ESCUELA MILITAR DE INGENIERÍA
UNIDAD ACADÉMICA LA PAZ
CARRERA DE INGENIERÍA DE SISTEMAS

# GUÍA DE INTRODUCCIÓN A LAS COMPETENCIAS DE PROGRAMACIÓN


**INGENIERÍA DE SISTEMAS**

**JOSÉ MISAEL MACHICADO SANJINÉS**

**JUAN MIGUEL MACHICADO SANJINÉS**

LA PAZ 2012

DEDICATORIA:

Agradecemos a Dios que con amor siempre esta guiando nuestros pasos, a nuestros queridos padres Felix Machicado y Margot Sanjinés por habernos dado el amor, la guía, educación y apoyo incondicional en todos los aspectos de nuestras vidas, a nuestros queridos abuelos Rosendo Sanjinés y Luisa Cruz, asimismo a nuestros hermanos Martha, Pablo, Danny y a nuestros familiares. A todos los docentes de la carrera de Ingeniería de Sistemas de la Escuela Militar de Ingeniería por los conocimientos impartidos, así también al Tcnl. DIM. Samuel Salgueiro Viaña y al Tcnl. Julio Cesar Narvaez Tamayo por el apoyo brindado en sus gestiones como jefes de carrera. Al Ing. Miguel Alejandro Zambrana Camberos por haber apoyado y supervisado este proyecto.

Juan Miguel Machicado Sanjinés y
José Misael Machicado Sanjinés

## Tabla de contenido


1. Introducción ..... 1
    1.1. Algoritmos ..... 1
    1.2. Lenguajes de Programación ..... 3
    1.3. Compiladores y entornos de ejecución ..... 5
    1.4. IDE ..... 6
    1.5. Competencias de programación ..... 6
    1.6. Juez virtual ..... 7
2. Estructura de datos ..... 12
    2.1. Tipos de datos ..... 12
    2.1.1. Números enteros ..... 12
    2.1.2. Números decimales ..... 16
    2.1.3. Caracteres ..... 17
    2.1.4. Booleanos ..... 18
    2.1.5. Cadenas ..... 19
    2.2. Estructuras ..... 23
    2.2.1. Estructura de datos lineales ..... 23
    2.2.1.1. Vectores estáticos ..... 23
    2.2.1.2. Vectores de tamaño variable ..... 29
    2.2.2. Pilas ..... 32
    2.2.3. Colas ..... 36
    2.2.4. Cola de prioridad ..... 40
    2.2.5. Listas enlazadas ..... 41
    2.2.6. Sets ..... 42
    2.2.7. Maps ..... 44
    3. Recursividad ..... 46
    3.1. Introducción a Backtracking (Proceso recursivo hacia atrás) ..... 55
    4. Introducción a la Programación dinámica ..... 57

# Capitulo 1

## 1. Introducción

Esta publicación pretende ser una guía inicial e intermedia en cuanto a algoritmos su aplicación en ejercicios de competencias de programación y también sobre el uso de diferentes jueces virtuales existentes y bastante conocidos en el ámbito de algoritmos, aclaramos que la presente guía debe ser complementada con lecturas sobre temas que están fuera del alcance de la presente, así mismo se recomienda que se acompañe la lectura de esta publicación con lecturas sobre el uso del lenguaje de programación de su preferencia ya que por el alcance y los objetivos de esta publicación muchos puntos de más profundidad no podrán ser cubiertos, también es importante aclarar desde un principio que el fin principal de las competencias de programación no es ganar una competencia nacional, regional o mundial, el verdadero fin de las competencias de programación es el de preparar a los estudiantes para que puedan aprender, aplicar, investigar y desarrollar algoritmos que optimicen y proporcionen una solución eficaz a problemas relativos a ciencias computacionales, aprender a trabajar en equipo, aprender a trabajar bajo presión, fomentar el espíritu competitivo en el estudiante (sana competencia OJO :D) .

En la presente guía se mostrara la configuración de todos los componentes necesarios para comenzar a practicar programación mostraremos dicho proceso tanto para Linux (IMPORTANTE en esta guía se utilizara Ubuntu 11.10 como distribución para los ejemplos) como para Windows (utilizaremos Windows XP, Windows Vista y Windows 7).

### 1.1. Algoritmos

Se comenzara por definir que es un algoritmo, un algoritmo en términos simplificados es una serie de pasos para concretar una cierta tarea por ejemplo para preparar una te de cocoa usualmente se siguen los siguientes pasos:

1. Obtener una taza para servir el te
2. Sacar un sobre del paquete de te
3. Quitar la envoltura del sobre de te e introducirlo en la taza
4. Vertir agua caliente en la taza
5. Colocar azúcar al gusto

Si bien se aplica esto cotidianamente, existen algoritmos para ser aplicados en diferentes ámbitos y para diferentes fines, el estudio de la optimización y aplicación de estos forman parte importante de las ciencias computacionales.

Cuando se analiza un problema a resolver muchas veces la primera solución intuitiva que se proporciona en muy pocos casos llega a ser una solución eficaz para el problema planteado, por ejemplo si se tiene el siguiente conjunto de números:

{ 2, 1, 4, 7, 6, 3, 9, 0}

y se quiere realizar varias consultas para saber si cierto numero “x” se encuentra en ese conjunto, la solución intuitiva seria para cada consulta buscar secuencialmente si dicho numero esta dentro de dicho conjunto ¿cierto? Es decir:

Consulta: ¿El numero 4 está en el conjunto de números?

{2, 1, 4, 7, 6, 3, 9, 0}

Algoritmo:

Paso 1. ¿4 es igual al primer elemento (2)?
Resultado 1. No, entonces se compara con el siguiente elemento
Paso 2. ¿4 es igual al segundo elemento (1)?
Resultado 2. No, entonces se compara con el siguiente elemento
Paso 2. ¿4 es igual al segundo elemento (4)?
Resultado 3. Si! Por lo tanto 4 se encuentra en el conjunto de datos

Pero esta no sería una solución eficaz si se tuviera una cantidad muy grande números dentro del conjunto ¿verdad? Pues existen algoritmos que ayudan a mejorar el tiempo de búsqueda y respuesta en conjuntos grandes de datos (no solo números), por ejemplo se podría ordenar el conjunto de números una sola vez para varias consultas y buscar utilizando el principio de bisección.

## 1.2. Lenguajes de Programación

Un lenguaje de programación permite escribir programas que puedan ser entendidos por el computador, permite las implementación de algoritmos en un computador, permite al programador elegir los datos que se deben usar, como serán guardados, como se presentaran al usuario final, para lo cual los lenguajes de programación intentan estar lo más cercano posible al lenguaje humano o natural.

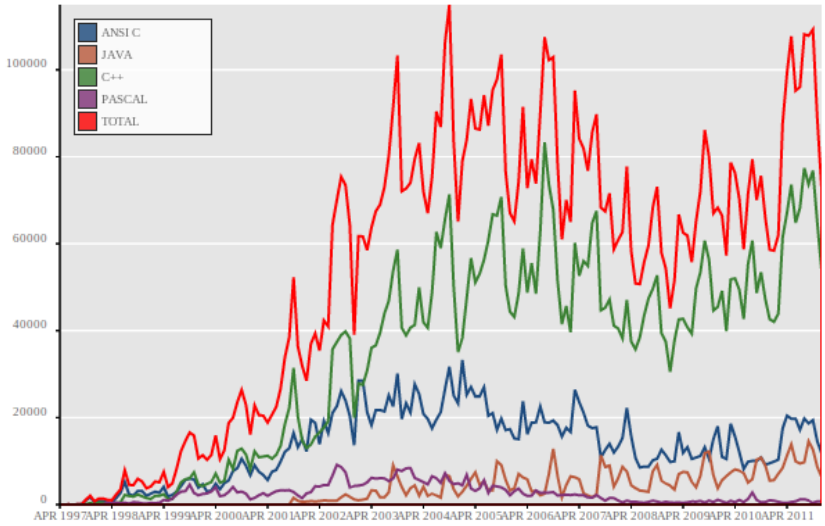
Los programas escritos deben ser compilados (traducir al lenguaje máquina que es necesario para que la computadora pueda entender las peticiones) y finalmente la computadora ejecuta y procesa los datos.

Una de las primeras decisiones a tomar es que lenguaje de programación utilizar para empezar a practicar e implementar algoritmos.

Tal vez en este momento el estudiante se pregunte como tomar dicha decisión y como saber si es una decisión acertada, pues se puede mencionar al respecto que nadie puede decir “ese y solamente ese lenguaje de programación le gana a todos los demás en lenguajes en todos los aspectos”, pues cada lenguaje de programación tiene sus pros y contras con respecto a los demás, pero para los fines de esta guía se mencionara los siguientes:

1. Pascal
2. ANSI C
3. C++
4. Java

Los anteriores cuatro lenguajes son los más utilizados hoy en día en competencias de programación, como base para esta afirmación se presenta el siguiente grafico extraído de la página de práctica para competencias de programación de la Universidad de Valladolid administrada y mantenida por Miguel Revilla




Como se puede ver en el anterior grafico el lenguaje menos usado es Pascal porque a pesar de ser bastante didáctico tiene una desventaja en cuanto a funcionalidades predefinidas (librerías) y además que va perdiendo fuerza en el mercado, podemos observar que ANSI C tiene una tendencia de decrecimiento esto ya que dentro de los paradigmas de programación la programación orientada a objetos es la tendencia actual en programación, y tanto ANSI C y Pascal están orientados  a la programación estructurada, C++ tiene todas las ventajas de ANSI C y además se presenta como lenguaje orientado a objetos y que además tiene varias librerías de bastante utilidad y es el más usado en competencias de programación, Java es un lenguaje completamente orientado a objetos que día a día va a ganando terreno por su característica multiplataforma e implícitamente dicha característica lo convierte en un lenguaje con mucho potencial en diversos campos, en esta guía no se pretende resaltar que tal o cual lenguaje es el mejor, sin embargo cabe mencionar que por tendencia y estadística se adoptara para las siguientes practicas tanto C++ por ser el lenguaje más utilizado y Java por su tendencia creciente, y cabe mencionar que ambos tienen características en sus librerías de utilidades bastante ventajosas, que se mencionara más adelante.

## 1.3. Compiladores y entornos de ejecución

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina sería capaz de interpretar, usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio o simplemente texto. (Fuente "Wikipedia, Articulo: Compilador, Revision: 3.5). Un entorno de ejecución es el que permite ejecutar los programas compilados en un cierto lenguaje. Todas las tareas mencionadas anteriormente (edición, compilación, y ejecución se pueden realizar a través de la línea de comandos del sistema operativo sin embargo existen herramientas que facilitan dichas tareas (Entornos de desarrollo integrados o IDE por sus siglas en ingles) sobre las que se entrara más en detalle en la siguiente sección). Para la presente se utilizara tanto C (ANSI C y C++) como Java y se mostrara el proceso de instalación de ambos lenguajes. En el caso de Java se necesitara instalar JDK y en el caso de C instalar GCC. A continuación se adjunta documentos de guía desde la descarga, instalación, hasta configuración de variables de entorno tanto para el JDK y el GCC (en dichas guías se entrara más en detalle sobre el uso de las variables de entorno del sistema operativo)

- JDK (Java Development Kit por sus siglas en ingles es el Kit de Desarrollo de Java)
  - Windows XP
  - Windows Vista - Windows 7
  - Linux (Ubuntu)

- GCC (GNU C Compiler por sus siglas en ingles es el Compilador de C GNU)
  - Windows XP (MinGW)
  - Windows Vista - Windows 7 (MinGW)
  - Linux (Ubuntu) (GCC y G++)

## 1.4. IDE

Los lenguajes mencionados cuentan con IDE's (Entorno de Desarrollo Integrado) que son de bastante ayuda y guía al momento de programar, ya que muchos de los IDE actuales cuentan con detección de errores y verificación de errores de sintaxis y semántica en la escritura de código fuente, compilación y ejecución automática de código fuente y también algunos cuentan con sugerencias de corrección, como ejemplo se puede mencionar NetBeans (Java), Codeblocks (ANSI C, C++), Eclipse (ANSI C, C++ y Java), Jgrasp (Multilenguaje), etc.

- Guía de Instalación y configuración de Eclipse para Java, ANSI C y C++
- Creación de un primer proyecto de Java, ANSI C y C++

## 1.5. Competencias de programación

Como se mencionó en la introducción el fin principal de las competencias de programación es preparar a los estudiantes para enfrentar retos de diferente naturaleza en las ciencias computacionales más específicamente en la solución algorítmica (de buena práctica y eficaz) de muchos de estos, existen competencias en las que participa gente profesional y estudiantil sin distinción de edad, podemos mencionar muchos beneficios de participar en las mismas entre los más importantes mencionamos los siguientes beneficios:

a. Ampliar conocimientos (así mismo no se logre clasificar a una competencia nacional, sudamericana o mundial el conocimiento que gane nadie se lo quitara).
b. Becas (es conocido que el hecho de haber participado en esta competencia es una muy buena carta de presentación para optar a muchas becas nacionales e internacionales)
c. Profesional (como se menciono es una muy buena carta de presentación para el mercado el haber participado en este tipo de competencias)
d. Aprender a trabajar bajo presión (por la misma naturaleza de la

competencia)

e. Aprender a trabajar y colaborar en equipo (ya que muchas de estas competencias se realizan en equipos pero vale aclarar que hay competencias que son de carácter individual)

Se puede mencionar muchas competencias de programación entre las más importantes se tiene a la ACM-ICPC (International Collegiate Programming Contest) organizada por la ACM que es sino la más grande una de las más grandes redes mundialmente reconocida de estudiantes y profesionales del área computacional, que fomenta la investigación y desarrollo de este campo para beneficio de la sociedad, esta competencia cuenta principalmente con el patrocinio de IBM entre otras empresas que apoyan este torneo en la que actualmente se participa en equipos de tres personas donde usualmente se plantean de 5 a 10 ejercicios de distintos niveles de dificultad y temática teniendo usualmente 5 horas para la resolución de los mismos, empezando su participación posiblemente en un torneo interno dentro de su universidad para posteriormente participar usualmente en el torneo nacional cuyos clasificados participan en la competencia regional correspondiente a su país que finalmente culmina con la instancia del torneo mundial de programación, se puede mencionar también la IOI (International Olimpiad in Informatics) que es el torneo internacional para escuelas secundarias de similares características a la ACM-ICPC, Facebook Hacker Cup es también otra competencia importante de programación, Challenge 24, Google Code Jam, Top Coder Competition, Java Cup, entre muchas otras.

## 1.6. Juez virtual

En dichas competencias o torneos de programación se utiliza un juez virtual (que es simplemente un programa computacional) que básicamente en base a los ejercicios propuestos indica si el ejercicio fue resuelto correctamente o no.

¿Cómo funciona un juez virtual? Es bueno entender esto ya que así se tiene una idea general de él porque un juez virtual genera un cierto tipo de respuesta.

Cuando un ejercicio es registrado en un juez virtual además de su planteamiento, contiene un archivo de entrada de datos que utiliza para probar su programa y un archivo de salida que utiliza para comparar su respuesta con la respuesta proporcionada, en base a esto se devuelve un cierto veredicto.

Se dara un ejemplo ficticio simple para entenderlo mejor, no se preocupe en este punto por como dar una solución en programa a este ejercicio, el objetivo es comprender el ejercicio.

**Ejercicio: Suma de números**

Adrian esta en el segundo curso de primaria y tiene como tarea realizar mas de cien sumas y esta en problemas porque esa tarea se la dieron hace un mes y debería haber hecho todo ese trabajo en este mes, el se encuentra desesperado y recurre a su ayuda y le pide que le realice un programa para ingresar dos números y obtener la respuesta.

El formato de la entrada de datos será el siguiente, primeramente se le proporcionara el número de sumas a realizar N (N es el número de sumas a realizar) donde N siempre será 0<=N<=500 (esto ya es una pista para saber que no tendremos que controlar que n es mayor a 0 así que al resolver es una preocupación menos, así mismo se menciona que N que es el número de sumas estará entre 0 y 500) , posteriormente se le proporcionara N * 2 números a ser sumados, es decir después de N se le dará dos números por cada N para realizar la suma correspondiente y mostrar en pantalla el resultado.

Ejemplo de entrada:

3 → Según la descripción nos indica que tendremos que hacer 3 sumas
1 2 → Primer caso de entrada, tendremos que sumar estos dos números
2 6 → Segundo caso de entrada, tendremos que sumar estos dos números
3 2 → Tercer caso de entrada, tendremos que sumar estos dos números

Ejemplo de salida

3 → Resultado de la primera suma
8 → Resultado de la segunda suma
5 → Resultado de la tercera suma

-------Fin del Ejercicio-------

Como se pudo observar por lo genera los ejercicios de practica incluyen una descripción didáctica del problema, cosa que es buena ya que se plantean casos ficticios para los que se tiene que idear soluciones traducidas en programas.

Como se pudo ver según la descripción el anterior ejercicio simplemente recibe el número de sumas a realizar y los dos números a sumar para cada una.

Una vez entendido esto seguramente escribirá un programa que resuelva esta problemática, una vez probado y seguro de que resuelve este mismo, está listo para enviar su solución y obtener el veredicto del juez, cuando envía su solución (es decir el código fuente que puede ser a través de internet o cual sea el medio de recepción del juez virtual usado), el juez recibe el dicho código fuente, en ese momento empieza la calificación automatizada de dicho ejercicio siguiendo los siguientes pasos:

1. Compilación de código fuente
2. ¿Hubo errores en la compilación? Si es así devolver “Error de compilación” y terminar la comprobación
3. Una vez compilado el código empieza la ejecución del programa, teniendo un tiempo límite para la misma, en este ejemplo daremos 5 segundos para la ejecución
4. Si el programa durante su ejecución tuvo algún evento inesperado considerado como excepción (Ejemplo: división de un número entre cero), se termina la ejecución y se devuelve “Error en Tiempo de Ejecución” y termina la comprobación

5. Si el programa tarda más de 5 segundos en ejecutarse se devuelve “Tiempo Limite excedido” y termina la comprobación
6. Si termino de ejecutarse se pasa a la etapa de comprobación

   Cuando se registra un problema se registra además de su descripción, tiempo límite de ejecución, un archivo de entrada de datos por ejemplo para el ejercicio anterior los siguientes datos estarán guardados en un archivo de texto:

   ```
   4
   1 2
   2 7
   3 6
   4 1
   ```

   Como pudo observar los datos son diferentes a los datos de ejemplo, por lo tanto también se tienen respuestas diferentes que están guardados en otro archivo de texto, para este caso se tendría guardado:

   ```
   3
   9
   9
   5
   ```

   Teniendo estos dos archivos, se compara con la salida generada por el programa del usuario
7. Por ejemplo, si la salida del usuario fuese la siguiente:

   ```
   3
   9
   9
   5
   ```

Como la salida es idéntica se devuelve “Aceptado” y se termina la comprobación

8. Si la comparación tiene diferencias, por ejemplo:

   4

   10

   10

   6

   Se devuelve “Respuesta Incorrecta” y se termina la comprobación

El proceso que se describió anteriormente es una forma de calificación sin embargo el funcionamiento entre jueces virtuales varia sin embargo es una idea general del funcionamiento de un juez virtual, usualmente las respuestas que generan los jueces virtuales son las siguientes:

- Accepted (ACC).- Significa que el código esta correcto (compilado y ejecutado correctamente), se ejecutó en un tiempo igual o menor al tiempo máximo y las respuestas de su programa coinciden con las del juez.
- Wrong Answer (WA).- Significa que el código esta correcto (compilado y ejecutado correctamente), se ejecutó en un tiempo igual o menor al tiempo máximo, pero sus respuestas no coinciden con las respuestas del juez.
- Time Limite Exceeded (TLE).- Significa que compilo bien pero que su código se ejecuta en más del tiempo establecido para ese problema.
- Runtime Error (RE).- Significa que compilo bien, pero en el momento de ejecutarse su código este tiene error (desborde de estructuras u otros).
- Compilation Error (CE).- Significa que su código no compilo bien es decir tiene fallo en sintaxis.

Guía de uso del Juez virtual de la Universidad de Valladolid (UVa)

# 2. Estructuras de datos

Los distintos lenguajes de programación usan datos primitivos para todos sus entornos de trabajo tal como lo son los enteros, decimales, cadenas, booleanos y otros.

Se tiene datos que serán variables y constantes (su valor no cambia a través del tiempo), donde se puede mencionar que los valores variables son las más utilizadas sin duda alguna.

En este capítulo se explicara lo básico de como declarar y leer las distintas variables en C++ y java.

*Referencia ejemplo en capitulo2.java y capitulo2.cpp*

## 2.1. Tipos de datos

A continuación se explicara los distintos tipos de datos existentes, así también del como manipularlos

### 2.1.1. Números enteros

Existen distintos tipos de datos enteros (es decir que no soportan la fracción decimal, ej: 2,74 no es soportado sin embargo 2 si es soportado por un entero) los cuales son soportados en Java, C y C++ sin embargo existen algunas diferencias entre estos más que todo en el rango que soportan es decir el tamaño que se le asigna a cada uno de estos en bits.

En Java se tiene los siguientes tipos de datos:

| Tipo de dato | Tam. en bits | Rango que soporta | Valor por defecto |
|---|---|---|---|
| byte | 8 bits | De -128 a 127 | 0 |
| short | 16 bits | De -32768 a 32767 | 0 |
| int | 32 bits | De -2147483648 a 2147483647 | 0 |

| | | | |
|---|---|---|---|
| long | 64 bits | De -9223372036854775808 a 9223372036854775807 | 0 |

En ANSI C:

| **Tipo de dato** | **Tam. en bits** | **Rango que soporta** | **Valor por defecto** |
|---|---|---|---|
| char | 8 bits | De -128 a 127 | x |
| unsigned char | 8 bits | De 0 a 255 | x |
| signed char | 8 bits | De -128 a 127 | x |
| int | 16 bits | De -32768 a 32767 | x |
| unsigned int | 16 bits | De 0 a 65535 | x |
| signed int | 16 bits | De -32768 a 32767 | x |
| short int | 16 bits | De -32768 a 32767 | x |
| unsigned short int | 16 bits | De 0 a 65535 | x |
| signed short int | 16 bits | De -32768 a 32767 | x |
| long int | 32 bits | De -2147483648 a 2147483647 | x |
| signed long int | 32 bits | De -2147483648 a 2147483647 | x |
| unsigned long int | 32 bits | De 0 a 4294967295 | x |

En C++:

| **Tipo de dato** | **Tam. en bits** | **Rango que soporta** | **Valor por defecto** |
|---|---|---|---|
| char | 8 bits | De -128 a 127 | x |
| unsigned char | 8 bits | De 0 a 255 | x |
| signed char | 8 bits | De -128 a 127 | x |
| int | 16 bits | De -32768 a 32767 | x |
| unsigned int | 16 bits | De 0 a 65535 | x |
| signed int | 16 bits | De -32768 a 32767 | x |
| short int | 16 bits | De -32768 a 32767 | x |

| | | | |
|---|---|---|---|
| unsigned short int | 16 bits | De 0 a 65535 | x |
| signed short int | 16 bits | De -32768 a 32767 | x |
| long int | 32 bits | De -2147483648 a 2147483647 | x |
| signed long int | 32 bits | De -2147483648 a 2147483647 | x |
| unsigned long int | 32 bits | De 0 a 4294967295 | x |
| long long | 64 bits | De -9223372036854775808 a 9223372036854775807 | x |
| unsigned long long | 64 bits | De 0 a 18446744073709551616 | x |
| signed long long | 64 bits | De -9223372036854775808 a 9223372036854775807 | x |

Observando las diferencias entre los lenguajes se puede resaltar que en Java no existe la opción de “unsigned” lo que llevaría a la pregunta ¿entonces Java no puede almacenar números del rango de C o C++? La respuesta es NO, sin embargo Java en ese sentido cuenta con una ventaja que tanto C y C++ no tienen, Java dentro de su librería “java.math” cuenta con la clase BigInteger que soporta números enteros muy grandes (más grandes que los presentados en los rangos límite de C y C++) donde básicamente el límite de almacenamiento se encuentra dado por la cantidad de bits utilizados para almacenar la cadena (conjunto de caracteres) que es manejada dinámicamente al manipular estos, sin embargo no se puede decir que puede almacenar un número infinito ya que obviamente se tendrá un límite de memoria de almacenamiento.

Para dar una idea se mencionara en palabras sencillas de que una clase es un término propio del paradigma de programación orientado a objetos que sin ser exactos se podría decir que se refiere a un contenedor de valores y métodos que cumplen ciertas tareas, donde dicho contenedor vendría a ser una plantilla para crear entidades que contengan dichas propiedades y funcionalidades especificadas en la declaración de la clase.

Sin embargo quedaría la siguiente pregunta ¿entonces se puede almacenar en C y C++ números grandes? La respuesta es SI sin embargo se tendría que construir funciones propias para manejar este tipo de datos tanto C y C++ contienen librerías de utilidades al

igual que Java pero no tienen dentro de sus librerías estándar un equivalente al BigInteger de Java, el modo de construcción de dicha librería en C o C++ seria básicamente imitando la manera manual de realizar estas operaciones, para ser más ilustrativo imitar el modo como nos enseñaron en la escuela a realizar estas operaciones es decir si sumariamos 193 + 12 primeramente se sumaria los dos últimos dígitos de la derecha es decir 3 + 2 es 5, los siguientes dos dígitos 9+1 es 10 donde se tendría que llevar de acarreo uno para los siguientes dígitos, finalmente se toma en cuenta el acarreo 1 + 1 + 0 seria 2 por lo tanto el resultado seria 205, de igual manera se tendría que imitar el proceso manual para la resta, multiplicación y división.

Otra diferencia que se puede observar es de que al momento de ejecutar el programa Java asigna valores iniciales por defecto a nuestras variables, sin embargo en C y C++ a veces el compilador asigna un valor por defecto o se asocia dicho valor a cierta porción de memoria que contiene basura informática es decir valores anteriores almacenados en ese espacio de memoria y finalmente también se puede percatar de que se mencionó a “char” como tipo de dato entero primeramente se debe aclarar que el tipo de dato char es utilizado para almacenar caracteres (ejemplos de caracteres: ‘a’,’c’,’!’, etc) por naturaleza sin embargo es válido también utilizarlos para almacenar valores numéricos por qué se debe a que el computador asocia un cierto número a un carácter en un cierto estándar, los más conocidos son ASCII y Unicode de los que se entrara en más detalles en la sección de caracteres más adelante, como un ejemplo de asociación numérica a un caracteres se podría mencionar el tan conocido valor “64” que está asociado al carácter ‘@’, se puede hacer la prueba de esto presionando en cualquier editor de texto la combinación de la tecla ALT y sin soltar ALT se presiona también el número “064”.

Como aclaración también se puede indicar que si es que se tiene un número decimal y es asignado a un número entero es posible truncar la parte decimal es decir solo tomar en cuenta la parte entera en ese caso se podría hacer lo siguiente:

```
int a=(int) 5.3;          en la variable a se guardo 5;
```

Este truco funciona tanto en C++ como en java, que es una manera rápida de convertir los datos, en este caso de los enteros no los redondea para esto existe funciones como lo es round().

(Mirar los códigos de ejemplo para mas detalles)

### 2.1.2. Números decimales

Existe float y double tanto en C, C++ y Java. Al igual que con los enteros se vera las tablas comparativas.

En Java se tiene:

| Tipo de dato | Tam. en bits | Rango que soporta | Valor por defecto |
|---|---|---|---|
| float | 32 bits (Norma IEEE 754) | De 1.40129846432481707e-45 a 3.40282346638528860e+38 | 0 |
| double | 64 bits (Norma IEEE 754) | De 4.94065645841246544e-324d a 1.79769313486231570e+308d | 0 |

En ANSI C y C++:

| Tipo de dato | Tam. en bits | Rango que soporta | Valor por defecto |
|---|---|---|---|
| float | 32 bits | De 3.4e-38 a 3.4e+38 | x |
| double | 64 bits | De 1.7E-308 a 1.7e+308 | x |
| long double | 64 bits | De 1.7E-308 a 1.7e+308 | x |

Simplemente se mencionara que en el caso de los valores por defecto pasa lo mismo que con los tipos de datos enteros, al igual que para los enteros Java tiene dentro de la librería "java.math" la clase BigDecimal que almacena números decimales muy grandes, situación en la que también C y C++ no cuentan con esa funcionalidad estándar y donde de igual manera se tendría que construir una función propia para manejar esto.

**NOTA:** En *capitulo211y2.cpp* se explica el uso de printf (método de mostrar información en consola donde solamente se muestran caracteres un ejemplo ilustrativo de consola seria la línea de comandos de Windows o la terminal de Linux) que también java lo tiene aplicado como System.out.printf( *argumentos* );

*Referencia ejemplo en capitulo211y2.java y capitulo211y2.cpp*

Resolver los siguientes ejercicios de la UVA:

| |
|---|
| UVA Online judge - Problem Jumping Mario - 11764 |
| UVA Online judge – Problem 11332 - Summing Digits |
| UVA Online judge – Problem 100 - The 3n+1 problem |
| UVA Online judge – Problem 11799 - Horror Dash |

### 2.1.3. Caracteres

Un carácter es una representación de una letra, símbolo, figura, número.

En el computador los caracteres son asociados a valores numéricos como se mencionó anteriormente con el ejemplo del carácter '@' que está asociado al número 64 dentro de la tabla de representación del computador, para dicha representación se siguen estándares entre los que mencionaremos como principales a ASCII y Unicode, ASCII tiene variantes ya que para los caracteres algunos idiomas se necesitan caracteres especiales por ejemplo para el idioma español necesitaremos el carácter 'ñ' que no será necesitado en el idioma ingles o árabe, de igual manera para el idioma japonés necesitaremos caracteres especiales que requerirán otra variante de ASCII, tanto C y C++ utilizan ASCII un carácter ASCII ocupa 8 bits, sin embargo Java utiliza Unicode que a pesar de utilizar más espacio para almacenar un carácter ya que utiliza 8, 16 o 32 bits según la arquitectura del computador, engloba varios caracteres utilizados en múltiples idiomas.

***char*** → o carácter, este almacena solo una letra, figura o numero Ejemplo:

char a='b';  → todo carácter se declara en comillas simples. En la variable a se guardo el carácter b

*Referencia ejemplo en capitulo213.java y capitulo213.cpp*

Resolver los siguientes ejercicios de la UVA:

| |
|---|
| UVA Online judge - Problem 575 - Skew Binary<br>UVA Online judge – Problem 272 - TeX Quotes<br>UVA Online judge – Problem 458 - The Decoder<br>UVA Online judge – Problem 10878 Decode the tape |

### 2.1.4. Booleanos

En este tipo de dato se maneja o bien verdad o falsedad, se utilizan como banderas o para comprobación o para control de flujo (es decir cuando debe acceder a donde).

Este tipo de datos solo maneja 0 para falso y 1 para true en el caso de c++ y solo se maneja true y false en el caso de java.

Java

boolean sw=true;  en sw se guardo verdad.
boolean sw=false;  en sw se guardo falso.

C++

bool sw=1; en sw se guardo verdad.
bool sw=0; en sw se guardo falso.

*Referencia ejemplo en capitulo214.java y capitulo214.cpp*

### 2.1.5. Cadenas

Una cadena es un conjunto de caracteres (unión de 2 o más caracteres), el manejo varia para ANSI C, C++ y Java.

**Cadenas en java**

En java se tiene la clase String que nos ayuda al manejo de cadenas, esta clase contiene varias funciones útiles para el tratamiento de estas así mismo el tamaño de las mismas es "variable", sin embargo se puso variable entre comillas ya que técnicamente no es exacto decir que el tamaño es variable; se declara una cadena de la siguiente manera:

String cad="nueva cadena"; → en esta caso es con doble comilla.

Para entender mejor lo que se explicaba anteriormente de que las cadenas de Java son de tamaño "variable", al crear la anterior cadena "nueva cadena" en memoria se creara un objeto (creado a partir de una clase) llamado "cad", si se quisiera agregar más texto a la misma por ejemplo el texto " de muestra" a la ya existente "nueva cadena" almacenada como "cad" y que sea "nueva cadena de muestra" se podría realizar lo siguiente para concatenar (unir las dos cadenas):

cad = cad + " de muestra";

Es importante resaltar que se debe asignar la operación a dicha variable ya que de otra manera no se guardara el cambio, por ejemplo si se realizara lo siguiente:

cad + " de muestra"; → no se asigna el resultado a una variable

No se almacenaría, ya que en la memoria de Java se tiene un "pool" (que se puede definir como un repositorio o almacén de objetos) y la manera en cómo funciona es guardando referencias, por ejemplo el valor "nueva cadena" inicialmente es referenciado a través de la

variable llamada “cad”:

| **Variable** | | **Valor en pool** |
|---|---|---|
| cad | ← | “nueva cadena” |

Al realizar la operación:

```
cad = cad +  “ de muestra”;
```

Se tendría lo siguiente:

| **Variable** | | **Valor en pool** |
|---|---|---|
| cad | ← | “nueva cadena de muestra” |
| Sin referencia | ← | “nueva cadena” |

Como se puede observar se crea un nuevo valor en pool y se hace referencia a cad y el anterior valor “nueva cadena” queda sin referencia sin embargo se debe tener en cuenta que el entorno de ejecución de Java (Maquina Virtual de Java) es el que se encarga de la limpieza de todos estos valores en pool que quedaron sin referencia, también se puede hacer explícitamente esta limpieza, sin embargo el propio entorno de ejecución es el que se encarga de esto.

Una vez explicado todo esto se tendría que aclarar que en la memoria de Java quedarían dos instancias diferentes: una seria “nueva cadena” y la segunda seria “nueva cadena de muestra”, es decir la primera instancia no se une con la segunda para formar una sola cadena, si no que se crean dos instancias diferentes, sin embargo si no se dio una explicación muy profunda de dichos conceptos es bueno conocer por lo menos a grandes rasgos estos para entender ciertos errores.

¿Cómo accedo a la posición de la cadena?

En java se accede a la clase de la cadena escribiendo un punto delante de ella, ejemplo:

```
nombre_de_la_cadena.charAt( posicion );
```

Haciendo un punto los distintos editores en los cuales se trabaje mostraran todo lo que se puede realizar con las cadenas como ejemplo se mostraran los más usados:

Para saber el tamaño de la cadena:

```
nombre_de_la_cadena.length();
```

Para convertir de minusculas a mayusculas las letras de l.a cadena y viceversa:

```
nombre_de_la_cadena=nombre_de_la_cadena.toUpperCase();  →  de mayusculas a minusculas
nombre_de_la_cadena=nombre_de_la_cadena.toLowerCase();→  de minusculas a mayusculas
```

Cabe mencionar de que existe otras clases llamadas “StringBuffer” y “StringBuilder” (recomendado) para el tratamiento de cadenas donde el contenido de dichas cadenas si es variable (sin comillas).

Para más detalles sobre la implementación observe el código de ejemplo *capitulo215.java*

**Cadenas en ANSI C y C++**

A diferencia de Java en ANSI C y C++ se puede almacenar cadenas en un vector (podemos definir un vector como un conjunto de elementos, se entrara más en detalle sobre vectores más adelante) de caracteres para manejar cadenas, sin embargo se debe dar un tamaño para almacenar dicha información, darle un valor inicial o podríamos utilizar las librerías existentes en ANSI C y C++ para dichos fines, por ejemplo en C++ se podría declarar una

cadena dándole un valor inicial de la siguiente manera:

char nombre_de_la_cadena[] = " Aca usted ingresa la cadena "; -para declarar se escribe con doble comilla.

Para poder ver la cadena:

printf( "%s", nombre_de_la_cadena );

Cabe aclarar de que para ANSI C se tiene la librería string.h y para C++ dentro de la STL de C++ tenemos la clase string que cumple funciones similares a la clase String de Java.

*Referencia de ejemplo en capitulo215.java y capitulo215.cpp*

¡Momento! ¿Qué es la STL de C++? en este punto es bueno mencionar que tanto C++ como Java tienen librerías de utilidades, en el caso de C++ la librería de plantillas de utilidades estándar es la STL (Standard Templates Library) donde se puede encontrar muchas funciones, por ejemplo en este preciso punto sirve para declarar y utilizar cadenas de manera mucho más sencilla, en el caso de Java también se tiene una librería de utilidades que esta dentro de “java.util” que cumple funciones similares a la STL de C++ sin embargo vale la pena aclarar que las implementaciones de funciones equivalentes para C++ y Java pueden variar, no necesariamente son las mismas, si se profundiza en las implementaciones de las funciones para C++ y Java podrá percatarse de esta situación, en varios casos se utilizan en C++ y Java algoritmos diferentes (implementación) para realizar una misma tarea.

Resolver los siguientes ejercicios de la UVA:

UVA Online judge - Problem 575 - Skew Binary
UVA Online judge – Problem 644 - Immediate Decodability

## 2.2. Estructuras

La base para resolver problemas es saber que estructura se utilizara (Algunos problemas solo requieren una estructura para ser resueltos). Se debe identificar y analizar el problema:

- ¿En el problema existen actualizaciones?
- ¿En el problema existen eliminaciones?
- ¿Es un problema en el cual se deba ordenar los datos?, en este caso ¿se requiere eficiencia? ¿Cuál es la cantidad de datos a ordenar?

Toda estructura tiene su límite de memoria entonces identificar el espacio de memoria es vital para evitar un RTE, identificar la eficiencia que debe tener el algoritmo ayuda a evitar un TLE.

En este capítulo se hare referencia a las distintas librerías con las cuales cuenta C++ y Java, como se menciono anteriormente se hare referencia al STL de C++ y a la librería “java.util” que cuentan con librerías que serán de mucha utilidad en este tópico.

### 2.2.1. Estructura de datos lineales

Se llama estructura de datos lineal si es que sus elementos forman una secuencia (es decir se forman linealmente), su acceso a estos son rápidos es por esta razón que los concursantes usan estos casi en todos los problemas.

#### 2.2.1.1. Vectores estáticos

Un vector es un conjunto de elementos guardado bajo una referencia general a la que se puede hacer referencia hacia los elementos mediante índices, para graficar un poco esto se da el siguiente ejemplo:

Dado un conjunto desordenado de números del 1 al 10:

{6, 3, 8, 5, 1, 7, 10, 9, 2, 4}

Se llama a este conjunto “A” para hacer referencia posteriormente a él, entonces se tendría:

A = {6, 3, 8, 5, 1, 7, 10, 9, 2, 4}

Se puede hacer referencia al primer elemento del conjunto “A” mediante su índice, es decir:

A[1] = 6

De igual forma con el quinto elemento de dicho conjunto: A[5] = 1

Es decir simplemente se guarda los elementos bajo una referencia (A en este ejemplo) y posteriormente simplemente se hace referencia a cada uno de sus elementos por su índice, en cuanto a los vectores se cuenta tanto con los vectores llamados estáticos y vectores dinámicos, la diferencia entre estos dos es que uno es de tamaño fijo y de tamaño variable correspondientemente, por el mismo nombre se puede entender que existe una ventaja al contar con un vector de tamaño variable es decir que se podrá agregar y quitar elementos de este y simplemente ocupar el espacio necesario para este fin, sin embargo existe una diferencia en velocidad entre ambos ya que el vector de tamaño fijo a pesar de dar esa restricción es de acceso más veloz a cada uno de los elementos esto se debe ya que a que cuando se crea el vector de tamaño fijo en la memoria del ordenador se crean espacios contiguos para todos sus elementos.

Todos estos datos son guardados en la memoria temporal del ordenador o en la memoria de una maquina virtual (caso de Java), junto con otros datos, por ejemplo si se representaría gráficamente una memoria de 40 espacios representando cada uno de estos como una casilla con capacidad para 8 bits, y asignando un número a cada una de estas empezando desde uno se tendría la siguiente figura:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Asumiendo que varias aplicaciones (incluyendo un programa propio) harán uso de este espacio de memoria, por ejemplo en un inicio se empiezan a ejecutar la aplicación 1 y la aplicación 2, la aplicación 1 (de color rojo) ocupa con sus datos los primeros ocho espacios, la aplicación 2 (de color azul) ocupa 4 espacios:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Y en este momento el programa propio (de color amarillo) empieza su ejecución (lo único que hará esta aplicación será crear un vector llamado “A” de tamaño fijo de tamaño 6), entonces nuestro programa pide espacio al ordenador en memoria para almacenar 6 elementos (cada uno de 8 bits), en ese momento el ordenador reserva 6 espacios para nuestros datos bajo la referencia de “A”:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Como se pudo observar se reservaron 6 espacios contiguos para la colección de elementos “A”, obviamente esta representación está bastante lejos de cómo en realidad se asigna los

espacios y como si pudiéramos ver por colores (por aplicación), se vería la memoria del ordenador bajo una representación similar, ya que en caso de verla así se vería una mezcla impresionante de colores por el hecho de que continuamente múltiples procesos y aplicaciones que se están ejecutando piden espacio en memoria o liberan espacio que previamente ocuparon, tal vez la pregunta en este momento sea ¿porque se toma el tiempo de entender todo esto, si lo que se trata de entender son algoritmos?, esto es importante ya que al momento de escoger entre una y otra opción se debe saber cuál es la opción conveniente y las implicaciones que la elección entre un vector estático y un vector dinámico conllevan.

Vale la pena aclarar que tanto en C++ y en Java los vectores y otras estructuras de datos están en base cero, que simplemente es enumerar los elementos desde 0, por ejemplo un conjunto de cuatro elementos estarían indexados como elemento 0 (primer), elemento 1 (segundo elemento), elemento 2 (tercer elemento), elemento 3 (cuarto elemento).

**Vectores estáticos en java**

Para evitar un RTE se debe saber hasta cuanto se puede reservar en memoria para un Vector, en el caso de java en un vector 1D soporta hasta un tamaño cercano a 10 000 000 por defecto.

La forma de declarar un vector 1D es de la siguiente manera:

```
tipo_de_dato    nombre_vector[]=new tipo_de_dato [tamaño];
```

La forma de declarar un vector 2D (una matriz) es de la siguiente manera

```
tipo_de_dato nombre_vector[][]=new tipo_de_dato [tamaño_en_filas]
[tamaño_en_columnas];
```

**Ejemplos:**

```
int Vector_prueba[]=new int [10000000];
```

```
String Vector_prueba[]=new String [10000000];
```

Java por defecto inicializa las variables con sus valores por defecto según el tipo de dato elegido para el vector, en el caso de declarar:

```
int Vec[]=new int [10];
```

Java hace lo siguiente:

| Posicion | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Vec | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

En el caso de declarar:

```
String Vec[]=new String [10];
```

Java hace lo siguiente:

| Posicion | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Vec | null | null | null | null | null | null | null | null | null | null |

En Java existe una clase que pertenece al paquete útil (java.util) que se llama Arrays, esta clase trabaja con los vectores estáticos de Java, por ejemplo si queremos ordenar ese vector:

```
Arrays.sort(nombre_vector_a_ordenar); //Complejidad O (n log n)
```

**Vectores estáticos en c++**

Como se explico anteriormente en la sección de tipos de datos se mencionara que Java inicializa las variables con un valor por defecto, sin embargo el modo de actuar de ANSI C y C++ es diferente (a veces con basura, es decir datos que estaban en esa posición de memoria).

Para evitar un RTE se debe saber hasta cuanto se puede reservar en memoria para un Vector, en el caso de C++ en un vector 1D soporta hasta un tamaño cercano a 10 000 000 por defecto.

La forma de declarar un vector 1D es de la siguiente manera:

```
tipo_de_dato   nombre_del_vector [tamaño];
int array[4] = {2,3,4,1};
```

En este caso se declaro un vector de tamaño 4 con los datos 2, 3, 4, 1, valores que los guarda de la siguiente manera:

{2,3,4,1} el 2 en la posicion 0
{2,3,4,1} el 3 en la posicion 1
{2,3,4,1} el 4 en la posicion 2
{2,3,4,1} el 1 en la posicion 3

¿Cómo se accede a una posición? Para acceder a una posicion en c++ solo se escribe:

```
nombre_del_vector[ posicion ]
```

Ejemplo de cómo ver con printf:

```
printf("%d", nombre_array[ posicion ]);
```

¿Cómo ordenar un vector?

En c++ existen distintas maneras de ordenar un vector pero para estos algoritmos existen clases predefinidas, para esto debemos importar.

```
#include <algorithm>  → este es el conocido c++ STL algorithm
sort(nombre_array, nombre_array + tamaño); //Complejidad O (n log n)
```

NOTA: Existen otros métodos de ordenación se explicaran en el código cpp( solo en c++, java solo tiene sort)

*Referencia ejemplo en capitulo2211.java y capitulo2211.cpp*

### 2.2.1.2. Vectores de tamaño variable

En este tipo de vectores no es necesario declarar el tamaño, sino que si se elimina un dato el tamaño reduce en 1, si se aumenta un dato el tamaño aumenta en 1.

Para entender la diferencia con un vector estático se continuara con la misma explicación que se realizo para los vectores estáticos.

Si se representaría gráficamente una memoria de 40 espacios representando cada uno de estos como una casilla con capacidad para 8 bits, y asignándoles un número a cada una de estas empezando desde uno se tendría la siguiente figura:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Asumiendo que varias aplicaciones incluyendo un programa propio harán uso de este espacio de memoria, por ejemplo en un inicio se empiezan a ejecutar la aplicación 1 y la aplicación 2, la aplicación 1 (de color rojo) ocupa con sus datos los primeros ocho espacios, la aplicación 2 (de color azul) ocupa 4 espacios:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Y en este momento el programa propio (de color amarillo) empieza su ejecución (lo único que hará esta aplicación será crear un vector llamado “A” de tamaño variable de 6 elementos), entonces el programa pide espacio al ordenador en memoria para agregar un elemento de 8 bits, en ese momento el ordenador reserva 1 espacios para dicho elemento bajo la referencia de “A”:

El programa propio sigue en ejecución al igual que los otros programas, mientras nuestro el programa no reserva más memoria los otros programas siguen haciendo uso de la misma, por ejemplo si la aplicación 1 pediría dos espacios de memoria más, la aplicación 2 pediría 3 espacios de memoria más, y recién la aplicación propia solicita agregar 2 elementos más, hipotéticamente sucedería lo siguiente:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

Como se puede observar el espacio reservado para el conjunto de elementos “A” no se encuentra contiguamente almacenado, obviamente esta representación está bastante lejos de cómo en realidad se asigna los espacios y como si pudiéramos ver por colores (por

aplicación), sin embargo se puede resaltar que ya que los elementos se encuentran dispersos se debe mantener una referencia entre uno y otro elemento para acceder al siguiente o anterior elemento, para lo que se utilizan punteros (que como su nombre indica apuntan al siguiente y/o anterior elemento), lo que hace mucho más lento el acceso a los elementos que en un vector estático.

Explicación para java:

En java también se utilizan los ArrayList(conjuntos de datos formados como lista) que es más rápido que el vector.

Primero se debe importar el util:

```
import java.util.*;
```

Ahora se debe conocer que no se trabaja con los tipos de datos explicados al principio del capitulo, se utilizara como se explica:

En vez de int se utiliza Integer
En vez de double se utiliza Double
En vez de char se utiliza Character
En vez de boolean se utiliza Boolean

Ejemplo:

Vector<Integer> A=new Vector<Integer>(); → Se esta declarando un vector de tamaño variable llamado A de tipo de dato entero.

Vector<Integer> A=new Vector<Integer>(); → Se declara un vector de tamaño variable llamado A de tipo de dato entero.

ArrayList<Character> A=new ArrayList<Character>(); → Se esta declarando un

ArrayList de tipo de dato char o caracter.

*Referencia ejemplo en capitulo2212.java y capitulo2212.cpp*

Resolver los siguientes ejercicios de la UVA:

| |
|---|
| UVA Online judge - Problem 591 - Box of bricks<br>UVA Online judge – Problem 541- Error Correction<br>UVA Online judge – Problem 594 - One Little, Two Little, Three Little Endians<br>UVA Online judge – 482 - Permutation Arrays<br>UVA Online judge – 644 - Immediate Decodability (uso de vector) |

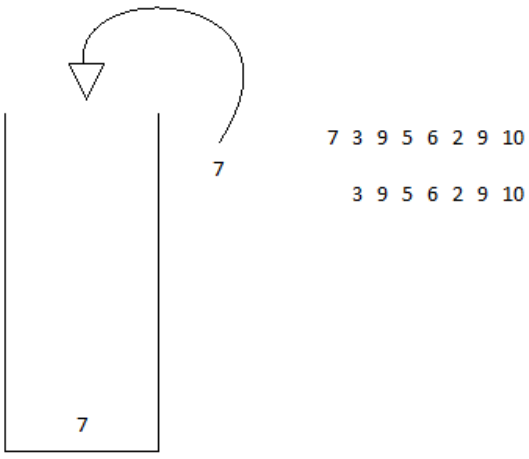
### 2.2.2. Pilas

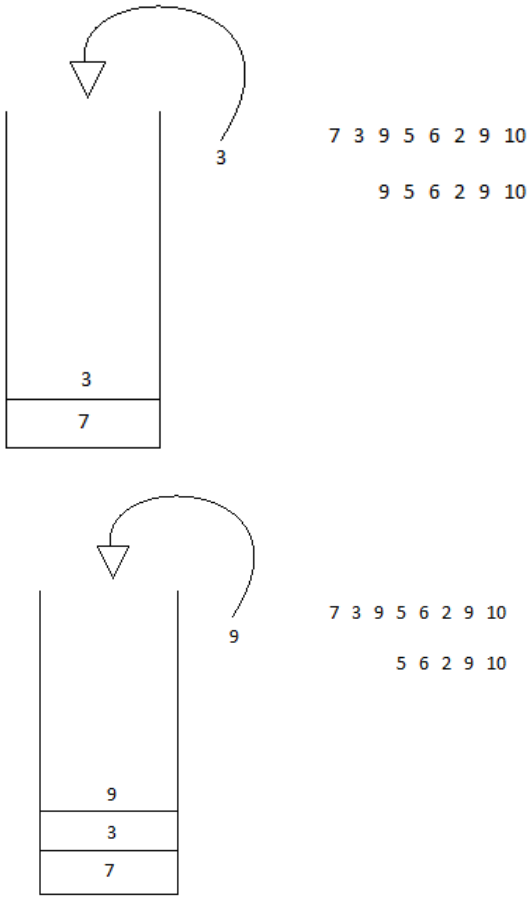
(Last input First output) o LIFO, Ultimo en entrar primero en salir.

Pila es una estructura que trabaja de la siguiente manera:

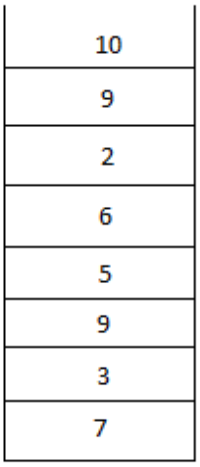
Se inserta los siguientes datos:

7 3 9 5 6 2 9 10

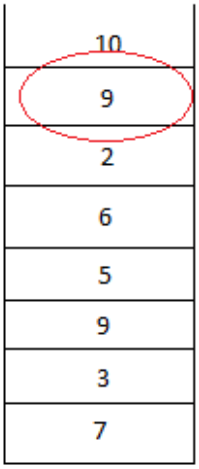





Así sucesivamente hasta:



Si se requiere sacar el último 9:



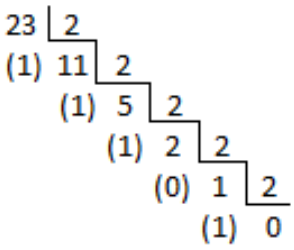

Necesariamente se tiene que sacar el 10 para poder sacar el 9.

Entonces se deduciría que las pilas ayudan a guardar un orden de inserción y cuando se saquen los datos se los tendrá de atrás hacia adelante (inverso).

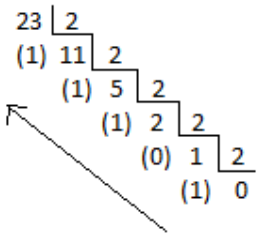
Ejemplo:

Para convertir a binario un numero primeramente se debe efectuar una serie de divisiones entre 2 tomando solamente la parte entera y posteriormente en orden inverso recolectar el resto de las divisiones efectuadas para obtener el numero en base 2, como se ejemplifica a continuación

La división:



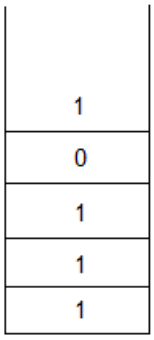
Escrita en orden inverso.



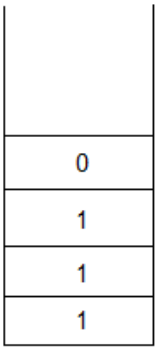

23(10) a binario es 10111

¿Problema?, los datos se generan 11101 y ese no es el 23, entonces se debe invertir los datos, para esto:

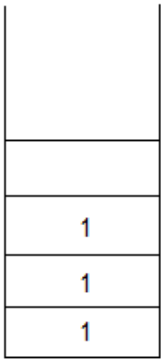
Se generó en orden 11101 > se inserta a la pila uno tras uno mientras se generen y quedara de la siguiente manera:




Entonces al sacar los datos uno por uno se podría realizar los siguiente, se toma una cadena “x” para almacenar los valores, por ejemplo se saca el primer elemento



Y se guarda en “x” momentáneamente “1”, se saca el siguiente elemento y se lo concatena al lado izquierdo y se obtiene:




Y en “x” se obtiene “01”, de igual forma con el resto de los elementos, y se obtendría finalmente “11101”.

Esta solución se demostrara en códigos con el siguiente ejemplo.

*Referencia ejemplo en capitulo222.java y capitulo222.cpp*

Resolver los siguientes ejercicios de la UVA:

| UVA Online judge - Problem 673 - Parentheses Balance |
|---|

### 2.2.3. Colas

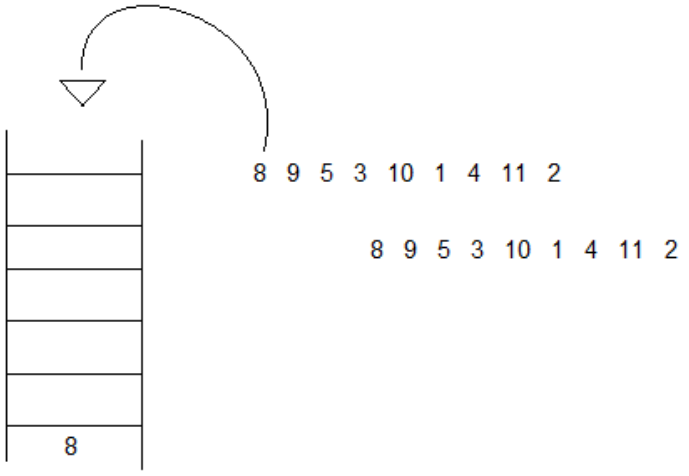
Las colas son una estructura de datos en la que la inserción y eliminación de un elemento se puede realizar por alguno de los extremos de esta, generando variaciones, por ejemplo las colas:

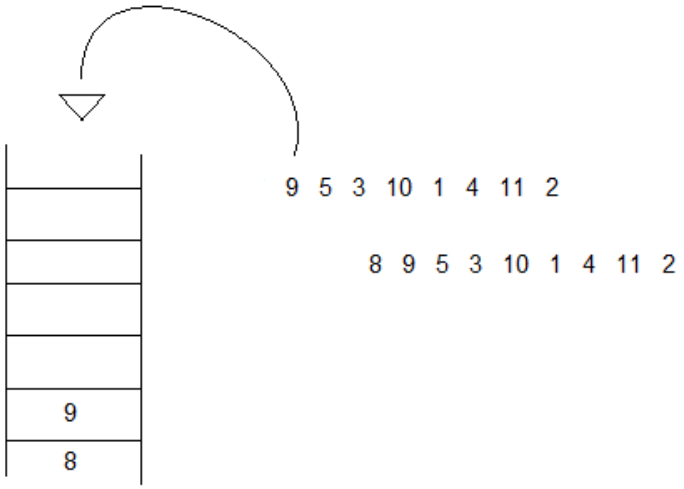
(First input First output) o FIFO, Primero en entrar primero en salir.

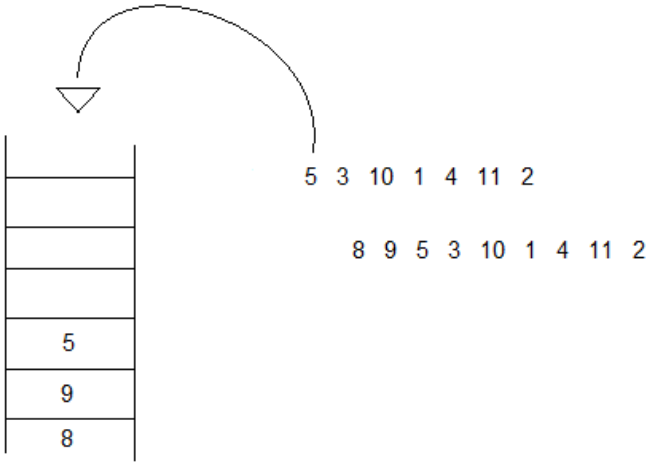
Si se requiere insertar los datos:

8 9 5 3 10 1 4 11 2

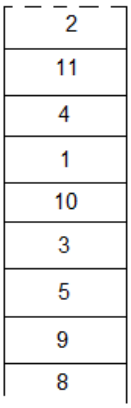
El proceso seria el siguiente:



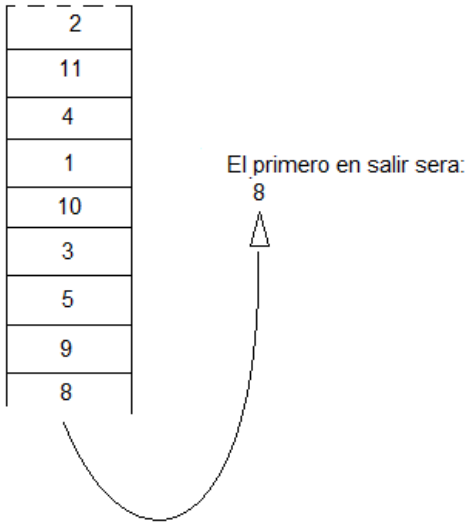



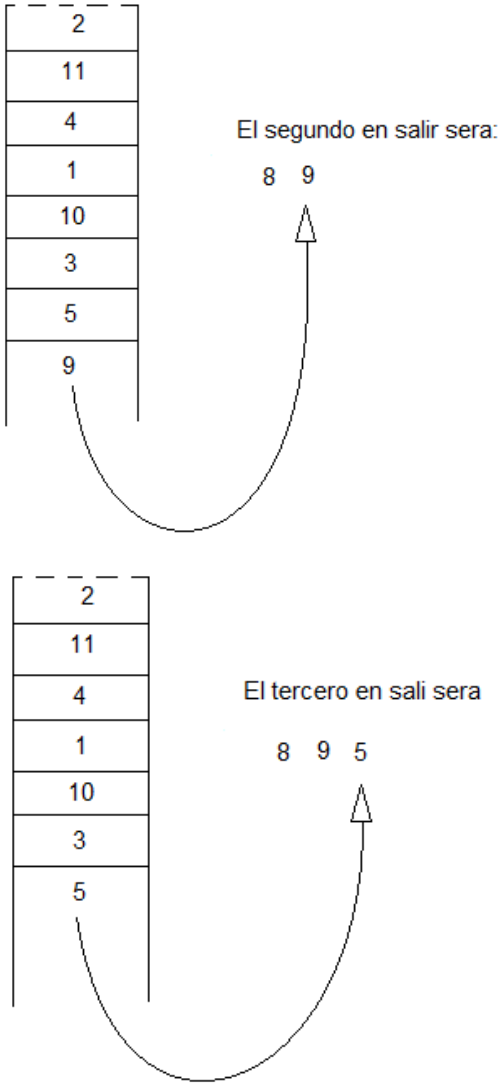



Hasta que quede de la siguiente manera:




Para la eliminación de datos el proceso se lo realizaría de la siguiente forma:






Así se ve que el orden de entrada era: 8 9 5 3 10 1 4 11 2

Y el orden de salida es: 8 9 5 3 10 1 4 11 2

Ejemplo de aplicación:

Se requiere separar paréntesis de llaves pero se debe respetar el orden de entrada:

ENTRADA: }))({})({}))){}}))}(){}{}{}{}(((({()}}){(){(){(){}}}){}{}({{{()}{()}{}}{{}{}{

SALIDA:

PARENTESIS: ))()()))()(()()()((()))()()()))(()((()()()()

LLAVES: }{{}}}}{{}}{}{{{}}}{}{{}{}}}{{{}{

Solución: Se crea 2 colas y en el orden de entrada se compara si es llave o si es paréntesis, asi insertando en su respectiva cola, y por ultimo solo sacamos todo de la cola, y como nos respeta orden de entrada la cola devolverá esa salida deseada.

*Referencia ejemplo en capitulo223.java y capitulo223.cpp*

### 2.2.4. Cola de prioridad

Colas de prioridad ya es una estructura mas compleja ya que es un árbol que se forma al introducir datos a el a diferencia de las pilas y colas este no guarda los elementos por orden de inserción, sino guarda los elementos de menor a mayor o viceversa, tiene 2 propiedades min heap(de menor a mayor) y max heap(de mayor a menor).

Ejemplo:

Se requiere ordenar de mayor 2 pares de elementos, pero los primeros elementos son iguales ordena de mayor a menor por el segundo elemento.

Si se introduce los datos:

2 5

2 2

2 7

2 1

3 1

3 2

La salida será:

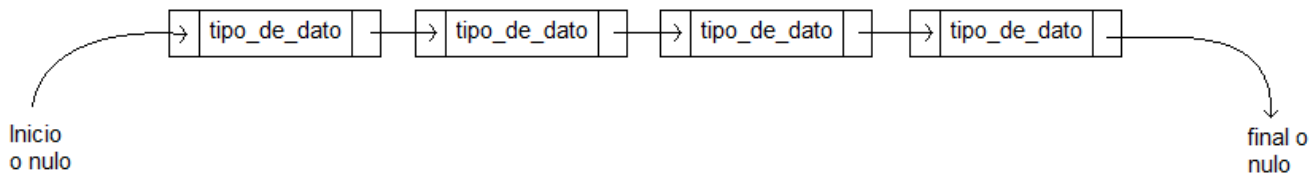
3 2
3 1
2 7
2 5
2 2
2 1

Entonces la solución mas simple será hacerlo con priority queue ya que ordena los datos mientras los ingresa y esto nos ayuda en tiempo de ejecución.

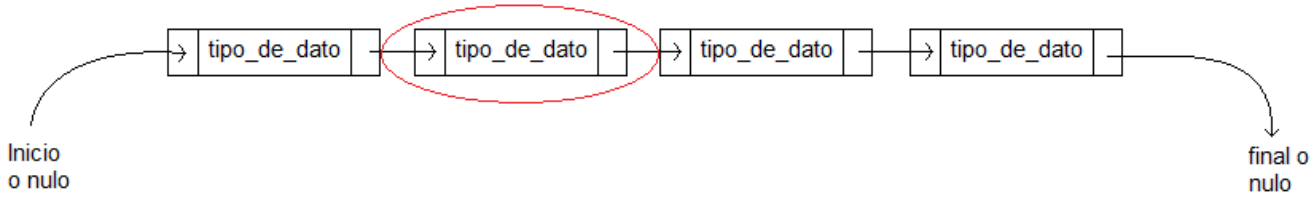
*Referencia ejemplo en capitulo224.java y capitulo224.cpp*

### 2.2.5. Listas enlazadas

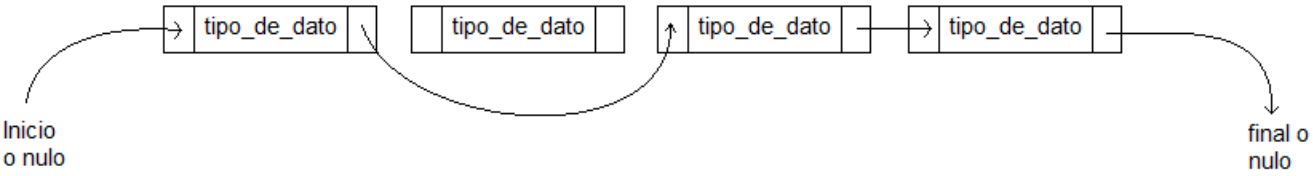
Una lista enlazada es una estructura que apunta a un elemento en específico, y este apunta a otro y así sucesivamente hasta el final.



Si se requiere eliminar




Entonces lo único que hace es:




Es decir para realizar modificaciones a la lista lo que se realiza es una serie de modificaciones en los punteros de un elemento a otro como se ilustro anteriormente con la eliminación de un dato.

*Referencia ejemplo en capitulo225.java y capitulo225.cpp*

### 2.2.6. Sets

Los sets son una estructura tipo árbol (red black o rojo negro, para una explicación e implementación más detallada se recomienda el libro Introduction to Algorithms del MIT) que ordena un tipo de dato al ingresarlo, esta estructura no permite repetidos es decir si se ingresarían los datos:

3 5 6 7 1 4 1 7 6 1

El resultado seria:

1 3 4 5 6 7

¿Por qué el uso de esta estructura?

Esta estructura es eficaz ya que lo hace en tiempo logarítmico, vale la pena aclarar que no acepta datos repetidos, en el caso de necesitar datos duplicados se puede hacer uso de Priority Queue explicado en el punto 2.2.4. .

**Ejemplo:**

El ejercicio pide que se mencione cuantos nombres distintos existen y después se muestren todos esos nombres.

**ENTRADA:**

Miguel
Jose
Alejandro
Pablo
Miguel
Alejandro
Mario
Jose

**SALIDA:**

5

Alejandro

Jose

Mario

Miguel

Pablo

**SOLUCIÓN**: El uso de set en estos problemas es lo recomendado en este tipo de ejercicios.

*Referencia ejemplo en capitulo226.java y capitulo226.cpp*

### 2.2.7. Maps

Es un conjunto de llave, por ejemplo, si se tiene los siguientes datos nombre, edad:

Miguel 21
Jose 22
Alejandro 35
Pablo 28

El nombre es la llave y la edad es el valor, es decir con el nombre se accede a la edad, pero los maps no aceptan repetidos es decir si en el conjunto anterior se inserta:

Jose 25

El map queda de la siguiente manera:

Alejandro 35
Jose 25 -> modifico de 21 a 25
Miguel 21
Pablo 28

¿Por qué el uso de maps?

Es bastante útil ya que elimina los repetidos es decir se asegura que no existan llaves repetidas, además que ordena la entrada como un árbol y con un valor se puede apuntar a un objeto y esto ayuda a tener tanto un acceso rápido como referencias por valor de llave.

**Ejemplo:**

Se requiere un diccionario, para palabras y una descripción, por ejemplo con la palabra pez tendríamos una descripción de la palabra pez, además que debe mostrar los datos ordenados.

Si tuviéramos de entrada para un map los siguientes datos:

| **Llave** | **Valor** |
|---|---|
| Pez | “Descripción total sobre pez” |
| Pez | “Descripción total sobre pez 2” |
| Pez | “Descripción total sobre pez 3” |
| Computadora | “Descripción total sobre Computadora” |
| Sobre | “Descripción total sobre Sobre” |

Obtendríamos lo siguientes resultados:

Si se consulta el tamaño del map se obtendría: 3
Si se consulta por la llave “Computadora” se obtendría “Descripción total sobre

Computadora”

Si se consulta por la llave “Pez” se obtendría “Descripción total sobre pez 3”, acá es importante resaltar que se sobrescribió el contenido y se guardó únicamente la última descripción.

Finalmente si se consulta por la llave “Sobre” obtendríamos “Descripción total sobre Sobre”

*Referencia ejemplo en capitulo227.java y capitulo227.cpp*

## 3. Recursividad

La recursividad es una técnica de programación para llamar a una función desde una misma función, este concepto puede ser algo confuso y es necesario aclararlo, este proceso se realiza en memoria (se crea un árbol), que se genera en segundo plano, con restricciones este proceso termina de manera adecuada, por ejemplo:
Para generar el factorial de un cierto número se realiza una multiplicación sucesiva de sus predecesores y de sí mismo, es decir:

9! = 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1

En formula se podría resumir de la siguiente manera, tomando “n” como el número para obtener el factorial de dicho número se tendría generalizando:

n! = n * (n-1)!

Teniendo como caso base 0, ya que el factorial de 0 es 1 por definición es decir el proceso terminaría cuando n es 0, para ejemplificarlo si se quisiera el factorial de 4 se tendría lo siguiente:

4! = 4 * 3!
3! = 3 * 2!

2! = 2 * 1!

1! = 1 * 0!

0! = 1

Se reemplaza secuencialmente:

1! = 1 * 1 → 1

2! = 2 * 1 → 2

3! = 3 * 2 → 6

4! = 4 * 6 → 24

Generando el factorial de un número con recursividad:
Lo principal es encontrar el algoritmo de recursividad este caso sería:

¿Por qué se denota el 0?

Es importante denotar cuando termina el algoritmo de recursividad en otras palabras dar un caso base para su terminación, en este caso se indicaría que cuando sea 0 termina, es decir devuelve 1, ya que 1 no varía el resultado en la multiplicación y factorial de 0 es 1 como se mencionó anteriormente.

```
int Factorial ( n )   -> función factorial que devuelve un entero
{
        Si n==0 ?    Entonces    devolvemos 1;
        Si no          devolvemos  n*Factorial(n-1)
}
```

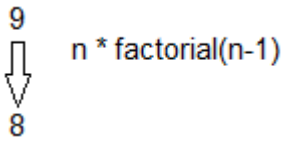
¿Pero, usted entiende como la memoria trabaja la recursividad?

Es importante entender como la memoria trabaja este proceso, porque así se puede entender el cómo modificar este proceso. Se lo explicara con el siguiente ejemplo para obtener el factorial de 9:

Primero se crea:

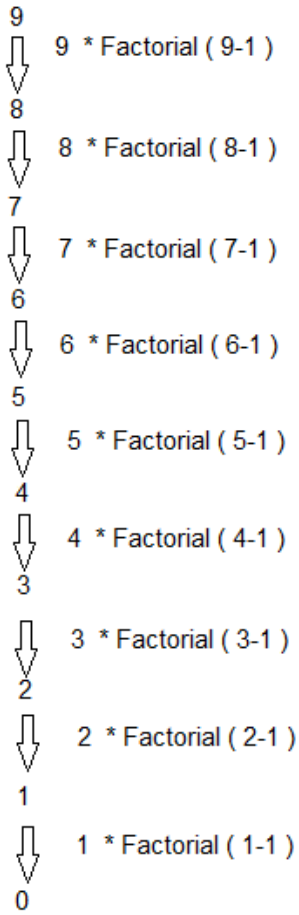
9

Pero como se puede observar si es que no es 0 (como vemos 9 no es 0), retorna n*factorial(n-1)




Aun no realiza la multiplicación ya que necesita Factorial(n-1) para realizarla.

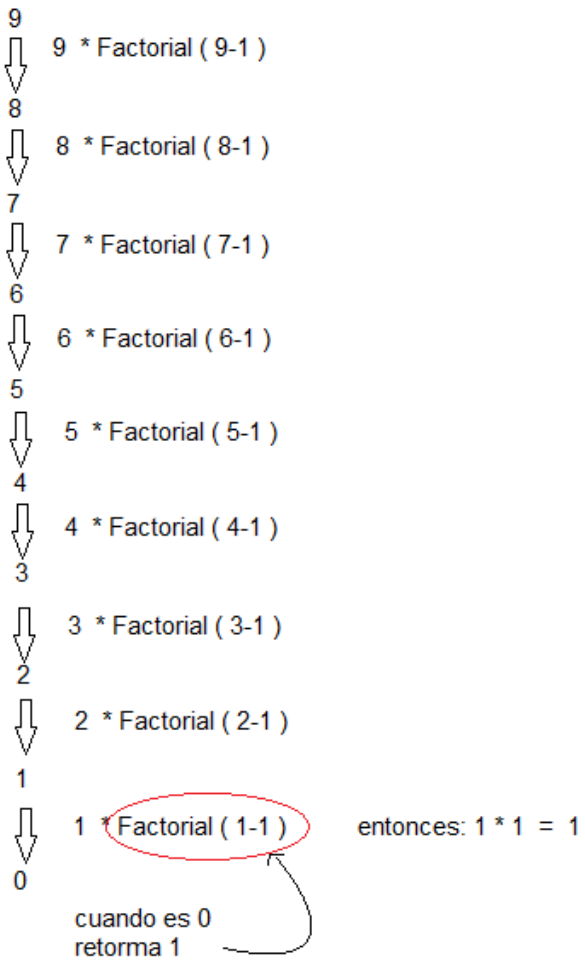
Entonces el algoritmo llegara hasta que el número sea 0

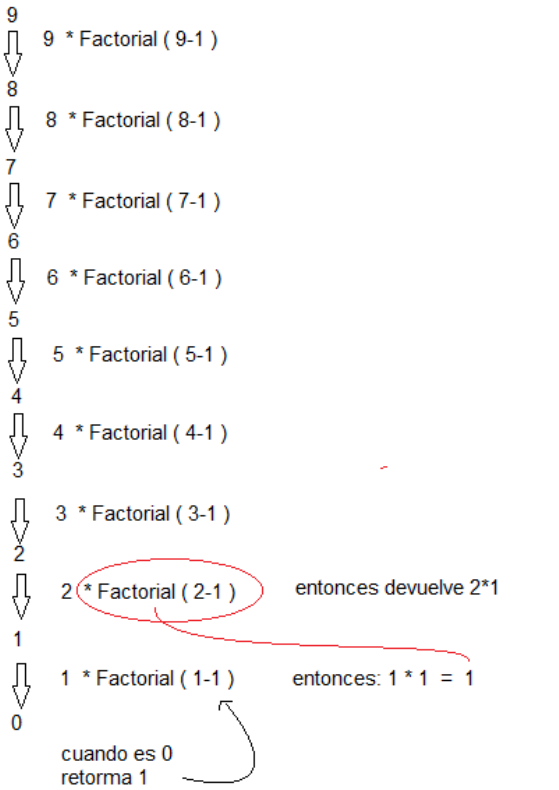


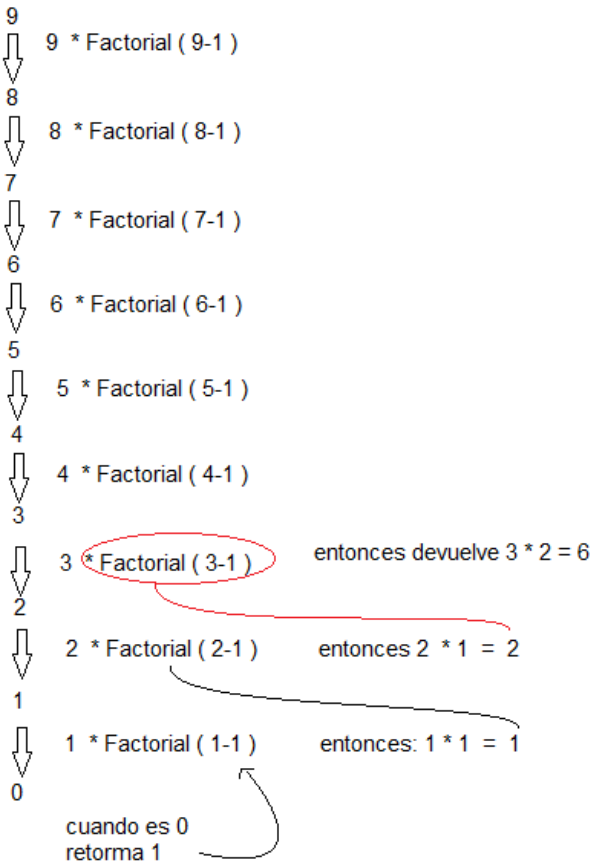

NOTA: Todo este proceso es como se ve en forma de árbol, en la memoria se generó todo

esto, es por eso tan importante entenderlo.

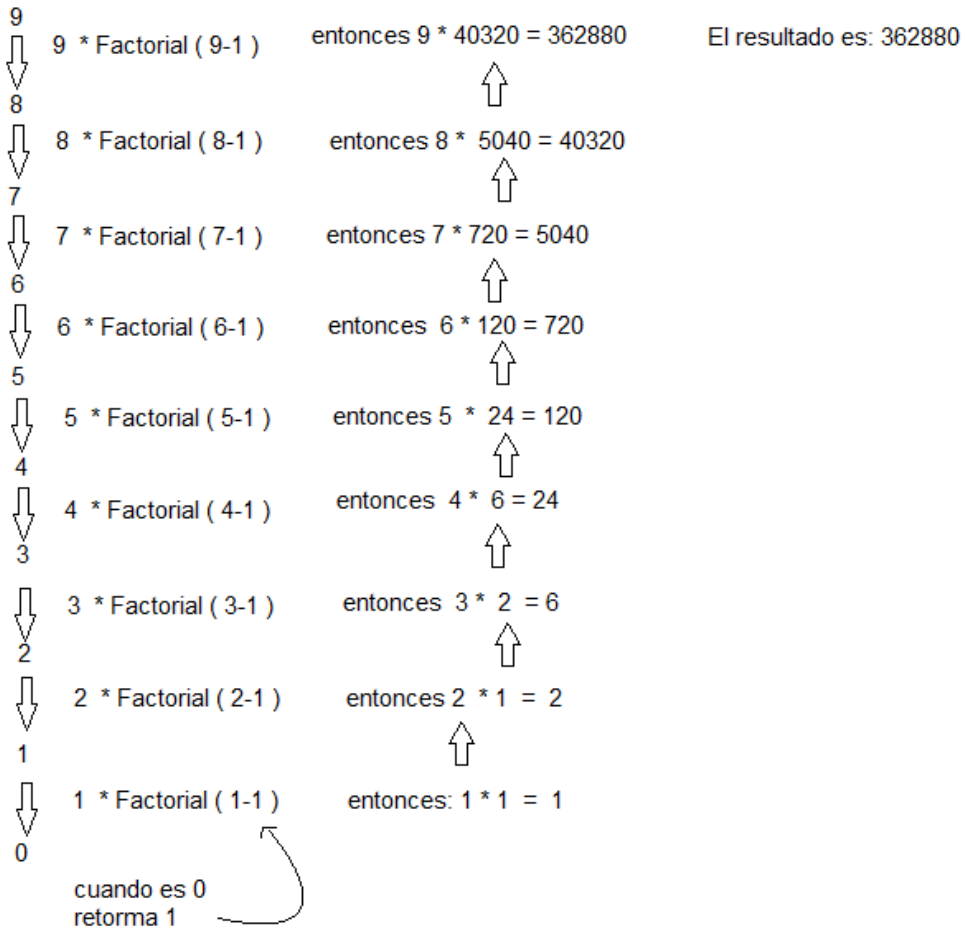
El algoritmo termina cuando es 0 ya no llama a la misma función factorial es decir devuelve 1, es decir, desde ahora comienza a hacer los cálculos:




9
9 * Factorial ( 9-1 )
8
8 * Factorial ( 8-1 )
7
7 * Factorial ( 7-1 )
6
6 * Factorial ( 6-1 )
5
5 * Factorial ( 5-1 )
4
4 * Factorial ( 4-1 )
3
3 * Factorial ( 3-1 )
2
2 * Factorial ( 2-1 )
entonces devuelve 2*1
1
1 * Factorial ( 1-1 )
entonces: 1 * 1 = 1
0
cuando es 0
retorma 1


9
9 * Factorial ( 9-1 )
8
8 * Factorial ( 8-1 )
7
7 * Factorial ( 7-1 )
6
6 * Factorial ( 6-1 )
5
5 * Factorial ( 5-1 )
4
4 * Factorial ( 4-1 )
3
3 * Factorial ( 3-1 )
entonces devuelve 3 * 2 = 6
2
2 * Factorial ( 2-1 )
entonces 2 * 1 = 2
1
1 * Factorial ( 1-1 )
entonces: 1 * 1 = 1
0
cuando es 0
retorma 1

Como puede notar género los datos de arriba hacia abajo, pero comienza a realizar las operaciones de abajo hacia arriba, entonces el proceso en total seria:




Así generando el factorial de 9 que sería: 362880

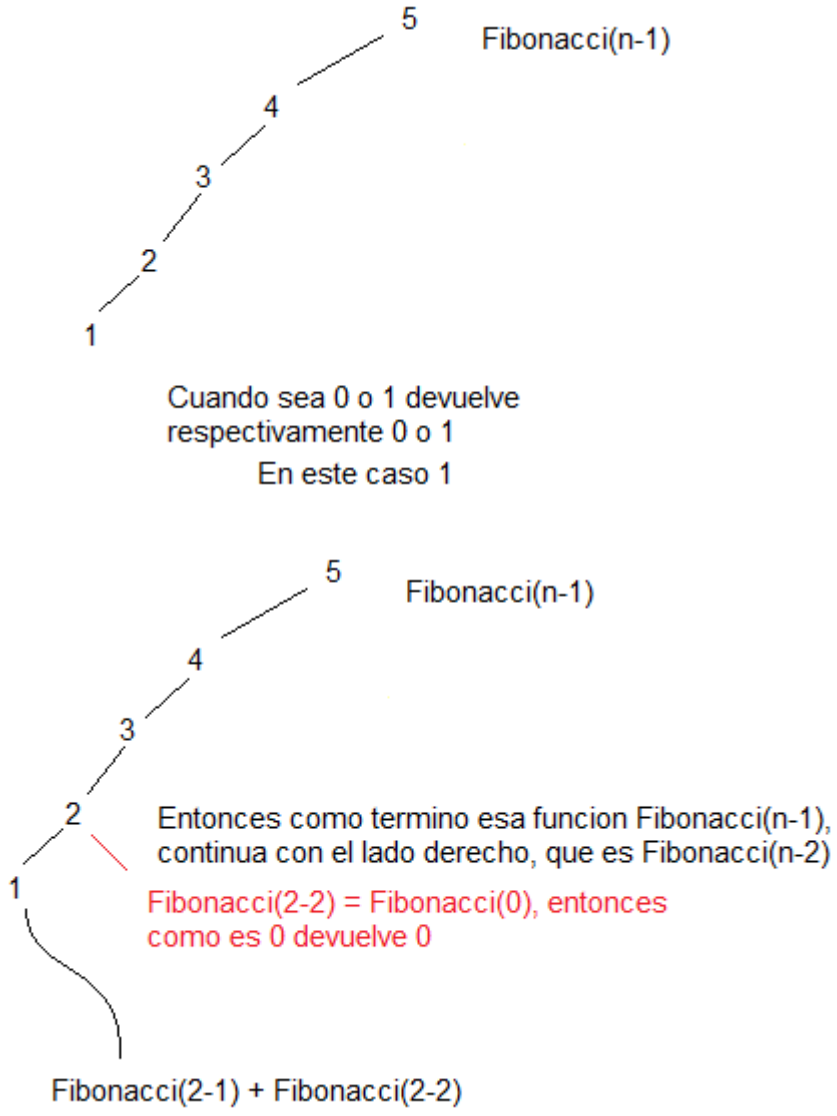
*Referencia ejemplo en factorial.java y factorial.cpp*

Para entender mejor realice la función recursiva que devuelva el n-esimo numero Fibonacci. En caso de no entender se explica la función y se grafica la solución:

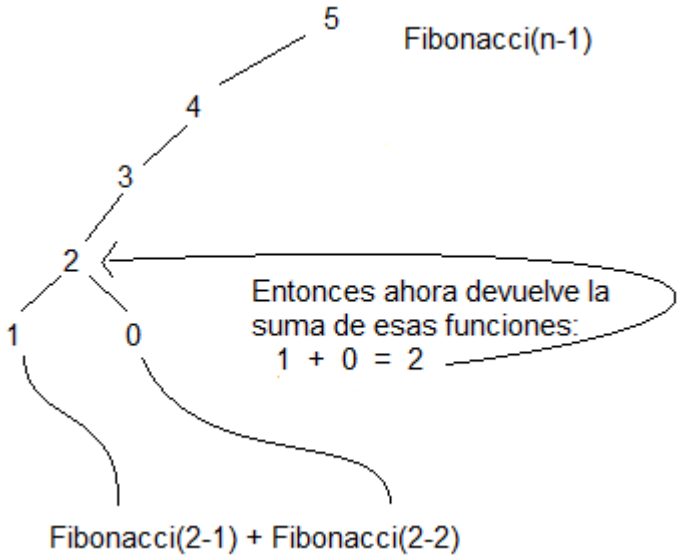
La función seria:

```
int Fibonacci(int n)
    If(n==0 ) devuelve 0;
    If(n==1 ) devuelve 1;
    Si no es ni 1 ni 0 devuelve Fibonacci(n-1)+ Fibonacci(n-2)
```

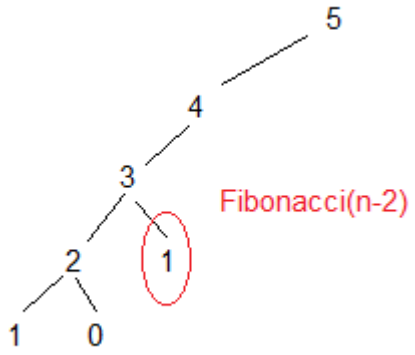
Como estamos llamando 2 veces a la función, ¿Cómo realiza esto la memoria?, primero ira por la primera llamada es decir: Fibonacci(n-1) que esta a la izquierda: ***Fibonacci(n-1)***+ Fibonacci(n-2)

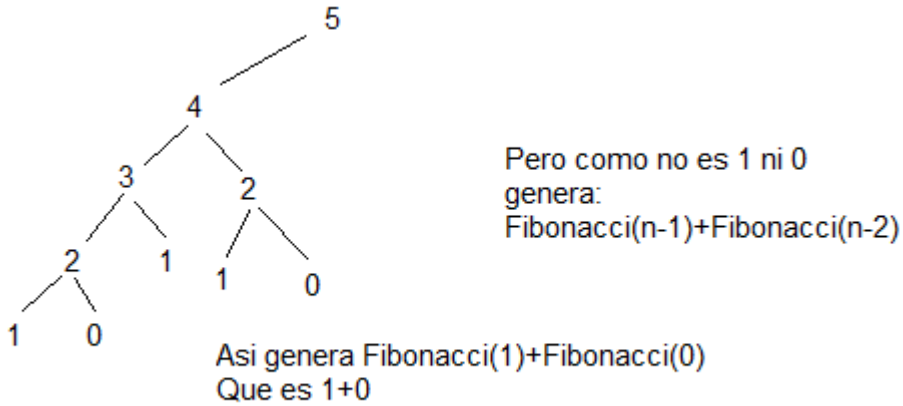


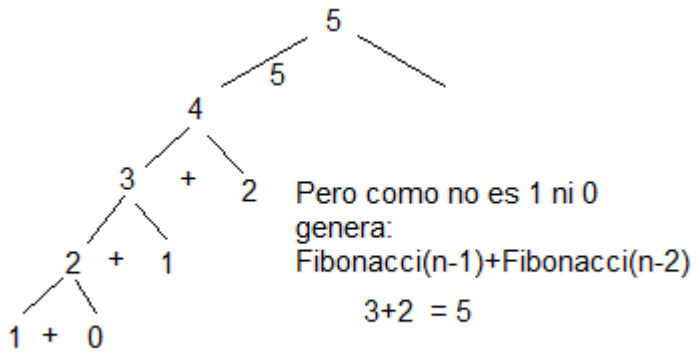
Entonces tendríamos:

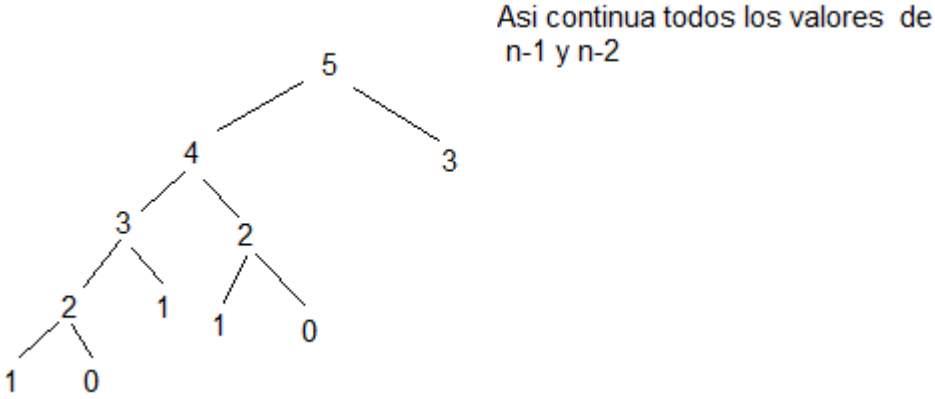



Así sucesivamente:

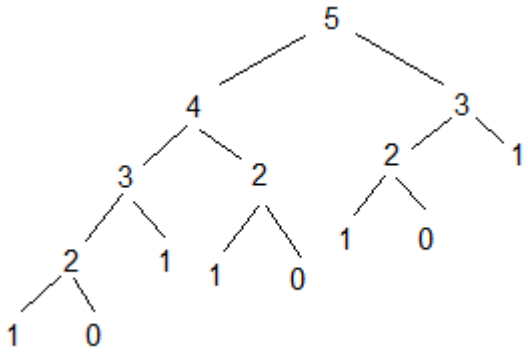


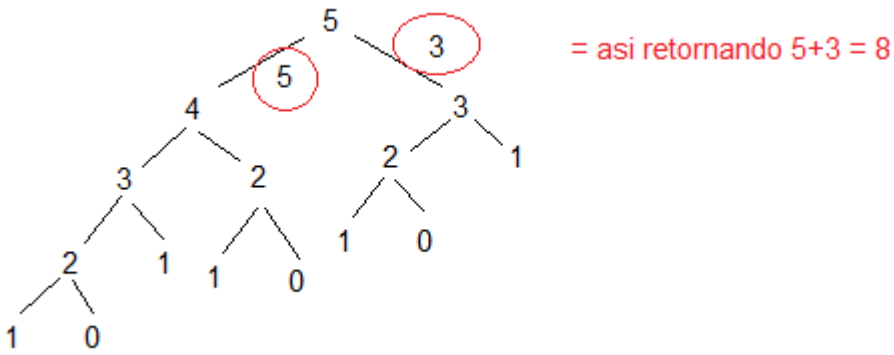






Teniendo por resultado






NOTA: Esta función genera el n-esimo número Fibonacci a partir del término 0 (imagine que genera la posición 0 como lo hacen en los vectores) es decir:

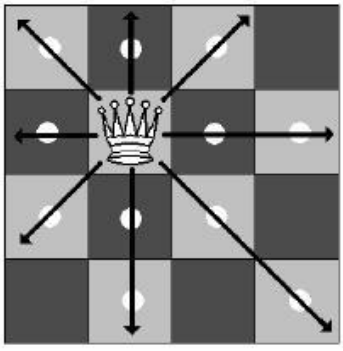
0 1 1 2 3 5, desde 1 1 2 3 5. Entonces al llamar a la función con tal solo hacer n-1, se solucionaría el problema.

Entenderá esto analizando el código.

*Referencia ejemplo en fibonacci.java y fibonacci.cpp*

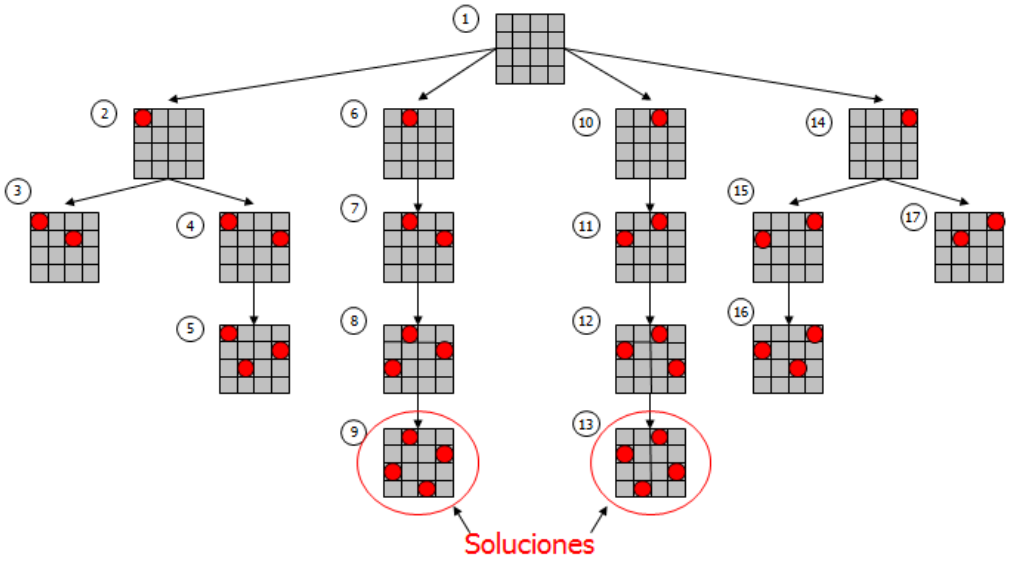
### 3.1. Introducción a Backtracking (Proceso recursivo hacia atrás)

Esta técnica es utilizada cuando se requiere recorrer todo el espacio de soluciones, es decir, un ejemplo: contar todas las posibles soluciones de n-reinas en un tablero (problema que trata de acomodar reinas en el tablero de ajedrez, pero estas reinas no deben acomodarse en posiciones donde puedan atacarse entre ellas).



El problema de backtracking es que es lento ya que probar todas las posibles combinaciones(con recursividad) resultaría un algoritmo exponencial es decir: $a^b$, en el caso de las 8 reinas dependiendo del algoritmo y condiciones seria o bien: $8^8$ (pura fuerza bruta, es decir nada de optimización ni restricciones) o 8! (caso en el que damos todas las restricciones necesarias), back tracking se puede utilizar cuando las posibles combinaciones son menores a 15 (preferentemente 10), es decir en el caso de las 8 reinas el tablero es de 10x10, en casos superiores seria lento.

**Ejemplo para un tablero 4x4:**

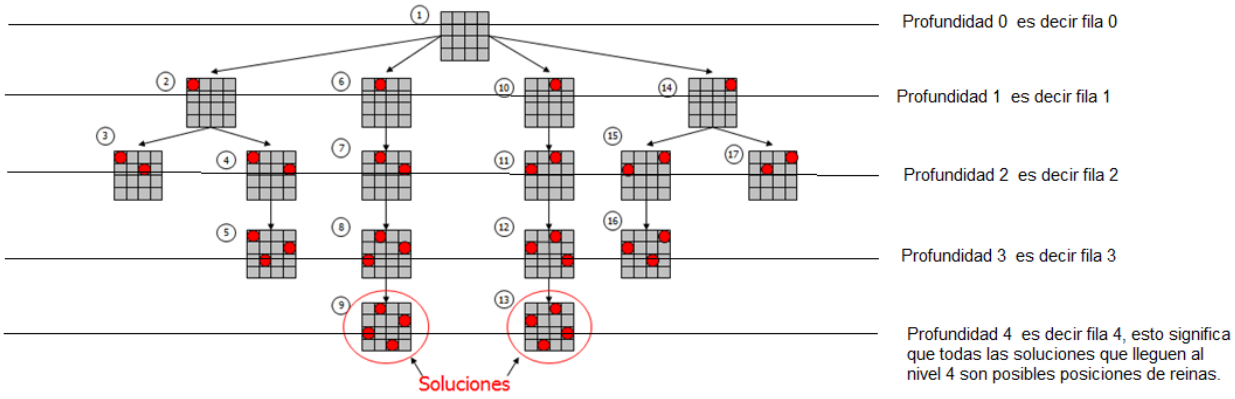



Las soluciones se las puede representar en un vector, una primera solución seria: 2 4 1 3

Es decir:

FILA: 1 2 3 4
COLUMNA: 2 4 1 3

Es decir la profundidad que genera la recursividad seria de 4, como se muestra a continuación.




*Referencia ejemplo en ocho_reinas.java y ocho_reinas.cpp*

## 4. Introducción a la Programación dinámica

Esta técnica que es utilizada para reducir el tiempo de ejecución de un programa computacional se utiliza sub-problemas, sub-estructuras y se va almacenando según pasa la ejecución en memoria sub-ejecuciones para no volver a realizarlas ahorrar tiempo, es aplicada cuando se necesita mayor rapidez en los algoritmos es decir, la búsqueda exhaustiva (búsqueda de todas las posibles soluciones, backtracking o fuerza bruta) no es suficiente, así que puede también ser llamado una búsqueda inteligente y rápida. El objetivo de la programación dinámica es la optimización, es típico encontrar problemas que requieran hallar un máximo o mínimo, o contar distintas estados, son en estos en los cuales la programación dinámica es la mejor manera de resolver estos ejercicios. Los estados que se presentan en los procesos o sub procesos son guardados así para no volverlos a repetir.

Ejemplo:

El problema que se describe a continuación es un problema de un campeonato realizado en la regional de Bolivia, la cual describía que:

Existía un guitarrista que quería saber la máxima y mínima nota a la que podía llegar,

dependiendo del cambio de tonos, donde la mínima nota es 0 y la máxima nota es la máxima capacidad de volumen de la guitarra:

Si los cambios de tonos son:

5 2 3 6 4 5 15 donde su estado inicial es: 5 y su máxima nota es 19 y su mínima nota es 0.
Se ilustra a continuación el estado inicial:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

Entonces el primer cambio será 5, entonces a las notas que podría llegar es 1 y 11:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |

El segundo cambio será 2 entonces los posibles cambios respecto a las 2 notas a las que pudo llegar en el primer cambio serán:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |
| | | V | | | | | | V | | | | V | | | | | | | |

Así sucesivamente hasta:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |
| | | V | | | | | | V | | | | V | | | | | | | |
| | | | | | V | | | | V | | V | | | | V | | | | |

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | V | | V | | | | V | | V | | | | V | | V | | |
| | V | | | | V | | V | | V | | V | | V | | V | | | | V |
| V | | V | | V | | V | | V | | V | | V | | V | | V | | V | |
| | V | | V | | | | | | | | | | | | V | | V | | V |

Entonces la respuesta seria:

1 19, donde 1 es la mínima nota y 19 es la máxima nota.

Aplicando una tabla de memoria (tabla que se utiliza en Programación dinámica) se puede generar rápido la máxima y mínima solución en comparación a buscar esta misma a través de la búsqueda exhaustiva en todo el espacio de soluciones posibles.

Descripción y casos de entrada y salida:

Entrada:

Primero se leerá el número de casos de prueba N, Luego para cada caso leerá M que es la nota máxima de esa canción, luego se leerá P que es la nota inicial en la que comenzó esa canción, luego será K que es el número de cambios que realizara, donde 0 <=M<= 1000, y 0 <=k<= 100.

La siguiente línea contendrá los K cambios que el guitarrista tiene que hacer.

Salida:

Deberá imprimir 2 números que son la mínima y máxima nota a la que el guitarrista podrá llegar en esa canción. Cada cado de prueba deberá estar separado por una línea en blanco.

Entrada

2
19 5 7
5 2 3 6 4 5 15
19 6 5
6 8 7 6 9

Salida:

1 19
8 19

Nota: Intente solucionar el problema por cuenta propia, en caso de no poder se le adjunta la solución del problema.

*Referencia ejemplo en Guitarrista.java y Guitarrista.cpp*

¿Qué solución le daría usted si necesitara hacer 100 000 cambios de tonos?

Si es que se intenta leer una matriz 100 000 X 100 000 se podría dar un desborde de memoria, un truco para estos casos es:

Solo se reserva 2 filas, una la actual y otra la siguiente nota, en este caso 5 es el estado inicial, donde la fila 0 contiene la actual y la fila 1 contiene la siguiente:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

Entonces se cambia de nota a 5, donde la fila actual será 0 y la siguiente 1:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | V | | | | | | | | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |

Pero la fila 0 ya no es de utilidad ya que solo interesa la fila 1, entonces se la elimina:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |

Así ahora la fila actual seria la 1 y la siguiente la 0, ahora el cambio será de 2:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V | | | | | | V | | | | V | | | | | | | |
| V | | | | | | | | | | V | | | | | | | | | |

Entonces la fila 1 es la que ya no nos sirve, entonces la borramos:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V | | | | | | V | | | | V | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

El siguiente cambio de nota será 3, donde la fila actual será 0 y la siguiente 1:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V | | | | | | V | | | | V | | | | | | | |
| | | | | | V | | | | V | | V | | | | V | | | | |

Así sucesivamente.

Entonces como se puede notar ya no se reserva las k líneas sino solo 2, optimizando de esta manera el proceso.

*Referencia ejemplo en Guitarrista2.java y Guitarrista2.cpp*