    for j in range(V):
        if G[i][j] == -1:
            wr(f'inf\t')
        else:
            wr(f'{G[i][j]}\t')
    wr('\n')
```

15.2) Lista de adyacencia

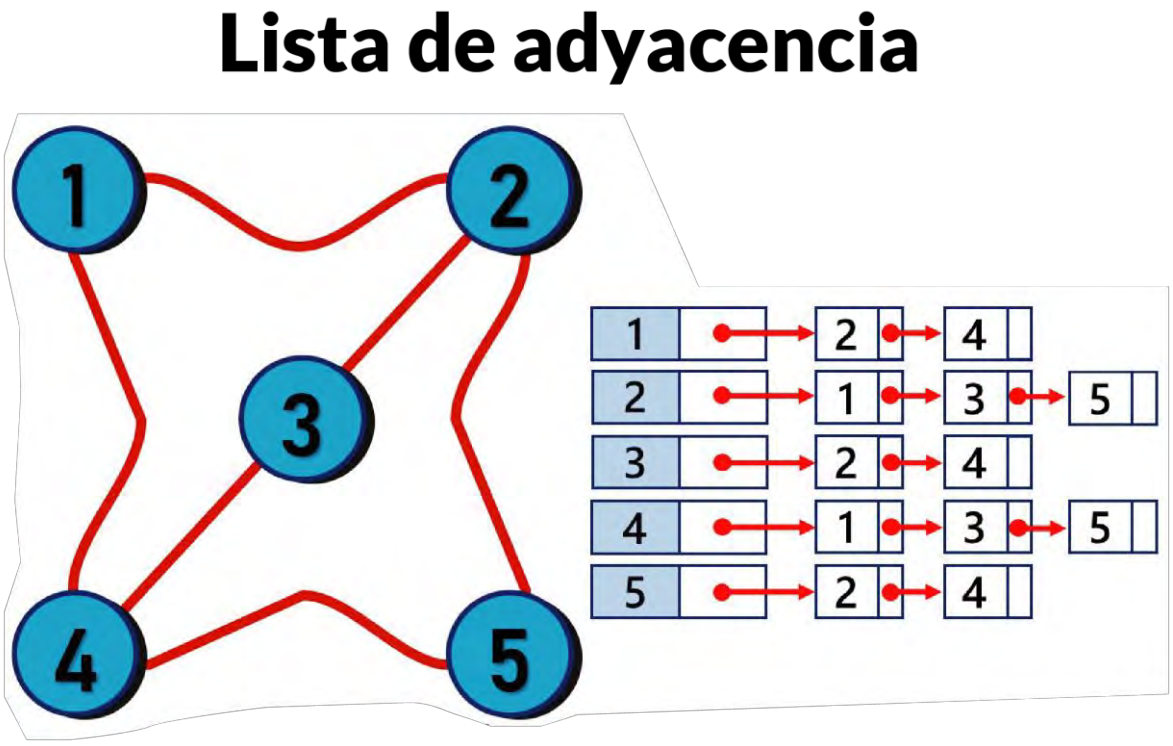


Ilustración 15-2 Lista de listas de adyacencia de un grafo

En teoría de grafos, una lista de adyacencia es una representación de todas las aristas o arcos de un grafo mediante una lista.

Si el grafo es no dirigido, cada entrada es un conjunto o multiconjunto de dos vértices conteniendo los dos extremos de la arista correspondiente. Si el grafo es dirigido, cada entrada es una tupla de dos nodos, uno denotando el nodo fuente y el otro denotando el nodo destino del arco correspondiente.

Típicamente, las listas de adyacentes no son ordenadas.

Complejidad de tiempo

Mejor caso : O(n) **Peor caso :** O(n) **Promedio:** O(n)

JAVA

```java
//Implementación java de una lista de adyacencia
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;


import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    static Vertex[] G;
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {
        int V = sc.nextInt(), E = sc.nextInt();
        G = new Vertex[V];
        for (int i = 0; i < V; i++) {
            G[i] = new Vertex();
            G[i].adj = new ArrayList<>();
        }
        for (int i = 0; i < E; i++) {
            int aux1, aux2, aux3;
            aux1 = sc.nextInt();
            aux2 = sc.nextInt();
            aux3 = sc.nextInt();
            //Dirigido
            G[aux1].adj.add(new Edge(aux2, aux3));
            //Descomentar para no dirigido
            //G[aux2].adj.add(new Edge(aux1, aux3));
        }
        for (int i = 0; i < G.length; i++) {
            System.out.print(i+"->" );
```

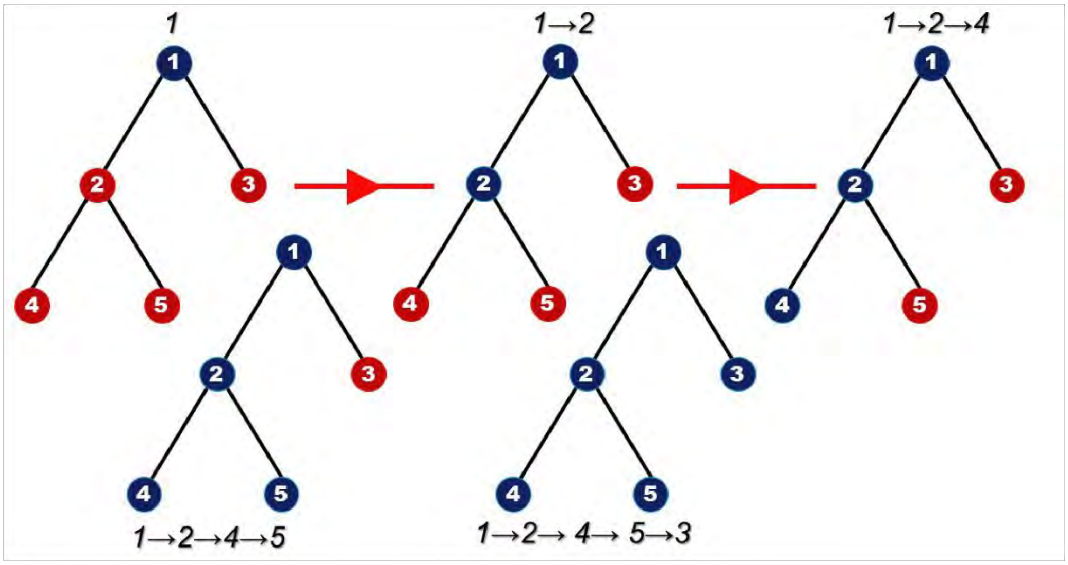
```
            for (Edge e : G[i].adj) {
                System.out.print(e.to + " " + e.w+"|");
            }
            System.out.println("");
        }
    }

    static class Vertex {
        List<Edge> adj;
        public Vertex() {
            adj = new ArrayList<>();
        }
    }

    static class Edge {
        int to, w;
        public Edge(int to, int w) {
            this.to = to;
            this.w = w;
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define MAX_V 101

using namespace std;

struct Edge {
    int to = 0, w = 0;

    Edge(int _to, int _w) {
        to = _to;
        w = _w;
    }
};

struct Vertex {
    list <Edge> adj;
};

static Vertex G[MAX_V];

int main(int argc, char *argv[]) {
    memset(G, 0, sizeof G);
    int V, E;
    scanf("%i %i", &V, &E);
    Vertex G [V];
    for (int i = 0; i < V; i++) {
        G[i] = Vertex();
        G[i].adj;
    }
```

```
    for (int i = 0; i < E; i++) {
        int desde, hasta, peso;
        scanf("%i %i %i", &desde, &hasta, &peso);
        G[desde].adj.push_back(Edge(hasta, peso));
        G[hasta].adj.push_back(Edge(desde, hasta));
    }
    int size = sizeof (G) / sizeof (G[0]);
    for (int i = 0; i < size; i++) {
        cout << i << "-> ";
        for (Edge e : G[i].adj) {
            cout << e.to << " " << e.w << "|";
        }
        cout << "" << endl;
    }
    return 0;
}
```

15.3) DFS (Depth First Search)



Guia del programador competitivo

Ilustración 15-3 Ejemplo de búsqueda en profundidad

Una Búsqueda en profundidad (en inglés DFS o Depth First Search) es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol (teoría

de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Completitud: DFS es completo si y solo si usamos búsqueda basada en grafos en espacios de estado finitos, pues todos los nodos serán expandidos.

Optimalidad: DFS en ningún caso asegura la optimalidad, pues puede encontrar una solución más profunda que otra en una rama que todavía no ha sido expandida.

Complejidad temporal: en el peor caso, O(b^m) siendo b el factor de ramificación (número promedio de ramificaciones por nodo) y m la máxima profundidad del espacio de estados.

Complejidad espacial: O(b^d) siendo b el factor de ramificación y d la profundidad de la solución menos costosa, pues cada nodo generado permanece en memoria, almacenándose la mayor cantidad de nodos en el nivel meta.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```java
// Programa java que imprime DFS transverso en un grafo

import java.util.*;

public class DFS {

    public static void main(String args[]) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println("Siguiendo la primera busqueda en profundidad "
                + "(iniciando desde vertice  2)");
        g.DFS(2);
    }

    static class Graph {
```

```
        private int V;  // número de vertices
        // Array de listas de adyacencia
        private LinkedList<Integer> adj[];
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
            }
        }
        //Agregando caminos
        void addEdge(int v, int w) {
            adj[v].add(w);
        }
        void DFSUtil(int v, boolean visited[]) {
            // Marca el nodo actual como visitado y lo imprime
            visited[v] = true;
            System.out.print(v + " ");
            // Recorre todos los vertices adyacentes a este vertice
            Iterator<Integer> i = adj[v].listIterator();
            while (i.hasNext()) {
                int n = i.next();
                if (!visited[n]) {
                    DFSUtil(n, visited);
                }
            }
        }
        void DFS(int v) {
            //Marca todos los vertices como no visitados (Falso)
            boolean visited[] = new boolean[V];
            DFSUtil(v, visited);
        }

    }
}
```

PYTHON

```
class Grafo:
    V = 0
    adj = [[], [], [], [], [], []]
    def addEdge(self, v, w):
        self.adj[v].append(w)
    def DFSUtil(self, v, visited=[]):
        visited[v] = True
        print(v, " ")
        for i in range(len(self.adj[v])):
            n = self.adj[v][i]
            if not visited[n]:
                self.DFSUtil(n, visited)
    def DFS(self, v):
        visited = [False, False, False, False, False, False]
        self.DFSUtil(v, visited)
```

```
g = Grafo()
g.V = 4
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
inicial = 0
print("Nodo inicial" + str(inicial))
g.DFS(inicial)
```

15.4) BFS (Breath First Search)

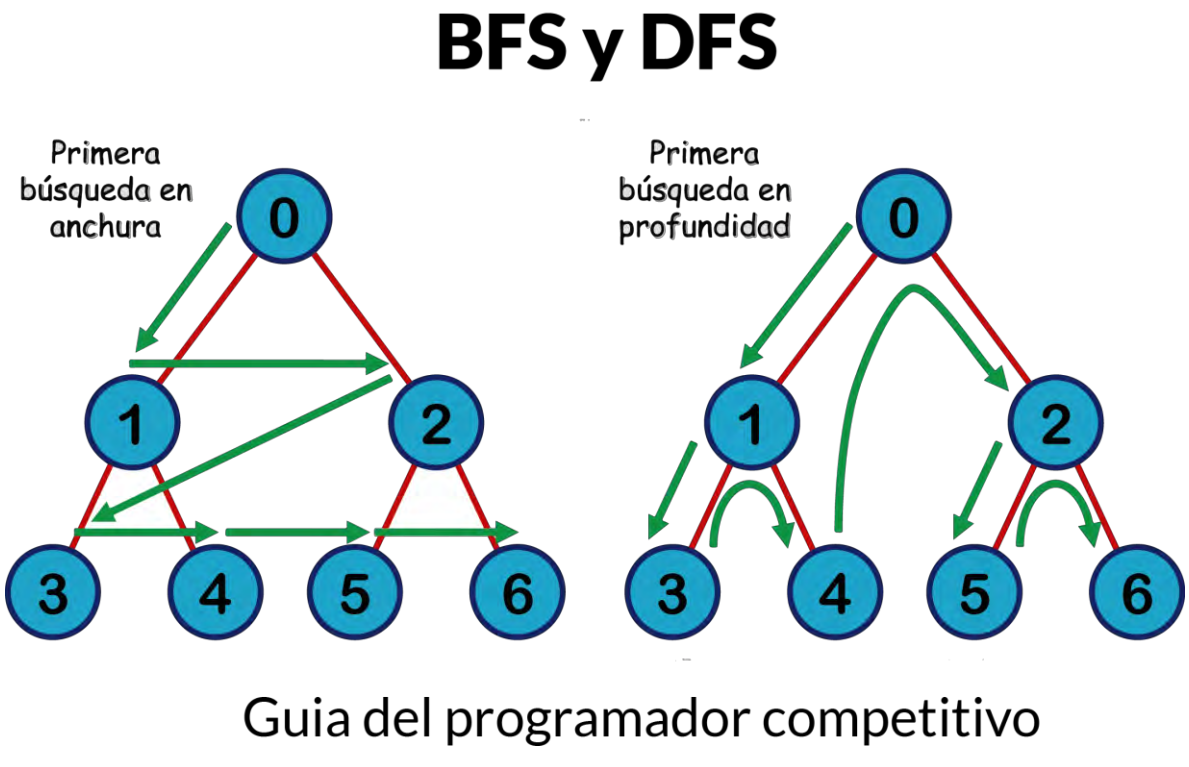


Ilustración 15-4 Búsqueda en profundidad y en anchura

Búsqueda en anchura (en inglés BFS - Breadth First Search) es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo (usado frecuentemente

sobre árboles). Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

Formalmente, BFS es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución. El algoritmo no usa ninguna estrategia heurística.

- Dado un vértice fuente s, Breadth-first search sistemáticamente explora los vértices de G para “descubrir” todos los vértices alcanzables desde s.
- Calcula la distancia (menor número de vértices) desde s a todos los vértices alcanzables.
- Después produce un árbol BF con raíz en s y que contiene a todos los vértices alcanzables.
- El camino desde dt a cada vértice en este recorrido contiene el mínimo número de vértices. Es el camino más corto medido en número de vértices.
- Su nombre se debe a que expande uniformemente la frontera entre lo descubierto y lo no descubierto. Llega a los nodos de distancia k, sólo tras haber llegado a todos los nodos a distancia k-1.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

BFS y DFS

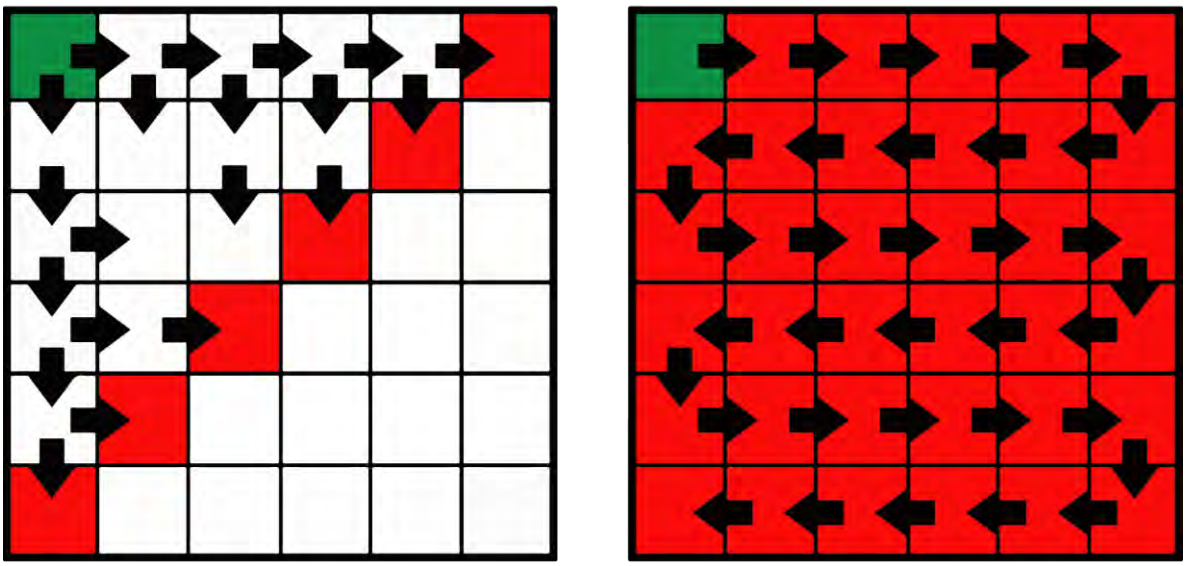


Ilustración 15-5 BFS Y DFS en una matriz

La complejidad computacional del algoritmo se puede expresar como O(|V|+|E|) O(|V|+|E|), donde |V| |V| es el número de vértices y |E| es el número de aristas. El razonamiento es porque en el peor caso, cada vértice y cada arista serán visitados por el algoritmo.

JAVA

```
// Programa java que imprime BFS (Busqueda en anchura) transverso
// desde un vertice (nodo) dado como inicio
// BFS(int s) atraviesa vertices alcanzables desde s

import java.util.*;
//Esta clase representa un grafo dirigido usando listas de adyacencia

public class BFS {

    public static void main(String args[]) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
```

```
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println("Siguiendo su primera busqueda en anchura transverso"
                + "(iniciando desde el vertice 2)");
        //Vertice de inicio
        g.BFS(2);
    }

    static class Graph {
        private final int V;   // número de vertices
        private LinkedList<Integer> adj[]; //lista de ayacencia
        // Constructor
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
            }
        }
        // funcion que agrega un camino al grafo
        void addEdge(int v, int w) {
            adj[v].add(w);
        }
        void BFS(int s) {
            //Marka todos los verticoes como no visitados (Falso)
            boolean visited[] = new boolean[V];
            //Crea una cola para la BFS
            // Create a queue for BFS
            LinkedList<Integer> queue = new LinkedList<>();
            //Marca el nodo actual como vistiado y lo encola
            visited[s] = true;
            queue.add(s);
            while (!queue.isEmpty()) {
                //Desencola un vertice de la cola y lo imprime
                s = queue.poll();
                System.out.print(s + " ");
                /* Obtiene todos los vertices adyacentes del
                vertice desencolado, si un adyacente no ha
                sido visitado, lo marca lo visita y lo encola*/
                Iterator<Integer> i = adj[s].listIterator();
                while (i.hasNext()) {
                    int n = i.next();
                    if (!visited[n]) {
                        visited[n] = true;
                        queue.add(n);
                    }
                }
            }
        }
    }
}
```

C++

```
#include <bits/stdc++.h>
```

```cpp
#define MAX_V 101
using namespace std;
vector<int> adj[MAX_V];
vector<int>que;

struct Graph {
    int V;

    void addEdge(int v, int w) {
        adj[v].push_back(w);
    }

    void BFS(int s) {
        bool visited[V];
        visited[s] = true;
        que.push_back(s);
        while (!que.empty()) {
            s = que.front();
            que.erase(que.begin());
            cout << s << " ";
            for (int i = 0; i < adj[s].size(); ++i) {
                int n = adj[s][i];
                if (!visited[n]) {
                    visited[n] = true;
                    que.push_back(n);
                }
            }
        }
    }
};

int main() {
    Graph gra;
    gra.V = 4;
    gra.addEdge(0, 1);
    gra.addEdge(0, 2);
    gra.addEdge(1, 2);
    gra.addEdge(2, 0);
    gra.addEdge(2, 3);
    gra.addEdge(3, 3);
    int initial = 0;
    gra.BFS(initial);
    return 0;
}
```

PYTHON

```python
def add_edge(v, w):
    ady[v].append(w)
def bfs(s):
    visited = [[] for i in range (V)]
    queue = []
    visited[s] = True;
    queue.append(s)
    while len(queue) > 0:
```

```
        s = queue[0]
        queue = queue[1:]
        print(str(s) + " ")
        aux = ady[s]
        for j in range (len(aux)):
            n = aux[j]
            if(visited[n] != True):
                visited[n] = True
                queue.append(n)
V = 4
ady = [[] for i in range (V)]
add_edge(0, 1)
add_edge(0, 2)
add_edge(1, 2)
add_edge(2, 0)
add_edge(2, 3)
add_edge(3, 3)
print(ady)
bfs(0)
```

15.5) BFS todos los caminos

Utilizando el algoritmo de búsqueda en anchura, y modificándolo se puede realizar la búsqueda de todos los caminos existentes desde un origen hasta un destino, permitiendo también que se impriman los recorridos de estos caminos.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
//Programa java que imprime todos los caminos
// desde un inicio hasta un destino usando BFS

import java.util.ArrayList;
import java.util.List;
//Un grafo dirigido usando lista de adyacencia

public class BFSAllPaths {
    // Número de vertices
    static int v;
    // lista de adyacencia
    static ArrayList<Integer>[] adjList;
```

```
static void initGraph(int vertices) {
    //Inicializa número de vertices
    v = vertices;
    // inicializa lista de adyacencia
    initAdjList();
}

static void initAdjList() {
    adjList = new ArrayList[v];
    for (int i = 0; i < v; i++) {
        adjList[i] = new ArrayList<>();
    }
}
//Agrega camino de u a v
static void addEdge(int u, int v) {
    // Agrega v a la lista de u
    adjList[u].add(v);
}
//Imrpime todos los caminos de
// 's' a 'd'
static void printAllPaths(int s, int d) {
    boolean[] isVisited = new boolean[v];
    ArrayList<Integer> pathList = new ArrayList<>();
    //Agrega origen al path[]
    pathList.add(s);
    printAllPathsUtil(s, d, isVisited, pathList);
}
/*Función recursiva que imprime
Todos los caminos de u a d.
isVisited[] mantiene rastro de
los vertices en el camino actual
localPathList<> almacena vertices actuales
en el camino actual*/
static void printAllPathsUtil(Integer u, Integer d,
        boolean[] isVisited,
        List<Integer> localPathList) {
    // Marca el nodo actual
    isVisited[u] = true;
    if (u.equals(d)) {
        System.out.println(localPathList);
        // Si coincidencia encontrada entonces
        // no se necesita atravesar más profundo
        isVisited[u] = false;
        return;
    }
    // Recorre todos los vertices
    //adyacentes al actual vertice
    for (Integer i : adjList[u]) {
        if (!isVisited[i]) {
            // Almacena el nodo actual
            // en path[]
            localPathList.add(i);
            printAllPathsUtil(i, d, isVisited, localPathList);
            localPathList.remove(i);
```

```
            }
        }
        // Marca el nodo actual
        isVisited[u] = false;
    }

    public static void main(String[] args) {
        //Crea el grafo
        initGraph(4);
        //camino desde hasta
        addEdge(0, 1);
        addEdge(0, 2);
        addEdge(0, 3);
        addEdge(2, 0);
        addEdge(2, 1);
        addEdge(1, 3);
        // inicio arbitrario
        int s = 0;
        // destino arbitrario
        int d = 3;
        System.out.println("Los siguientes son todos los diferentes"
                + "caminos de " + s + " a " + d);
        printAllPaths(s, d);
    }
}
```

C++

```
#include <bits/stdc++.h>
using namespace std;

// función de utilidad para impresion
// encontrando todos los caminos
void printpath(vector<int>& path) {
    int size = path.size();
    for (int i = 0; i < size; i++)
        cout << path[i] << " ";
    cout << endl;
}
// Funcion para revisar si el vertice actual
// ya se encuentra presente en el camino
int isNotVisited(int x, vector<int>& path) {
    int size = path.size();
    for (int i = 0; i < size; i++)
        if (path[i] == x)
            return 0;
    return 1;
}
// Funcion de utilidad para encontrar los caminos dentro de un grafo
// desde el inicio hasta el fin dado
void findpaths(vector<vector<int>> &g, int inicio, int destino, int v) {
    // Crear una cola que almacena los caminos
    queue<vector<int>> q;
    // vector de caminos que almacena el camino actual
    vector<int> path;
```

```
    path.push_back(inicio);
    q.push(path);
    while (!q.empty()) {
        path = q.front();
        q.pop();
        int last = path[path.size() - 1];
        // Si el ultimo vertice es el destino deseado
        // entonces se imprime el camino
        if (last == destino)
            printpath(path);
        // Atravesar a todos los nodos conectados al vÃ©rtice actual
        // y empujar una nueva ruta a la cola
        for (int i = 0; i < g[last].size(); i++) {
            if (isNotVisited(g[last][i], path)) {
                vector<int> newpath(path);
                newpath.push_back(g[last][i]);
                q.push(newpath);
            }
        }
    }
}

int main() {
    vector<vector<int>> g;
    // Numero de vertices
    int v = 4;
    g.resize(4);
    // Contruccion del grafo
    g[0].push_back(3);
    g[0].push_back(1);
    g[0].push_back(2);
    g[1].push_back(3);
    g[2].push_back(0);
    g[2].push_back(1);
    //inicio y destino
    int inicio = 2, destino = 3;
    cout << "Los caminos desde " << inicio
            << " hasta " << destino << " son \n";
    // Llamada a la funcion que permitirá¡ encontrar los caminos
    // recibiendo como parametros la matriz de caminos, el inicio, el
    // y la cantidad de vertices
    findpaths(g, inicio, destino, v);
    return 0;
}
```

PYTHON

```
from collections import defaultdict

class Graph:
    def __init__(self, vertices):
        self.V = vertices
```

```
        self.graph = defaultdict(list)
    def addEdge(self, u, v):
        self.graph[u].append(v)
    def printAllPathsUtil(self, u, d, visited, path):
        visited[u] = True
        path.append(u)
        if u == d:
            print path
        else:
            for i in self.graph[u]:
                if visited[i] == False:
                    self.printAllPathsUtil(i, d, visited, path)
        path.pop()
        visited[u] = False

    def printAllPaths(self, s, d):
        visited = [False] * (self.V)
        path = []
        self.printAllPathsUtil(s, d, visited, path)
g = Graph(4)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(0, 3)
g.addEdge(2, 0)
g.addEdge(2, 1)
g.addEdge(1, 3)
s = 2 ; d = 3
print ("Siguiendo los diferentes caminos desde %d a %d :" %(s, d))
g.printAllPaths(s, d)
```

15.6) Domino DFS

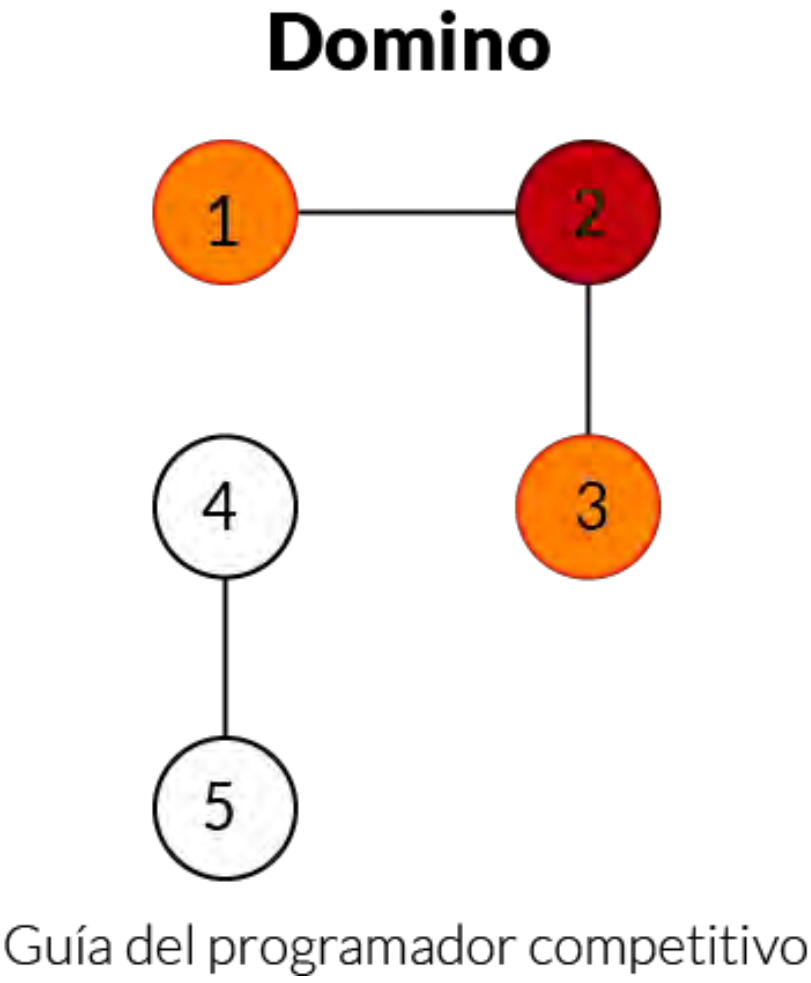


Ilustración 15-6 Fichas de domino vistas como un grafo

El efecto dominó o reacción en cadena es el efecto acumulativo producido cuando un acontecimiento origina una cadena de otros acontecimientos similares.

Se produce cuando un pequeño cambio origina un cambio similar a su lado, que a su vez causa otro similar, y así sucesivamente en una secuencia lineal. Recibe este nombre, por analogía con la caída de una hilera de fichas de dominó colocadas en posición vertical. El efecto dominó también puede hacer referencia a una cadena de acontecimientos no materiales.

El término, en sus distintos usos, se ha hecho popular por su analogía al efecto mecánico, una fila de fichas de dominó al caer una ficha detrás de otra, aunque típicamente se refiere

a una secuencia enlazada de acontecimientos donde el tiempo entre acontecimientos sucesivos es relativamente pequeño.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```java
//Programa java que usando DFS busca cuantos dominos
// caen desde un origen

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

/* EJEMPLO DE INGRESO
9 6 3
1 2
2 5
5 3
4 3
6 7
7 8
1
6
4*/
public class DominoDFS {
    static final int MAX = 10001;
    static ArrayList<ArrayList<Integer>> ady
            = new ArrayList<ArrayList<Integer>>(MAX);
    static Scanner sc = new Scanner(System.in);
    //la cantidad total de dominos que caerán
    static int total;
    //arreglo de domino caido
    static boolean visitado[] = new boolean[MAX];

    public static void main(String[] args) {
        //número de dominos, cantidad de enlaces, consultas
        int V, M, Q, x, y, origen;
        V = sc.nextInt();
        M = sc.nextInt();
        Q = sc.nextInt();
        for (int i = 0; i < V; i++) {
            ady.add(new ArrayList<>());
        }
        while (M > 0) {
            //domino x hace caer a domino y
            x = sc.nextInt();
            y = sc.nextInt();
            ady.get(x).add(y);
            M--;
```

```
        }
        while (Q > 0) {
            //domino origen
            origen = sc.nextInt();
            total = 0;
            Arrays.fill(visitado, false);
            dfs(origen);
            System.out.printf("%d\n", total);
            Q--;
        }

    }

    static void dfs(int u) { //domino origen
        //aumento en mi respuesta la caida de un domino
        total++;
        //domino "u" cayo
        visitado[u] = true;
        //verifico los demás posibles
        //domino que caeran si impulso "u"
        for (int v = 0; v < ady.get(u).size(); ++v) {
            //si el domino adyacente no
            //cayó entonces es elsiguiente a evaluar
            if (!visitado[ady.get(u).get(v)]) {
                //recursivamente veo que dominos
                //caeran a partir del adyacente de "u"
                dfs(ady.get(u).get(v));
            }
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#define MAX 10001
using namespace std;
vector <vector<int> >ady(MAX);
int total = 0;
bool visited[MAX];

void DFS(int u) {
    total++;
    visited[u] = true;
    for (int v = 0; v < ady[u].size(); v++) {
        if (!visited[ady[u][v]]) {
            DFS(ady[u][v]);
        }
    }
}

int main() {
    int dominos, caminos, busquedas, desde, hasta, origen;
    cin>>dominos;
    cin>>caminos;
```

```
    cin>>busquedas;
    while (caminos > 0) {
        cin >> desde>>hasta;
        ady[desde].push_back(hasta);
        caminos--;
    }
    while (busquedas) {
        cin>>origen;
        total = 0;
        memset(visited, 0, sizeof visited);
        DFS(origen);
        cout << "Dominos tumbados : " << total << endl;
        busquedas--;
    }
    return 0;
}
```

15.7) Laberintos BFS

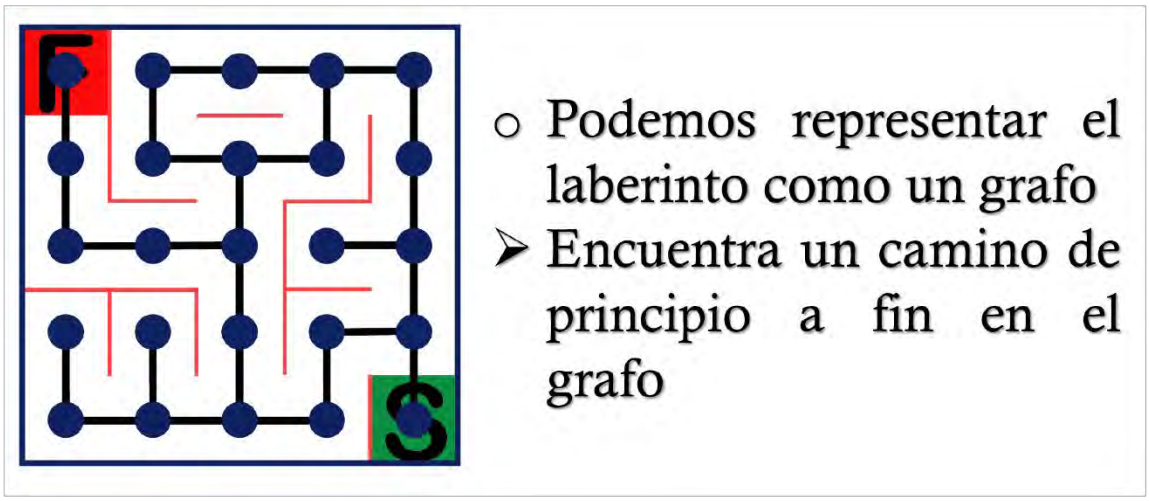


Ilustración 15-7 Salir del laberinto

Un laberinto es un pasatiempo gráfico consistente en trazar una línea desde un punto de origen situado en el exterior de un laberinto a uno de destino situado generalmente en el

centro o bien en el lado opuesto. La dificultad consiste en encontrar un camino directo hasta el lugar deseado. El laberinto, por su propia configuración, contiene diferentes vías sin salida (de mayor o menor longitud) y solo un recorrido correcto.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```java
// Programa java que realiza laberintos con BFS

import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class ExitTheMazeBFS {

    /* Ejemplo de ingreso
8 8
.......I
.#######
.#......
.#.S...S
.###.#.#
.#...#.#
.#.###.#
........
     */
    //máximo número de filas y columnas del laberinto
    static final int MAX = 100;
    static Scanner sc = new Scanner(System.in);
    //laberinto
    static char ady[][] = new char[MAX][MAX];
    //arreglo de estados visitados
    static boolean visitado[][] = new boolean[MAX][MAX];
    //incremento en coordenada x
    static int dx[] = {0, 0, 1, -1};
    //incremento en coordanada y
    static int dy[] = {1, -1, 0, 0};
    //altura y ancho del laberinto
    static int h, w;
    //Arreglo para mostrar la ruta que se siguio
    static Estado prev[][] = new Estado[MAX][MAX];

    static class Estado {
        int x; // Fila del estado
        int y; // Columna del estado
        int d; // Distancia del estado
        // Constructor
```

```
    Estado(int x1, int y1, int d1) {
        this.x = x1;
        this.y = y1;
        this.d = d1;
    }

    Estado() {
    }
}

public static void main(String[] args) {
    int x = 0, y = 0;
    System.out.println("Ingrese altura del laberinto: ");
    h = sc.nextInt();
    System.out.println("Ingrese ancho del laberinto: ");
    w = sc.nextInt();
    sc.nextLine();
    System.out.printf("\nIngrese el laberinto, con un solo "
            + "valor inicial I, valor final sera S: \n");
    for (int i = 0; i < h; ++i) {
        String aux = sc.nextLine();
        for (int j = 0; j < w; ++j) {
            ady[i][j] = aux.charAt(j);
            //obtengo coordenada de valor inicial
            if (ady[i][j] == 'I') {
                x = i;
                y = j;
            }
        }
    }
    //MOSTRAMOS LABERINTO
    for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; ++j) {
            System.out.printf("%c", ady[i][j]);
        }
        System.out.printf("\n");
    }
    int min = BFS(x, y, h, w);
    if (min != -1) {
        System.out.printf("Menor número de pasos: %d\n", min);
    } else {
        System.out.println("No se pudo llegar al destino");
    }

}
//Funcion para imprimir la ruta encontrada
//Ingresan las coordenadas del nodo final
static void print(int x, int y) {
    //El arreglo prev posee las coordenadas del nodo
    //anterior, por ello empezamos desde el final
    //El proceso termina al momento de preguntar
    //por el anterior del nodo inicial, como pusimos -1
    //Preguntamos hasta que nuestro anterior sea diferente de -1
    for (int i = x, j = y; prev[i][j].d != -1; i
```

```
            = prev[x][y].x, j = prev[x][y].y) {
        ady[i][j] = '*';
        x = i;
        y = j;
    }

    System.out.printf("Camino con menor número de pasos\n");
    for (int i = 0; i < h; ++i) {
        for (int j = 0; j < w; ++j) {
            System.out.printf("%c", ady[i][j]);
        }
        System.out.printf("\n");
    }
}
//coordenadas de inicial "I" y dimensiones de laberinto
static int BFS(int x, int y, int h, int w) {
    //Estado inicial, distancia = 0
    Estado inicial = new Estado(x, y, 0);
    //Cola de todos los posibles Estados por
    //los que se pase para llegar al destino
    Queue<Estado> Q = new LinkedList<>();
    //Insertamos el estado inicial en la Cola.
    //marcamos como no visitado
    Q.offer(inicial);
    for (int i = 0; i < MAX; i++) {
        Arrays.fill(visitado[i], false);
    }
    //el inicial no tiene una ruta anterior puesto que es  primero
    prev[x][y] = new Estado(-1, -1, -1);
    //Mientras cola no este vacia
    while (!Q.isEmpty()) {
        //Obtengo de la cola el estado actual,
        //en un comienzo será el inicial
        Estado actual = Q.peek();
        //Saco el elemento de la cola
        Q.poll();
        //Si se llego al destino (punto final)
        if (ady[actual.x][actual.y] == 'S') {
            //imprimo la ruta del camino más corto
            print(actual.x, actual.y);
            //Retornamos distancia recorrida hasta ese momento
            return actual.d;
        }
        //Marco como visitado dicho estado para no volver a recorrerlo
        visitado[actual.x][actual.y] = true;
        //Recorremos hasta 4 porque tenemos 4 posibles adyacentes
        for (int i = 0; i < 4; ++i) {
            //nx y ny tendran la coordenada adyacente
            int nx = dx[i] + actual.x;
            //ejemplo en i=0 y actual
            //(3,4) -> 3+dx[0]=3+0=3,
            //4+dy[0]=4+1=5, nueva coordenada (3,5)
            int ny = dy[i] + actual.y;
            //aqui comprobamos que la coordenada
```

```
                //adyacente no sobrepase las dimensiones del laberinto
                //además comprobamos que no sea
                //pared "#" y no este visitado
                if (nx >= 0 && nx < h && ny >= 0
                        && ny < w && ady[nx][ny] != '#' && !visitado[nx][ny]) {
                    //Creamos estado adyacente aumento en
                    //1 la distancia recorrida
                    Estado adyacente = new Estado(nx, ny, actual.d + 1);
                    //Agregamos adyacente a la cola
                    Q.offer(adyacente);
                    //El previo del nuevo nodo es el actual.
                    prev[nx][ny] = actual;
                }
            }
        }
        return -1;
    }
}
```

C++

```
#include<bits/stdc++.h>
#define MAX 100
using namespace std;

struct Estado {
    int x;
    int y;
    int d;
};
char ady[MAX][MAX];
bool visited[MAX][MAX];
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
int h, w;
Estado prev[MAX][MAX];

void print(int x, int y) {
    for (int i = x, j = y; prev[i][j].d != -1; i = prev[x][y].x, j =
prev[x][y].y) {
        ady[i][j] = '*';
        x = i;
        y = j;
    }
    cout << "camino con menor pasos" << endl;
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            cout << ady[i][j];
        }
        cout << endl;
    }
}

int BFS(int x, int y, int h, int w) {
    Estado inicial;
```

```
    inicial.x = x;
    inicial.y = y;
    inicial.d = 0;
    queue<Estado> Q;
    Q.push(inicial);
    for (int i = 0; i < MAX; ++i) {
        memset(visited[i], false, sizeof visited[i]);
    }
    Estado nuevo;
    nuevo.x = -1;
    nuevo.y = -1;
    nuevo.d = -1;
    prev[x][y] = nuevo;
    while (!Q.empty()) {
        Estado actual = Q.front();
        Q.pop();
        if (ady[actual.x][actual.y] == 'S') {
            print(actual.x, actual.y);
            return actual.d;
        }
        visited[actual.x][actual.y] = true;
        for (int i = 0; i < 4; i++) {
            int nx = dx[i] + actual.x;
            int ny = dy[i] + actual.y;
            if (nx >= 0 && nx < h && ny >= 0 && ny < w && ady[nx][ny] != '#' &&
!visited[nx][ny]) {
                Estado adyacente;
                adyacente.x = nx;
                adyacente.y = ny;
                adyacente.d = actual.d + 1;
                Q.push(adyacente);
                prev[nx][ny] = actual;
            }
        }
    }
    return -1;
}

int main() {
    int x = 0;
    int y = 0;
    cout << "altura\n";
    cin>>h;
    cout << "ancho\n";
    cin>>w;
    cin.ignore();
    for (int i = 0; i < h; i++) {
        string aux;
        getline(cin, aux);
        for (int j = 0; j < w; j++) {
            ady[i][j] = aux[j];
            if (ady[i][j] == 'I') {
                x = i;
                y = j;
```

```
            }
        }
    }
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            cout << ady[i][j];
        }
        cout << endl;
    }
    int mini = BFS(x, y, h, w);
    if (mini != -1) {
        cout << "El menor numero de pasos es " << mini;
    } else {
        cout << "No se pudo llegar";
    }
    return 0;
}
```

15.8) Conteo de caminos DFS

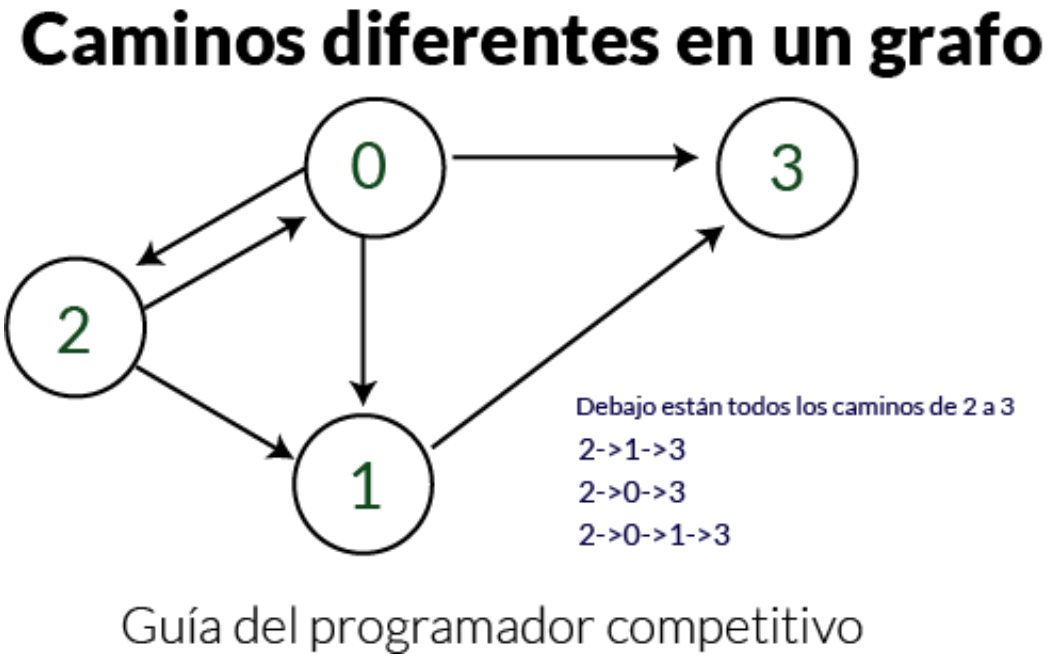


Ilustración 15-8 Todos los caminos posibles entre 2 y 3

Cuente el número total de caminos o vías que existen entre dos vértices en un grafo dirigido, estos caminos no contienen un ciclo, la simple razón de esto es que un ciclo contiene infinito número de caminos y esto crea problema.

El problema puede ser resuelto usando backtracking, esto es si tomamos un camino y empezamos a andar por él, si nos lleva al vértice de destino entonces contamos el camino y nos devolvemos a tomar otro camino, si el camino no nos lleva al vértice destino, descartamos este camino.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que cuenta todos los caminos de un
// inicio a un destino.

import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;

public class CountAllPaths {

    public static void main(String args[]) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(0, 3);
        g.addEdge(2, 0);
        g.addEdge(2, 1);
        g.addEdge(1, 3);
         //Origen - destino
        int s = 2, d = 3;
        System.out.println(g.countPaths(s, d));
    }

    static class Graph {
        // Número de vertices
        private int V;
        // array de listas de adyacencia
        private LinkedList<Integer> adj[];
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList<>();
            }
```

```
        }
        //agrgar caminos en un grafo
        void addEdge(int v, int w) {
            //agrega w a las listas de v
            adj[v].add(w);
        }
        // Un metodo recursivo que cuenta
        // todos los caminos de u a d
        int countPathsUtil(int u, int d,
                boolean visited[],
                int pathCount) {
            //Marca el nodo actual como visitado
            //y lo imprime
            visited[u] = true;
            // Si el vertice actual es igual
            // al destino, incrementa el conteo
            if (u == d) {
                pathCount++;
            } // Recore todos los vertices
            // adyacentes a este vertice
            else {
                Iterator<Integer> i = adj[u].listIterator();
                while (i.hasNext()) {
                    int n = i.next();
                    if (!visited[n]) {
                        pathCount = countPathsUtil(n, d,
                                visited,
                                pathCount);
                    }
                }
            }
            visited[u] = false;
            return pathCount;
        }
        // Retorna conteo de caminos desde s a d
        int countPaths(int s, int d) {
            //Marca todos los vertices como no visitados
            boolean visited[] = new boolean[V];
            Arrays.fill(visited, false);
            int pathCount = 0;
            pathCount = countPathsUtil(s, d,
                    visited,
                    pathCount);
            return pathCount;
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#define MAX 101
using namespace std;
vector<int> adj[MAX];
```

```
struct Graph {
    int V;

    void addEdge(int v, int w) {
        adj[v].push_back(w);
    }

    int countPathsUtil(int u, int d, bool visited[], int pathCount) {
        visited[u] = true;
        if (u == d) {
            pathCount++;
        } else {
            for (int i = 0; i < adj[u].size(); i++) {
                int n = adj[u][i];
                if (!visited[n]) {
                    pathCount = countPathsUtil(n, d, visited, pathCount);
                }
            }
        }
        visited[u] = false;
        return pathCount;
    }

    int countPaths(int s, int d) {
        bool visited[V];
        memset(visited, false, sizeof visited);
        int pathcount = 0;
        pathcount = countPathsUtil(s, d, visited, pathcount);
        return pathcount;
    }
};

int main() {
    Graph g;
    g.V = 4;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(2, 0);
    g.addEdge(2, 1);
    g.addEdge(1, 3);
    int s = 2, d = 3;
    cout << g.countPaths(s, d);
    return 0;
}
```

15.9) Ciclo en un grafo dirigido

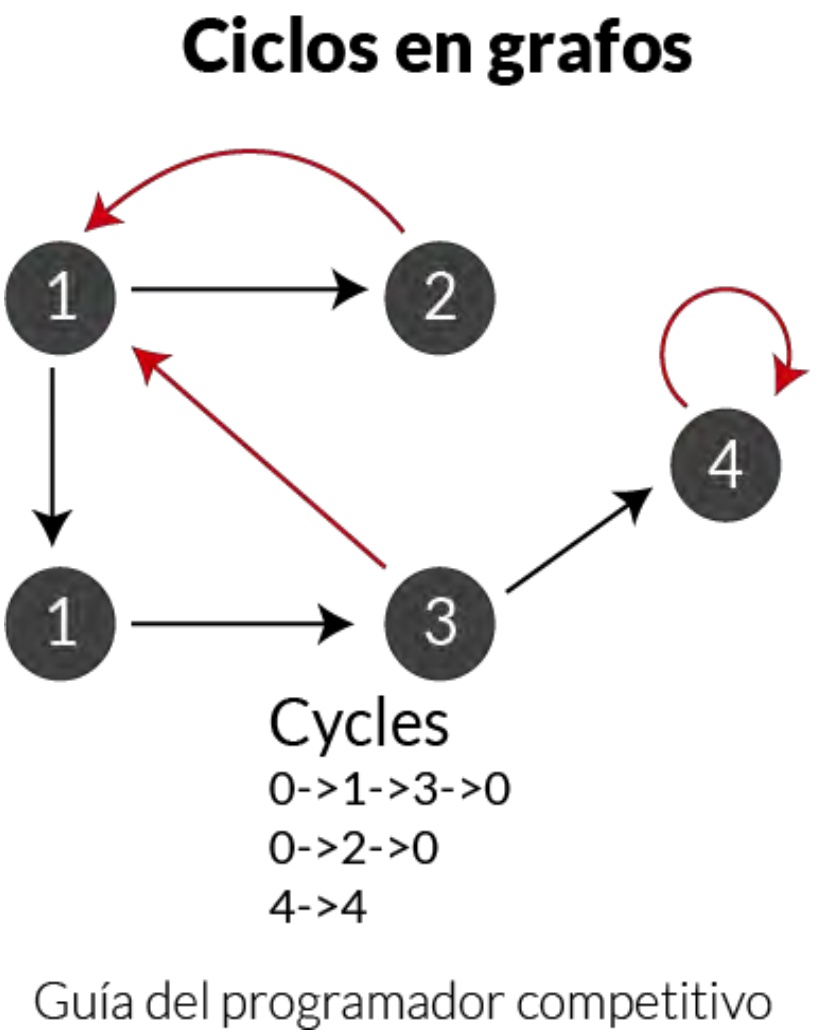


Ilustración 15-9 Ejemplos de ciclos (Cycles) dentro de un grafo dirigido

Dado un grafo dirigido, verificar si el grafo contiene un ciclo o no, la función debe retornar true si el grafo dado contiene al menos un ciclo, de lo contrario retorne false.

DFS puede ser usado para detectar un ciclo en un grafo, DFS para un grafo conectado produce un árbol, hay un ciclo en un grafo solo si hay un camino de regreso presente en el grafo. Un camino de regreso es un camino de un nodo a sí mismo, o uno de sus antecesores en el árbol producido por el DFS.

Para un grafo desconexo, tenemos el bosque DFS como salida, para detectar un ciclo, podemos verificar los arboles individuales en búsqueda de caminos de regreso.

Para detectar un camino de regreso, podemos rastrear los vértices actuales en una pila de recursión de la función de DFS transverso, si llegamos a un vértice que ya está en la pila de recursión entonces hay un ciclo en el árbol, el camino que conecta el vértice actual a un vértice en la pila recursión es el camino de regreso, Usamos recStack[] para mantener rastreado los vértices de la pila de recursión.

La complejidad de tiempo de este método es la misma complejidad de tiempo de un DFS transverso la cual es O(V+E) siendo V la cantidad de vértices y E la cantidad de caminos.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que detecta ciclo en un grafo

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class CycleInADirectedGraph {

    static class Graph {
        private final int V;
        private final List<List<Integer>> adj;
        public Graph(int V) {
            this.V = V;
            adj = new ArrayList<>(V);
            for (int i = 0; i < V; i++) {
                adj.add(new LinkedList<>());
            }
        }
        private boolean isCyclicUtil(int i, boolean[] visited,
                boolean[] recStack) {
            /* Marca el nodo actual como visitado
            y parte de la pila de recursión*/
            if (recStack[i]) {
                return true;
            }
            if (visited[i]) {
                return false;
            }
            visited[i] = true;
            recStack[i] = true;
            List<Integer> children = adj.get(i);
//funcion lambda
if (children.stream().anyMatch((c) -> (isCyclicUtil(c, visited, recStack)))) {
```

```
                return true;
            }
//  for (Integer c: children)
//  if (isCyclicUtil(c, visited, recStack))
//  return true;             recStack[i] = false;
            return false;
        }
        private void addEdge(int source, int dest) {
            adj.get(source).add(dest);
        }
        /* Retorna true si el grafo tiene un ciclo, si no falso*/
        private boolean isCyclic() {
            //Marca todos los vetices como no visitados
            // y no parte de la pila de recursión
            boolean[] visited = new boolean[V];
            boolean[] recStack = new boolean[V];
            for (int i = 0; i < V; i++) {
                if (isCyclicUtil(i, visited, recStack)) {
                    return true;
                }
            }
            return false;
        }

        public static void main(String[] args) {
            Graph graph = new Graph(4);
            graph.addEdge(0, 1);
            graph.addEdge(0, 2);
            graph.addEdge(1, 2);
            graph.addEdge(2, 0);
            graph.addEdge(2, 3);
            graph.addEdge(3, 3);
            if (graph.isCyclic()) {
                System.out.println("El grafo contiene un ciclo");
            } else {
                System.out.println("El grafo no contiene"
                        + "un ciclo");
            }
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#define MAX 101
using namespace std;
vector<vector<int> > adj(MAX);

struct Graph {
    int V;

    void addEdge(int source, int dest) {
        adj[source].push_back(dest);
    }
```

```
    bool isCycleUtil(int i, bool visited[], bool recStack[]) {
        if (recStack[i]) {
            return true;
        }
        if (visited[i]) {
            return false;
        }
        visited[i] = true;
        recStack[i] = true;
        vector<int> children = adj[i];
        for (int c = 0; c < children.size(); c++) {
            if (isCycleUtil(children[c], visited, recStack)) {
                return true;
            }
        }
        recStack[i] = false;
        return false;
    }

    bool iscyclic() {
        bool visit[V];
        bool recStack[V];
        memset(visit, false, sizeof visit);
        memset(recStack, false, sizeof recStack);
        for (int i = 0; i < V; i++) {
            if (isCycleUtil(i, visit, recStack)) {
                return true;
            }
        }
        return false;
    }
};

int main() {
    Graph g;
    g.V = 4;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    if (g.iscyclic()) {
        cout << "el grafo SI tiene un ciclo";
    } else {
        cout << "el grafo NO tiene un ciclo";
    }
    return 0;
}
```

15.10) DFS Cerramiento transitivo

Dado un grafo dirigido, encontrar si un vértice v es alcanzable desde otro vértice u para todos los pares de vértices (u,v) en el grafo dado, aquí alcanzable significa que existe un camino desde el vértice u a v, la matriz de habilidad de alcance es llamada cerramiento transitivo de un grafo.

La solución está basada en el algoritmo de Floyd Warshall,

Los pasos abstractos de este algoritmo son:

- Crear una matriz tc[V][V] que pueda tener finalmente el cerramiento transitivo de un grafo dado, inicializar todas sus entradas como 0.
- Llamar DFS por cada nodo del grado para marcar vértices alcanzables en tc[][]. En llamadas recursivas de DFS no podemos llamar DFS para un vértice adyacente si este ya fue marcado como alcanzable en tc[].

El código usa listas de adyacencia para el grafo de entrada y construye una matriz tc[V][V] tal que tc[u][v] será true si v es alcanzable desde u.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
//Programa java que imprime el cerramiento transitivo de un grafo

import java.util.ArrayList;
import java.util.Arrays;

public class DFSTransitiveClosure {

    public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println("Matriz de cerramiento transitivo es ");
```

```
        g.transitiveClosure();
    }

    static class Graph {
        // número de vertices
        private int vertices;
        // lista de adyacencia
        private ArrayList<Integer>[] adjList;
        // para almacenar el cerramiento transitivo
        private int[][] tc;
        // Constructor
        public Graph(int vertices) {
            // inicializa el conteo de vertices
            this.vertices = vertices;
            this.tc = new int[this.vertices][this.vertices];
            // initializa lista de adyacencia
            initAdjList();
        }
        private void initAdjList() {
            adjList = new ArrayList[vertices];
            for (int i = 0; i < vertices; i++) {
                adjList[i] = new ArrayList<>();
            }
        }
        // Agregar caminos
        public void addEdge(int u, int v) {
            adjList[u].add(v);
        }
        // buscando cerramiento transitivo
        public void transitiveClosure() {
            for (int i = 0; i < vertices; i++) {
                dfsUtil(i, i);
            }
            for (int i = 0; i < vertices; i++) {
                System.out.println(Arrays.toString(tc[i]));
            }
        }

        private void dfsUtil(int s, int v) {
            // Marca alcance desde s a v como true
            tc[s][v] = 1;
            // Encuentra todos los vertices alcanzables
            // atraves de v
            for (int adj : adjList[v]) {
                if (tc[s][adj] == 0) {
                    dfsUtil(s, adj);
                }
            }
        }
    }
}
```

15.11) BFS para grafos desconexos

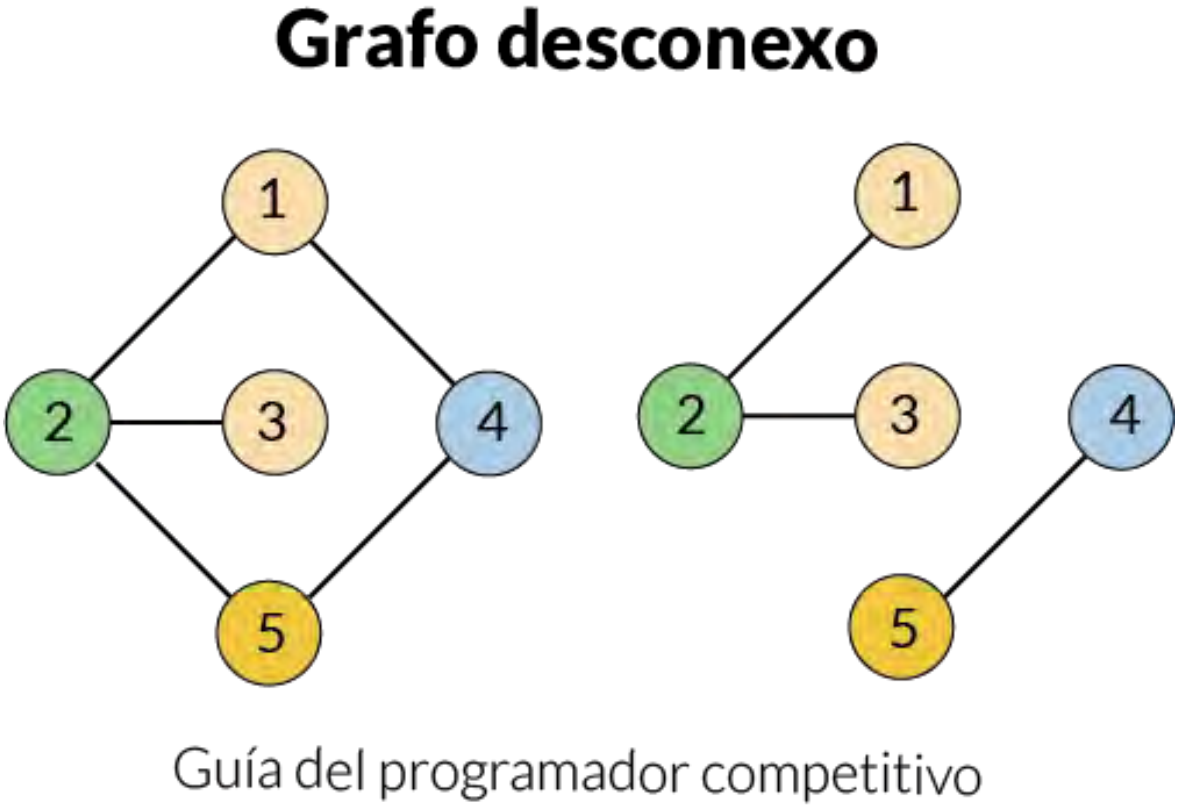


Ilustración 15-10 Ejemplos de grafos desconexos

Por ejemplo asumamos que todos los vértices son alcanzables desde un vértice inicial, pero en el caso de un grafo desconexo o que cualquier vértice es inalcanzable desde todos los vértices, un BFS normal no nos da la salida deseada, por lo que se utiliza esta modificación del BFS.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Implementación de BFS modificado

import java.util.*;
```

```java
public class DisconnectedGraphBFS {
    // Implementando grafo usando HashMap
    static HashMap<Integer, LinkedList<Integer>> graph = new HashMap<>();
    // Agregar caminos al grafo
    public static void addEdge(int a, int b) {
        if (graph.containsKey(a)) {
            LinkedList<Integer> l = graph.get(a);
            l.add(b);
            graph.put(a, l);
        } else {
            LinkedList<Integer> l = new LinkedList<>();
            l.add(b);
            graph.put(a, l);
        }
    }

    public static void bfshelp(int s, ArrayList<Boolean> visited) {
        // Crea una cola para el BFS
        LinkedList<Integer> q = new LinkedList<>();
        //Marca el nodo actual como visitado y lo encola
        q.add(s);
        visited.set(s, true);
        while (!q.isEmpty()) {
            // Desencola un vertice de la cola y la imprime
            int f = q.poll();
            System.out.print(f + " ");
            // Verifica cuando el nodo actual esta conectado
            // a otro nodo o no
            if (graph.containsKey(f)) {
                Iterator<Integer> i = graph.get(f).listIterator();
                // Obtiene todos los nodos adyacentes
                // del nodo desencolado f, si no ha sido visitado
                // lo marca y lo encola
                while (i.hasNext()) {
                    int n = i.next();
                    if (!visited.get(n)) {
                        visited.set(n, true);
                        q.add(n);
                    }
                }
            }
        }
    }

    //Fución BFS que verifica cada nodo
    public static void bfs(int vertex) {
        ArrayList<Boolean> visited = new ArrayList<>();
        //Marcando cada nodo como no visitado
        for (int i = 0; i < vertex; i++) {
            visited.add(i, false);
        }
        for (int i = 0; i < vertex; i++) {
            //Verificando cuantos nodos no han sido visitados
```

```
            if (!visited.get(i)) {
                bfshelp(i, visited);
            }
        }
    }

    public static void main(String[] args) {
        int v = 5;
        addEdge(0, 4);
        addEdge(1, 2);
        addEdge(1, 3);
        addEdge(1, 4);
        addEdge(2, 3);
        addEdge(3, 4);
        bfs(v);
    }
}
```

15.12) Ciclo de Euler en un grafo dirigido

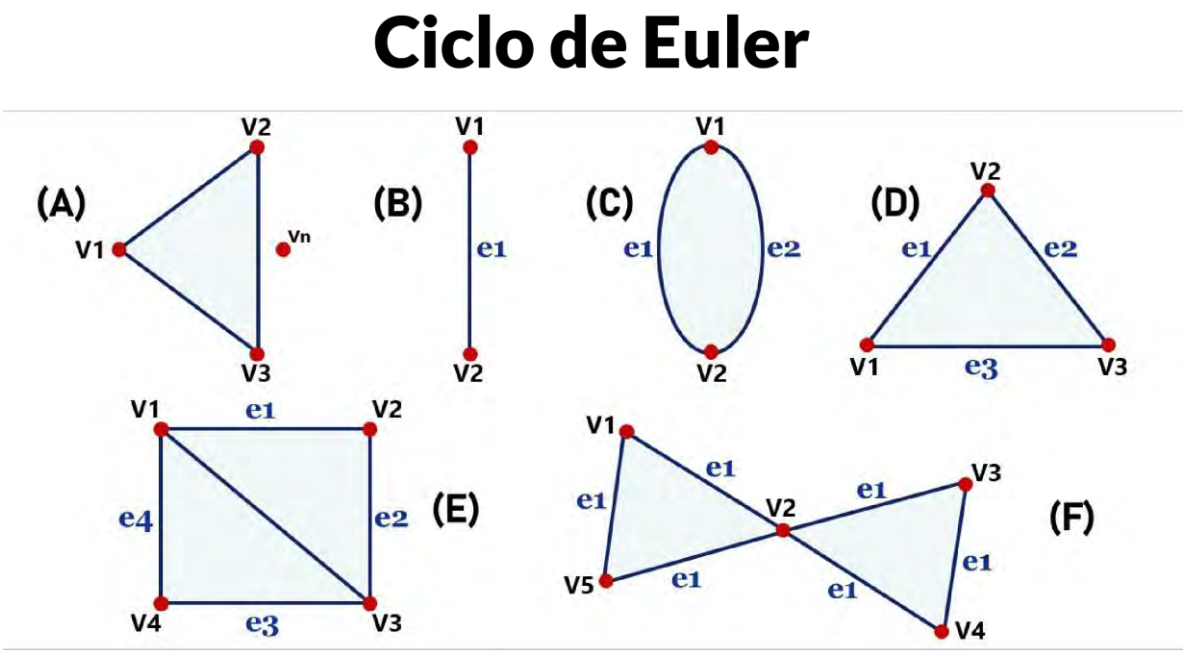


Ilustración 15-11 Ejemplo de diferentes grafos

Un camino de Euler es un camino en un grafo que visita cada camino exactamente una vez, el circuito de Euler es un camino de Euler que empieza y termina en el mismo vértice.
Un grafo es euleriano si tiene un ciclo de Euler.
Un grafo dirigido tiene un ciclo de Euler si las siguientes condiciones son verdad:

1) Todos los vértices con grado no cero pertenecen a una sola componente fuertemente conectada.
2) En los grados, el grado de entrada es igual al grado de salida

Podemos detectar componentes fuertemente conectadas usando el DFS de Kosaraju.
Para comparar los grados de entrada y salida, necesitamos almacenar los grados de entrada y salida de cada vértice, el grado de salida puede ser obtenido por el tamaño de la lista de adyacencia, en grado de entrada puede ser almacenado creando un array de igual tamaño al número de vértices.
La complejidad de tiempo de esta implementación es de O(V+E), luego de correr el algoritmo de Kosaraju, atravesamos todos los vértices y comparamos los grados de salida y entrada, esto toma O(V) tiempo.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que verifica si un grafo es Euleriano

import java.util.*;
import java.util.LinkedList;

public class EulerianCycleDirectedGraph {

    public static void main(String[] args) throws java.lang.Exception {
        Graph g = new Graph(5);
        g.addEdge(1, 0);
        g.addEdge(0, 2);
        g.addEdge(2, 1);
        g.addEdge(0, 3);
        g.addEdge(3, 4);
        g.addEdge(4, 0);
        if (g.isEulerianCycle()) {
            System.out.println("El grafo dado es euleriano ");
```

```
        } else {
            System.out.println("El grafo dado no es euleriano ");
        }
    }

    static class Graph {
        private int V;   // Número de vertices
        private LinkedList<Integer> adj[];//Lista de adyacencia
        private int in[];            //Manteniendo los grados
        //Constructor
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            in = new int[V];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
                in[i] = 0;
            }
        }
        //Agregar caminos
        void addEdge(int v, int w) {
            adj[v].add(w);
            in[w]++;
        }

        void DFSUtil(int v, Boolean visited[]) {
            // Marca nodo actual como visitado
            visited[v] = true;
            int n;
            //  recorre todos los nodos adyacentes a este vertice
            Iterator<Integer> i = adj[v].iterator();
            while (i.hasNext()) {
                n = i.next();
                if (!visited[n]) {
                    DFSUtil(n, visited);
                }
            }
        }
        // Retornar el transpuesto de este grafo
        Graph getTranspose() {
            Graph g = new Graph(V);
            for (int v = 0; v < V; v++) {
                Iterator<Integer> i = adj[v].listIterator();
                while (i.hasNext()) {
                    g.adj[i.next()].add(v);
                    (g.in[v])++;
                }
            }
            return g;
        }
        // Verifica si el grafo esta fuertemente conectado
        Boolean isSC() {
            /*Paso 1: Marca todos los vertices como
            no visitados (Primer DFS)*/
```

```
            Boolean visited[] = new Boolean[V];
            for (int i = 0; i < V; i++) {
                visited[i] = false;
            }
            /* Paso 2: Hace DFS transverso
            iniciando del primer vertice*/
            DFSUtil(0, visited);
            //  Si DFS no visita todos los nodos, retorna falso
            for (int i = 0; i < V; i++) {
                if (visited[i] == false) {
                    return false;
                }
            }
            /* Paso 3: Crea un grafo reversado*/
            Graph gr = getTranspose();
            /* Paso 4: marca todos los vertices
            como no visitados (Segundo dfs)*/
            for (int i = 0; i < V; i++) {
                visited[i] = false;
            }
            /* Paso 5: hacer DFS para el grafo reverso
            iniciando desde el vertice primero
            debe ser el mismo que el primer DFS*/
            gr.DFSUtil(0, visited);
            // Si todos los vertices no son visitados en el segund
            // DFS retorna falso
            for (int i = 0; i < V; i++) {
                if (visited[i] == false) {
                    return false;
                }
            }
            return true;
        }

        /* Esta función retorna true si encuentra un
        ciclo de euler, falso si no*/
        Boolean isEulerianCycle() {
            // verifica si todos los vertices con grado no cero
            // viendo si estan conectados
            if (isSC() == false) {
                return false;
            }
            // Verifica si en grado de entrada y
            // salida cada vertice es igual
            for (int i = 0; i < V; i++) {
                if (adj[i].size() != in[i]) {
                    return false;
                }
            }
            return true;
        }
    }
}
```

C++

```cpp
#include <bits/stdc++.h>
#define FAST ios_base::sync_with_stdio(false);cin.tie(NULL);

using namespace std;

const int MAX_V = 90;

struct Graph {
    int V;
    vector<int> adj[MAX_V];
    int in[MAX_V];

    Graph(int v) {
        V = v;
        for (int i = 0; i < V; i++) {
            in[i] = 0;
        }
    }

    void addEdge(int source, int destiny) {
        adj[source].push_back(destiny);
        in[destiny]++;
    }

    void DFSUtil(int v, bool visited[]) {
        visited[v] = true;
        int n;
        for (int i = 0; i < adj[v].size(); ++i) {
            n = adj[v][i];
            if (!visited[n]) {
                DFSUtil(n, visited);
            }
        }
    }

    Graph getTranspose() {
        Graph ge(V);
        for (int v = 0; v < V; v++) {
            for (int i : adj[v]) {
                ge.adj[i].push_back(v);
                ge.in[v]++;
            }
        }
        return ge;
    }

    bool isSC() {
        bool visited[V];
        memset(visited, false, V);
        DFSUtil(0, visited);
        for (int i = 0; i < V; i++) {
            if (visited[i] == false) {
                return false;
            }
```

```
        }
        Graph gr = getTranspose();
        memset(visited, false, sizeof visited);
        gr.DFSUtil(0, visited);
        for (int i = 0; i < V; i++) {
            if (visited[i] == false) {
                return false;
            }
        }
        return true;
    }

    bool isEulerian() {
        if (isSC() == false) {
            return false;
        }
        for (int i = 0; i < V; i++) {
            if (adj[i].size() != in[i]) {
                return false;
            }
        }
        return true;
    }
};

int main() {
    FAST
            int vertices, caminos;
    cin >> vertices >> caminos;
    Graph g(vertices);
    for (int i = 0; i < caminos; i++) {
        int inicio, destino;
        cin >> inicio >> destino;
        g.addEdge(inicio, destino);
    }
    if (g.isEulerian()) {
        cout << "El grafo dado es Euleriano" << endl;
    } else {
        cout << "El grafo dado no es Euleriano" << endl;
    }
    return 0;
}
```

15.13) Camino de Euler en un grafo no dirigido

Camino eureliano

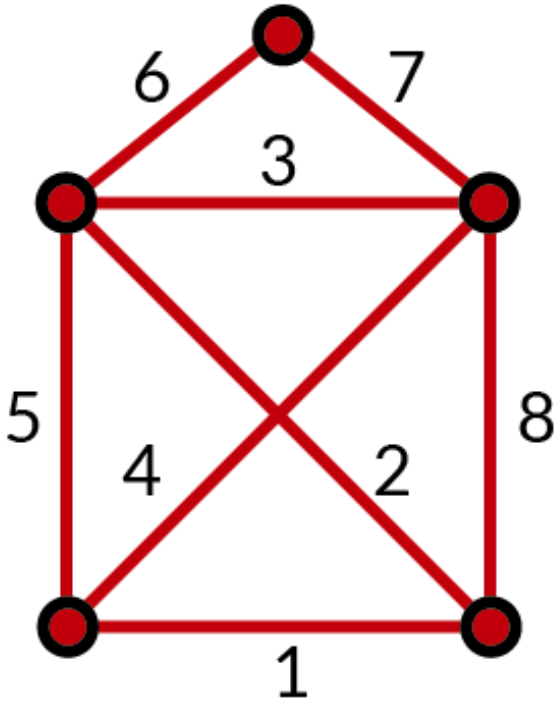


Ilustración 15-12 La casita, usada como ejemplo en muchos problemas de grafos

¿Es possible dibujar un grafo dado sin levantar el lápiz del papel y sin pasar por los caminos más de una vez?

Un grafo es llamado euleriano si tiene un ciclo de Euler y es llamado semi-euleriano si tiene un camino de Euler, el problema es similar al camino Hamiltoniano, podemos verificar si un grafo es euleriano o no en tiempo polinómico, de O(V+E).

Las siguientes son propiedades interesantes de los grafos no dirigidos con un camino euleriano y ciclo, podemos usar esta propiedades para encontrar si un grafo es euleriano o no.

Ciclo euleriano

Un grafo no dirigido tiene un ciclo de Euler si las siguientes dos condiciones son ciertas:

- Todos los vértices con grado no cero estan conectados, no nos importa los vértices con grado cero porque no pertenecen a el ciclo de Euler o el camino, solo estamos considerando los caminos.
- Todos los vértices tienen grado par.

Camino euleriano

Un grafo no dirigido tiene un camino euleriano si las dos siguientes condiciones se cumplen:

- La misma primera condición de un ciclo de Euler
- Si dos vértices tienen grado impar y todos los demás vértices tienen grado par, note que solo un vértice con grado impar no es posible en un grafo no dirigido, la suma de todos los grados es siempre par en un grafo no dirigido.

Note que un grafo sin caminos se considera euleriano porque no hay caminos que atravesar. En el camino euleriano, cada vez que visitamos un vértice v, nosotros caminamos a través de dos caminos no visitados, con un punto de fin como v, por lo tanto todos los vértices medios en el camino euleriano deben tener grado par, para el ciclo euleriano cualquier vértice puede ser vértice medio, sin embargo todos los vértices deben tener grado par.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que busca camino euleriano
// de un grafo

import java.util.*;
import java.util.LinkedList;

public class EulerianPathUndirectedGraph {

    public static void main(String args[]) {
        // Creamos varios ejemplos de grafos para probar
        Graph g1 = new Graph(5);
        g1.addEdge(1, 0);
        g1.addEdge(0, 2);
        g1.addEdge(2, 1);
        g1.addEdge(0, 3);
        g1.addEdge(3, 4);
```

```
        g1.test();
        Graph g2 = new Graph(5);
        g2.addEdge(1, 0);
        g2.addEdge(0, 2);
        g2.addEdge(2, 1);
        g2.addEdge(0, 3);
        g2.addEdge(3, 4);
        g2.addEdge(4, 0);
        g2.test();
        Graph g3 = new Graph(5);
        g3.addEdge(1, 0);
        g3.addEdge(0, 2);
        g3.addEdge(2, 1);
        g3.addEdge(0, 3);
        g3.addEdge(3, 4);
        g3.addEdge(1, 3);
        g3.test();
        /* Crearemos un grafo con tres vertices
        conectados en forma de ciclo*/
        Graph g4 = new Graph(3);
        g4.addEdge(0, 1);
        g4.addEdge(1, 2);
        g4.addEdge(2, 0);
        g4.test();
        /* Creamos un grafo con vertices con grado cero*/
        Graph g5 = new Graph(3);
        g5.test();
    }

    static class Graph {

        private int V;
        private LinkedList<Integer> adj[];
        // Constructor
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
            }
        }
        //Agregar caminos
        void addEdge(int v, int w) {
            adj[v].add(w);
            adj[w].add(v); //El grafo es no dirigido
        }

        void DFSUtil(int v, boolean visited[]) {
            visited[v] = true;
            Iterator<Integer> i = adj[v].listIterator();
            while (i.hasNext()) {
                int n = i.next();
                if (!visited[n]) {
                    DFSUtil(n, visited);
```

```
        }
    }
}

boolean isConnected() {
    boolean visited[] = new boolean[V];
    int i;
    for (i = 0; i < V; i++) {
        visited[i] = false;
    }
    for (i = 0; i < V; i++) {
        if (!adj[i].isEmpty()) {
            break;
        }
    }
    //Si no hay caminos en el grafo, retorna true
    if (i == V) {
        return true;
    }
    DFSUtil(i, visited);
    for (i = 0; i < V; i++) {
        if (visited[i] == false && adj[i].size() > 0) {
            return false;
        }
    }
    return true;
}

int isEulerian() {
    if (isConnected() == false) {
        return 0;
    }
    // Cuenta vertices con grado impar
    int odd = 0;
    for (int i = 0; i < V; i++) {
        if (adj[i].size() % 2 != 0) {
            odd++;
        }
    }
    // Si cuenta es más de 2, el grafo no es euleriano
    if (odd > 2) {
        return 0;
    }
    //si odd es 2, es semieuleriano
    //Si odd es 0, es eulerian0
    return (odd == 2) ? 1 : 2;
}
void test() {
    int res = isEulerian();
    switch (res) {
        case 0:
            System.out.println("Grafo no es euleriano");
            break;
        case 1:
```

```
                    System.out.println("Grafo tiene un camino de euler");
                    break;
                default:
                    System.out.println("Grafo tiene ciclo de euler");
                    break;
            }
        }
    }
}
```

C++

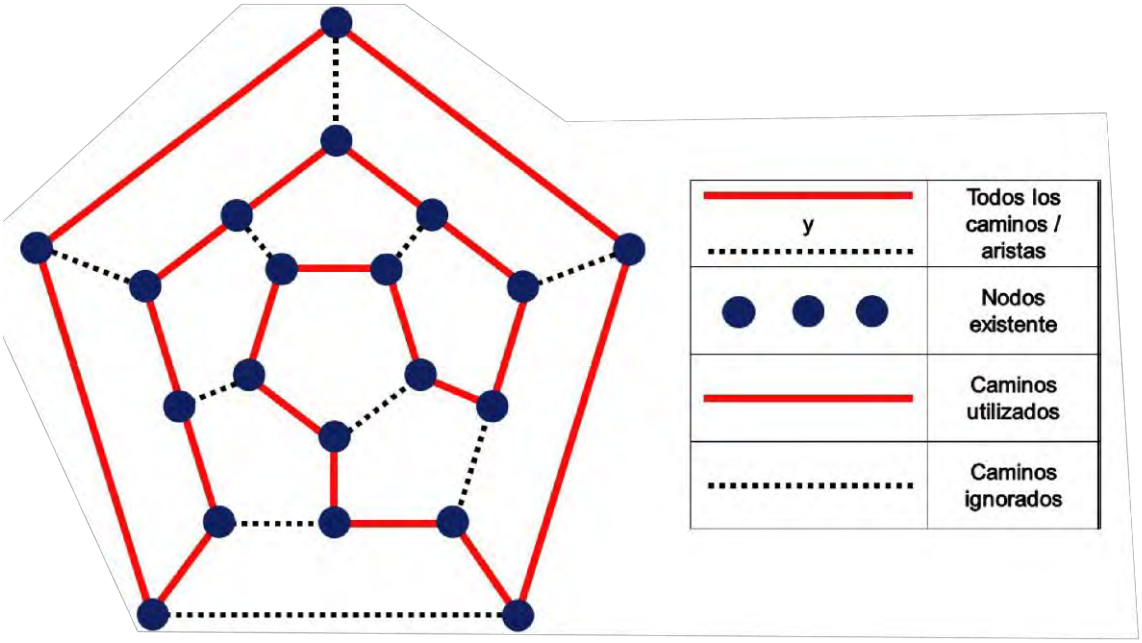
```
#include <bits/stdc++.h>
#define MAX 101
using namespace std;
vector<vector<int> >adj(MAX);
int in[MAX];

void clean() {
    adj.clear();
    adj.erase(adj.begin(), adj.begin() + adj.size());
    memset(in, 0, sizeof in);
}

struct Graph {
    int V;

    void addEdge(int v, int w) {
        adj[v].push_back(w);
        adj[w].push_back(v);
    }

    void DFSUtil(int v, bool visited[]) {
        visited[v] = true;
        int n;
        for (int i = 0; i < adj[v].size(); ++i) {
            n = adj[v][i];
            if (!visited[n]) {
                DFSUtil(n, visited);
            }
        }
    }

    bool isConnected() {
        bool visited[V];
        int i;
        memset(visited, false, sizeof visited);
        for (i = 0; i < V; i++) {
            if (!adj[i].empty()) {
                break;
            }
        }
        if (i == V) {
            return true;
        }
```

```
        DFSUtil(i, visited);
        for (i = 0; i < V; i++) {
            if (visited[i] == false && adj[i].size() > 0) {
                return false;
            }
        }
        return true;
    }

    int isEulerian() {
        if (isConnected() == false) {
            return 0;
        }
        int odd = 0;
        for (int i = 0; i < V; i++) {
            if (adj[i].size() % 2 != 0) {
                odd++;
            }
        }
        if (odd > 2) {
            return 0;
        }
        return (odd == 2) ? 1 : 2;
    }

    void test() {
        int res = isEulerian();
        switch (res) {
            case 0:
                cout << "Grafo no euleriano" << endl;
                break;
            case 1:
                cout << "Grafo tiene camino de euler" << endl;
                break;
            case 2:
                cout << "Grafo tiene ciclo de euler" << endl;
                break;
        }
    }

    void init() {
        adj.clear();
        memset(in, 0, sizeof in);
    }

};

int main() {
    /*
    Graph g1;
    g1.init();
    g1.V=5;
    g1.addEdge(1, 0);
    g1.addEdge(0, 2);
```

```
        g1.addEdge(2, 1);
        g1.addEdge(0, 3);
        g1.addEdge(3, 4);
        g1.test();
        Graph g2;
        g2.init();
        g2.V=5;
        g2.addEdge(1, 0);
        g2.addEdge(0, 2);
        g2.addEdge(2, 1);
        g2.addEdge(0, 3);
        g2.addEdge(3, 4);
        g2.addEdge(4, 0);
        g2.test();
         */
        Graph g3;
        g3.init();
        g3.V = 5;
        g3.addEdge(1, 0);
        g3.addEdge(0, 2);
        g3.addEdge(2, 1);
        g3.addEdge(0, 3);
        g3.addEdge(3, 4);
        g3.addEdge(1, 3);
        g3.test();
        /*
        Graph g4;
        g4.init();
        g4.V=3;
        g4.addEdge(0, 1);
        g4.addEdge(1, 2);
        g4.addEdge(2, 0);
        g4.test();
        Graph g5;
        g5.init();
        g5.V=3;
        g5.test();
         */

}
```

15.14) Ciclo Hamiltoniano

Ciclo de Hamilton



Guia del programador competitivo

Ilustración 15-13 Ciclo Hamiltoniano en un grafo

El camino Hamiltoniano de un grafo no dirigido es un camino que visita cada vértice exactamente una vez, un ciclo Hamiltoniano es un camino Hamiltoniano que tiene un camino desde el último vértice al primer vértice del camino Hamiltoniano, debemos determinar si un grafo tiene ciclo Hamiltoniano o no, si lo contiene imprimir el camino.

Nuestra entrada será un array graph[v][v] donde v es el número de vértices en el grafo y graph[][] es la matriz de adyacencia representando el grafo, un valor graph[i][j] es 1 si existe un camino directo de I a j, de lo contrario será 0.

Nuestra salida será un array path[v] que deberá contener el camino Hamiltoniano, path[i] puede representar el iesimo vértice en el camino Hamiltoniano. El código debe también retornar si no existe ciclo Hamiltoniano en el grafo.

Se crea un array de camino vacío y se le agrega el vértice 0 a él, se agregan los otros vértices iniciando desde el vértice 1, antes de añadir un vértice, se verifica por cual es el adyacente

del anterior añadido y si no se ha añadido ya, si encontramos tal vértice, añadimos el vértice como parte de la solución, si no lo encontramos retornamos false.

Note que el código siempre imprime el ciclo iniciando de 0, el punto de inicio no importa ya que el ciclo puede empezar de cualquier punto, si se quiere cambiar el punto de inicio, se deben hacer dos cambios al código de abajo.

Cambie "path[0] = 0;" por "path[0] = s;" donde s es el nuevo punto de inicio, también cambie el ciclo "for (int v = 1; v < V; v++)" en hamCycleUtil() por "for (int v = 0; v < V; v++)".

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
/* Programa java que soluciona el ciclo de Hamilton
usando backtracking*/

public class HamiltonianCycle {

    static final int V = 5;
    static int path[];

    /* Una función de utilidad para comprobar si el vértice v puede ser
       añadido en el índice 'pos'en el ciclo hamiltoniano
       construido hasta ahora (almacenado en 'path[]')*/
    static boolean isSafe(int v, int graph[][], int path[], int pos) {
        /*Verifica si este vertice es adyacente del
        anterior vertice */
        if (graph[path[pos - 1]][v] == 0) {
            return false;
        }
        /* Verifica si el vertice ya esta incluido*/
        for (int i = 0; i < pos; i++) {
            if (path[i] == v) {
                return false;
            }
        }
        return true;
    }

    static boolean hamCycleUtil(int graph[][], int path[], int pos) {
        /* Caso base:Si todos los vertices estan incluidos en
           el ciclo hamiltoniano*/
        if (pos == V) {
            // Y si hay un camino de el ultimo al primer vertice
            return graph[path[pos - 1]][path[0]] == 1;
        }
```

```
        /*Prueba diferentes vértices como próximo candidato
        en el ciclo hamiltoniano. No intentamos con 0,
        ya que incluimos 0 como punto de partida en hamCycle ()*/
        for (int v = 1; v < V; v++) {
            /* Verifica si este vertice puede ser añadido al
            ciclo hamiltoniano*/
            if (isSafe(v, graph, path, pos)) {
                path[pos] = v;
                /* Recorre hasta construir el camino*/
                if (hamCycleUtil(graph, path, pos + 1) == true) {
                    return true;
                }
                path[pos] = -1;
            }
        }
        /* Si no hay vertice para añadir al ciclo
        retorna falso*/
        return false;
    }

    /* Esta función resuelve el problema del ciclo hamiltoniano usando
       backtracking. Utiliza principalmente hamCycleUtil () para resolver el
       problema. Devuelve falso si no hay ciclo hamiltoniano.
       posible, de lo contrario devuelve verdadero e imprime la ruta.
       Tenga en cuenta que puede haber más de una solución,
       Esta función imprime una de las soluciones factibles.*/
    static int hamCycle(int graph[][]) {
        path = new int[V];
        for (int i = 0; i < V; i++) {
            path[i] = -1;
        }
        /* Pongamos el vértice 0 como el primer vértice en el camino.
           Si hay un ciclo hamiltoniano, entonces el camino puede ser
           Comenzó desde cualquier punto del ciclo ya que la gráfica es
           no dirigido*/
        path[0] = 0;
        if (hamCycleUtil(graph, path, 1) == false) {
            System.out.println("\nNo existe solución");
            return 0;
        }
        printSolution(path);
        return 1;
    }

    // Imprimir solución
    static void printSolution(int path[]) {
        System.out.println("Solución existe: Este"
                + " es uno de los ciclos hamiltoniano");
        for (int i = 0; i < V; i++) {
            System.out.print(" " + path[i] + " ");
        }
        System.out.println(" " + path[0] + " ");
    }
```

```
    public static void main(String args[]) {
        /*Tenemos el siguiente grafo
           (0)--(1)--(2)
            |   / \   |
            |  /   \  |
            | /     \ |
           (3)-------(4)    */
        int graph1[][] = {{0, 1, 0, 1, 0},
        {1, 0, 1, 1, 1},
        {0, 1, 0, 0, 1},
        {1, 1, 0, 0, 1},
        {0, 1, 1, 1, 0},};
        // Imprimir solución
        hamCycle(graph1);
        /*Tenemos el siguiente grafo
           (0)--(1)--(2)
            |   / \   |
            |  /   \  |
            | /     \ |
           (3)       (4)    */
        int graph2[][] = {{0, 1, 0, 1, 0},
        {1, 0, 1, 1, 1},
        {0, 1, 0, 0, 1},
        {1, 1, 0, 0, 0},
        {0, 1, 1, 0, 0},};
        // Imprimir solución
        hamCycle(graph2);
    }
}
```

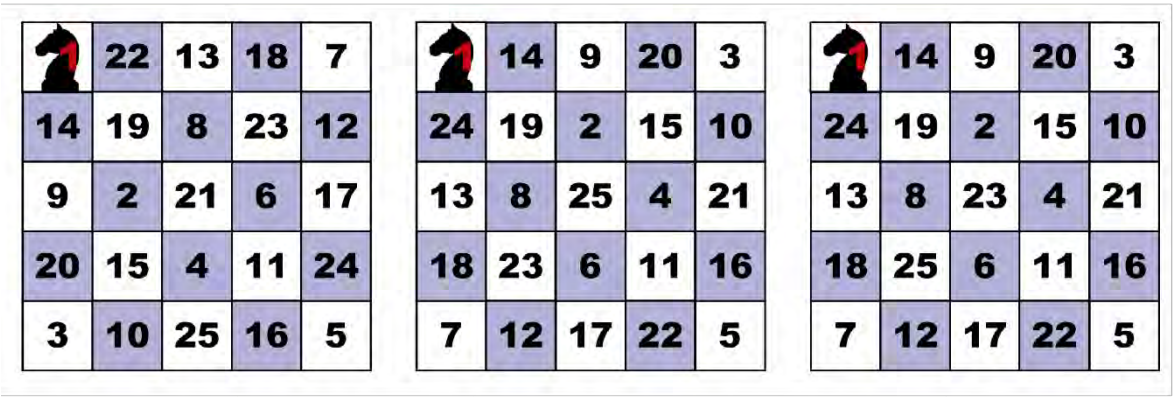
C++

```
#include<bits/stdc++.h>
#include<cstdlib>
#define MAX 256
using namespace std;
const int V = 5;
int path[MAX];

bool isSafe(int v, int graph[V][V], int path[], int pos) {
    if (graph[path[pos - 1]][v] == 0)return false;
    for (int i = 0; i < pos; i++) {
        if (path[i] == v)return false;
    }
    return true;
}

bool hamCycleUtil(int graph[V][V], int path[], int pos) {
    if (pos == V)return graph[path[pos - 1]][path[0]] == 1;
    for (int v = 1; v < V; v++) {
        if (isSafe(v, graph, path, pos)) {
            path[pos] = v;
            if (hamCycleUtil(graph, path, pos + 1) == true)return true;
        }
        path[pos] = -1;
```

```
    }
    return false;
}

void printSolution(int path[]) {
    cout << "Solucion existente" << endl;
    for (int i = 0; i < V; i++) {
        cout << path[i] << " ";
    }
    cout << endl;
}

int hamCycle(int graph[V][V]) {
    path[V];
    memset(path, -1, sizeof path);
    path[0] = 0;
    if (hamCycleUtil(graph, path, 1) == false) {
        cout << "NO existe ciclo hamiltoniano" << endl;
        return 0;
    }
    printSolution(path);
    return 1;
}

int main() {
    int graph1[V][V] = {
        {0, 1, 0, 1, 0},
        {1, 0, 1, 1, 1},
        {0, 1, 0, 0, 1},
        {1, 1, 0, 0, 1},
        {0, 1, 1, 1, 0},};
    // Imprimir soluciÃ³n
    hamCycle(graph1);
    int graph2[V][V] = {
        {0, 1, 0, 1, 0},
        {1, 0, 1, 1, 1},
        {0, 1, 0, 0, 1},
        {1, 1, 0, 0, 0},
        {0, 1, 1, 0, 0},};
    // Imprimir soluciÃ³n
    hamCycle(graph2);
}
```

15.15) El tour del caballo de ajedrez

Knight Tour 01



Guía del programador competitivo

Ilustración 15-14 La travesía del caballero en un tablero de ajedrez

El backtracking es un paradigma algorítmico que intenta diferentes soluciones hasta que encuentra una solución que "Funciona". Problemas los cuales son típicamente resueltos con técnicas de backtracking tienen una propiedad en común, estos problemas puede solo ser resueltos intentando cada posible configuración y cada configuración es intentada una sola vez, una solución ingenua para estos problemas es intentar todas las configuraciones e imprimir una configuración que siga las restricciones dadas por el problema. Backtracking funciona en forma incremental y es una optimización sobre las soluciones ingenuas donde todas las posibles configuraciones son generadas e intentadas.

Knight Tour 02

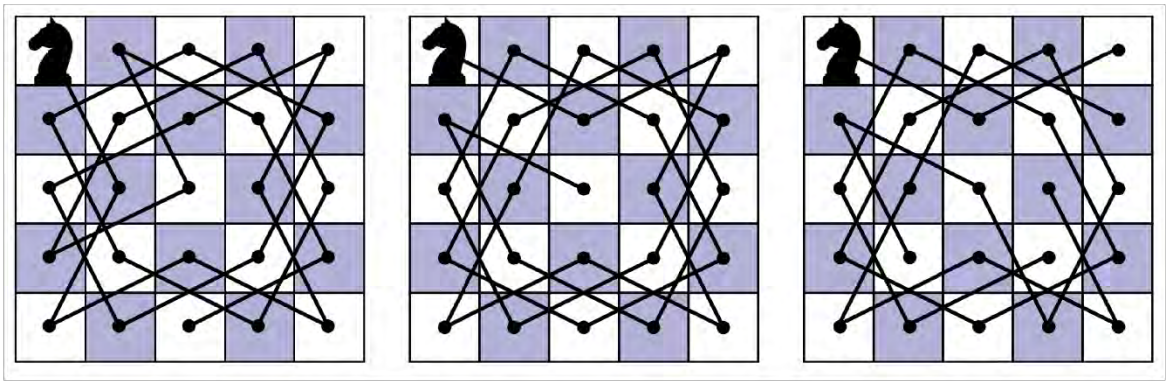


Ilustración 15-15 Formas de solución para el caballero en el tablero de ajedrez

Backtracking funciona de forma incremental para atacar problemas, típicamente iniciamos desde un vector de solución vacío, y uno por uno vamos agregando ítems, cuando agregamos un ítem podemos verificar si agregando el siguiente ítem violamos alguna de las restricciones del problema, si lo hace eliminamos ese elemento e intentamos otras alternativas. Si ninguna de las alternativas funciona entonces volvemos a la fase previa y removemos el ítem anterior dado en la fase anterior. Si alcanzamos la fase inicial entonces decimos que no existe solución, si agregamos un valor que no viola alguna restricción entonces recursivamente agregamos ítems uno por uno, si el vector de solución se completa imprimimos la solución.

El siguiente es el backtracking del problema del tour del caballo en el tablero de ajedrez.

- Si todos los cuadrados son visitados imprima la solución

- Si no

a) Agregue uno de los siguientes movimientos posibles al vector de solución y recursivamente verifique si este movimiento lleva a una solución (Un caballo puede hacer máximo 8 movimientos, aquí escogemos alguno de esos 8 movimientos).
b) Si el movimiento escogido arriba no lleva a una solución entonces removemos este movimiento del vector de solución e intentamos otros movimientos alternativos.
c) Si ninguna de las alternativas funciona, retornamos falso (Retornando falso podemos remover el anterior ítem agregado en recursión y si el falso es retornado a la recursión inicial entonces no existe solución.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java para el problema del tour del caballo
public class KnightTourBacktracking {

    static int N = 8;

    /* Una función que verifica si i.j es
    indice valido para un tablero n*n*/
    static boolean isSafe(int x, int y, int sol[][]) {
        return (x >= 0 && x < N && y >= 0
                && y < N && sol[x][y] == -1);
    }

    /* Imprimir la solución*/
    static void printSolution(int sol[][]) {
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                System.out.print(sol[x][y] + "\t");
            }
            System.out.println();
        }
    }
        /*Esta función resuelve el problema de Knight Tour.
       utilizando Backtracking. Esta función principalmente
       utiliza solveKTUtil () para resolver el problema. Eso
       devuelve falso si no es posible realizar un recorrido completo,
       De lo contrario, devuelve true e imprime el recorrido.
       Tenga en cuenta que puede haber más de una
       soluciones, esta función imprime una de las
       soluciones viables.*/
    static boolean solveKT() {
        int sol[][] = new int[N][N];
        /* Inicializando la matriz de solución*/
```

```
        for (int x = 0; x < N; x++) {
            for (int y = 0; y < N; y++) {
                sol[x][y] = -1;
            }
        }
        /* xMove[] y yMove[] define el siguiente movimiento del caballo
          xMove[] para siguiente valor en x
          yMove[] para siguiente valor en y */
        int xMove[] = {2, 1, -1, -2, -2, -1, 1, 2};
        int yMove[] = {1, 2, 2, 1, -1, -2, -2, -1};
        //Desde que el caballero inicie en el primer bloque
        sol[0][0] = 0;
        if (!solveKTUtil(0, 0, 1, sol, xMove, yMove)) {
            System.out.println("No existe la solución");
            return false;
        } else {
            printSolution(sol);
        }
        return true;
    }

    static boolean solveKTUtil(int x, int y, int movei,
            int sol[][], int xMove[],
            int yMove[]) {
        int k, next_x, next_y;
        if (movei == N * N) {
            return true;
        }
        /* Intenta todos los movimientos desde la coordenada
        x  y y*/
        for (k = 0; k < 8; k++) {
            next_x = x + xMove[k];
            next_y = y + yMove[k];
            if (isSafe(next_x, next_y, sol)) {
                sol[next_x][next_y] = movei;
                if (solveKTUtil(next_x, next_y, movei + 1,
                        sol, xMove, yMove)) {
                    return true;
                } else {
                    sol[next_x][next_y] = -1;// backtracking
                }
            }
        }
        return false;
    }
    public static void main(String args[]) {
        solveKT();
    }
}
```

C++

```
#include <bits/stdc++.h>
#define tablero 101
using namespace std;
```

```
int N = 12;

bool isSafe(int x, int y, int sol[tablero][tablero]) {
    return (x >= 0 && x < N && y >= 0 && y < N && sol[x][y] == -1);
}

void printSolution(int sol[tablero][tablero]) {
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            cout << sol[x][y] << "\t";
        }
        cout << endl;
    }
}

bool solveKUtil(int x, int y, int movei, int sol[tablero][tablero], int
xMove[tablero], int yMove[tablero]) {
    int k, next_x, next_y;
    if (movei == N * N) {
        return true;
    }
    for (k = 0; k < N; k++) {
        next_x = x + xMove[k];
        next_y = y + yMove[k];
        if (isSafe(next_x, next_y, sol)) {
            sol[next_x][next_y] = movei;
            if (solveKUtil(next_x, next_y, movei + 1, sol, xMove, yMove)) {
                return true;
            } else {
                sol[next_x][next_y] = -1;
            }
        }
    }
    return false;
}

bool solveKT() {
    int sol[tablero][tablero];
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            sol[x][y] = -1;
        }
    }
    int xMove[8] = {2, 1, -1, -2, -2, -1, 1, 2};
    int yMove[8] = {1, 2, 2, 1, -1, -2, -2, -1};
    sol[0][0] = 0;
    if (!solveKUtil(0, 0, 1, sol, xMove, yMove)) {
        cout << "No existe la solucion" << endl;
        return false;
    } else {
        printSolution(sol);
    }
    return true;
}
```

```
int main() {
    solveKT();
}
```

15.16) Kosaraju DFS Componentes fuertemente conexas

Dado un grafo dirigido, encontrar si el grafo se encuentra fuertemente conectado o no, un grado es fuertemente conectado si hay un camino entre cualquier par de vértices.

Esto es fácil para un grafo no dirigido, solo tenemos que hacer BFS y DFS comenzando desde cualquier vértice, si BFS o DFS visita todos los vértices, entonces el grado no dirigido dado está conectado, esta aproximación no funciona con un grafo dirigido.

Una simple idea es usar un algoritmo de todos los caminos más cortos entre todos los pares como Floyd Warshall o encontrar el cerramiento transitivo del grafo, la complejidad de tiempo de este método es de O (v^3).

Podemos también hacer DFS V veces iniciando desde cada vértice, si alguno de los DFS no visita todos los vértices entonces el grafo no es fuertemente conectado. Este algoritmo toma O (V*(V+E)) en complejidad de tiempo, el cual puede ser el mismo del cerramiento transitivo para un grafo denso.

Una mayor idea puede ser el algoritmo de componentes fuertemente conectadas (SCC), podemos encontrar todos los SCC en O (V+E) tiempo, si el número de SCC es 1, entonces el grafo es fuertemente conectado, el algoritmo de SCC hace trabajo extra cuando busca todos los SCC.

El siguiente es un algoritmo simple basado en el DFS de Kosaraju el cual realiza dos DFS transversos en el grafo.

1) Inicializa todos los vértices como no visitados.

2) Hace un DFS transverso del grado iniciando desde cualquier vértice v, si este DFS transverso no visita todos los vértices, entonces retorna falso.
3) Reversa todos los arcos (O encuentra transpuesta o reversa de grafo)
4) Marque todos los vértices como no visitados en el grafo reverso
5) Hacer un DFS transverso del grafo reversado iniciando desde el mismo vértice v, si el DFS transverso no visita todos los vértices entonces retorna falso, de otro modo retorna true.

La idea es, si cada nodo puede ser alcanzado de un vértice v, y cada nodo puede alcanzar v, entonces el grafo es fuertemente conectado. En el paso dos podemos verificar si todos los vértices son alcanzables desde v, en el paso 4 verificamos si todos los vértices puede alcanzar v (En el grafo reverso, si todos los vértices son alcanzables desde v entonces todos los vértices pueden alcanzar v en el grafo original).
La complejidad de tiempo de esta implementaciones es la misma de DFS, el cual es O (V+E) si el grafo está representado usando listas de adyacencia.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que verifica si un grafo es
// fuertemente conectado

import java.util.*;
import java.util.LinkedList;

public class KosarajuDFSStronglyConnected {

    public static void main(String args[]) {
        Graph g1 = new Graph(5);
        g1.addEdge(0, 1);
        g1.addEdge(1, 2);
        g1.addEdge(2, 3);
        g1.addEdge(3, 0);
        g1.addEdge(2, 4);
        g1.addEdge(4, 2);
        if (g1.isSC()) {
            System.out.println("Si");
        } else {
            System.out.println("No");
```

```
        }
        Graph g2 = new Graph(4);
        g2.addEdge(0, 1);
        g2.addEdge(1, 2);
        g2.addEdge(2, 3);
        if (g2.isSC()) {
            System.out.println("Si");
        } else {
            System.out.println("No");
        }
    }

    static class Graph {
        private int V;
        private LinkedList<Integer> adj[];
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
            }
        }
        void addEdge(int v, int w) {
            adj[v].add(w);
        }

        void DFSUtil(int v, Boolean visited[]) {
            visited[v] = true;
            int n;
            Iterator<Integer> i = adj[v].iterator();
            while (i.hasNext()) {
                n = i.next();
                if (!visited[n]) {
                    DFSUtil(n, visited);
                }
            }
        }
        // Obtiene el transpuesto de un grafo
        Graph getTranspose() {
            Graph g = new Graph(V);
            for (int v = 0; v < V; v++) {
                Iterator<Integer> i = adj[v].listIterator();
                while (i.hasNext()) {
                    g.adj[i.next()].add(v);
                }
            }
            return g;
        }
        //Verifica si el grafo esta fuertemente conectado
        Boolean isSC() {
            /*Paso 1: marcar todos los vertices como no visitados
            (primer DFS)*/
            Boolean visited[] = new Boolean[V];
            for (int i = 0; i < V; i++) {
```

```
                visited[i] = false;
            }
            //Paso 2: DFS transverso desde el primer vertice
            DFSUtil(0, visited);
            // si no visita todos, retorna falso
            for (int i = 0; i < V; i++) {
                if (visited[i] == false) {
                    return false;
                }
            }
            // Paso 3: crear grafo transpuesto
            Graph gr = getTranspose();
            // Paso 4: marcar todos los vertices como no visitados
            // (Segundo DFS)
            for (int i = 0; i < V; i++) {
                visited[i] = false;
            }
            /* Paso 5: hacer DFS en el grafo reversado*/
            gr.DFSUtil(0, visited);
            /* Si todos los vertices no son visitados, retorne falso*/
            for (int i = 0; i < V; i++) {
                if (visited[i] == false) {
                    return false;
                }
            }
            return true;
        }
    }
}
```

15.17) Mínimo de movimientos de un caballo de ajedrez

Dado una mesa de ajedrez cuadrada de tamaño NxN, la posición de un caballo y la posición objetivo, necesitamos encontrar la mínima cantidad de pasos que un caballo toma para llegar a la posición objetivo.

Este problema puede verse como el camino más corto en un grafo sin pesos, sin embargo usamos BFS ´para resolver este problema, intentamos todos las 8 posibles posiciones donde un caballo puede llegar desde su posición, si la posición alcanzable no ha sido visitada ya y está dentro del tablero, agregamos este estado dentro de la cola con una distancia de 1 más

que su estado padre, finalmente retornamos la distancia de la posición objetivo cuando sale de la cola.

El siguiente código implementa BFS para la búsqueda a través de las celdas, donde cada celda contiene sus coordenadas y distancias desde el nodo inicial, en el peor de los casos el código visita todas las celdas del tablero, haciendo que el la complejidad del peor de los casos sea O(n^2).

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** $O(v^2)$ **Promedio:** O(|v|+|e|)

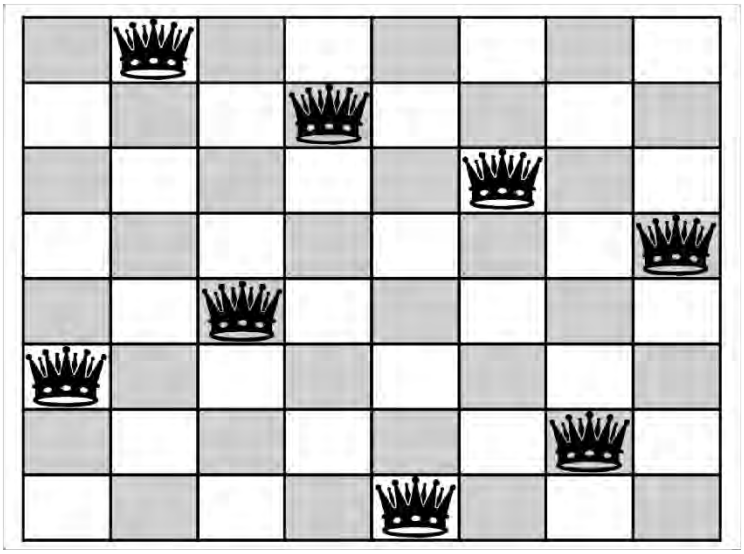
JAVA

```java
//Programa java que encuentra la minima cantidad de pasos
// para alcanzar una celda especifica con un caballo

import java.util.Vector;

public class MinimumMovesKnight {

    /*Clase que almacena los datos de una celda*/
    static class cell {

        int x, y;
        int dis;

        public cell(int x, int y, int dis) {
            this.x = x;
            this.y = y;
            this.dis = dis;
        }
    }

    /* Utilidad que retorna s (x,y) yace dentro del tablero*/
    static boolean isInside(int x, int y, int N) {
        return x >= 1 && x <= N && y >= 1 && y <= N;
    }

    /* Retorna los minimos pasos para llegar al objetivo*/
    static int minStepToReachTarget(int knightPos[], int targetPos[],
            int N) {
        // dirección x y y, donde el caballo puede llegar
        int dx[] = {-2, -1, 1, 2, -2, -1, 1, 2};
        int dy[] = {-1, -2, -2, -1, 1, 2, 2, 1};
        // Vector para almacenar los estados del caballo
        Vector<cell> q = new Vector<>();
        // Agrega la posicion inicial con distancia 0
        q.add(new cell(knightPos[0], knightPos[1], 0));
```

```
        cell t;
        int x, y;
        boolean visit[][] = new boolean[N + 1][N + 1];
        //Hacer todas las celdas sin visitar
        for (int i = 1; i <= N; i++) {
            for (int j = 1; j <= N; j++) {
                visit[i][j] = false;
            }
        }

        // visitar estado inicial
        visit[knightPos[0]][knightPos[1]] = true;
        // ciclo hasta que quede solo un valor
        while (!q.isEmpty()) {
            t = q.firstElement();
            q.remove(0);
            /* Si la celda actual es igual al objetivo
            retorne su distancia*/
            if (t.x == targetPos[0] && t.y == targetPos[1]) {
                return t.dis;
            }
            // Ciclo de todos los estados alcancables
            for (int i = 0; i < 8; i++) {
                x = t.x + dx[i];
                y = t.y + dy[i];
                if (isInside(x, y, N) && !visit[x][y]) {
                    visit[x][y] = true;
                    q.add(new cell(x, y, t.dis + 1));
                }
            }
        }
        return Integer.MAX_VALUE;
    }

    public static void main(String[] args) {
        int N = 30;
        int knightPos[] = {1, 1};
        int targetPos[] = {30, 30};
        System.out.println(minStepToReachTarget(knightPos, targetPos, N));
    }
}
```

15.18) El problema de las N reinas

El problema de las N reinas



Guía del programador competitivo

Ilustración 15-16 El problema de las N reinas en un tablero de ajedrez

El problema de las N reinas es aquel en donde se colocan N reinas en un tablero de NxN de tal manera que dos reinas no puedan atacarse una con otra.

La idea es colocar reinas una por una en diferentes columnas, iniciando desde la columna de más a la izquierda, cuando colocamos una reina en una columna, verificamos por colisiones con las reinas ya colocadas, en la columna actual, si encontramos una fila en la cual no hay colisión marcamos esta columna y fila como parte de la solución, si nosotros no encontramos tal fila en donde haya colisión entonces se retrocede y se retorna falso.

1) Iniciar desde la columna de más a la izquierda.
2) Si todas las reinas han sido colocadas, retornar true
3) Intentar todas las filas de la columna actual

   Realizar lo siguiente para cada columna intentada

   a) si la reina puede ser colocada seguramente en esta fila entonces marcar esta [fila, columna] como parte de la solución y recursivamente verificar si colocando una reina aquí se llega a la solución.

b) Si colocando una reina en [fila,columa] se llega a una solución, entonces retorne true.

c) Si colocando una reina no se llega a la solución entonces desmarque esta fila y columna (Paso atrás) y vaya al paso A para intentar otras filas.

4) Si todas las filas han sido probadas y ninguna funciona retorne falso para activar el backtracking.

Complejidad de tiempo

Mejor caso : $O(v^3)$ **Peor caso :** $O(v^3)$ **Promedio:** $O(v^3)$

JAVA

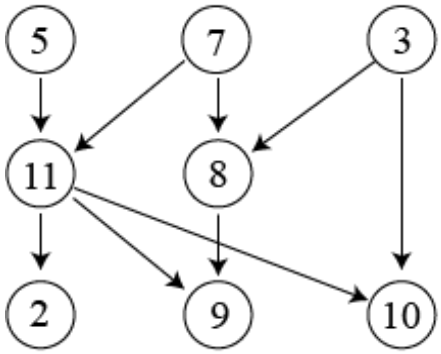
```
/* Programa JAVA que resuelve el problema de las N reinas
usando backtracking*/
public class NQueenProblem {
    final int N = 4;

    void printSolution(int board[][]) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(" " + board[i][j]
                        + " ");
            }
            System.out.println();
        }
    }
/*Una función de utilidad para comprobar si una reina puede
       colocarse en board[row][col]. Tenga en cuenta que
       La función se llama cuando "col" reinas ya están
       Colocadas en columnas de 0 a col -1. Así que necesitamos
       para comprobar sólo el lado izquierdo para las reinas atacantes*/
    boolean isSafe(int board[][], int row, int col) {
        int i, j;
        /*Verifica esta fila en el lado izquierdo*/
        for (i = 0; i < col; i++) {
            if (board[row][i] == 1) {
                return false;
            }
        }
        /*Verifica la diagonal superior de la izquierda*/
        for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
```

```
    /*Diagonal baja desde la izquierda*/
    for (i = row, j = col; j >= 0 && i < N; i++, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }
    return true;
}

boolean solveNQUtil(int board[][], int col) {
    /* Caso base: Si todas las reinas estan
    puestas, retorne true*/
    if (col >= N) {
        return true;
    }
    /*Considere esta columna e intente colocando esta
    reina en todas las filas una por una*/
    for (int i = 0; i < N; i++) {
        /* Verifica si la reina puede ser puesta
        en board[i][col] */
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            /*Recursivamente pone todas las reinas*/
            if (solveNQUtil(board, col + 1) == true) {
                return true;
            }
            board[i][col] = 0; // BACKTRACK
        }
    }
    /*Si la reina no puede ser puesta en
    ninguna fila en esta columna,r retorna falso*/
    return false;
}

/*Esta función resuelve el problema de N Queen usando
Backtracking. Utiliza principalmente solveNQUtil()
para resolver el problema. Devuelve false si no se
pueden colocar las reinas; de lo contrario, devuelve
true e imprime la ubicación de las reinas en forma de
1s. Tenga en cuenta que puede haber más de una solución,
esta función imprime una de las soluciones posibles.*/
boolean solveNQ() {
    int board[][] = {{0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
    };
    if (solveNQUtil(board, 0) == false) {
        System.out.print("Solución no existe");
        return false;
    }
    printSolution(board);
    return true;
}
```

```
    public static void main(String args[]) {
        NQueenProblem Queen = new NQueenProblem();
        Queen.solveNQ();
    }
}
```

15.19) Ordenamiento topológico

Ordenamiento Topológico



El gráfico que semuestran a la izquierda tiene muchas clases topológicas válidos, incluyendo:

- 5, 7, 3, 11, 8, 9, 10 (visual de izquierda a derecha, de arriba a abajo).
- 3, 5, 7, 8,11, 2, 9, 10 (más pequeño número de vértice disponible primero).
- 5, 7, 3, 8, 11, 10, 9, 2 (bordes menor número primero)
- 7, 5, 11, 3, 10, 8, 9, 2 (más grande con números vértice disponible primero)
- 5, 7, 11, 2, 3, 8, 9, 10 (intentar de arriba a abajo, de izquiera a derecha)
- 3, 7, 8, 5, 11, 10, 2, 9 (arbitrario)

Guía del programador competitivo

Ilustración 15-17 Diferentes formas de ordenamiento topológico

Ordenamiento topológico de un grafo dirigido aciclico (DAG) es un ordenamiento lineal de vértices tales que por cada camino dirigido uv, el vértice u venga antes de v en el orden, si el grafo no es DAG no es posible el ordenamiento topológico.

En DFS imprimimos un vértice y luego recursivamente llamamos DFS para los vértices adyacentes, en el ordenamiento topológico necesitamos imprimir un vértice antes de sus vértices adyacentes.

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```java
// Programa java que imprime el ordenamiendo topologico
// de un grafo dirigido

import java.util.*;

public class TopologicalSorting {
    public static void main(String args[]) {
        Graph g = new Graph(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
        g.addEdge(3, 1);
        System.out.println("Siguiendo el ordenamiento topologico "
                + "del grafo dado ");
        g.topologicalSort();
    }

    static class Graph {
        private int V;
        private LinkedList<Integer> adj[];
        Graph(int v) {
            V = v;
            adj = new LinkedList[v];
            for (int i = 0; i < v; ++i) {
                adj[i] = new LinkedList();
            }
        }

        void addEdge(int v, int w) {
            adj[v].add(w);
        }

        void topologicalSortUtil(int v, boolean visited[],
                Stack stack) {
            visited[v] = true;
            Integer i;
            Iterator<Integer> it = adj[v].iterator();
            while (it.hasNext()) {
                i = it.next();
                if (!visited[i]) {
                    topologicalSortUtil(i, visited, stack);
                }
            }
            stack.push(new Integer(v));
        }
        void topologicalSort() {
            Stack stack = new Stack();
            boolean visited[] = new boolean[V];
```

```
            for (int i = 0; i < V; i++) {
                visited[i] = false;
            }
            for (int i = 0; i < V; i++) {
                if (visited[i] == false) {
                    topologicalSortUtil(i, visited, stack);
                }
            }
            // Imprime el contenido de la pila
            while (stack.empty() == false) {
                System.out.print(stack.pop() + " ");
            }
        }
    }
}
```

15.20) Algoritmo de Kahn para ordenamiento topológico

Ordenamiento topológico para un grafo dirigido aciclico (DAG) es un ordenamiento lineal de vértices el cual por cada camino dirigido UV, el vértice U viene antes de V en el ordenamiento, el ordenamiento topológico no es posible si el grafo no es un DAG.

Los pasos de este algoritmo son los siguientes:

1) Calcule los grados de entrada de cada vértice en el DAG presente e inicialice el conteo de los nodos visitados como 0.
2) Tome todos los vértices con grado de entrada como o y agréguelos en una cola
3) Remueva un vértice de la cola y entonces incremente el conteo de nodos visitados por 1
4) Decremente el grado de entrada en 1 en todos los nodos vecinos, si el grado de entrada de los nodos vecinos es cero, agréguelos a la cola.
5) Repita el paso 3 hasta que la cola este vacía.
6) Si el conteo de nodos visitados no es igual al número de notos en el grado entonces el ordenamiento topológico no es posible en este grafo.

¿Cómo encontrar el grado de entrada en cada nodo?

Existen dos vías para encontrar el grado de entrada de cada nodo.

Tomaremos un array de grado de entrada para mantener rastreo de estos.

1) Atraviese el array de nodos y simplemente incremente el conteo del nodo de destino en 1-

```
for each node in Nodes
   indegree[node] = 0;
for each edge(src,dest) in Edges
   indegree[dest]++
```

2) Atraviese la lista de cada nodo e incremente el grado de entrada de todos los nodos conectados con él en 1

```
for each node in Nodes
      If (list[node].size()!=0) then
      for each dest in list
         indegree[dest]++;
```

La complejidad de tiempo se da por: el ciclo externo será ejecutado V número de veces y el interno será ejecutado E número de veces, la complejidad resultante será de O(V+E).

Complejidad de tiempo

Mejor caso : O(|v|+|e|) **Peor caso :** O(|v|+|e|) **Promedio:** O(|v|+|e|)

JAVA

```
// Programa java que imprime el ordenamiento topologico
// de un grafo

import java.util.*;

public class KahnAlgorithmTopologicalSorting {

    public static void main(String args[]) {
        Graph g = new Graph(6);
        g.addEdge(5, 2);
        g.addEdge(5, 0);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(2, 3);
```

```
        g.addEdge(3, 1);
        System.out.println("Siguiendo su ordenamiento topologico ");
        g.topologicalSort();
    }

    static class Graph {
        int V;
        List<Integer> adj[];
        public Graph(int V)// Constructor
        {
            this.V = V;
            adj = new ArrayList[V];
            for (int i = 0; i < V; i++) {
                adj[i] = new ArrayList<>();
            }
        }
        // Agregar caminos
        public void addEdge(int u, int v) {
            adj[u].add(v);
        }

        public void topologicalSort() {
            // Crea un array para almaccenar los indegrees
            // de todos los vertices e inicializa en 0
            int indegree[] = new int[V];
            // Atraviesa las listas de adyacencia para llenar
            // grados de los vertices
            for (int i = 0; i < V; i++) {
                ArrayList<Integer> temp = (ArrayList<Integer>) adj[i];
                temp.forEach((node) -> {
                    indegree[node]++;
                });
            }
            /* Crea una cola y encola todos los vertices
            con grado 0*/
            Queue<Integer> q = new LinkedList<>();
            for (int i = 0; i < V; i++) {
                if (indegree[i] == 0) {
                    q.add(i);
                }
            }
            // Inicializa el conteo de vertices visitados
            int cnt = 0;
            // Crea un vector que almacena el resultado
            ArrayList<Integer> topOrder = new ArrayList<Integer>();
            while (!q.isEmpty()) {
                int u = q.poll();
                topOrder.add(u);
                 //Funcion lamba Reemplazable con un for
                adj[u].stream().filter((node) -> (--indegree[node] ==
0)).forEachOrdered((node) -> {
                    q.add(node);
                });
                cnt++;
```

```
            }
            // Verifica si hay ciclo
            if (cnt != V) {
                System.out.println("There exists a cycle in the graph");
                return;
            }
            //Imprime el ordenamiento topologico
            for (int i : topOrder) {
                System.out.print(i + " ");
            }
        }
    }
}
```

15.21) Caminos más cortos mediante Dijkstra

Dijkstra

STEP	N°	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	u	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyw					

Construir el árbol de ruta más corta rastreando los nodos predecesores.

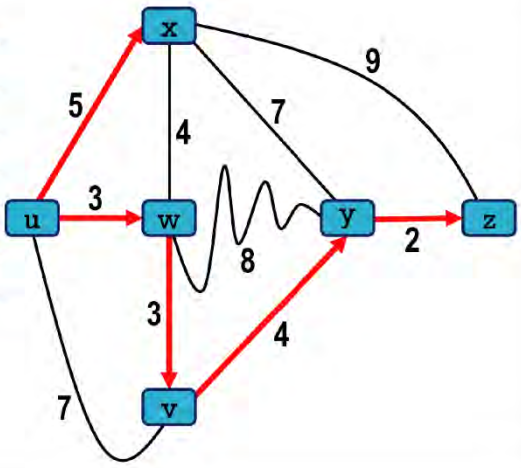


Ilustración 15-18 Ejemplo de búsqueda del camino más corto usando el algoritmo de Dijkstra

Dado un grafo y un vértice origen en el grafo, encuentre los caminos más cortos del origen a todos los vértices en el grafo dado.

La complejidad de tiempo de esta implementación es de O (V*E) si el grafo de entrada está representado usando una lista de adyacencia. Tener en cuenta que Dijkstra no procesa pesos negativos y no detecta ciclos negativos.

Complejidad de tiempo

Mejor caso : O(v*e) **Peor caso :** O(v^2) **Promedio:** O(v*e)

JAVA

```java
// Implementación java del algoritmo de Dijkstra que
// busca el camino más corto de un nodo al resto
/*
 EJEMPLO DE INPUT
 5 9
 1 2 7
 1 4 2
 2 3 1
 2 4 2
 3 5 4
 4 2 3
 4 3 8
 4 5 5
 5 3 5
 1
 */
import java.util.*;

public class DijkstraSP {

    //similar a los defines de C++
    static final int MAX = 10005; //maximo número de vértices
    static final int INF = 1 << 30; //definimos un valor
    //grande que represente la distancia infinita
    //inicial, basta conque sea superior al maximo
    //valor del peso en alguna de las aristas
    //En el caso de java usamos una clase que
    //representara el pair de C++

    static class Node implements Comparable<Node> {

        int first, second;

        Node(int d, int p) { //constructor
            this.first = d;
            this.second = p;
        }

        @Override
        public int compareTo(Node other) { //es necesario
            //definir un comparador para el
```

```
            //correcto funcionamiento del PriorityQueue
            if (second > other.second) {
                return 1;
            }
            if (second == other.second) {
                return 0;
            }
            return -1;
        }
    };

    static Scanner sc = new Scanner(System.in); //para lectura de datos
    static List< List< Node>> ady = new ArrayList< List< Node>>(); //lista de
adyacencia
    static int distancia[] = new int[MAX]; // distancia de vértice inicial
    //vértice con ID = u
    static boolean visitado[] = new boolean[MAX]; //para vértices visitados
    static PriorityQueue< Node> Q = new PriorityQueue<Node>();
    //usamos el comparador definido para
    //que el de menor valor este en el tope
    static int V; //número de vertices
    static int previo[] = new int[MAX]; //para la impresion de caminos

    //función de inicialización
    static void init() {
        for (int i = 0; i <= V; ++i) {
            distancia[i] = INF; //inicializamos todas
            //las distancias con valor infinito
            visitado[i] = false; //inicializamos todos
            //los vértices como no visitados
            previo[i] = -1; //inicializamos el previo
            //del vertice i con -1
        }
    }

    //Paso de relajacion
    static void relajacion(int actual, int adyacente, int peso) {
        //Si la distancia del origen al vertice actual +
        //peso de su arista es menor a la distancia del
        //origen al vertice adyacente
        if (distancia[actual] + peso < distancia[adyacente]) {
            //relajamos el vertice actualizando la distancia
            distancia[adyacente] = distancia[actual] + peso;
            //a su vez actualizamos el vértice previo
            previo[adyacente] = actual;
            //agregamos adyacente a la cola de prioridad
            Q.add(new Node(adyacente, distancia[adyacente]));
        }
    }
    //Impresion del camino más corto desde el vertice inicial y final ingresados
    static void print(int destino) {
        if (previo[destino] != -1) //si aun poseo un vertice previo
        {
            print(previo[destino]); //recursivamente sigo explorando
```

```
        }
        //terminada la recursion imprimo los vertices
        //recorridos
        System.out.printf("%d ", destino);
    }

    static void dijkstra(int inicial) {
        init(); //inicializamos nuestros arreglos
        //Insertamos el vértice inicial en la Cola de Prioridad
        Q.add(new Node(inicial, 0));
        //Este paso es importante, inicializamos la distancia del
        //inicial como 0
        distancia[inicial] = 0;
        int actual, adyacente, peso;
        while (!Q.isEmpty()) { //Mientras cola no este vacia
            //Obtengo de la cola el nodo con menor peso, en un
            //comienzo será el inicial
            actual = Q.element().first;
            Q.remove(); //Sacamos el elemento de la cola
            if (visitado[actual]) {
                continue; //Si el vértice actual ya fue visitado entonces sigo
            }//sacando elementos de la cola
            visitado[actual] = true; //Marco como visitado el vértice actual
            //reviso sus adyacentes del
            //vertice actual
            for (int i = 0; i < ady.get(actual).size(); ++i) {
                adyacente = ady.get(actual).get(i).first; //id del vertice
adyacente
                //peso de la arista que une actual
                //con adyacente ( actual , adyacente )
                peso = ady.get(actual).get(i).second;
                //si el vertice adyacente no fue visitado
                if (!visitado[adyacente]) {
                    //realizamos el paso de relajacion
                    relajacion(actual, adyacente, peso);
                }
            }
        }

        System.out.printf("Distancias más cortas iniciando en vertice %d\n",
inicial);
        for (int i = 1; i <= V; ++i) {
            System.out.printf("Vertice %d , distancia más corta = %d\n", i,
distancia[i]);
        }

        System.out.println("\n**************Impresion de camino más
corto**************");
        System.out.printf("Ingrese vertice destino: ");
        int destino;
        destino = sc.nextInt();
        print(destino);
        System.out.printf("\n");
    }
```

```
    public static void main(String[] args) {
        int E, origen, destino, peso, inicial;

        V = sc.nextInt();
        E = sc.nextInt();
        for (int i = 0; i <= V; ++i) {
            ady.add(new ArrayList<Node>()); //inicializamos lista de
        }//adyacencia
        for (int i = 0; i < E; ++i) {
            origen = sc.nextInt();
            destino = sc.nextInt();
            peso = sc.nextInt();
            ady.get(origen).add(new Node(destino, peso)); //grafo dirigido
            //ady.get( destino ).add( new Node( origen , peso ) ); //no dirigido
        }
        System.out.print("Ingrese el vertice inicial: ");
        inicial = sc.nextInt();
        dijkstra(inicial);
    }
}
```

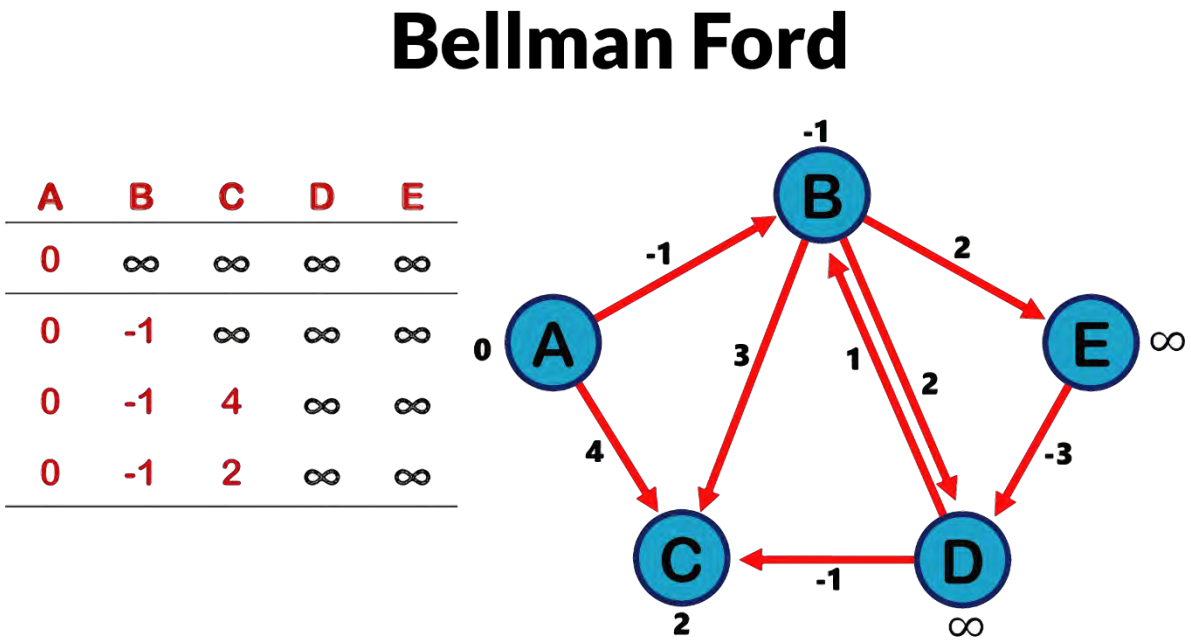
C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
#define MAX 10005
const int INF = 1 << 30;
//---------------//
using namespace std;

struct Node {
    int destino, peso;

    Node(int _destino, int _peso) : destino(_destino), peso(_peso) {
    }

    Node() : destino(-1), peso(-1) {
    }
};

struct State {
    int destino;
    int peso;

    State(int _destino, int _peso) : destino(_destino), peso(_peso) {
    }

    bool operator<(const State &b) const {
        return peso > b.peso;
    }
};
vector <vector<Node> >ady(MAX);
int distancia[MAX];
```

```
bool visited[MAX];
priority_queue<State> Q;
int previo[MAX];

void init(int V) {
    for (int i = 1; i <= V; i++) {
        distancia[i] = INF;
    }
}

void relajacion(int actual, int adyacente, int peso) {
    if (distancia[actual] + peso < distancia[adyacente]) {
        distancia[adyacente] = distancia[actual] + peso;
        previo[adyacente] = actual;
        Q.push(State{adyacente, distancia[adyacente]});
    }
}

void print(int destino) {
    if (previo[destino] != -1) {
        print(previo[destino]);
    }
    cout << destino << " ";
}

void dijkstra(int inicial, int V) {
    init(V);
    Q.push(State{inicial, 0});
    distancia[inicial] = 0;
    int actual, adyacente, peso;
    while (!Q.empty()) {
        actual = Q.top().destino;
        Q.pop();
        if (visited[actual]) {
            continue;
        }
        visited[actual] = true;
        for (int i = 0; i < ady[actual].size(); i++) {
            adyacente = ady[actual][i].destino;
            peso = ady[actual][i].peso;
            if (!visited[adyacente]) {
                relajacion(actual, adyacente, peso);
            }
        }
    }
    cout << "distancia mas corta iniciada desde " << inicial << endl;
    for (int i = 1; i <= V; i++) {
        cout << "vertice " << i << " distancia mas corta = " << distancia[i] <<
endl;
    }
    cout << "impresion del camino mas corto" << endl;
    int destino;
    cin>>destino;
    print(destino);
```

```
    cout << endl;
}

int main() {
    memset(previo, -1, sizeof previo);
    int V, E, origen, destino, peso, inicial;
    cin >> V>>E;
    for (int i = 0; i < E; i++) {
        cin >> origen >> destino>>peso;
        ady[origen].push_back(Node{destino, peso}); //dirigido
        //ady[destino].push_back(Node{origen,peso});// no dirigido
    }
    cout << "Inserte el verice inicial" << endl;
    cin>>inicial;
    dijkstra(inicial, V);
}
```

15.22) Caminos más cortos mediante Bellman-Ford



Guia del programador competitivo

Ilustración 15-19 Ejemplo de búsqueda del camino más corto usando el algoritmo de Bellman-Ford

Dado un grafo y un vértice de origen src en el grafo, encontrar los caminos más cortos desde src a todos los vértices en el grafo dado, el grafo puede contener caminos con pesos negativos. Si hay un ciclo de peso negativo, entonces las distancias más cortas no son calculadas, se reporta el ciclo negativo.

1) Este paso inicializa las distancias desde el origen de todos los vértices como infinito y la distancia al origen en si como 0, crea un array dist[] de tamaño V con todos los valores como infinito excepto dist[src] donde src es el vértice origen.
2) Este paso calcula las distancias más cortas, esto se realiza V-1 veces.
3) si dist[v] > dist[u] + peso del camino uv, entonces actualice dist[] en dist[v] = dist[u] + weight of edge uv
4) Este paso verifica si hay un ciclo negativo en el grafo, se realiza lo siguiente:
5) si dist[v] > dist[u] + peso del camino uv, entonces "El grafo contiene un ciclo negativo"

La idea del paso 3 es, el paso 2 garantiza las distancias más cortas si el grafo no contiene un ciclo de peso negativo, si iteramos a través de todos los caminos una vez más y obtenemos un camino más corto para cualquier vértice, entonces ahí hay un ciclo negativo.

Como en otros problemas de programación dinámica, el algoritmo calcula los caminos más cortos de manera del atrás hacia adelante, primero calcula las distancias más cortas las cuales tienen al menos una arista en el camino, luego calcula los caminos más cortos con al menos dos aristas, y así en adelante, luego de la iesima iteración del ciclo exterior, los caminos más cortos con al menos i aristas son calculados, ahí puede haber un máximo de V-1 aristas en un camino simple, por eso el ciclo externo se corre V-1 veces, la idea es, asumiendo que ahí no hay ciclo negativo, si calculamos los caminos más cortos con al menos i aristas, entonces una interacción sobre todos las aristas garantiza darnos el camino más corto con al menos i+1 artistas.

Ejemplo basado en la imagen anterior:

Dado el vértice origen 0, inicializamos todas las distancias como infinito, excepto la distancia al origen mismo, el número total de vértices en el grafo es 5 y todos los caminos deben ser procesados 4 veces.

Todos las aristas son procesadas en el siguiente orden, (B, E), (D, B), (B, D), (A, B), (A, C), (D, C), (B, C), (E, D). Tenemos las siguientes distancias cuando todos las aristas son procesadas por primera vez, la primera fila

La primera iteración garantiza darnos todos los caminos más cortos los cuales tienen un largo de una arista, obtenemos las distancias siguientes cuando todos las aristas son procesadas por segunda vez.

La segunda iteración garantiza darnos todos los caminos más cortos que sean de 2 aristas más larga, el algoritmo procesa los caminos 2 veces más, las distancias son minimizadas luego de la segunda iteración, y la tercera y cuarta no actualiza distancias...

1) Pesos negativos son encontrados en varias aplicaciones de grafos, por ejemplo en vez de pagar el costo por un camino, podemos obtener ventaja si seguimos el camino.
2) Bellman-Ford trabaja mejor que Dijkstra para sistemas distribuidos, a diferencia de Dijkstra en donde necesitamos encontrar el valor menor de todos los vértices, en Bellman-Ford necesitamos considerar uno por uno.

Complejidad de tiempo

Mejor caso : $O(v^2*e)$ **Peor caso :** $O(v^3)$ **Promedio:** $O(v^2*e)$

JAVA

```
// Implementación java del algoritmo de BellmanFord
// para la busqueda del camino más corto de un vertice al
// resto y con capacidad de detección de ciclo negativo

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class BellmanFordSP {
```

```java
static final int MAX = 105;
static final int INF = 1 << 30;
static int[] previo = new int[MAX];
static int[] distancia = new int[MAX];
static int vertices;
static List<List<Node>> adyacencia = new ArrayList<List<Node>>();
static Scanner sc = new Scanner(System.in);

static void inicializacion() {
    for (int i = 0; i <= vertices; i++) {
        distancia[i] = INF;
        previo[i] = -1;
    }
}

static class Node {
    int first, second;
    public Node(int destino, int peso) {
        this.first = destino;
        this.second = peso;
    }
}

static void print(int destino) {
    if (previo[destino] != -1) {
        print(previo[destino]);
    }
    System.out.printf("%d ", destino);
}

static boolean relajacion(int actual, int adyacente, int peso) {
    if (distancia[actual] + peso < distancia[adyacente]) {
        distancia[adyacente] = distancia[actual] + peso;
        previo[adyacente] = actual;
        return true;
    }
    return false;
}

static void bellmanFord(int inicial) {
    inicializacion();
    distancia[inicial] = 0;
    for (int i = 0; i < vertices - 1; i++) {
        for (int actual = 0; actual < vertices; actual++) {
            for (int j = 0; j < adyacencia.get(actual).size(); j++) {
                int adyacente = adyacencia.get(actual).get(j).first;
                int peso = adyacencia.get(actual).get(j).second;
                relajacion(actual, adyacente, peso);
            }
        }
    }
    for (int actual = 0; actual < vertices; actual++) {
        for (int j = 0; j < adyacencia.get(actual).size(); j++) {
            int adyacente = adyacencia.get(actual).get(j).first;
```

```
                int peso = adyacencia.get(actual).get(j).second;
                if (relajacion(actual, adyacente, peso)) {
                    System.out.println("Existe ciclo negativo");
                    return;
                }
            }
        }
        System.out.println("No existe ciclo negativo");
        System.out.printf("Distancias más corteas iniciando en el nodo %d\n",
inicial);
        for (int i = 0; i < vertices; i++) {
            System.out.printf("Nodo %d , distancia más corta = %d\n", i,
distancia[i]);
        }
        System.out.println("\n ________Camino más corto____");
        System.out.println("Ingrese vertice destino: ");
        int destino = sc.nextInt();
        print(destino);
        System.out.println("");
    }

    public static void main(String[] args) {
        int E, origen, destino, peso, inicial;
        vertices = sc.nextInt();
        E = sc.nextInt();
        for (int i = 0; i < 10; i++) {
            adyacencia.add(new ArrayList<>());
        }
        for (int i = 0; i < E; i++) {
            origen = sc.nextInt();
            destino = sc.nextInt();
            peso = sc.nextInt();
            adyacencia.get(origen).add(new Node(destino, peso));
        }
        System.out.printf("Ingrese el nodo inicial: ");
        inicial = sc.nextInt();
        bellmanFord(inicial);
    }
}
```

C++

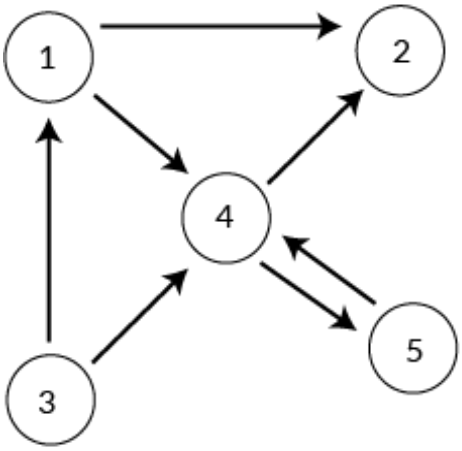
```
#include<bits/stdc++.h>
#include<cstdlib>
//---------------//
#define MAX 105
using namespace std;
int previo[MAX];
int distancia[MAX];

void init(int vertices) {
    for (int i = 0; i <= vertices; i++) {
        distancia[i] = INT_MAX;
        previo[i] = -1;
    }
```

```
}

void printPath(int destino) {
    if (previo[destino] != -1)printPath(previo[destino]);
    printf("%d ", destino);
}

bool relajacion(int actual, int adyacente, int peso) {
    if (distancia[actual] + peso < distancia[adyacente]) {
        distancia[adyacente] = distancia[actual] + peso;
        previo[adyacente] = actual;
        return true;
    }
    return false;
}
vector<vector<pair <int, int> > >ady(MAX);

void BellmanFord(int inicial, int vertices) {
    init(vertices);
    distancia[inicial] = 0;
    for (int i = 0; i < vertices; i++) {
        for (int actual = 0; actual < vertices; actual++) {
            for (int j = 0; j < ady[actual].size(); j++) {
                int adyacente = ady[actual][j].first;
                int peso = ady[actual][j].second;
                relajacion(actual, adyacente, peso);
            }
        }
    }
    for (int actual = 0; actual < vertices; actual++) {
        for (int j = 0; j < ady[actual].size(); j++) {
            int adyacente = ady[actual][j].first;
            int peso = ady[actual][j].second;
            if (relajacion(actual, adyacente, peso)) {
                printf("Existe ciclo negativo\n");
                return;
            }
        }
    }
    printf("No existe ciclo negativo\ndistancias mas corta desde el nodo %d",
inicial);
    for (int i = 0; i <= vertices; i++) {
        printf("nodo %d distancia mas corta = %d\n", i, distancia[i]);
    }
    printf("camino mas corto hasta un destino\n");
    int destino;
    cin>>destino;
    printPath(destino);
    cout << endl;
}

int main() {
    int E, origen, destino, peso, inicial, vertices;
    cin >> vertices>>E;
```

```
    for (int i = 0; i < E; i++) {
        cin >> origen >> destino>>peso;
        ady[origen].push_back(make_pair(destino, peso)); //dirigido
        //ady[destino].push_back(make_pair{origen,peso});//NO dirigido
    }
    printf("Nodo inicial : ");
    cin>>inicial;
    BellmanFord(inicial, vertices);
}
```

15.23) Caminos más cortos entre todos los nodos mediante Floyd-Warshall

Floyd warshall



	1	2	3	4	5
1	False	True	False	True	False
2	False	True	False	False	False
3	True	False	False	True	False
4	False	True	False	False	True
5	False	False	False	True	False

Guía del programador competitivo

Ilustración 15-20 Matriz de alcance entre nodos del grafo dado

El algoritmo de Floyd Warshall se usa para la resolución de todos los caminos más cortos de todos los pares, el problema es encontrar las distancias más pequeñas entre cada par de vértices dado un grafo de caminos con pesos.

Inicializamos la matriz de la solución igual que la matriz de entrada del grafo en el primer paso, luego actualizamos la matriz de solución considerando todos los vértices en un vértice intermedio, la idea es uno por uno tomar todos los vértices y actualizar todos los caminos más cortos los cuales incluyen el vértice seleccionado como el vértice intermedio en el camino más corto. Cuando tomamos el vértice de número k como un vértice intermedio ya estamos considerando vértices {0,1,2…k-1} como intermedios, para cada par (i,j) del origen al destino respetivamente existen dos posibles casos.

1) k no es un vértice intermedio en el camino más corto de i a j, mantenemos el valor de dist[i][j] como esta.
2) K es un vértice intermedio en el camino más corto de i a j, actualizamos el valor de dist[i][j] como dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

La complejidad de tiempo es: O(V^3) o O(N^3).

El programa solo imprime las distancias más cortas, podemos modificar la solución para imprimir el camino más corto ordenando la información del predecesor en una matriz separada.

El valor de INF puede ser tomado de INT_MAX o Integer.MAX_VALUE, lo que nos permite manejar el los valores máximos posibles.

Complejidad de tiempo

Mejor caso : $O(v^3$ **Peor caso :** $O(v^3)$ **Promedio:** $O(v^3)$

JAVA

```java
// Programa java que busca todos los caminos más cortos
// en un grafo

import java.util.ArrayList;
import java.util.Scanner;

public class Main {
```

```
static int INF = 9999;
static int V = 0;
//grafo en matriz fija
static int[][] graph;
//matriz que almacena el camino recorrido
static int[][] next;

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    V = sc.nextInt();
    graph = new int[V][V];
    next = new int[V][V];
    int E = sc.nextInt();
    //Inicializa todo en infinito
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            graph[i][j] = INF;
        }
    }
    // De un nodo al mismo nodo es 0
    for (int i = 0; i < V; i++) {
        graph[i][i] = 0;
    }
    for (int i = 0; i < E; i++) {
        graph[sc.nextInt()][sc.nextInt()] = sc.nextInt();
    }
    floydWarshall(graph);
    ArrayList<Integer> path;
    System.out.println("El camino mas corto desde 0 a 3: ");
    //Construit camino
    path = constructPath(0, 3);
    //imprimir camino
    printPath(path);
}

static void floydWarshall(int graph[][]) {
    int dist[][] = new int[V][V];
    int i, j, k;
    /*Inicialice la matriz de solución igual que
        la matriz del gráfico de entrada. O podemos
        decir que los valores iniciales de las distancias
        más cortas se basan en las rutas más cortas,
        considerando que no hay vértice intermedio. */
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
            // Si no hay camino entre i y j
            if (graph[i][j] == INF) {
                next[i][j] = -1;
            } else {
                next[i][j] = j;
            }
        }
    }
```

```
        /*Agregue todos los vértices uno por uno al conjunto
        de vértices intermedios.
        ---> Antes del inicio de una iteración, tenemos
        distancias más cortas entre todos los pares de vértices,
        de modo que las distancias más cortas consideran solo
        los vértices en el conjunto {0, 1, 2, .. k-1} como
        vértices intermedios.
        ----> Después del final de una iteración, el vértice
        número k se agrega al conjunto de vértices intermedios
        y el conjunto se convierte en {0, 1, 2, ... k} */
    for (k = 0; k < V; k++) {
        //Toma todos los vertices como inicio uno por uno
        for (i = 0; i < V; i++) {
            /* Toma todos los vertices como destino del
                incio del origen seleccionado*/
            for (j = 0; j < V; j++) {
                // Si vertice k esa en el camino más corto
                // desde i a j, actualiza el valor de dist[i][j]
                // No podemos viajar por un camino si no existe
                if (dist[i][k] == INF|| dist[k][j] == INF) {
                    continue;
                }
                if (dist[i][j] > dist[i][k]+ dist[k][j]) {
                    dist[i][j] = dist[i][k]+ dist[k][j];
                    next[i][j] = next[i][k];
                }
            }
        }
    }
    //Imprime la matriz de distancias cortas
    printSolution(dist);
}

static ArrayList<Integer> constructPath(int u, int v) {
    // si no hay camino devuelve una lista vacia
    if (next[u][v] == -1) {
        return new ArrayList<>();
    }
    // Almacenando el camino en el vector
    ArrayList<Integer> path = new ArrayList<>();
    path.add(u);
    while (u != v) {
        u = next[u][v];
        path.add(u);
    }
    return path;
}

static void printPath(ArrayList<Integer> path) {
    int n = path.size();
    for (int i = 0; i < n - 1; i++) {
        System.out.print(path.get(i) + " -> ");
    }
```

```
        System.out.println(path.get(n - 1) + " -> ");
    }

    static void printSolution(int dist[][]) {
        System.out.println("La siguiente matriz muestra las distancias "
                + "más cortas entre cada par de vertices");
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                if (dist[i][j] == Main.INF) {
                    System.out.print("INF ");
                } else {
                    System.out.print(dist[i][j] + " ");
                }
            }
            System.out.println();
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
//--------------//
#define INF 9999
#define MAX 256
using namespace std;
const int V = 4;
int next[V][V];

void printPath(vector<int> path) {
    int n = path.size() - 1;
    for (int i = 0; i < n - 1; i++) {
        cout << path[i] << " -> ";
    }
    cout << path[n - 1] << " -> " << endl;
}

int constructPath(int u, int v) {
    if (next[u][v] == -1) {
        return 0;
    }
    vector<int>path;
    path.push_back(u);
    while (u != v) {
        u = next[u][v];
        path.push_back(u);
    }
    printPath(path);
    return 1;
}

void printSolution(int dist[V][V]) {
    cout << "La siguiente matriz muestra las distancias mas cortas entre cada
par de vertices\n";
```

```
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            if (dist[i][j] == INF) {
                cout << "INF \t";
            } else {
                cout << dist[i][j] << "\t";
            }
        }
        cout << endl;
    }
}

void floydWarshall(int graph[V][V]) {
    int dist[V][V];
    int i, j, k;
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            dist[i][j] = graph[i][j];
            if (graph[i][j] == INF) {
                next[i][j] = -1;
            } else {
                next[i][j] = j;
            }
        }
    }
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] == INF || dist[k][j] == INF) {
                    continue;
                }
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                    next[i][j] = next[i][k];
                }
            }
        }
    }
    printSolution(dist);
}

int main() {
    int E, a, b, c;
    cin>>E;
    int graph[V][V];
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            graph[i][j] = INF;
        }
    }
    for (int i = 0; i < V; i++) {
        graph[i][i] = 0;
    }
    for (int i = 0; i < E; i++) {
```

```
        cin >> a >> b>>c;
        graph[a][b] = c;
    }
    floydWarshall(graph);
    constructPath(0, 3);
}
```

15.24) Caminos más cortos en un grafo binario

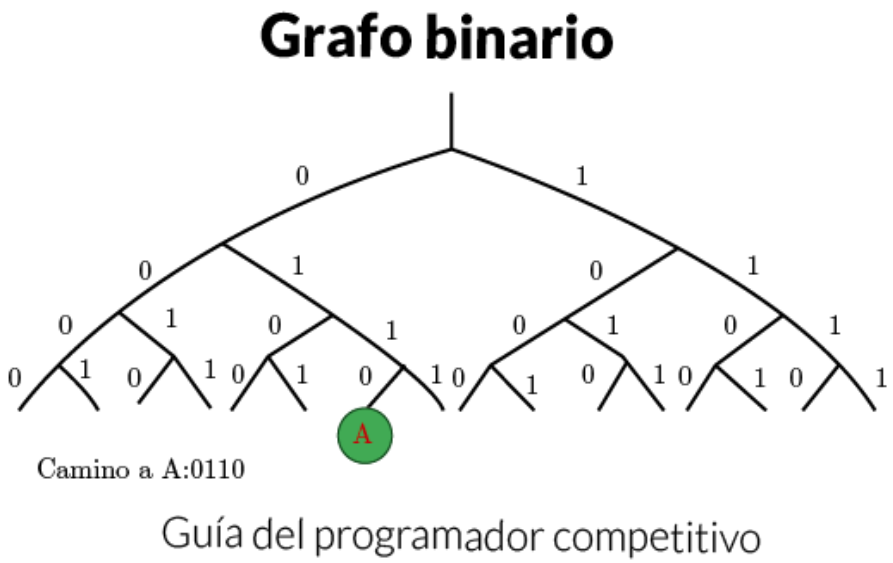


Ilustración 15-21 Camino en un grafo binario

Dado un grafo donde cada camino tiene un peso de 0 o 1, un vértice inicial también es dado, encuentre el camino más corto desde el origen hasta cualquier otro vértice.

En el BFS normal de un grafo, todos los caminos son de igual peso, pero en BFS 0-1 ALGUNOS caminos pueden tener peso 0 y algunos 1, en este no podemos usar un array de booleanos para marcar los nodos visitados pero en cada paso podemos verificar la condición de distancia optima, usamos una cola de doble fin para almacenar el nodo, mientras realizamos BFS si un camino es encontrado con peso 0 el nodo es empujado al frente de la doble cola y si el camino tiene peso 1 es empujado atrás en la doble cola.

Esta aproximación es similar a Dijkstra en que si la distancia más corta a un nodo es relajada por el nodo anterior entonces solo será empujada en la cola.

La idea de arriba trabaja en todos los casos, cuando sacamos un vértice, es el vértice de mínimo peso de todos los vértices restantes, si hay un vértice de peso 0 adyacente a el entonces este adyacente tiene la misma distancia, si hay un adyacente de peso 1, entonces este adyacente tiene la máxima distancia de todos los vértices en la cola doble, (Porque todos los otros vértices son adyacentes de actual eliminado vértice o adyacente de los anteriores eliminados).

Este problema puede ser resuelto con Dijkstra pero el tiempo de complejidad seria de O(E + V Log V) donde por BFS puede ser de O(V+E).

Complejidad de tiempo

Mejor caso : O(v+e) **Peor caso :** O(v+e) **Promedio:** O(v+e)

JAVA

```java
//Programa java que implementa el camino más corto
// en un grafo binario

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;

public class BinaryGraphShortestPath {
    //Número de vertices
    static int V = 9;
    // Lista de listas que almacena los caminos
    static ArrayList<ArrayList<node>> edges = new ArrayList<ArrayList<node>>();
    public static void main(String[] args) {
        for (int i = 0; i < V; i++) {
            edges.add(new ArrayList<>());
        }
        addEdge(0, 1, 0);
        addEdge(0, 7, 1);
        addEdge(1, 7, 1);
        addEdge(1, 2, 1);
        addEdge(2, 3, 0);
        addEdge(2, 5, 0);
        addEdge(2, 8, 1);
        addEdge(3, 4, 1);
        addEdge(3, 5, 1);
        addEdge(4, 5, 1);
        addEdge(5, 6, 1);
        addEdge(6, 7, 1);
```

```
        addEdge(7, 8, 1);
        int src = 0;//Nodo inicial
        zeroOneBFS(src);
    }

    static void zeroOneBFS(int src) {
        // Inicializa las distancias desde el nodo inicial
        int dist[] = new int[V];
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        //cola doble para el BFS.
        Deque<Integer> Q = new ArrayDeque<>();
        dist[src] = 0;
        Q.add(src);
        while (!Q.isEmpty()) {
            int v = Q.getFirst();
            Q.removeFirst();
            for (int i = 0; i < edges.get(v).size(); i++) {
                //Busca la distancia optima
                if (dist[edges.get(v).get(i).to] > dist[v] +
edges.get(v).get(i).weight) {
                    dist[edges.get(v).get(i).to] = dist[v] +
edges.get(v).get(i).weight;
                    /* pone peso caminos de peso 0 al frente y 1 atras para
                    que los vertices puedan ser procesados en orden ascendente
                    por sus pesos*/
                    if (edges.get(v).get(i).weight == 0) {
                        Q.addFirst(edges.get(v).get(i).to);
                    } else {
                        Q.addLast(edges.get(v).get(i).to);
                    }
                }
            }
        }
        //imprimiendo los caminos más cortos
        for (int i = 0; i < V; i++) {
            System.out.print(dist[i] + " ");
        }
        System.out.println("");
    }
    static void addEdge(int u, int v, int wt) {
        edges.get(u).add(new node(v, wt));
        edges.get(v).add(new node(u, wt));
    }
    // una estructura que representa los caminos
    static class node {
        // Dos variables, una denora el nodo
        // destino y otra el peso
        int to, weight;
        public node(int to, int weight) {
            this.to = to;
            this.weight = weight;
        }
```

```
    }
}
```

15.25) Grafo bipartito

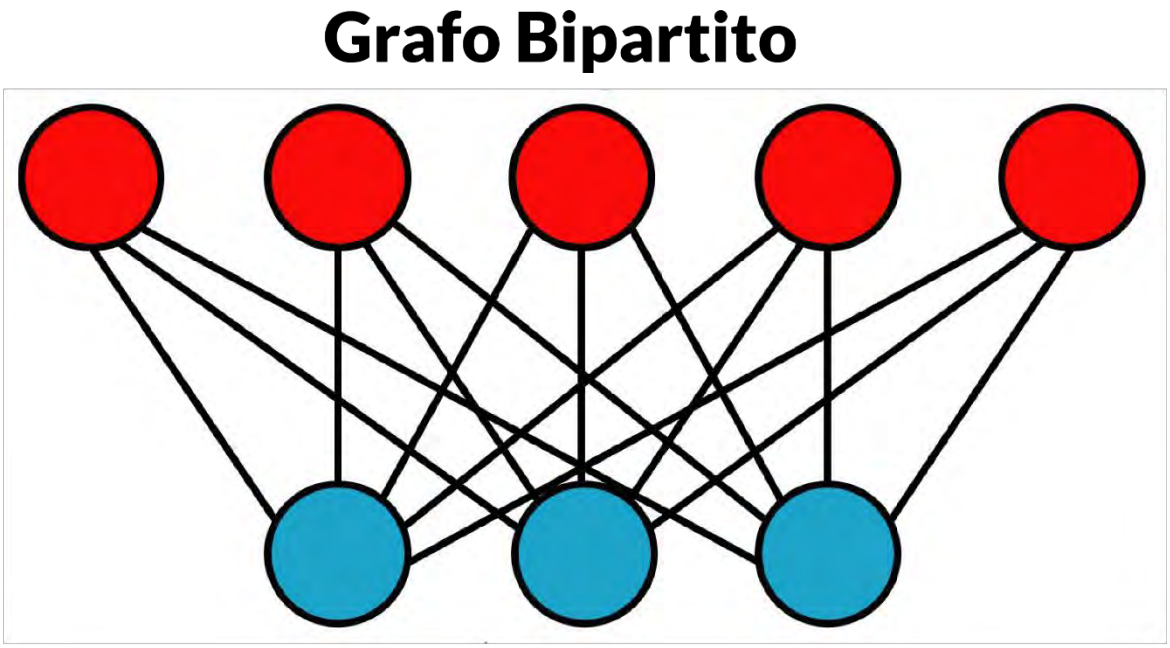


Ilustración 15-22 Ejemplo de coloración de un grafo bipartito

Un grafo bipartito es un grafo cuyos vértices pueden ser divididos en dos sets independientes, U y V en donde cada arista (u, v) conecta un vértice de U a V o un vértice de V a U. En otras palabras para cada arista (u,v) u pertenece a U y v pertenece a V, o viceversa, también podemos decir que no hay aristas que conecte vértices del mismo set.

Un grafo bipartito es posible si el coloramiento del grafo es posible usando dos colores los cuales los vértices de un set son coloreados del mismo color.

Una aproximación para verificar si un grafo es bipartito es verificar si el grafo es coloreable o no usando backtracking.

El siguiente es un algoritmo simple de verificación de bipartito usando BFS.

1) Asigna el color ROJO al vértice origen (Poniéndolo en el set U)

2) Colorea todos los vecinos con color AZUL (poniéndolos en el set V)
3) Colorear todos los vecinos del vecino anterior de color ROJO (Poniéndolos en U)
4) De esta forma se asigna color a todos los vértices los cuales satisfacen todas las restricciones del coloramiento.
5) Mientras asignamos colores, si encontramos un vecino que esta coloreado del mismo color del actual vértice entonces el grafo no puede ser coloreado, es decir no es bipartito.

El algoritmo solo funciona si el grafo es fuertemente conectado, en el código siempre empezamos desde el origen 0 y asumimos que los vértices son visitados desde el, una observación importante es un grafo sin aristas es también bipartito,
La complejidad de tiempo de esta aproximación es la misma del BFS, O(V^2) si el grafo es representado con listas de adyacencia, se convierte en O(V+E).

Complejidad de tiempo

Mejor caso : O(v+e) **Peor caso :** O(v+e) **Promedio:** O(v+e)

JAVA

```java
//Programa java que encuentra si un grafo es
//bipartito o no

import java.util.*;

public class BipartiteGraph {

    final static int V = 4;

    static boolean isBipartite(int G[][], int src) {
        /*Crear una matriz de colores para almacenar los
        colores asignados a todas las verificaciones.
        El número de vértice se utiliza como índice en
        esta matriz. El valor '-1' de colorArr [i] se
        usa para indicar que no se asigna ningún color
        al vértice 'i'. El valor 1 se utiliza para indicar
        que el primer color está asignado y el valor 0
        indica que el segundo color está asignado.*/
        int colorArr[] = new int[V];
        for (int i = 0; i < V; ++i) {
            colorArr[i] = -1;
        }
        //Asigna primer color al origen
```

```
        colorArr[src] = 1;
        //Crea una cola de número de vertices
        // y encola el vertice origen
        LinkedList<Integer> q = new LinkedList<>();
        q.add(src);
        while (!q.isEmpty()) {
            // Descola un vertice de la cola
            int u = q.poll();
            //Retorna falso si hay un autociclo
            if (G[u][u] == 1) {
                return false;
            }
            //Encuentra todos los vertoces adyacentes sin color
            for (int v = 0; v < V; ++v) {
                // Un camino de u a v existe
                // y destino v no esta coloreado
                if (G[u][v] == 1 && colorArr[v] == -1) {
                    //Asigna color alternativo para esta adyacencia
                    colorArr[v] = 1 - colorArr[u];
                    q.add(v);
                } /*Un camino de u a v existe y el destino
                esta del mismo color que u*/
                else if (G[u][v] == 1 && colorArr[v] == colorArr[u]) {
                    return false;
                }
            }
        }
        // Si llegamos aqui, todos los vertoces adyacentes pueden
        // ser coloreados con color alternativo
        return true;
    }

    public static void main(String[] args) {
        int G[][] = {{0, 1, 0, 1},
        {1, 0, 1, 0},
        {0, 1, 0, 1},
        {1, 0, 1, 0}
        };
        if (isBipartite(G, 0)) {
            System.out.println("Si");
        } else {
            System.out.println("No");
        }
    }
}
```

C++

```
#include<bits/stdc++.h>
#include<cstdlib>
using namespace std;
const int V = 4;

bool isBipartite(int G[V][V], int src) {
    int colorArr[V];
```

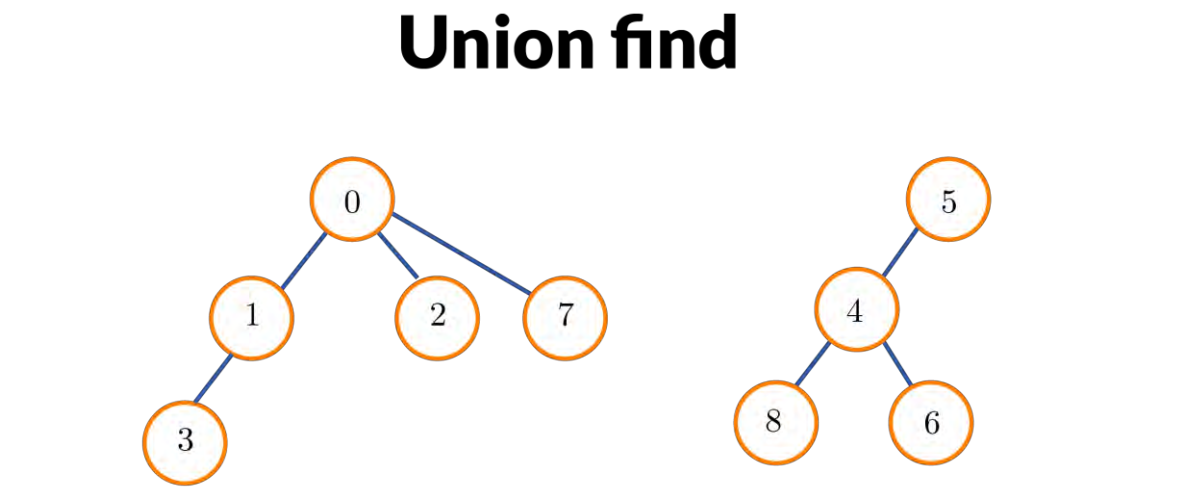
```
    memset(colorArr, -1, sizeof colorArr);
    colorArr[src] = 1;
    vector<int>q;
    q.push_back(src);
    while (!q.empty()) {
        int u = q.front();
        q.erase(q.begin());
        if (G[u][u] == 1) {
            return false;
        }
        for (int v = 0; v < V; v++) {
            if (G[u][v] == 1 && colorArr[v] == 1) {
                colorArr[v] = 1 - colorArr[u];
                q.push_back(v);
            } else if (G[u][v] == 1 && colorArr[v] == colorArr[u]) {
                return false;
            }
        }
    }
    return true;
}

int main() {
    int G[V][V] = {
        {0, 1, 0, 1},
        {1, 0, 1, 0},
        {0, 1, 0, 1},
        {1, 0, 1, 0}};
    if (isBipartite(G, 0)) {
        cout << "Es bipartito" << endl;
    } else {
        cout << "NO Es bipartito" << endl;
    }
}
```

PYTHON

```
V = 4

def isBipartite(G, src):
    global V
    colorarr = [-1 for x in range(V)]
    colorarr[src] = 1
    q = []
    q.append(src)
    while not len(q) == 0:
        u = q.pop()
        if G[u][u] == 1:
            return False
    for v in range(V):
        if G[u][v] == 1 and colorarr[v] == -1:
            colorarr[v] = 1 - colorarr[u]
            q.append(v)
        elif G[u][v] and colorarr[v] == colorarr[u]:
            return False
```

```
    return True

G = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]]
if isBipartite(G, 0):
    print("Es bipartitio")
else:
    print("No es bipartitio")
```

15.26) Union Find



Vértices	0	1	2	3	4	5	6	7	8
Padre	0	0	0	1	5	5	4	0	4

Guía del programador competitivo

Ilustración 15-23 Padres e hijos en un grafo

Una estructura de datos para conjuntos disjuntos, es una estructura de datos que mantiene un conjunto de elementos particionados en un número de conjuntos disjuntos no se solapan los conjuntos). Un algoritmo Unión-Buscar es un algoritmo que realiza dos importantes operaciones en esta estructura de datos:

Buscar: Determina a cual subconjunto pertenece un elemento. Esta operación puede usarse para verificar si dos elementos están en el mismo conjunto.

Union: Une dos subconjuntos en uno solo.

La otra operación importante CrearConjunto es generalmente trivial, esta crea un conjunto con un elemento dado. Con estas tres operaciones, muchos problemas prácticos de particionamiento pueden ser resueltos.

Con el fin de definir estas operaciones más precisamente, es necesario representar los conjuntos de alguna manera. Una aproximación común es seleccionar un elemento fijo de cada conjunto, llamado el representativo, para representar el conjunto como un todo. Entonces Buscar(x) retorna el elemento representativo del conjunto al cual x pertenece, y Unión toma como argumento dos elementos representativos de dos conjuntos respectivamente.

Complejidad de tiempo

Mejor caso : O(v+e) **Peor caso :** O(v+e) **Promedio:** O(v+e)

JAVA

```
//Programa java que usa el algoritmo UNION FIND
// para verificar componentes conexas
/*
INPUT
9 7
2 0
7 0
3 1
1 0
6 4
8 5
4 5
 */
import java.util.*;

public class UnionFind {

    static final int MAX = 10005; //maximo número de vértices
    static int padre[] = new int[MAX]; //Este arreglo contiene el padre del i-
esimo nodo
    static int rango[] = new int[MAX]; //profundidad de cada vértice

    //Método de inicialización
    static void MakeSet(int n) {
```

```
        for (int i = 0; i < n; ++i) {
            padre[i] = i; //Inicialmente el padre de cada vértice es el mismo
vértice
            rango[i] = 0; //Altura o rango de cada vértice es 0
        }
    }
    //Método para encontrar la raiz del vértice actual X
    static int Find(int x) {
        if (x == padre[x]) { //Si estoy en la raiz
            return x; //Retorno la raiz
        } //else return Find( padre[ x ] ); //De otro modo busco el padre del
vértice actual, hasta llegar a la raiz.
        else {
            return padre[x] = Find(padre[x]); //Compresion de caminos
        }
    }
    //Método para unir 2 componentes conexas
    static void Union(int x, int y) {
        int xRoot = Find(x); //Obtengo la raiz de la componente del vértice X
        int yRoot = Find(y); //Obtengo la raiz de la componente del vértice Y
        padre[xRoot] = yRoot; //Mezclo ambos arboles o conjuntos, actualizando
su padre de alguno de ellos como la raiz de otro
    }

    //Método para unir 2 componentes conexas usando sus alturas (rangos)
    static void UnionbyRank(int x, int y) {
        int xRoot = Find(x); //Obtengo la raiz de la componente del vértice X
        int yRoot = Find(y); //Obtengo la raiz de la componente del vértice Y
        if (rango[xRoot] > rango[yRoot]) { //en este caso la altura de la
componente del vértice X es
            //mayor que la altura de la componente del vértice Y.
            padre[yRoot] = xRoot; //el padre de ambas componentes será el de
mayor altura
        } else { //en este caso la altura de la componente del vértice Y es
mayor o igual que la de X
            padre[xRoot] = yRoot; //el padre de ambas componentes será el de
mayor altura
            if (rango[xRoot] == rango[yRoot]) { //si poseen la misma altura
                rango[yRoot]++; //incremento el rango de la nueva raíz
            }
        }
    }

    static int root[] = new int[MAX]; //tendra las raices de las componentes
conexas luego de aplicar el método
    static int numComponentes; //variable para el número total de componentes
conexas
    //Método para obtener el número de componentes conexas luego de realizar las
conexiones respectivas

    static int getNumberConnectedComponents(int n) {
        numComponentes = 0;
        for (int i = 0; i < n; ++i) {
```

```
            if (padre[i] == i) { //Si el padre del vértice i es el mismo vértice
entonces es raíz
                //if( Find( i ) == i ){ //podemos usamos find para el mismo
proposito y
                //para que se realice compresion de caminos
                root[numComponentes++] = i; //almaceno la raiz de cada nueva
componente
                // numComponentes++;
            }
        }
        return numComponentes;
    }

    static int numVertices[] = new int[MAX]; //almacenara la cantidad de
vértices para la i-esima raiz.
    //Método para obtener la raiz y el número de vértices de cada componente
conexa
    //será necesario primero tener la cantidad de componentes conexas
    //podemos llamar 1ero al metodo getNumberConnectedComponents o incluir
porcion de su codigo en este

    static void getNumberNodes(int n) {
        Arrays.fill(numVertices, 0); //inicializo mi contador de vértices
        for (int i = 0; i < n; ++i) {
            numVertices[Find(i)]++; //incremento la raíz del vértice i
        }
        for (int i = 0; i < numComponentes; ++i) {
            System.out.printf("Componente %d: Raiz = %d , Nro nodos = %d.\n", i
+ 1, root[i], numVertices[root[i]]);
        }
    }

    //Método que me determina si 2 vértices estan o no en la misma componente
conexa
    static boolean sameComponent(int x, int y) {
        if (Find(x) == Find(y)) {
            return true; //si poseen la misma raíz
        }
        return false;
    }

    public static void main(String[] args) {
        int V, E, origen, destino;
        Scanner sc = new Scanner(System.in);
        V = sc.nextInt();
        E = sc.nextInt(); //tengamos número de vertices y aristas
        MakeSet(V); //inicializamos los conjuntos
        for (int i = 0; i < E; ++i) {
            origen = sc.nextInt();
            destino = sc.nextInt();
            UnionbyRank(origen, destino); //union de elementos
        }
        System.out.printf("El número de componentes conexas es: %d\n",
getNumberConnectedComponents(V));
```

```
            getNumberNodes(V);
        }
    }
```

15.27) Árbol de expansión mínima de Kruskal

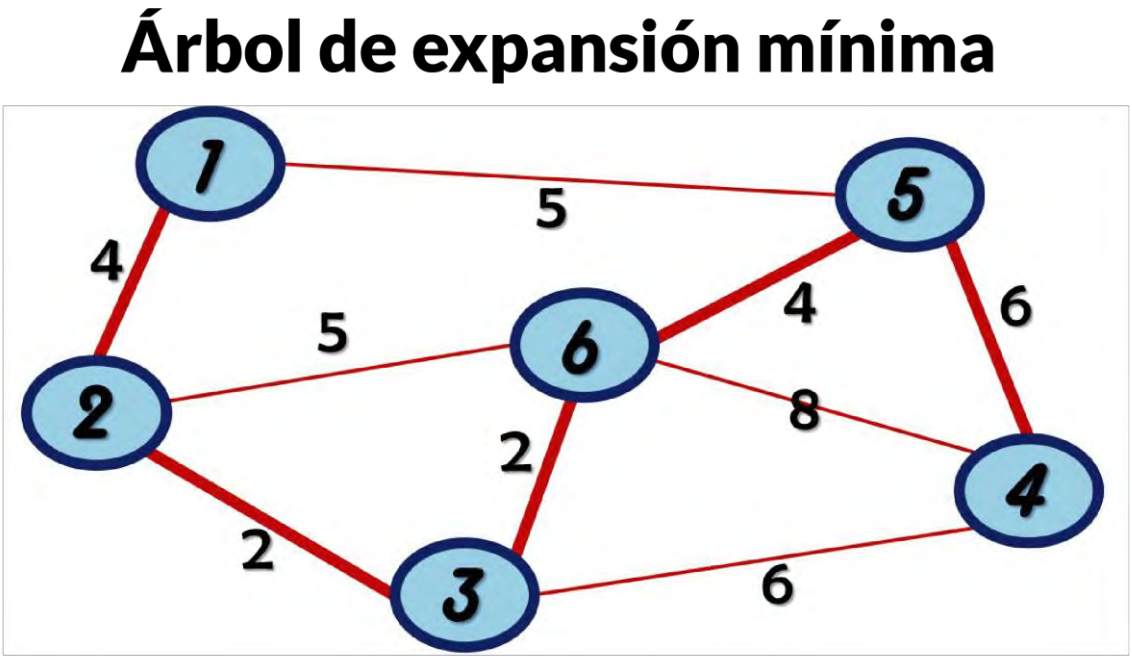


Ilustración 15-24 Árbol de expansión mínima en un grafo

Dado un grafo conexo y no dirigido, un árbol de expansión del grafo es el subgrafo que es el árbol que conecta todos los vértices juntos, un solo grafo puede tener muchos árboles de expansión diferentes, un árbol de expansión mínima (MST) o árbol de expansión de peso mínimo para un grafo ponderado, conexo y no dirigido es el árbol de expansión con el peso menor o igual a cualquier otro árbol de expansión. El peso de un árbol de expansión es la suma de los pesos dados en cada camino del árbol de expansión.

¿Cuantos caminos tiene un árbol de expansión mínima?

Un MST tiene (V-1) caminos donde V es el número de vértices en el grafo dado.

Los siguientes son los pasos para encontrar el MST con el algoritmo de Kruskal.

1) ordene todos los caminos en orden incremental dado su peso.
2) Tome el camino más pequeño, verifique si forma un ciclo con el árbol formado hasta ahora, si el ciclo no es formado, incluya este camino, si no descártelo
3) Repita el paso 2 hasta que haya (V-1) caminos en el árbol de expansión.

El paso 2 usa el algoritmo Union-Find para detectar ciclos.

El algoritmo es un algoritmo voraz, la voracidad escogida es tomar el camino de tamaño más pequeño lo que no causa un ciclo en el MST construido hasta ese punto

La complejidad de tiempo de este algoritmo es de O(ElogE) or O(ElogV). Ordenar los caminos toma O(ELogE) tiempo, luego de ordenar iteramos a través de todos los caminos y aplicamos el algoritmo Union-Find, Las operaciones de unir y encontrar puede tomar al menos O(LogV) tiempo, entonces la complejidad completa es O(ELogE + ELogV) tiempo. El valor de W puede ser al menos O (V^2) entonces O (LogV) son O (LogE) iguales.

Complejidad de tiempo

Mejor caso : O(eLog(e)+ eLog(v)) **Peor caso :** O(eLog(e)+ eLog(v))

Promedio: O(eLog(e)+ eLog(v))

JAVA

```
//Implementación java de busqueda del
// arbol de expansión minima usando el algoritmo de Kruskal
/*
 EJEMPLO DE INPUT
 9 14
 1 2 4
 1 8 9
 2 3 9
 2 8 11
 3 4 7
 3 9 2
 3 6 4
 4 5 10
 4 6 15
 5 6 11
 6 7 2
 7 8 1
 7 9 6
 8 9 7
```

```
 EJEMPLO VERIFICACION DE MST
9 11
 1 2 4
 1 8 9
 2 3 9
 2 8 11
 3 9 2
 7 8 1
 7 9 6
 8 9 7
 4 5 10
 4 6 15
 5 6 11
 */
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;

public class KruskalMST {

    static final int MAX = 1005; //maximo número de vértices
    //UNION-FIND
    static int padre[] = new int[MAX]; //Este arreglo contiene el padre del i-
esimo nodo
    //Método de inicialización
    static void MakeSet(int n) {
        for (int i = 1; i <= n; ++i) {
            padre[i] = i;
        }
    }
    //Método para encontrar la raiz del vértice actual X
    static int Find(int x) {
        return (x == padre[x]) ? x : (padre[x] = Find(padre[x]));
    }
    //Método para unir 2 componentes conexas
    static void Union(int x, int y) {
        padre[Find(x)] = Find(y);
    }
    //Método que me determina si 2 vértices estan o no en la misma componente
conexa
    static boolean sameComponent(int x, int y) {
        if (Find(x) == Find(y)) {
            return true;
        }
        return false;
    }
    ///FIN UNION-FIND
    static int V, E; //número de vertices y aristas
    //Estructura arista( edge )

    static class Edge implements Comparator<Edge> {
        int origen; //Vértice origen
        int destino; //Vértice destino
```

```
        int peso; //Peso entre el vértice origen y destino
        Edge() {
        }
        //Comparador por peso, me servira al momento de ordenar lo realizara en
orden ascendente
        //Ordenar de forma descendente para obtener el arbol de expansion maxima
        @Override
        public int compare(Edge e1, Edge e2) {
            //return e2.peso - e1.peso; //Arbol de expansion maxima
            return e1.peso - e2.peso; //Arbol de expansion minima
        }
    };

    static Edge arista[] = new Edge[MAX]; //Arreglo de aristas para el uso en
kruskal
    static Edge MST[] = new Edge[MAX]; //Arreglo de aristas del MST encontrado
    static void KruskalMST() {
        int origen, destino, peso;
        int total = 0; //Peso total del MST
        int numAristas = 0; //Número de Aristas del MST
        MakeSet(V); //Inicializamos cada componente
        Arrays.sort(arista, 0, E, new Edge()); //Ordenamos las aristas por su
comparador
        for (int i = 0; i < E; ++i) {
            origen = arista[i].origen; //Vértice origen de la arista actual
            destino = arista[i].destino; //Vértice destino de la arista actual
            peso = arista[i].peso; //Peso de la arista actual
            //Verificamos si estan o no en la misma componente conexa
            if (!sameComponent(origen, destino)) { //Evito ciclos
                total += peso; //Incremento el peso total del MST
                MST[numAristas++] = arista[i]; //Agrego al MST la arista actual
                Union(origen, destino); //Union de ambas componentes en una sola
            }
        }

        //Si el MST encontrado no posee todos los vértices mostramos mensaje de
error
        //Para saber si contiene o no todos los vértices basta con que el número
        //de aristas sea igual al número de vertices - 1
        if (V - 1 != numAristas) {
            System.out.println("No existe MST valido para el grafo ingresado, el
grafo debe ser conexo.");
            return;
        }
        System.out.println("-----El MST encontrado contiene las siguientes
aristas-----");
        for (int i = 0; i < numAristas; ++i) {
            System.out.printf("( %d , %d ) : %d\n", MST[i].origen,
MST[i].destino, MST[i].peso);
        }
        System.out.printf("El costo minimo de todas las aristas del MST es :
%d\n", total);
    }
```

```
    //( vertice u , vertice v ) : peso
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); //para lectura de datos
        V = sc.nextInt();
        E = sc.nextInt();
        //Realizamos el ingreso del grafo, almacenando las aristas en un arreglo
con los datos respectivos
        for (int i = 0; i < E; ++i) {
            arista[i] = new Edge();
            arista[i].origen = sc.nextInt();
            arista[i].destino = sc.nextInt();
            arista[i].peso = sc.nextInt();
            //arista[ i ] = new Arista( sc.nextInt() , );
        }
        KruskalMST();
    }
}
```

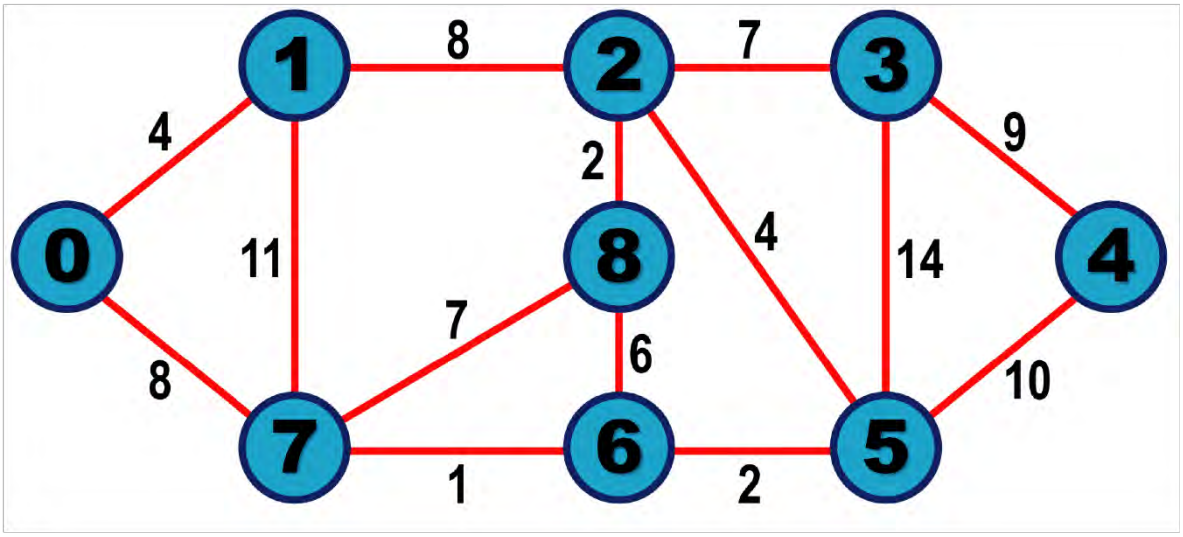
C++

```
#include <bits/stdc++.h>
#define MAX 1005
using namespace std;
int padre[MAX];

struct Edge {
    int origen, destino, peso;

    bool operator<(const Edge &t) {
        return peso < t.peso;
    }
};
//set de nodos para indicar cuales son sus padre e hijos

void makeSet(int n) {
    for (int i = 0; i <= n; i++) {
        padre[i] = i;
    }
}

int find(int x) {
    return (x == padre[x] ? x : (padre[x] = find(padre[x])));
}

void Union(int x, int y) {
    padre[find(x)] = find(y);
}

bool sameComponent(int x, int y) {
    if (find(x) == find(y)) {
        return true;
    }
    return false;
}
Edge aristas[MAX];
```

```
Edge MST[MAX];
vector<Edge>vec;

void KruskalMST(int V, int E) {
    int origen, destino, peso;
    int total = 0;
    int numAristas = 0;
    makeSet(V);
    sort(vec.begin(), vec.end());
    for (int i = 0; i < E; i++) {
        origen = vec[i].origen;
        destino = vec[i].destino;
        peso = vec[i].peso;
        if (!sameComponent(origen, destino)) {
            total += peso;
            MST[numAristas++] = vec[i];
            Union(origen, destino);
        }
    }
    if (V - 1 != numAristas) {
        cout << "No existe MST valido para el grafo ingresado este debe ser
conexo" << endl;
        return;
    }
    cout << "MST encontrado contiene las siguientes aristas" << endl;
    for (int i = 0; i < numAristas; i++) {
        cout << MST[i].origen << "," << MST[i].destino << ":" << MST[i].peso <<
endl;
    }
    cout << "el costo minimo de todas las aristas de MST es : " << total <<
endl;
}

int main() {
    int V, E;
    cin >> V>>E;
    for (int i = 0; i < E; i++) {
        Edge aristas[i];
        cin >> aristas[i].origen;
        cin >> aristas[i].destino;
        cin >> aristas[i].peso;
        vec.push_back(aristas[i]);
    }
    KruskalMST(V, E);
}
```

15.28) Árbol de expansión mínima de Prim

MST



Guia del programador competitivo

Ilustración 15-25 Buscando el árbol de expansión mínima mediante PRIM

El algoritmo de Prim es también un algoritmo voraz, comienza con un árbol de expansión vacío, la idea es mantener dos conjuntos de vértices, el primero contiene los vértices ya incluidos en el MST, la otra contiene los vértices que no han sido incluidos aun, en cada paso, se considera todos los caminos que conectan dos sets, y toma el camino con mínimo peso de esos caminos, luego de tomar el camino, se mueve el otro punto final del camino al set conteniendo el MST.

Un grupo de caminos que conectan dos sets de vértices en un grafo es llamado corte en la teoría de grafos, entonces en cada paso del algoritmo de Prim, nosotros buscamos un corte (de dos conjuntos, uno contiene los vértices ya en el MST y el otro el resto de vértices), tome el camino de mínimo peso del corte e incluye este vértice al conjunto MST.

La idea detrás del algoritmo de Prim es simple, un árbol de expansión significa que todos los vértices deben estar conectados, entonces los dos conjuntos disjuntos de vértices deben

estar conectados para hacer un árbol de expansión. Y ellos deben estar conectados con el camino de peso mínimo para realizar un árbol de expansión mínima.

1) Crear un set mstSet que mantenga rastro de los vértices ya incluidos en el MST.
2) Asignar un valor key a todos los vértices en el grafo de entrada, inicializar los valores key como INFINITO, asignar el valor key como 0 para el primer vértice entonces este es tomado primero.
3) Mientras mstSet no incluya todos los vértices.
    a) Tomar un vértice u el cual no esté en el mstSet y tenga el valor key mínimo
    b) Incluir u al mstSet
    c) Actualizar el valor key de todos los vértices adyacentes de u, para actualizar los valores key, se itera a través de todos los vértices adyacentes v, si el peso del camino u-v es menos que el valor key anterior de v, actualice el valor key como peso de u-v.

La idea de usar valores key es tomar el camino de mínimo peso como el corte, los valores key son usados únicamente por los vértices que no han sido incluidos en el MST, el valor key de estos vértices indican los caminos de peso mínimo conectándolos con el set de vértices incluidos en el MST.
Por ejemplo:

- El set mstSet esta inicialmente vacío y los keys asignados a los vértices son {0, INF, INF, INF, INF, INF, INF, INF} donde INF indica infinito, ahora tomamos el vértices con el valor key menor el vértice 0 es tomado, se incluye en el mstSetm entonces mstSet se convierte en {0}, luego de incluir al mstSet, se actualizan los valores key de los vértices adyacentes, los vértices adyacentes de 0 son 1 y 7, los valores key de 1 y 7 son actualizados como 4 y 8.
- Se toma el vértice con el menor valor key y no incluido ya en el MST, (no en mstSet), el vértice 1 es tomado y añadido al mstSet por lo que se convierte en {0,1}, se

actualiza los valores key de los vértices adyacentes de 1, el valor key del vértice 2 se convierte en 8.

- Se toma el vértice con el valor key mínimo y no incluido en MST, podemos tomar el vértice 7 o 2, tomaremos el 7, entonces el mstSet es {0,1,7}, actualizamos los valores key de los vértices adyacentes de 7, el valor key del vértice 6 y 8 se convierten en finitos, en 1 y 7.
- Tome el vértice con el valor key mínimo, y no incluido ya en MST, el vértice 6 es tomado entonces el mstSet se convierte en {0,1,7,6}, se actualizan los valores key y los vértices adyacentes de 6m el valor de 5 y 8 es actualizado.
- Repetimos los pasos de arriba hasta que mstSet incluya todos los vértices del grafo dado.

Complejidad de tiempo

Mejor caso : $O(v^3)$ **Peor caso :** $O(v^3)$ **Promedio:** $O(v^3)$

JAVA

```
//Programa java que calcula el MST de un grafo

import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Comparator;

public class PrimMSTFULL {

    public static void main(String[] args) {
        int V = 9;
        Graph graph = new Graph(V);
        addEdge(graph, 0, 1, 4);
        addEdge(graph, 0, 7, 8);
        addEdge(graph, 1, 2, 8);
        addEdge(graph, 1, 7, 11);
        addEdge(graph, 2, 3, 7);
        addEdge(graph, 2, 8, 2);
        addEdge(graph, 2, 5, 4);
        addEdge(graph, 3, 4, 9);
        addEdge(graph, 3, 5, 14);
        addEdge(graph, 4, 5, 10);
        addEdge(graph, 5, 6, 2);
        addEdge(graph, 6, 7, 1);
```

```
        addEdge(graph, 6, 8, 6);
        addEdge(graph, 7, 8, 7);
        prims_mst(graph);
    }

    static class node1 {
        int dest;
        int weight;
        node1(int a, int b) {
            dest = a;
            weight = b;
        }
    }

    static class Graph {
        int V;
        LinkedList<node1>[] adj;
        Graph(int e) {
            V = e;
            adj = new LinkedList[V];
            for (int o = 0; o < V; o++) {
                adj[o] = new LinkedList<>();
            }
        }
    }

    static class node {
        int vertex;
        int key;
    }

    /* Comparador de la priorityQueue
     retorna 1 si node0.key > node1.key
     retorna 0 si node0.key < node1.key y
    retorna -1 otherwise */
    static class comparator implements Comparator<node> {

        @Override
        public int compare(node node0, node node1) {
            return node0.key - node1.key;
        }
    }
    static void addEdge(Graph graph, int src, int dest, int weight) {
        node1 node0 = new node1(dest, weight);
        node1 node = new node1(src, weight);
        graph.adj[src].addLast(node0);
        graph.adj[dest].addLast(node);
    }

    // Buscar MST
    static void prims_mst(Graph graph) {
        Boolean[] mstset = new Boolean[graph.V];
        node[] e = new node[graph.V];
        int[] parent = new int[graph.V];
```

```
        for (int o = 0; o < graph.V; o++) {
            e[o] = new node();
        }
        for (int o = 0; o < graph.V; o++) {
            //inicializar en falso
            mstset[o] = false;
            // Inicial valores key en infinito
            e[o].key = Integer.MAX_VALUE;
            e[o].vertex = o;
            parent[o] = -1;
        }
        // incluir el vertice inicial en el MST
        mstset[0] = true;
        e[0].key = 0;
        PriorityQueue<node> queue = new PriorityQueue<>(graph.V, new
comparator());
        for (int o = 0; o < graph.V; o++) {
            queue.add(e[o]);
        }
        while (!queue.isEmpty()) {
            node node0 = queue.poll();
            mstset[node0.vertex] = true;
            for (node1 iterator : graph.adj[node0.vertex]) {
                if (mstset[iterator.dest] == false) {
                    if (e[iterator.dest].key > iterator.weight) {
                        queue.remove(e[iterator.dest]);
                        e[iterator.dest].key = iterator.weight;
                        queue.add(e[iterator.dest]);
                        parent[iterator.dest] = node0.vertex;
                    }
                }
            }
        }
        // Imprimir el par de vertices del mst
        for (int o = 1; o < graph.V; o++) {
            System.out.println(parent[o] + " "
                    + "-"
                    + " " + o);
        }
    }
}
```

Version 2:

```
//Programa java que busca el arbol de expansión
// minima de minimo costo

import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Scanner;

public class PrimMSTMinValue {
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    N = sc.nextInt();
    G = new Vertex[N];
    for (int i = 0; i < G.length; i++) {
        G[i] = new Vertex();
    }
    E = sc.nextInt();
    for (int i = 0; i < E; i++) {
        int from, to, w;
        from = sc.nextInt();
        to = sc.nextInt();
        w = sc.nextInt();
        G[from].adj.add(new Edge(to, w));
        G[to].adj.add(new Edge(from, w));
    }
    System.out.println(prim());

}

static PriorityQueue<QueueItem> Q;
static int E;
static boolean[] intree;
static int N;
static Vertex[] G;

static class QueueItem implements Comparable<QueueItem> {
    int v, w;
    public QueueItem(int v, int w) {
        this.v = v;
        this.w = w;
    }
    @Override
    public int compareTo(QueueItem q) {
        if (this.w != q.w) {
            return this.w - q.w;
        }
        return this.v - q.v;
    }
}

static void process(int u) {
    intree[u] = true;
    G[u].adj.forEach((e) -> {
        int v = e.to, w = e.w;
        if (!intree[v]) {
            Q.offer(new QueueItem(v, w));
        }
    });
}

static int prim() {
    intree = new boolean[N];
    Q = new PriorityQueue<>();
```

```
        int cost = 0;
        process(0);
        while (!Q.isEmpty()) {
            QueueItem qi = Q.poll();
            int v = qi.v, w = qi.w;
            if (!intree[v]) {
                cost += w;
                process(v);
            }
        }
        return cost;
    }

    static class Vertex {
        List<Edge> adj;
        public Vertex() {
            adj = new ArrayList<>();
        }
    }

    static class Edge {
        int to, w;
        public Edge(int to, int w) {
            this.to = to;
            this.w = w;
        }
    }
}
```

15.29) **Problemas de repaso**

Ejercicios en Online Judge

314-Robot	341-Non-Stop Travel
315-Network	492-Pig-Latin
331-Mapping the Swaps	439- Knight Moves
334-Identifying Concurrent Events	558-Wormholes
352-The Seasonal War	125-Numbering Paths
382- Perfection	

Capitulo 16. Otros algoritmos y programación dinámica

La programación dinámica es un método para reducir el tiempo de ejecución de un algoritmo mediante la resolución de subproblemas, los cuales al juntarse forman la solución del problema completo.

16.1) Knapsack 0/1

Knapsack

	1	2	3	4	5	6	7	8	9	10
Valor	79	32	47	18	26	85	33	40	45	59
Tamaño	85	26	48	21	22	95	43	45	55	52

- Mochila con capacidad 101
- 10 "elementos" (por ejemplo, proyectos, ...) 1, ..., 10
- Solución trivial: mochila vacía, valor 0
- Solución codiciosa, evaluando los artículos después del valor
  - (0000010000), valor de 85
  - Mejor sugerencia?

Guía del programador competitivo

Ilustración 16-1 Ejemplo del problema de la moleta

Dados pesos y valores de n ítems, ponga esos ítems en una mochila de capacidad W, para obtener el máximo valor total en la mochila, en otras palabras dados dos arrays de enteros val[0..n-1] y wt[0…-1] los cuales representan los valores y pesos asociados a los n ítems respectivamente. También dado un entero W el cual representa la capacidad de la mochila, encuentre el máximo valor subset de val[] tal que esa suma de los pesos de este subset es

menor o igual a w, no se puede romper un ítem, se tiene que tomar o no tomar, (propiedad 0-1).

Una simple solución es considerar todos los subsets de ítems y calcular el peso total y valor de todos los subsets, considere solo los subsets los cuales el peso total es menor a W, de todos los subsets tome el que tenga el máximo valor de peso.

Considere todos los subsets de ítems, puede haber dos casos para cada ítem:

1) El ítem está incluido en el subset optimo
2) El ítem no está incluido en el subset optimo

Con esto, el máximo valor puede ser obtenido de n ítems es máximo siguiendo los dos siguientes valores,

1) El máximo valor obtenido por n-1 ítems y peso W (Excluyendo el enésimo ítem)
2) Valor del enésimo ítem más el máximo valor obtenido por n-1 ítems y W menos el peso del enésimo ítem (Incluyendo el enésimo ítem).

SI el peso del enésimo ítem es mayor que W, entonces el enésimo ítem no puede ser incluido y el caso 1 es la única posibilidad.

Desde que los subproblemas son evaluados de nuevo, este problema tiene la propiedad de sobreponer subproblemas. Entonces el problema de la mochila 0-1 tiene ambas propiedades de un problema de programación dinámica, como otros problemas de programación dinámica, recomputaciones de subproblemas iguales pueden ser evadidos construyendo una matriz temporal desde el fondo, el código usa esta metodología.

La complejidad de tiempo de este algoritmo es de O(nW) donde n es el número de ítems y W el peso de la mochila.

JAVA

```java
/*Implementación java del problema 0-1 Knapsack */
public class KnapsackRecursive {
//Función de utilidad que retorna el mayor de dos números

    static int max(int a, int b) {
        return (a > b) ? a : b;
    }
```

```
    // Retorna el maximo valor que puede ser
    //Puesto en una knapsack de capacidad w

    static int knapSack(int W, int wt[], int val[], int n) {
        // Caso base
        if (n == 0 || W == 0) {
            return 0;
        }
        /*Si el peso de el nesimo item es más
        que la capacidad el knapsack, entonces
        este item no puede ser icluido en una
        solución optima*/
        if (wt[n - 1] > W) {
            return knapSack(W, wt, val, n - 1);
        } // Retorna el maximo de dos casos:
        //1) nesimo termino incluido
        //2) no incluido
        else {
            return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),
                    knapSack(W, wt, val, n - 1));
        }
    }

    public static void main(String args[]) {
        int val[] = new int[]{60, 100, 120};
        int wt[] = new int[]{10, 20, 30};
        int W = 50;
        int n = val.length;
        System.out.println(knapSack(W, wt, val, n));
    }
}
// Implementación java que usa programación dinamica
// para solucionar el problema de 0-1 Knapsack

public class KnapsackDP {
    //Función de utilidad que retorna el maximo de dos enteros
    static int max(int a, int b) {
        return (a > b) ? a : b;
    }
    // Retorna el maximo valor que puede ser puesto en un
    //knapsack de capacidad w
    static int knapSack(int W, int wt[], int val[], int n) {
        int i, w;
        int K[][] = new int[n + 1][W + 1];
        // Construye la tabla K[][]de abajo hacia arriba
        for (i = 0; i <= n; i++) {
            for (w = 0; w <= W; w++) {
                if (i == 0 || w == 0) {
                    K[i][w] = 0;
                } else if (wt[i - 1] <= w) {
                    K[i][w] = max(val[i - 1]
                            + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
                } else {
                    K[i][w] = K[i - 1][w];
```

```
                }
            }
        }

        return K[n][W];
    }

    public static void main(String args[]) {
        int val[] = new int[]{60, 100, 120};
        int wt[] = new int[]{10, 20, 30};
        int W = 50;
        int n = val.length;
        System.out.println(knapSack(W, wt, val, n));
    }
}
```

16.2) Cambio de monedas

Dado un valor N, si queremos hacer el cambio de N centavos y tenemos suministros infinitos de cada uno de las S = {S1, S2,.., Sm} monedas con valor, ¿de cuantas formas podemos hacer el cambio? El orden de las monedas no importa

Dado un valor N, queremos hacer el cambio por N centavos, y tenemos suministros de cada una, por ejemplo, para N=4, y S={1,2,3}, existen 4 soluciones {1,1,1,1},{1,1,2},{2,2},{1,3}. Entonces la salida debe ser 4, para N=10, y S={2,5,3,6} existen 6 soluciones {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} y {5,5}. Entonces la salida debe ser 5.

Para contar el total el números de soluciones podemos dividir todos los sets solución en dos sets.

1) Soluciones que no contienen emesima moneda o Sm.
2) Soluciones que al menos contienen una Sm.

Dejaremos ser count(S[],m,n) la función de conteo del número de soluciones, luego estas pueden ser escritas como suma de count(S[], m-1, n) y count(S[], m, n-Sm).

Por lo tanto, el problema tiene una propiedad se subestructura optima haciendo que el problema pueda ser resuelto usando soluciones a subproblemas.

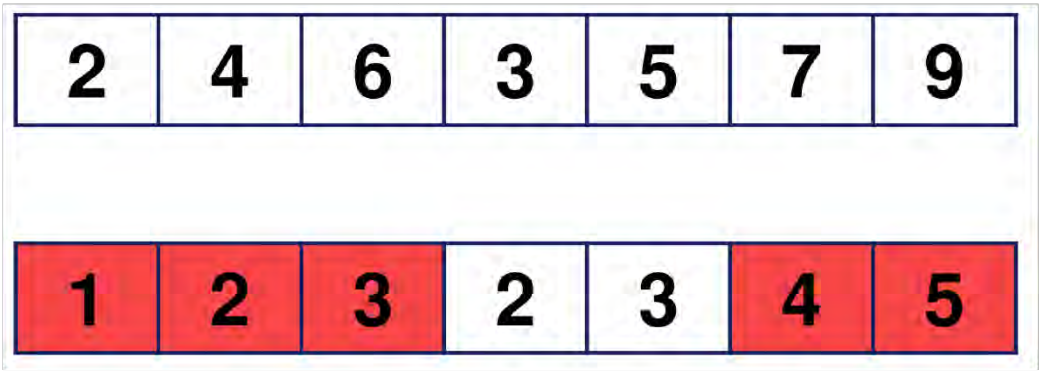
Complejidad de tiempo: O(mn)

JAVA

```java
// Programa java que resuelve el problema
// del cambio de monedas

public class CoinExchange {

    // Retorna el conteo de formás que podemos
    // sumar S[0...m-1] monedas para obtener n
    static int count(int S[], int m, int n) {
        // Si n es 0 entonces 1 es la solución
        // (No incluir ninguna moneda
        if (n == 0) {
            return 1;
        }
        // Si n es menos que 0 entonces no hay solución
        if (n < 0) {
            return 0;
        }
        /*Si no hay monedas y n es mayor que 0, no existe solución*/
        if (m <= 0 && n >= 1) {
            return 0;
        }
        // count es la suma de las soluciónes (i)
        // incluyendo S[m-1] (ii) excluyendo S[m-1]
        return count(S, m - 1, n)
                + count(S, m, n - S[m - 1]);
    }

    public static void main(String[] args) {
        int arr[] = {1, 2, 3};
        int m = arr.length;
        System.out.println(count(arr, m, 4));
    }
}
```

16.3) Longest Increasing Subsequence

Longest Increasing Subsequence



Guía del programador competitivo

Ilustración 16-2 Ejemplo LIS

El problema de la secuencia incremental más larga (LIS), es encontrar la longitud de la subsecuencia más larga dada una secuencia la cual todos los elementos de la subsecuencia están ordenados en orden ascendente, por ejemplo la longitud de a LIS para {10, 22, 9, 33, 21, 50, 41, 60, 80} es 6 y LIS es {10, 22, 33, 50, 60, 80}.

Dejaremos arr [0...n-1] ser el array de entrada y L(i) ser la longitud de la LIS finalizando en el índice I tal que arr[i] es el último elemento de la LIS.

Entonces L(i) puede ser recursivamente escrito como:

L(i) = 1 + max(L(j)) donde 0 < j < i y arr[j] < arr[i];

o

L(i) = 1, Si no existe tal j.

Para encontrar la LIS de un array dado, necesitamos retornar max(L(i)) donde 0 < i < n.

Por lo tanto vemos que el problema LIS satisface la propiedad de la subestructura óptima como el problema principal puede ser resuelto usando soluciones a subproblemas.

Note que la complejidad de tiempo de esta programación dinámica es O(nLogn).

JAVA

```java
//Programa java que implementa la LIS

public class LongestIncreasingSubsequence {

    static int max_ref;

    /* Para hacer uso de llamadas recursivas, esta función debe devolver.
   dos cosas:
   1) Longitud de LIS que termina con el elemento arr [n-1]. Usamos
      max_ending_here para este propósito
   2) Máximo global ya que el LIS puede terminar con un elemento
      antes de arr [n-1] max_ref se usa para este propósito.
   El valor de LIS de la matriz completa de tamaño n se almacena en
   * max_ref cual es nuestro resultado final*/
    static int _lis(int arr[], int n) {
        // Caso base
        if (n == 1) {
            return 1;
        }
        // 'max_ending_here' es el tamaño de LIS
        // terminando con arr[n-1]
        int res, max_ending_here = 1;
        for (int i = 1; i < n; i++) {
            res = _lis(arr, i);
            if (arr[i - 1] < arr[n - 1] && res + 1 > max_ending_here) {
                max_ending_here = res + 1;
            }
        }
        if (max_ref < max_ending_here) {
            max_ref = max_ending_here;
        }
        return max_ending_here;
    }

    static int lis(int arr[], int n) {
        // Guarda el resultado
        max_ref = 1;
        //Almacena su resultado en max
        _lis(arr, n);
        //Retorna el maximo
        return max_ref;
    }

    public static void main(String args[]) {
        int arr[] = {10, 22, 9, 33, 21, 50, 41, 60};
        int n = arr.length;
        System.out.println("Tamaño de la subsecuencia incremental más larga"
                + " es " + lis(arr, n) + "\n");
    }
}
```

16.4) Longest Common SubString

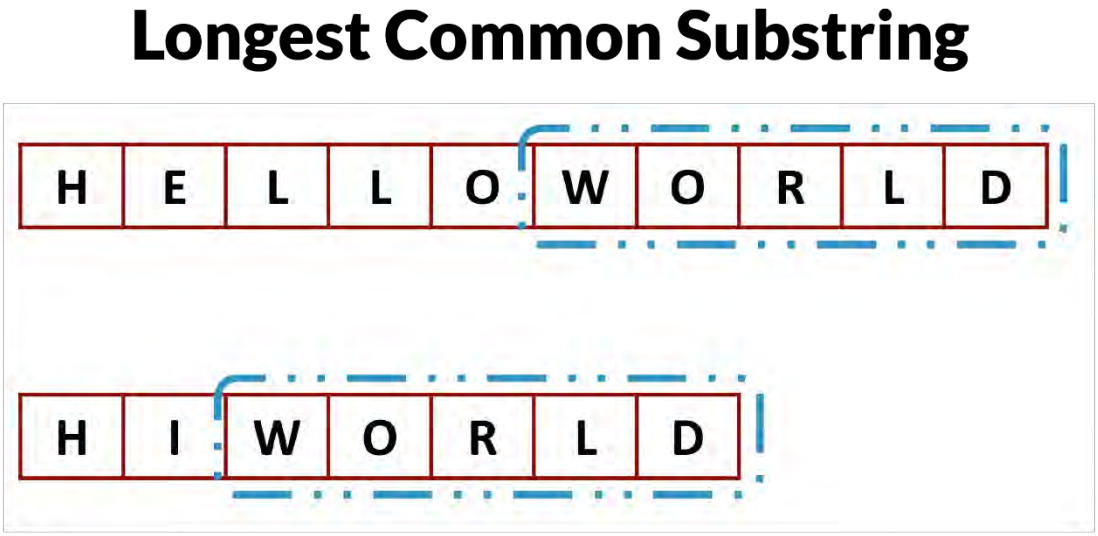


Ilustración 16-3 Ejemplo LCS

Dados dos strings X y Y encuentre el substring común más largo. Dejaremos que M y N sean las longitudes del primer y segundo string respectivamente.

Una simple solución es uno por uno considerar todos los substrings del primer string y por cada substring verificar su es un substring en el segundo substring, mantenemos rastro del string de longitud máxima, ahí pueden haber O(m^2) substrings y podemos encontrar si un string es substring de otro string en tiempo O(n), entonces el tiempo general de este método será O(n * m2)

Programación dinámica puede ser usada para encontrar el string común más largo en O(n*m) tiempo, la idea es encontrar el sufijo común más largo para todos los substrings de ambos strings y almacena estas longitudes en la tabla.

El sufijo común más largo tiene propiedad de subestructura propia.

Si el último carácter coincide, entonces reducimos ambas longitudes en 1.

- LCSuff(X, Y, m, n) = LCSuff(X, Y, m-1, n-1) + 1 if X[m-1] = Y[n-1]

Si el último carácter no coincide, entonces el resultado es 0, por ejemplo:

- LCSuff(X, Y, m, n) = 0 if (X[m-1] != Y[n-1])

Ahora consideramos sufijos de diferentes substrings terminando en diferentes índices, el sufijo común más largo, su longitud es el substring común más largo.

- LCSubStr(X, Y, m, n) = Max(LCSuff(X, Y, i, j)) donde 1 <= i <= m y 1 <= j <= n

Complejidad de tiempo: O(m*n)

JAVA

```java
/* Implementación java que encuentra el substring comun más
largo usando programación dinamica*/
public class LongestCommonSubString {

    static int LCSubStr(char X[], char Y[], int m, int n) {

        int LCStuff[][] = new int[m + 1][n + 1];
        int result = 0;  //Para almacenar el tamaño del substring
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                if (i == 0 || j == 0) {
                    LCStuff[i][j] = 0;
                } else if (X[i - 1] == Y[j - 1]) {
                    LCStuff[i][j] = LCStuff[i - 1][j - 1] + 1;
                    result = Integer.max(result, LCStuff[i][j]);
                } else {
                    LCStuff[i][j] = 0;
                }
            }
        }
        return result;
    }

    public static void main(String[] args) {
        String X = "hola mundo";
        String Y = "mundo";
        int m = X.length();
        int n = Y.length();
        System.out.println("El tamaño del substring cómun más largo es "
                + LCSubStr(X.toCharArray(), Y.toCharArray(), m, n));
    }
}
```

16.5) Triángulo de Pascal

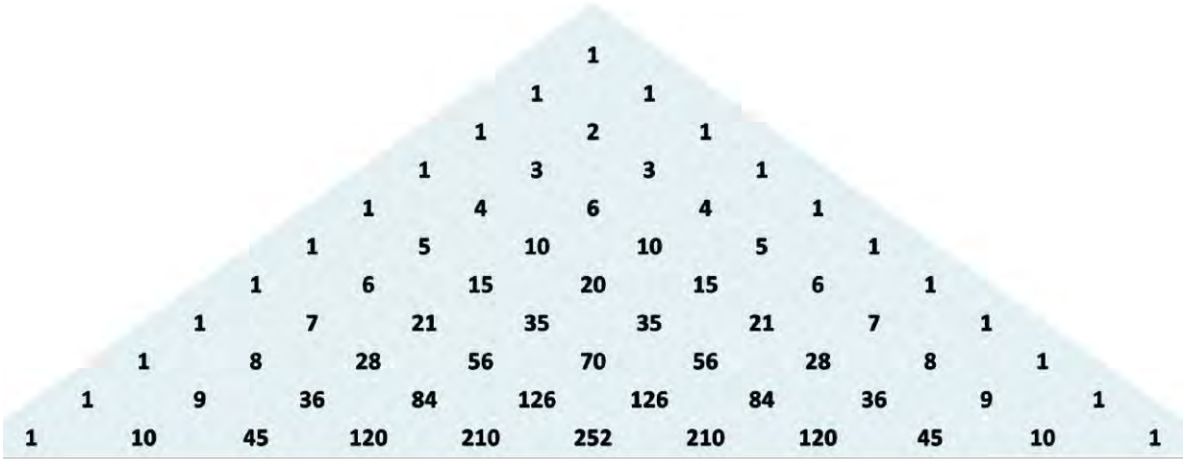


Ilustración 16-4 Triangulo de pascal de N=10

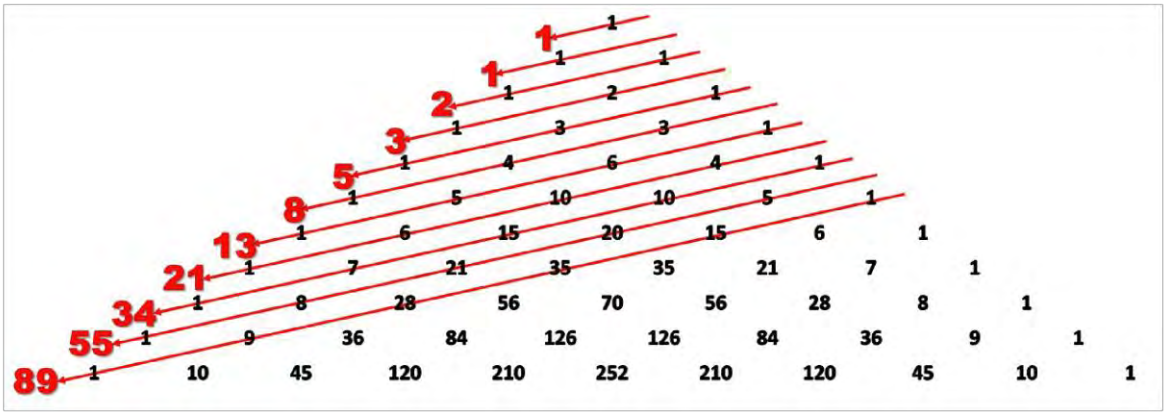


Ilustración 16-5 Números de Fibonacci en el triángulo de Pascal

El triángulo de Pascal es un triángulo de números enteros, infinito y simétrico Se empieza con un 1 en la primera fila, y en las filas siguientes se van colocando números de forma que cada uno de ellos sea la suma de los dos números que tiene encima. Se supone que los lugares fuera del triángulo contienen ceros, de forma que los bordes del triángulo están formados por unos.

- Los números del triángulo de Pascal coinciden con los números combinatorios.
- El número combinatorio Cm n (n sobre m) se encuentra en el triángulo en la fila n+1, en el lugar m+1.
- El número combinatorio Cm n (n sobre m) que representa el número de grupos de m elementos que pueden hacerse de entre un conjunto de n (por ejemplo, (4 sobre 2) nos da el número de parejas distintas que podrían hacerse en un grupo de cuatro personas), se encuentra en el triángulo en la fila n+1, en el lugar m+1.
- Podemos saber que el número de parejas posibles que decíamos antes es 6 si miramos el tercer número de la quinta fila.

Esto hace que el triángulo sea útil como representación de estos números, y proporciona una buena forma de intuir sus propiedades.

- La fórmula general del llamado Binomio de Newton (a + b)n está formada por unos coeficientes que coinciden con la línea número n+1 del triángulo de Pascal (la que empieza por 1 y n).
- Si el primer elemento de una fila es un número primo, todos los números de esa fila serán divisibles por él (menos el 1, claro). Así, en la fila 7: (1 7 21 35 35 21 7 1), los números 7,21 y 35 son divisibles por 7.
- La suma de los elementos de cualquier fila es el resultado de elevar 2 al número que define a esa fila.
- La serie de Fibonacci puede ser encontrada también en el triángulo de Pascal. Dividiendo al mismo según las líneas que mostramos en el diagrama, los números atrapados entre ellas suman cada uno de los elementos de esta sucesión.

JAVA

```java
//Código java que realiza el triángulo de Pascal

public class PascalTriangle {

    static void printPascal(int n) {
        /* Itera atraves de cada linea y la imprime
        con sus entradas*/
        for (int line = 0; line < n; line++) {
            // Cada linea tiene u número de enteror
            //igual al número de linea
            for (int i = 0; i <= line; i++) {
                System.out.print(binomialCoeff(line, i) + " ");
            }
            System.out.println();
        }
    }

    static int binomialCoeff(int n, int k) {
        int res = 1;
        if (k > n - k) {
            k = n - k;
        }
        for (int i = 0; i < k; ++i) {
            res *= (n - i);
            res /= (i + 1);
        }
        return res;
    }

    public static void main(String args[]) {
        int n = 7;
        printPascal(n);
    }
}
```

16.6) Problemas de repaso

Ejercicios en Online Judge

147-Dollars	1213-Sum Of Different Primes
357-Let Me Count The Ways	10130-SuperSale
990-Diving for Gold	1196-Tiling Up Blocks

10029-Edit Step Ladders

10534-Wavio Sequenc

3. Lista de tablas

Lista de ilustraciones

5. Bibliografía

ACM International Collegiate Programming Contest. (2001). *Latin American Regional Contest - Contest session .* ACM.

Amraii, S. A. (2006). Observations on Teamwork Strategies in the. *Crossroads, The ACM Student Magazine*, 1-3.

Arias, J. (2020, mayo 10). *Algorithms and more* . Retrieved from UTP: https://jariasf.wordpress.com/

Asociación Colombiana de Ingenieros de Sistemas. (2019, Septiembre 10). *ACIS*. Retrieved Septiembre 11, 2019, from https://acis.org.co/portal/

Bronson, G. (2007). *C++ para ingeniería y ciencias.* México, D.F.: Cengage Learning.

Combéfis, S. (2014). Programming Trainings and Informatics Teaching. *Olympiads in Informatics*, 21-34.

CPP Reference. (2019). *Cpp Reference*. Retrieved Septiembre 10, 2019, from https://en.cppreference.com/w/

Directi. (2019, Septiembre 10). *About CodeChef*. Retrieved from https://www.codechef.com/aboutus

Etheridge, D. (2009). *Java: The Fundamentals of Objects and Classes.* Ventus Publishing ApS.

Francesc Comellas. (2001). *Matemática discreta UPC.* Barcelona: Edicions de la Universitat Politècnica de Catalunya.

Francesc Comellas, J. F. (2001). *Matemática Discreta.* Catalunya: Edicions de la Universitat Politècnica de Catalunya, SL.

García de Jalón, J., Rodriguez , J. I., Mingo, I., & Brazáles, A. (2000). *Aprenda Java.* Navarra: Escuela Superior de Ingenieros Industriales de San Sebastian.

GeeksforGeeks. (2019, Septiembre 10). *GeeksforGeeks About US*. Retrieved from https://www.geeksforgeeks.org/about/

González, R. (2018). *Python para todos.* Madrid.

Johnsonbaugh, R. (2005). *Matematicas Discretas 6ta edición.* Ciudad de Mexico, Mexico: Prentice Hall.

Kiusalaas, J. (2013). *Numerical methods in engineering with python 3.* New York,: Cambridge University Press.

Laaksonen, A. (2018). *Competitive Programmer's Handbook.* Helsinki.

OEIS. (2020, Mayo 10). *La Enciclopedia On-Line de las Secuencias de Números Enteros*. Retrieved from https://oeis.org/?language=spanish

Oracle. (2019, Septiembre). *Oracle*. Retrieved Septiembre 10, 2019, from https://www.java.com/es/download/help/index_using.xml

Python Software Foundation. (2019, Septiembre 9). *Docs Python*. Retrieved from https://docs.python.org/3/

Rob Hoogerwoord, H. Z. (2016). *Discrete Structures.* 2IT50.

Steven Halim, F. H. (2019). *Competitive programming 3.* Lulu Enterprises. Inc.

Universidad Nacional de Colombia. (2020, mayo 10). *Programación competitiva UNAL*. Retrieved from https://sites.google.com/view/programacion-unal/inicio?authuser=0

Uva Online Judge. (2012, Julio 12). *UVa OJ Board*. Retrieved Septiembre 10, 2019, from https://uva.onlinejudge.org/board/viewtopic.php?t=71042

Zhu Jie-ao, S. M. (2005). Learning Software Engineering through Experience of ACM-ICPC Training and Practicing Exercises. *ACM/ICPC*.