

CAPÍTULO V

RECURSIVIDAD

1. INTRODUCCIÓN

Un procedimiento o función recursiva es aquella que se llama a si misma. Esta característica permite a un procedimiento recursivo repetirse con valores diferentes en sus parámetros. La recursión es una alternativa a la iteración muy elegante en la resolución de problemas, especialmente si estos tienen naturaleza recursiva.

Normalmente, una solución recursiva es menos eficiente en términos de tiempo de computadora que una solución iterativa debido al tiempo adicional de llamada a procedimientos.

En muchos casos, la recursión permite especificar una solución más simple y natural para resolver un problema que en otro caso sería difícil. Por esta razón la recursión (recursividad) es una herramienta muy potente para la resolución de problemas y la programación.

Se dice que un objeto es recursivo, si en parte está formado por sí mismo o se define en función de sí mismo.

Un objeto recursivo es aquel que forma parte de si mismo. Esta puede servir de ayuda para la definición de conceptos matemáticos. Así, la definición del conjunto de los números naturales es aquel conjunto en el que se cumplen las siguientes características.

- 0 es un número natural.
- El siguiente número de un número natural es otro número natural

Mediante una definición finita hemos representado un conjunto infinito.

El concepto de la recursividad es muy importante en programación. La recursividad es una herramienta muy eficaz para resolver diversos tipos de problemas; existen muchos algoritmos que se describirán mejor en términos recursivos.

Suponga que dispone de una rutina o que contiene una secuencia de llamada a si misma, o bien una sentencia de llamada a una segunda rutina que a su vez tiene una sentencia

de llamada a la rutina original. Entonces se dice que la rutina original es una rutina recursiva.

“... Como el costo de la programación aumenta con regularidad y el costo de la computación disminuye, hemos llegado al punto en donde la mayoría de los casos no vale la pena que un programador desarrolle laboriosamente una solución no recursiva para un problema que se resuelve con mayor naturalidad en forma recursiva”.

2. RECURSIÓN VERSUS ITERACIÓN

¿Cuándo debemos utilizar la iteración y cuándo la recursión? Hay por lo menos estos tres factores a considerar:

- 1) Las funciones iterativas son usualmente más rápidas que sus contrapartes recursivas. Si la velocidad es importante, normalmente usaríamos la iteración.
- 2) Si la memoria de pila es un limitante, se preferirá la iteración sobre la recursión.
- 3) Algunos procedimientos se programan de manera recursiva de forma muy natural, y resultan prácticamente inabordables iterativamente. Aquí la elección es clara.

3. RECURSIVIDAD

La recursividad desde el punto de vista de la programación es una poderosa herramienta que facilita la implementación de algunos algoritmos que de otro modo se harían muy complejos.

El ejemplo clásico de esto último es el recorrido de árboles.

4. FORMA DE ENFRENTAR PROBLEMAS RECURSIVOS

Propiedades

Un procedimiento o función recursivos han de cumplir dos propiedades generales para no dar lugar a un bucle infinito con las sucesivas llamadas:

- **Cumplir cierta condición o criterio base**, del que dependa la llamada recursiva.
- Cada vez que el procedimiento o función se llamen a si mismos, directa o indirectamente, debe **estar más cerca del cumplimiento de la condición** de que depende la llamada.

Debe existir un caso BASE.

Este debe ser un caso en que la función se puede computar en forma simple (sin llamar a ninguna "copia" de ella). Este es el caso en que se **detiene** la recursión.

Se debe confiar en que, al llamar a una **copia** de la función, **esa función va a resolver el problema** que le entregamos. No importa cuán difícil sea ese problema, debemos creer que lo soluciona.

Hay que verificar que el problema "más pequeño" sea efectivamente **más pequeño**, es decir, que el problema esté más cerca de solucionarse. La idea es que sepamos que en cada llamada recursiva estamos más cerca del **caso base**.

Ejemplo: La función factorial, se define en matemática como:

$$N! = n*(n-1)*(n-2)*(n-3)*.....3.2.1 \quad \text{donde} \quad 0! = 1$$

Factorial Iterativo:

A continuacion se muestra la función sin recursividad de la función factorial.

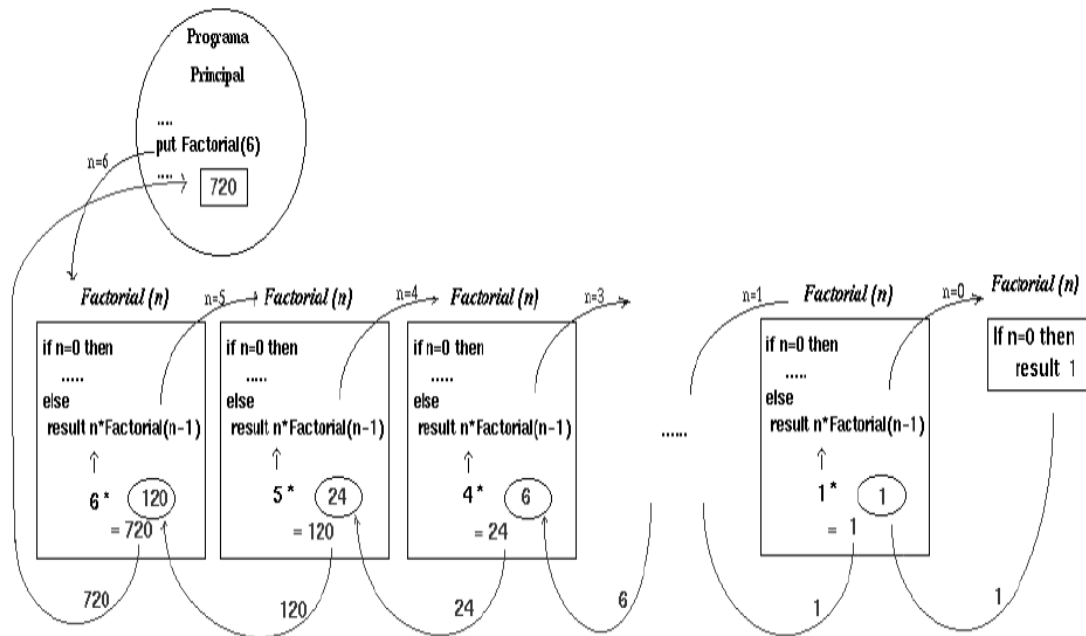
```
public static int factorial (int n)
{
    int fact = 1;
    while (n >= 1)
    {
        fact *= n;
        n--;
    }
    return fact;
}
```

Factorial Recursivo:

La function factorial en forma recursive, se define:

$$N! = \begin{cases} 1 & \text{si } N = 0 \\ N * (N - 1)! & \text{si } N > 0 \end{cases}$$

```
public static int factorial (int n)
{
    if (n==0)
        return 1;
    else
        return n * factorial (n-1);
}
```

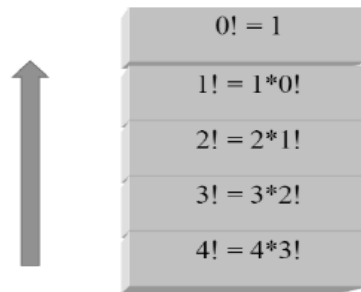


¿CÓMO FUNCIONA?

Es necesario llevar un registro de todas las operaciones que se dejan pendientes hasta que se llega al caso base.

El patrón que sigue este registro es el de una pila, en donde la operación original se deja en el fondo y en el tope aparecería el caso base.

Una vez que se llenó la pila, se efectúa la operación de vaciar la pila efectuando todos los cálculos que previamente se habían dejado pendientes.



En la gráfica anterior se observa claramente el comportamiento de pila de la implementación recursiva, en esta versión simplificada se omiten los detalles del funcionamiento al nivel de la máquina.

Para ser realistas, la recursión no solo pasa una copia de los parámetros (operandos de los cálculos pendientes), sino que también debe pasar una copia de las direcciones de llamada de función en cada paso para regresar al punto original del programa donde se inició el proceso de recursión.

Por lo general, los lenguajes de alto nivel como el C, C++ o Java, enmascaran todos los detalles de la implementación. A pesar de lo anterior, es necesario tomarlo en cuenta debido a que la pila exige una gran cantidad de recursos (memoria y tiempo) de la máquina a la hora de ponerla a funcionar.

Ejemplo.- Serie de Fibonacci

Fibonacci (n) = Fibonacci (n-1) + Fibonacci (n-2)

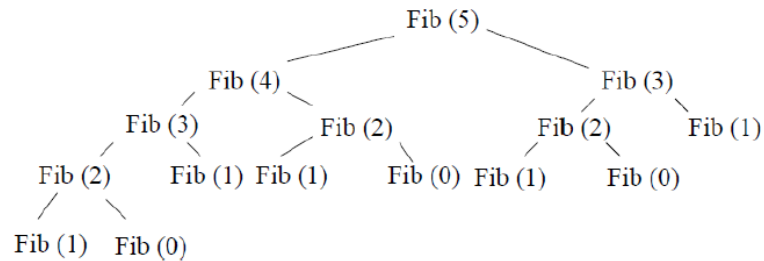
$$\text{Fibonacci}(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2) & \text{si } n \geq 2 \end{cases}$$

Fibonacci recursivo

```
public static int fibonacci (int n)
{
    if (n <= 1)
        return n;
    else
        return fibonacci(n-1) + fibonacci(n-2);
}
```

¿ES EFICIENTE LA IMPLEMENTACIÓN?

Existen muchas llamadas redundantes en la implementación recursiva de Fibonacci:



Para calcular:

- fibonacci (5) requiere de 15 llamadas recursivas.
- fibonacci(35) requiere de 29,860,703 llamadas recursivas (tiempo exponencial de ejecución).

Por lo tanto:

Utiliza recursión cuando se mejora el diseño del algoritmo y se ejecuta en un tiempo y espacio razonable.

Función recursiva de Fibonacci

```
int Fib( int n )
{
    if ( n <=1)
        return(1);
    else
        return ( Fib( n-1) + Fib(n-2) );
}
```

5. TIPOS DE RECURSIVIDAD

A) Recursividad simple: Aquella en cuya definición sólo aparece una llamada recursiva. Se puede transformar con facilidad en algoritmos iterativos.

Ejemplo: Función factorial recursiva

B) Recursividad múltiple: Se da cuando hay más de una llamada a sí misma dentro del cuerpo de la función, resultando más difícil de hacer de forma iterativa.

Ejemplo: Función fibonacci recursiva

C) Recursividad anidada: En algunos de los argumentos de la llamada recursiva hay una nueva llamada a sí misma.

Ejemplo: Función Ackerman recursiva

```
int Ack ( int n, int m ) /* ej: Ackerman */
{
    if ( n == 0 )
        return (m+1)
    else
        if ( m == 0 )
            return (Ack (n-1,1) );
        return( Ack (n-1, Ack(n,m-1) ) );
}
```

D) Recursividad cruzada o indirecta: Son algoritmos donde una función provoca una llamada a sí misma de forma indirecta, a través de otras funciones.

Ejemplo: Función Par o Impar:

```
int par ( int nump )
{
    if ( nump == 0 )
        return(1);
    return (impar (nump-1) );
}
```

```
int impar ( int numi )
{
    if ( numi == 0 )
        return(0);
    return( par ( numi -1) );
}
```

6. LA PILA DE RECURSIÓN

La memoria del ordenador se divide (de manera lógica, no física) en varios segmentos (4):

Segmento de código: Parte de la memoria donde se guardan las instrucciones del programa en código Máquina.

Segmento de datos: Parte de la memoria destinada a almacenar las variables estáticas.

Montículo: Parte de la memoria destinada a las variables dinámicas.

Pila del programa: Parte destinada a las variables locales y parámetros de la función que está siendo ejecutada.

Se reserva espacio en la pila para los parámetros de la función y sus variables locales.

- Se guarda en la pila la dirección de la línea de código desde donde se ha llamado a la función.
- Se almacenan los parámetros de la función y sus valores en la pila.
- Al terminar la función, se libera la memoria asignada en la pila y se vuelve a la instrucción actual.
- En el caso recursivo, cada llamada genera un nuevo ejemplar de la función con sus correspondientes objetos locales:
- La función se ejecutará normalmente hasta la llamada a sí misma. En ese momento se crean en la pila nuevos parámetros y variables locales.
- El nuevo ejemplar de función comienza a ejecutarse.
- Se crean más copias hasta llegar a los casos bases, donde se resuelve directamente el valor, y se va saliendo liberando memoria hasta llegar a la primera llamada (última en cerrarse)

Ejemplo.- Función potencia recursiva

$$3^2 = 9; \quad 3^0 = 1$$

```
float xelevn ( float base, int exp )
{
    if (exp == 0 )
        return(1);
    return( base * xelevn (base , exp-1));
}
```

Ejemplo.- Multiplicar 2 numero naturales, con sumas sucesivas recursivas:

$$3*2 = 6$$

```
int multi( int a, int b )
{
    if ( b == 0 )
        return(0);
    return( a + multi(a, b-1) );
}
```


Ejemplo.- Crear el programa para la aplicación de recursividad, AppRecursividad, en la misma codificar las siguientes funciones recursivas:

- a) Factorial de un numero.
- b) Multiplicación de dos números naturales.
- c) Calcular la potencia de un numero.
- d) Hallar la sumatoria de N términos para $1+1/3+1/5+1/7+\dots$

Programa 23 en Java

```
package ejercicios;
import java.util.Scanner;
public class AppRecursividad
{
    static Scanner sc=new Scanner(System.in);
    //Funcion factorial de N
    static int factorial(int N)
    {
        if (N==0)
            return 1;
        else
            return N*factorial(N-1);
    }

    // Funcion potencia recursiva
    static float xelevn ( float base, int exp )
    {
        if (exp == 0 )
            return(1);
        return( base * xelevn (base , exp-1));
    }

    // Funcion multiplicación de naturales recursiva
    static int multiplicar( int a, int b )
    {
        if ( b == 0 )
            return(0);
        return( a + multiplicar(a, b-1) );
    }
}
```

//Función sumatoria recursiva de N terminos 1+1/3+1/5+1/7+....

```
static float sumatoria(int N)
{
    if (N==1)
        return 1;
    else
        return sumatoria (N-1)+(float)1/(float)((N*2)-1);
}
```

//Funcion principal

```
public static void main(String[] args)
{
    int opcion, num, num2,R;
    int e;
    float b,P;
    do{
        System.out.print("\n\n M E N U R E C U R S I V I D A D \n 1. Factorial \n 2.
        Potencia \n 3. multiplicar \n 4. sumatoria de N terminos 1+1/3+1/5+...");
        System.out.print("\n 5. sumatoria de N terminos 3+6+9+12+...\n 6.Salir \n\n Elija su
        opcion...");
        opcion=sc.nextInt();
        switch(opcion)
        {
            case 1:
                System.out.println("\n\n Ingrese el numero para obtener su factorial...");
                num=sc.nextInt();
                R=factorial(num);
                System.out.println("El factorial de num es "+R);
                break;
            case 2:
                System.out.print("\n\n Ingrese la base ");
                b=sc.nextInt();
                System.out.print("Ingrese el exponente");
                e=sc.nextInt();
                P=xelevn(b,e);
                System.out.print(b+ "elevado a "+e+" es "+P);
                break;
            case 3:
                System.out.println("\n\n Ingrese el primer numero...");
                num=sc.nextInt();//lee un entero el nextInt() para un real es nextFloat(), cadena
                se lee como next() una linea se lee con nextLine()
                System.out.println("Ingrese el segundo numero...");
                num2=sc.nextInt();
                R=multiplicar(num,num2);
                System.out.println("la multiplicacion es "+R);
        }
    }
}
```

```
        break;
    case 4:
        System.out.println("Cuantos terminos desea sumar?? ");
        num=sc.nextInt();
        P=sumatoria(num);
        System.out.println("la sumatoria de 1 + 1/3+ 1/5 + 1/7+...es "+P); break;
    default:
        System.out.println("fin del programa");
    }
} while (opcion!=6);
}
```

Salida del programa



```
Console
AppRecursividad [Java Application] C:\Program Files (x86)\Java\jre7\bin\jav...

  MENU  R E C U R S I V I D A D
1. Factorial
2. Potencia
3. multiplicar
4. sumatoria de N terminos 1+1/3+1/5+...
5. sumatoria de N terminos 3+6+9+12+...
6.Salir

Elija su opcion...1

Ingrese el numero para obtener su factorial...
5
El factorial de num es 120

  MENU  R E C U R S I V I D A D
1. Factorial
2. Potencia
3. multiplicar
4. sumatoria de N terminos 1+1/3+1/5+...
5. sumatoria de N terminos 3+6+9+12+...
6.Salir

Elija su opcion...4
Cuantos terminos desea sumar??
5
la sumatoria de 1 + 1/3+ 1/5 + 1/7+...es 1.7873018
```

EJERCICIOS PROPUESTOS PARA LA PRÁCTICA # 6

En cada uno de los ejercicios se pide implmentar en el lenguaje Java las Funciones Recursivas correspondientes invocándolas desde el programa principal:

- 1) Hallar el máximo común divisor
- 2) Imprimir los N primeros números naturales
- 3) Realizar en forma recursiva la función de Stirling
- 4) Implementar en forma recursiva la función de ackerman
- 5) Implementar en forma recursiva el problema de las torres de Hanoi

CAPÍTULO V

ÁRBOLES

1. INTRODUCCIÓN

Los árboles junto con los grafos constituyen estructuras de datos no lineales. Las listas enlazadas tienen grandes ventajas o flexibilidad sobre la representación contigua de estructura de datos (los arrays), pero tienen una gran debilidad: son listas secuenciales; es decir, están dispuestas de modo que es necesario moverse a través de ellas, una posición cada vez. Los árboles superan estas desventajas utilizando los métodos de punteros y listas enlazadas para su implementación. Las estructuras de datos organizadas como árboles serán muy valiosas en una gama grande de aplicaciones, sobre todo problemas de recuperación de información.

2. DEFINICIÓN DE ÁRBOL

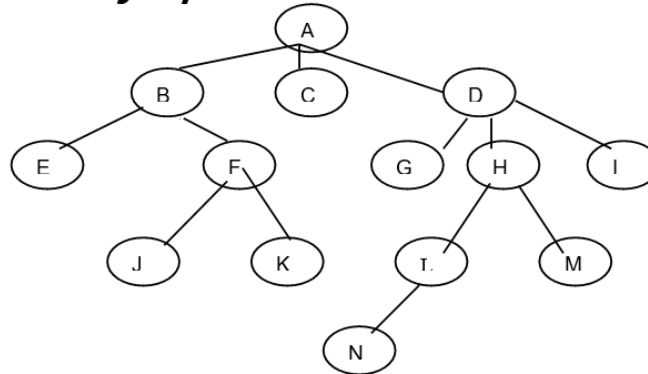
*“Un **árbol** es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos”.*

Un árbol (en inglés, tree) es una estructura que organiza sus elementos, denominados nodos, formando jerarquías. Los científicos utilizan los árboles generales para representar relaciones. Fundamentalmente, la relación clave es la de “padre - hijos” entre los nodos del árbol.

Un árbol es una estructura no lineal, consta de un conjunto de nodos juntos con una relación que impone una estructura jerárquica sobre los nodos y cumple las siguientes condiciones:

- A) Existe un solo nodo llamado raíz.
- B) El resto de los nodos se distribuyen en $n \geq 0$ conjuntos disjuntos $T_1, T_2, T_3, \dots, T_n$ donde cada uno de estos es a su vez un árbol y se los denomina sub-árboles.
- C) Se incluye la existencia del árbol nulo.

Ejemplo: Sea el árbol T

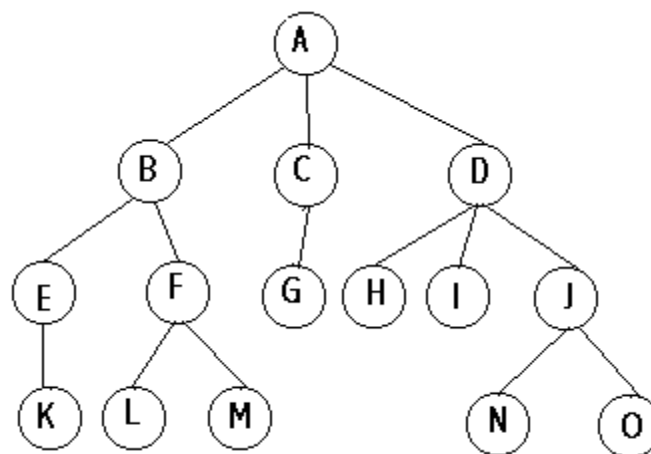


Número de nodos = 14
Subárbol izquierdo T1 = {B,E,F,J,K}
Subárbol central T2 = {C}
Subárbol derecho T3 = {D,G,H,L,M}

Debido a la naturaleza jerárquica de los árboles, se puede utilizar para representar en formación que sea jerárquica por naturaleza, por ejemplo, diagramas de organizaciones (Organigramas), árboles genealógicos, árboles de especies animales, etc.

3. CONCEPTOS BÁSICOS RELACIONADOS

Sea el árbol siguiente:



A) Nodo Hijo

Nodo hijo: cualquiera de los nodos apuntados por uno de los nodos del árbol. En el ejemplo, 'L' y 'M' son hijos de 'F'.

B) Nodo Padre

Nodo padre: nodo que contiene un puntero al nodo actual. En el ejemplo, el nodo 'A' es padre de 'B', 'C' y 'D'.

Los árboles con los que trabajaremos tienen otra característica importante: cada nodo sólo puede ser apuntado por otro nodo, es decir, cada nodo sólo tendrá un padre. Esto hace que estos árboles estén fuertemente jerarquizados, y es lo que en realidad les da la apariencia de árboles.

En cuanto a la posición dentro del árbol:

C) Nodo Raíz: nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol. En el ejemplo, ese nodo es el 'A'.

D) Nodo Hoja (Terminal): nodo que no tiene hijos. En el ejemplo hay varios: 'G', 'H', 'I', 'K', 'L', 'M', 'N' y 'O'.

E) Nodo No Terminal (Terminal): Son aquellos nodos que tienen descendientes

F) Nodo Rama: aunque esta definición apenas la usaremos, esos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'F' y 'J'.

Existen otros conceptos que definen las características del árbol, con relación a su tamaño:

G) Orden: es el número potencial de hijos que puede tener cada elemento de árbol. De este modo, diremos que un árbol en el que cada nodo puede apuntar a otros dos es de orden dos, si puede apuntar a tres será de orden tres, etc.

H) Grado de un Árbol: el número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es tres, ya que tanto 'A' como 'D' tienen tres hijos, y no existen elementos con más de tres hijos.

I) Grado de un Árbol: El máximo de la aridad de sus nodos (el grado máximo de los nodos del árbol).

J) Nivel: se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raíz es cero y el de sus hijos uno. Así sucesivamente. En el ejemplo, el nodo 'D' tiene nivel 1, el nodo 'G' tiene nivel 2, y el nodo 'N', nivel 3.

K) Altura de un Nodo: Es la longitud del camino más largo desde ese nodo hasta un nodo terminal.

L) Altura de Árbol: la altura de un árbol se define como el nivel del nodo de mayor nivel. Como la altura de un árbol se define como el nivel del nodo de mayor nivel. Como cada nodo de un árbol puede considerarse a su vez como la raíz de un árbol, también podemos hablar de altura de ramas. El árbol del ejemplo tiene altura 3, la rama 'B' tiene altura 2, la rama 'G' tiene altura 1, la 'H' cero, etc.

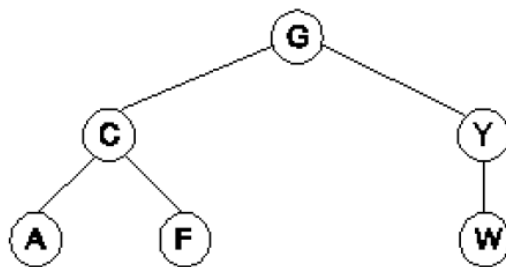
Es la altura de la raíz, es la longitud del camino más largo del nodo raíz.

M) Profundidad de un Nodo: Es la longitud del camino desde la raíz hasta ese nodo.

4. CARACTERÍSTICAS

A los arboles ordenados de grado dos se les conoce como arboles binarios ya que cada nodo del árbol no tendrá más de dos descendientes directos. Las aplicaciones de los arboles binarios son muy variadas ya que se les puede utilizar para representar una estructura en la cual es posible tomar decisiones con dos opciones en distintos puntos.

La representación gráfica de un árbol binario es la siguiente:

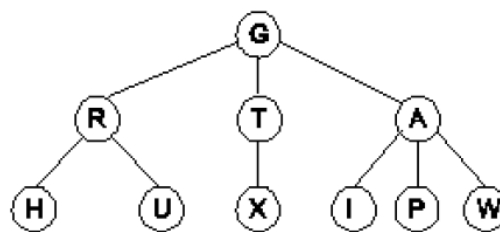


Los arboles representan las estructuras no lineales y dinámicas de datos más importantes en computación. Dinámicas porque las estructuras de árbol pueden cambiar durante la ejecución de un programa. No lineales, puesto que a cada elemento del árbol pueden seguirle varios elementos.

Los arboles pueden ser contruidos con estructuras estáticas y dinámicas. Las estáticas son arreglos, registros y conjuntos, mientras que las dinámicas están representadas por listas.

La definición de árbol es la siguiente: es una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos; uno de los cuales es conocido como raíz. además se crea una relación o parentesco entre los nodos dando lugar a términos como padre, hijo, hermano, antecesor, sucesor, ancestro, etc.. Formalmente se define un árbol de tipo T como una estructura homogénea que es la concatenación de un elemento de tipo T junto con un número finito de arboles disjuntos, llamados subarboles. Una forma particular de árbol puede ser la estructura vacía.

La figura siguiente representa a un árbol general.



Se utiliza la recursión para definir un árbol porque representa la forma más apropiada y porque los arboles tienen una gran variedad de aplicaciones.

Por ejemplo, se pueden utilizar para representar fórmulas matemáticas, para organizar adecuadamente la información, para construir un árbol genealógico, para el análisis de circuitos eléctricos y para numerar los capítulos y secciones de un libro, además es una característica inherente de los mismos

5. ARBOL BINARIO

Es un conjunto de nodos el cual puede ser vacío, o puede contener una raíz y dos sub-árboles denominados sub-árbol izquierdo y derecho. Cada uno de estos sub-árboles es, a su vez, un árbol binario.

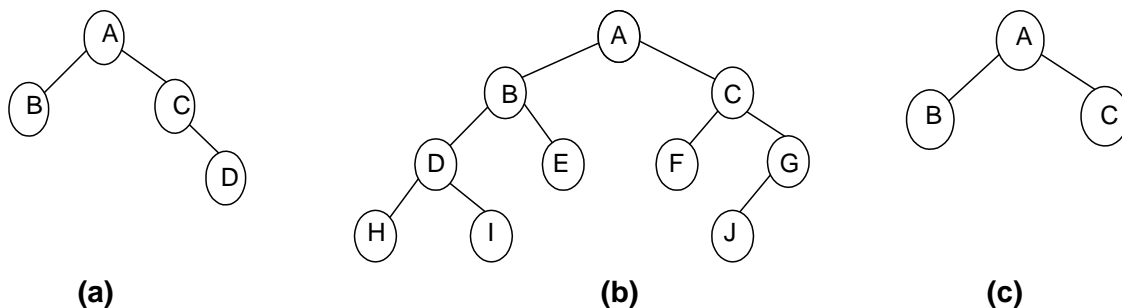


Figura 1. Diversos tipos de árboles binarios

La definición de árbol conlleva el hecho de que un árbol binario es un tipo de estructura de datos recursiva. Esto es, cada sub-árbol se define como un árbol más simple. La naturaleza recursiva del árbol binario ayuda a simplificar la operación con árboles binarios.

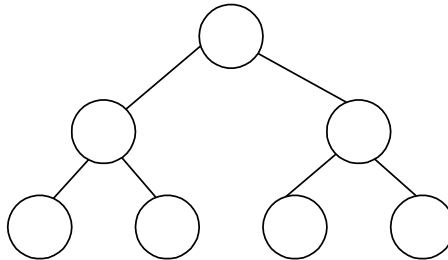


Figura 2. Arbol lleno de altura 3

Un árbol binario lleno es aquel en el que cada nodo tiene o dos hijos o ninguno si es una hoja.

Un **árbol binario completo es equilibrado**, mientras que un árbol binario lleno es totalmente equilibrado.

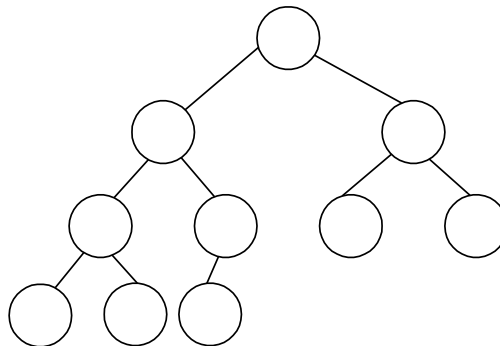


Figura 3. Arbol Binario Completo

Si un **árbol binario es lleno**, es necesariamente completo.

Un árbol binario es completamente (totalmente) equilibrado si los subárboles izquierdo y derecho de cada nodo tienen la misma altura.

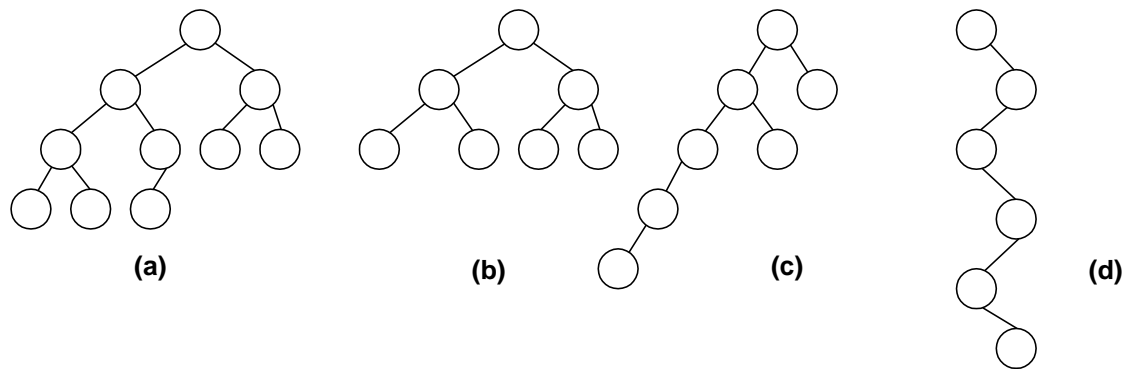


Figura 4. Arbol Binario (a) Equilibrado, (b) Completamente Equilibrado, (c) y (d) Arboles no Equilibrados

6. TRANSFORMACIÓN DE UN ÁRBOL GENERAL EN UN ÁRBOL BINARIO

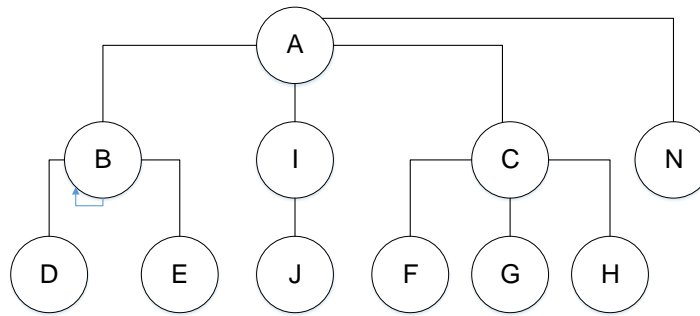
Estableceremos los mecanismos necesarios para convertir un árbol general en un árbol binario. Para esto, debemos seguir los pasos que se describen a continuación:

- 1) Enlazar los hijos de cada nodo en forma horizontal (los hermanos).
- 2) Enlazar en forma vertical el nodo padre con el nodo hijo que se encuentra más a la izquierda. Además, debe eliminarse el vínculo de ese padre con el resto de sus hijos.
- 3) Rotar el diagrama resultante aproximadamente 45 grados hacia la izquierda, y así se obtendrá el árbol binario correspondiente.

Otra forma más práctica:

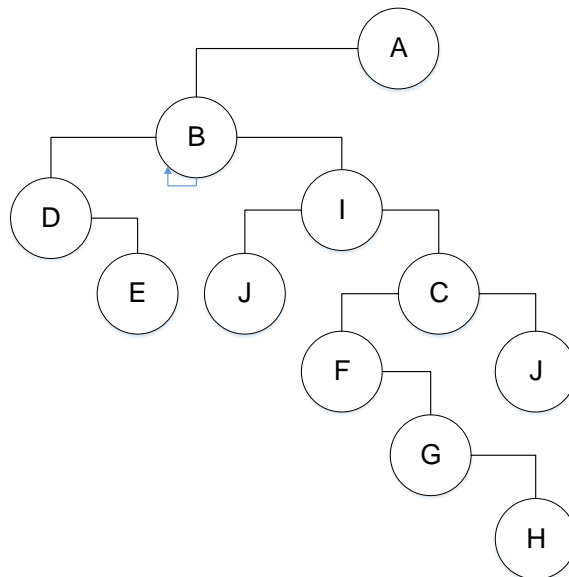
- 1ro. El primer hijo en aparecer de un nodo x en G será hijo izquierdo del nodo x en B.
- 2do. Los nodos x que tengan un siguiente hermano en G tendrán como hijo derecho a el siguiente hermano en B.
- 3ro. Si un nodo no tiene hijo en G, entonces no tiene hijo izquierdo en B.
- 4to. Si un nodo tiene un siguiente hermano en G, entonces no tiene hijo derecho en B.

ARBOL N-ARIO G



Al realizar el proceso de transformación del Arbol N-ario en un Arbol Binario tenemos:

ARBOL BINARIO B



7. REPRESENTACIÓN DE UN ÁRBOL EN MEMORIA

Hay dos formas tradicionales de representar un árbol binario en memoria:

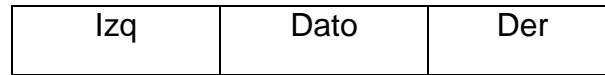
- Por medio de datos tipo punteros también conocidos como variables dinámicas o listas.
- Por medio de arreglos.

Sin embargo, la más utilizada es la primera, puesto que es la más natural para tratar este tipo de estructuras.

Los nodos del árbol binario serán representados como registros que contendrán como mínimo tres campos. En un campo se almacenará la información del nodo. Los dos

restantes se utilizarán para apuntar al subarbol izquierdo y derecho del subarbol en cuestión.

Cada nodo del árbol binario se representa gráficamente de la siguiente manera:



Nodo

El algoritmo de creación de un árbol binario es el siguiente:

Procedimiento crear(q:nodo)

inicio

```
mensaje("Rama izquierda?")
lee(respuesta)
si respuesta = "si" entonces
    new(p)
    q(li) <-- null
    crear(p)
en caso contrario
    q(li) <-- null
mensaje("Rama derecha?")
lee(respuesta)
si respuesta="si" entonces
    new(p)
    q(ld)<--p
    crear(p)
en caso contrario
    q(ld) <--null
```

fin

//Programa Función Principal

Inicio

```
new(p)
raiz<--p
crear(p)
```

Fin

8. CLASIFICACIÓN DE ÁRBOLES BINARIOS (A. B.)

Existen cuatro tipos de árboles binaris:

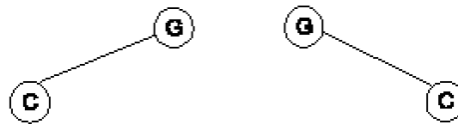
- A. B. Distinto.
- A. B. Similares.
- A. B. Equivalentes.
- A. B. Completos.

A continuación se hará una breve descripción de los diferentes tipos de árbol binario así como un ejemplo de cada uno de ellos.

1) A. B. Distinto.

Se dice que dos árboles binarios son distintos cuando sus estructuras son diferentes.

Ejemplo:



2) A. B. Similares.

Dos arboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos es diferente.

Ejemplo:



3) A. B. Equivalentes.

Son aquellos árboles que son similares y que además los nodos contienen la misma información.

Ejemplo:



4) A. B. Completos.

Son aquellos arboles en los que todos sus nodos excepto los del ultimo nivel, tiene dos hijos; el subarbol izquierdo y el subarbol derecho.

9. RECORRIDO DE UN ARBOL BINARIO

Hay tres manera de recorrer un árbol : en inorden, preorden y postorden. Cada una de ellas tiene una secuencia distinta para analizar el árbol como se puede ver a continuación:

1) INORDEN:

- Recorrer el subarbol izquierdo en inorden.
- Examinar la raíz.
- Recorrer el subarbol derecho en inorden.

2) POSTORDEN

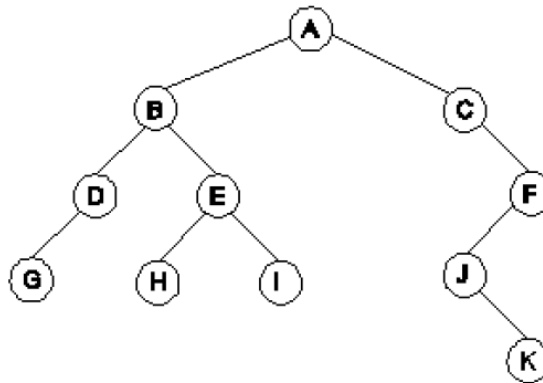
- Examinar la raíz.
- Recorrer el subarbol izquierdo en preorden.
- Recorrer el subarbol derecho en preorden.

3) PREORDEN

- Recorrer el subarbol izquierdo en postorden.
- Recorrer el subarbol derecho en postorden.
- Examinar la raíz.

Recorrido Pre-orden	Recorrido En-orden	Recorrido Post-orden
1. Visitar la raíz	1. Ir a bus-árbol izquierdo	1. Ir a sub-árbol izquierdo
2. ir a sub-árbol izquierdo	2. Visitar la raíz	2. Ir a sub-árbol derecho
3. Ir a sub-árbol derecho	3. ir a subárbol derecho	3. Visitar la raíz

A continuación se muestra un ejemplo de los diferentes recorridos en un árbol binario.

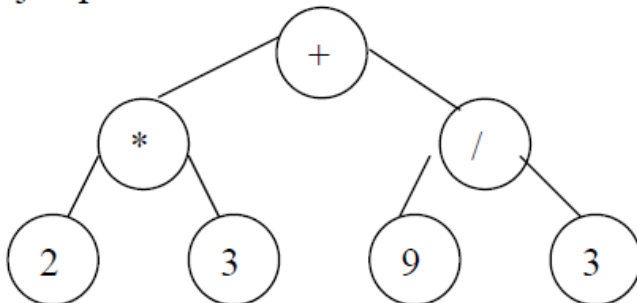


Recorrido	Resultado
Inorden:	GDBHEIACJKF
Preorden:	ABDGEHICFJK
Postorden:	GDHIEBKJFCA

10. ÁRBOLES DE EXPRESIONES

Los árboles binarios se utilizan para representar expresiones en memoria; esencialmente, en compiladores de lenguajes de programación. La Figura 5 muestra un árbol binario de expresiones para la expresión aritmética $(a+b*c)$.

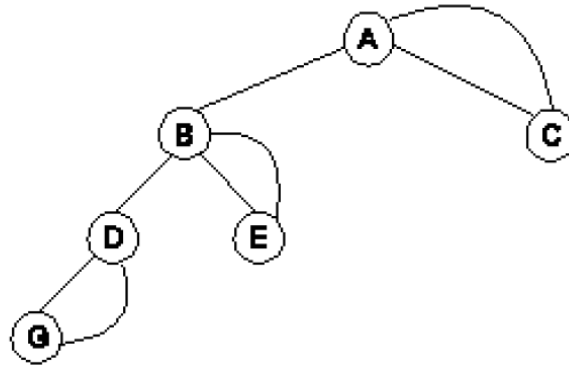
Ejemplo: $2 * 3 + 9 / 3$



Recorrido	Resultado
Preorden (Notación prefija)	+ * 2 3 / 9 3
Infija (Notación infija)	2 * 3 + 9 / 3
Postorden Notación Postfija)	2 3 * 9 3 / +

11. ÁRBOLES ENHEBRADOS

Existe un tipo especial de árbol binario llamado enhebrado, el cual contiene hebras que pueden estar a la derecha o a la izquierda. El siguiente ejemplo es un árbol binario enhebrado a la derecha.



A) ARBOL ENHEBRADO A LA DERECHA

Este tipo de árbol tiene un apuntador a la derecha que apunta a un nodo antecesor.

B) ARBOL ENHEBRADO A LA IZQUIERDA

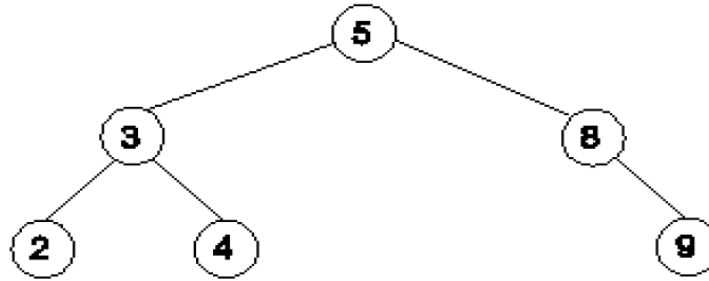
Estos arboles tienen un apuntador a la izquierda que apunta al nodo antecesor en orden.

12. ÁRBOLES BINARIOS DE BÚSQUEDA

Un árbol de búsqueda binaria es una estructura apropiada para muchas de las aplicaciones que se han discutido anteriormente con listas. La ventaja especial de utilizar un árbol es que se facilita la búsqueda.

Un árbol binario de búsqueda es aquel en el que el hijo de la izquierda (si existe) de cualquier nodo contiene un valor más pequeño que el nodo padre, y el hijo de la derecha (si existe) contiene un valor más grande que el nodo padre.

Un ejemplo de árbol binario de búsqueda es el siguiente:



IMPLEMENTACIÓN DE UN ÁRBOL BINARIO EN JAVA

Programamos el nodo del árbol binario en java:

Programa 24 en Java

```
package estructurasDinamicas;
public class NodoArbol
{
    //miembros de acceso
    NodoArbol nodoizquierdo;
    int datos;
    NodoArbol nododerecho;
    //iniciar dato y hacer de este nodo un nodo hoja
    public NodoArbol(int datosNodo)
    {
        datos = datosNodo;
        nodoizquierdo = nododerecho = null; //el nodo no tiene hijos
    }
    //buscar punto de insercion e insertar nodo nuevo
    public synchronized void insertar(int valorInsertar)
    {
        if(valorInsertar < datos)
        {
            //insertar en subarbol izquierdo
            if (nodoizquierdo == null)
                nodoizquierdo = new NodoArbol(valorInsertar);
            else //continua recorriendo subarbol izquierdo
                nodoizquierdo.insertar(valorInsertar);
        }
        //insertar nodo derecho
        else
        {
            if(valorInsertar > datos)
            {
                //insertar nuevo nodoArbol
                if(nododerecho == null)
                    nododerecho = new NodoArbol(valorInsertar);
                else
                    nododerecho.insertar(valorInsertar);
            }
        }
    }
    // fin del metodo insertar
}
```

Ahora implementamos la estructura de datos del Árbol Binario, junto con sus recorridos.

```
package estructurasDinamicas;
public class Arbol
{
    private NodoArbol raiz;
    //construir un arbol vacio
    public Arbol()
    {
        raiz = null;
    }
    //insertar un nuevo nodo en el arbol de busqueda binaria
    public synchronized void insertarNodo(int valorInsertar)
    {
        if(raiz == null)
            raiz = new NodoArbol(valorInsertar); //crea nodo raiz
        else
            raiz.insertar(valorInsertar); //llama al metodo insertar
    }

    // Empieza el Recorrido en Preorden
    public synchronized void recorridoPreorden()
    {
        ayudantePreorden(raiz);
    }
    //método recursivo para recorrido en preorden
    private void ayudantePreorden(NodoArbol nodo)
    {
        if(nodo == null)
            return;
        System.out.print(nodo.datos + " "); //mostrar datos del nodo
        ayudantePreorden(nodo.nodoizquierdo); //recorre subarbol izquierdo
        ayudantePreorden(nodo.nododerecho); //recorre subarbol derecho
    }

    //Empezar Recorrido Inorden
    public synchronized void recorridoInorden()
    {
        ayudanteInorden(raiz);
    }

    //Método Recursivo para Recorrido Inorden
    private void ayudanteInorden( NodoArbol nodo)
    {
        if(nodo == null)
            return;
        ayudanteInorden(nodo.nodoizquierdo);
        System.out.print(nodo.datos + " ");
        ayudanteInorden(nodo.nododerecho);
    }
}
```

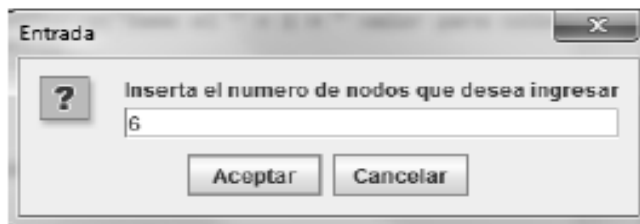
```
//Empezar Recorrido Posorden
public synchronized void recorridoPosorden()
{
    ayudantePosorden(raiz);
}
//Método Recursivo para Recorrido Posorden
private void ayudantePosorden(NodoArbol nodo)
{
    if( nodo == null )
        return;
    ayudantePosorden(nodo.nodoizquierdo);
    ayudantePosorden(nodo.nododerecho);
    System.out.print(nodo.datos + " ");
}
}
```

Luego implementamos la aplicación de árboles binarios:

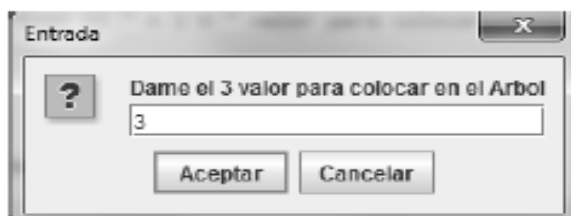
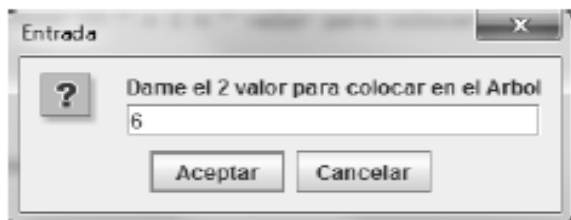
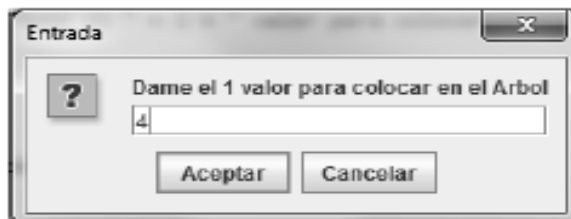
```
package estructurasDinamicas;
import javax.swing.JOptionPane;
public class PruebaArbol
{
    public static void main(String args [])
    {
        Arbol arbol = new Arbol();
        int valor;
        String Dato;
        System.out.println("Insertando los siguientes valores: ");
        Dato = JOptionPane.showInputDialog("Inserta el numero de nodos que desea ingresar");
        int n = Integer.parseInt(Dato);
        for(int i = 1; i <= n; i++ )
        {
            Dato = JOptionPane.showInputDialog("Dame el " + i + " valor para colocar en el Arbol");
            valor = Integer.parseInt(Dato);
            System.out.print(valor + " ");
            arbol.insertarNodo(valor);
        }
    }
}
```

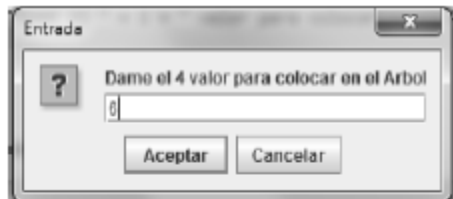
```
}  
System.out.println("\n\nRecorrido Preorden");  
arbol.recorridoPreorden();  
System.out.println("\n\nRecorrido Inorden");  
arbol.recorridoInorden();  
System.out.println("\n\nRecorrido Postorden");  
arbol.recorridoPosorden();  
}  
}
```

Salida del programa:



Luego llenamos





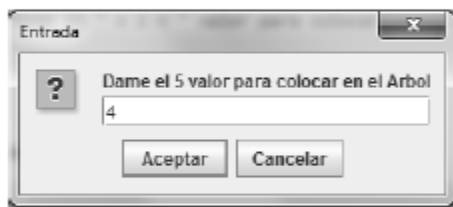
Entrada

?

Dame el 4 valor para colocar en el Arbol

0

Aceptar Cancelar



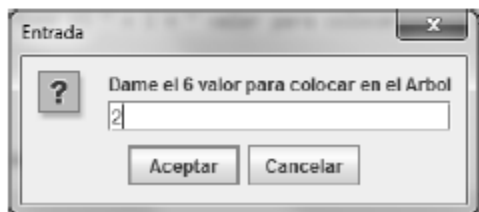
Entrada

?

Dame el 5 valor para colocar en el Arbol

4

Aceptar Cancelar



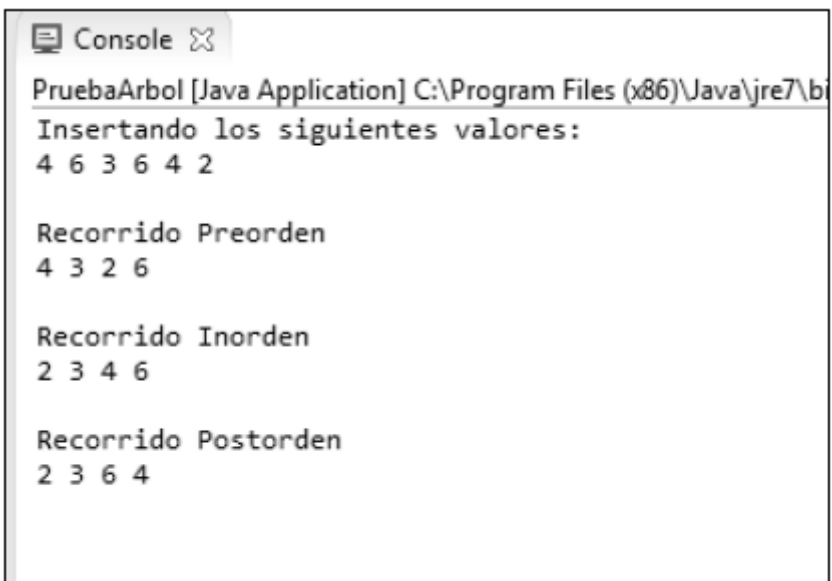
Entrada

?

Dame el 6 valor para colocar en el Arbol

2

Aceptar Cancelar



```
PruebaArbol [Java Application] C:\Program Files (x86)\Java\jre7\bin
Insertando los siguientes valores:
4 6 3 6 4 2

Recorrido Preorden
4 3 2 6

Recorrido Inorden
2 3 4 6

Recorrido Postorden
2 3 6 4
```

EJERCICIOS PROPUESTOS PARA LA PRÁCTICA NRO. 7

Realizar la implementación en Java de lo siguiente:

- 1) Dado un árbol binario de búsqueda, que contiene el nombre, el apellido y la edad de un grupo de personas, ordenados por edades.

Se pide: Hacer un algoritmo que visualice los datos de las personas de mayor a menor edad.
- 2) Dado un Arbol T, se pide invertir sus elementos:
- 3) Dado un Arbol Binario de Búsqueda (ABB) que contiene números enteros. Se pide: Hacer un procedimiento que reciba como parámetros: el puntero a raíz del arbol, y un número entero. El procedimiento debe buscar el elemento NUM y una vez encontrado visualizar todos sus antecesores (los anteriores).
- 4) Hacer una función que guarde en un archivo secuencial un árbol binario simétrico. Definir las estructuras necesarias.
- 5) Hacer una función que genere un árbol binario simétrico leyendo los datos desde un archivo secuencial. Definir las estructuras necesarias.

UNIVERSIDAD MAYOR DE SAN ANDRÉS
CARRERA INFORMÁTICA - F.C.P.N.
CURSO DE TEMPORADA 2023

CAPÍTULO VI

ÁRBOLES

MATERIA: ESTRUCTURA DE DATOS (INF – 131)

DOCENTE: M. Sc. ZARA YUJRA CAMA

GESTION 2023
LA PAZ - BOLIVIA

1. INTRODUCCION (1)

Los árboles junto con los grafos constituyen estructuras de datos no lineales.

Las **listas enlazadas** tienen grandes ventajas o flexibilidad sobre la representación contigua de estructura de datos (los arrays), pero tienen una **gran debilidad**: son **listas secuenciales**; es decir, están dispuestas de modo que es necesario moverse a través de ellas, una posición cada vez.

1. INTRODUCCION (2)

Los **árboles superan estas desventajas** utilizando los métodos de punteros y listas enlazadas para su implementación.

Las estructuras de datos organizadas como árboles serán muy valiosas en una gama grande de aplicaciones, sobre todo problemas de recuperación de información.

2. DEFINICIÓN DE ÁRBOL

“Un árbol es una estructura no lineal en la que cada nodo puede apuntar a uno o varios nodos”.

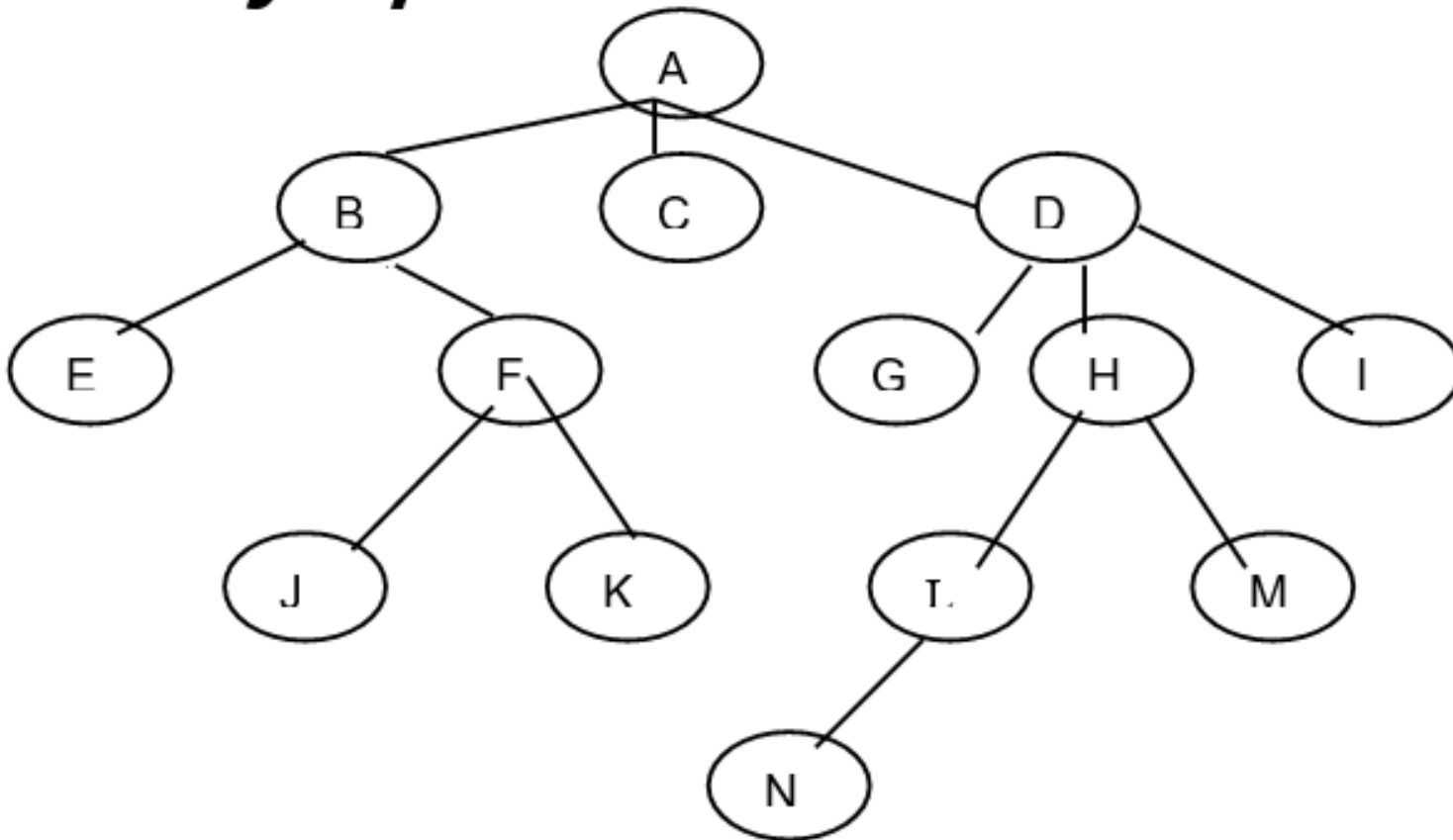
Un árbol (en inglés, tree) es una estructura que organiza sus elementos, denominados nodos, formando jerarquías. Los científicos utilizan los árboles generales para representar relaciones. Fundamentalmente, la relación clave es la de “**padre - hijos**” entre los nodos del árbol.

ÁRBOL - CONDICIONES QUE CUMPLE:

- Existe un solo nodo llamado raíz.
- El resto de los nodos se distribuyen en $n \geq 0$ conjuntos disjuntos $T_1, T_2, T_3, \dots, T_n$ donde cada uno de estos es a su vez un árbol y se los denomina sub-árboles.
- Se incluye la existencia del árbol nulo.

ÁRBOL - CONDICIONES QUE CUMPLE:

Ejemplo: Sea el árbol T



Número de nodos = 14

Subárbol izquierdo T1 =
{B,E,F,J,K}

Subárbol central T2 =
{C}

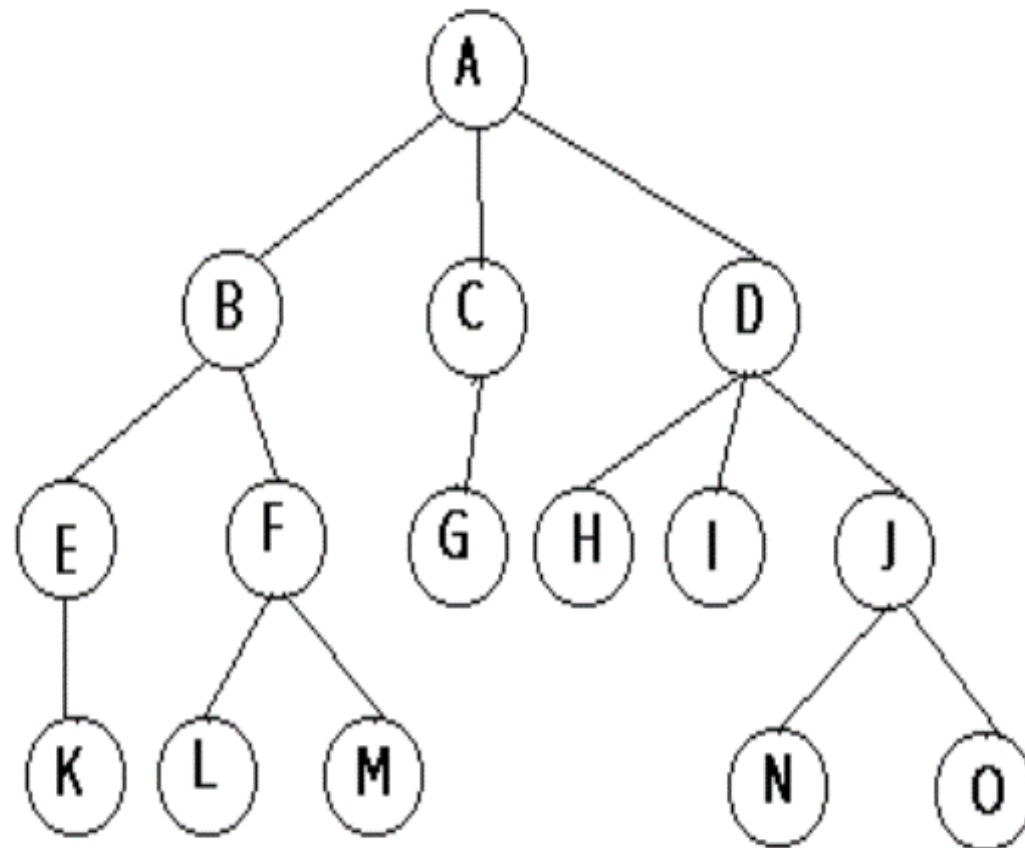
Subárbol derecho T3 =
{D,G,H,,L,M}

ÁRBOL - CONDICIONES QUE CUMPLE:

Debido a la naturaleza jerárquica de los árboles, se puede utilizar para representar en formación que sea jerárquica por naturaleza, por ejemplo, **diagramas de organizaciones** (Organigramas), **árboles genealógicos**, árboles de especies animales, etc.

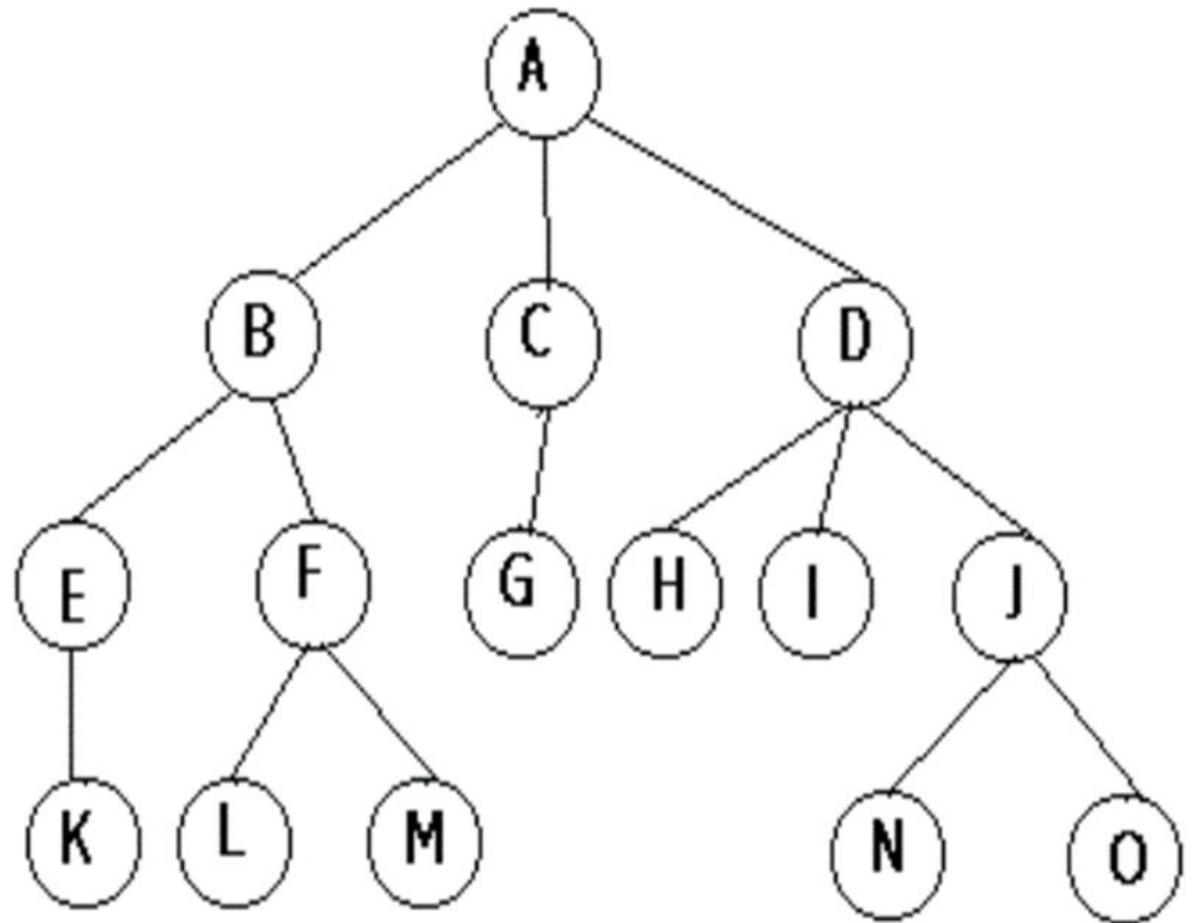
3. CONCEPTOS RELACIONADOS

Sea el árbol siguiente:



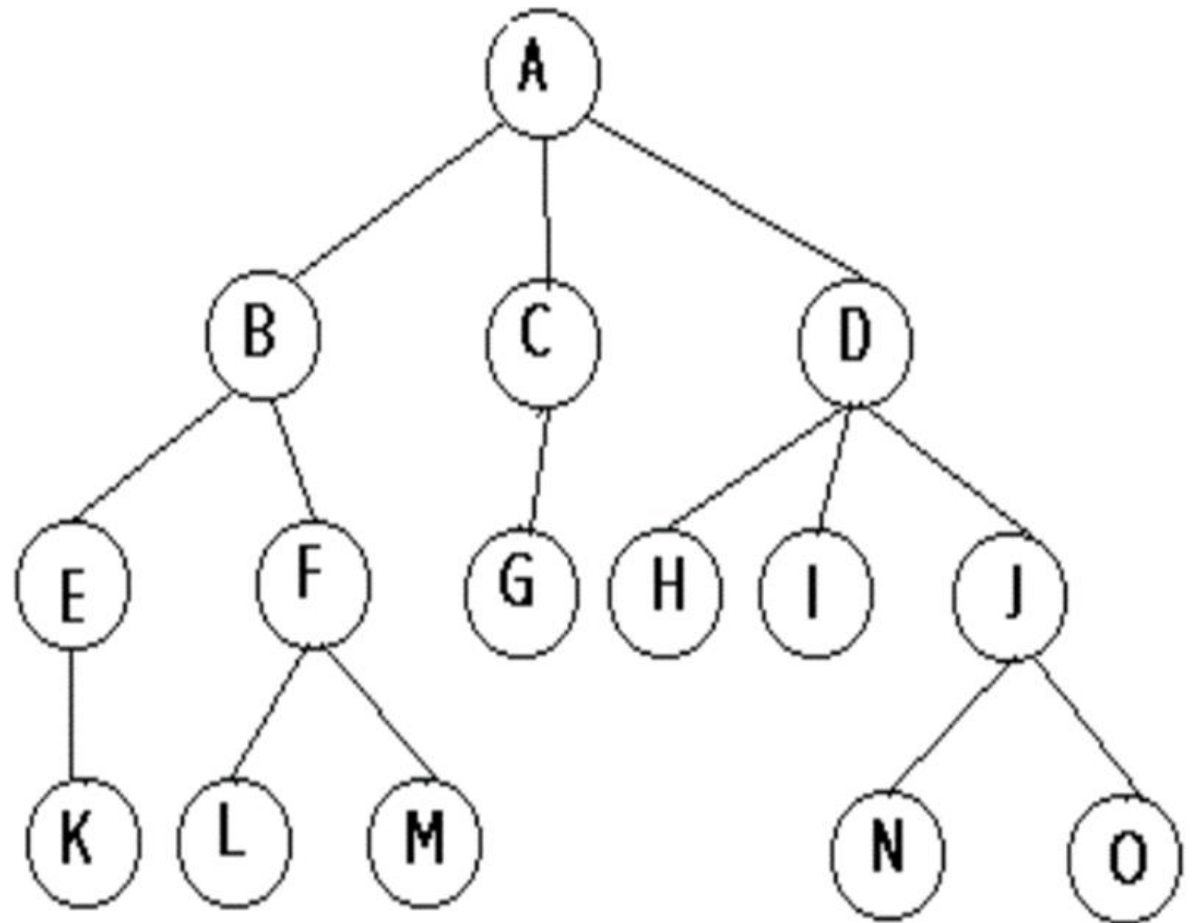
1) NODO HIJO

Nodo hijo: cualquiera de los nodos apuntados por uno de los nodos del árbol. En el ejemplo, 'L' y 'M' son hijos de 'F'.



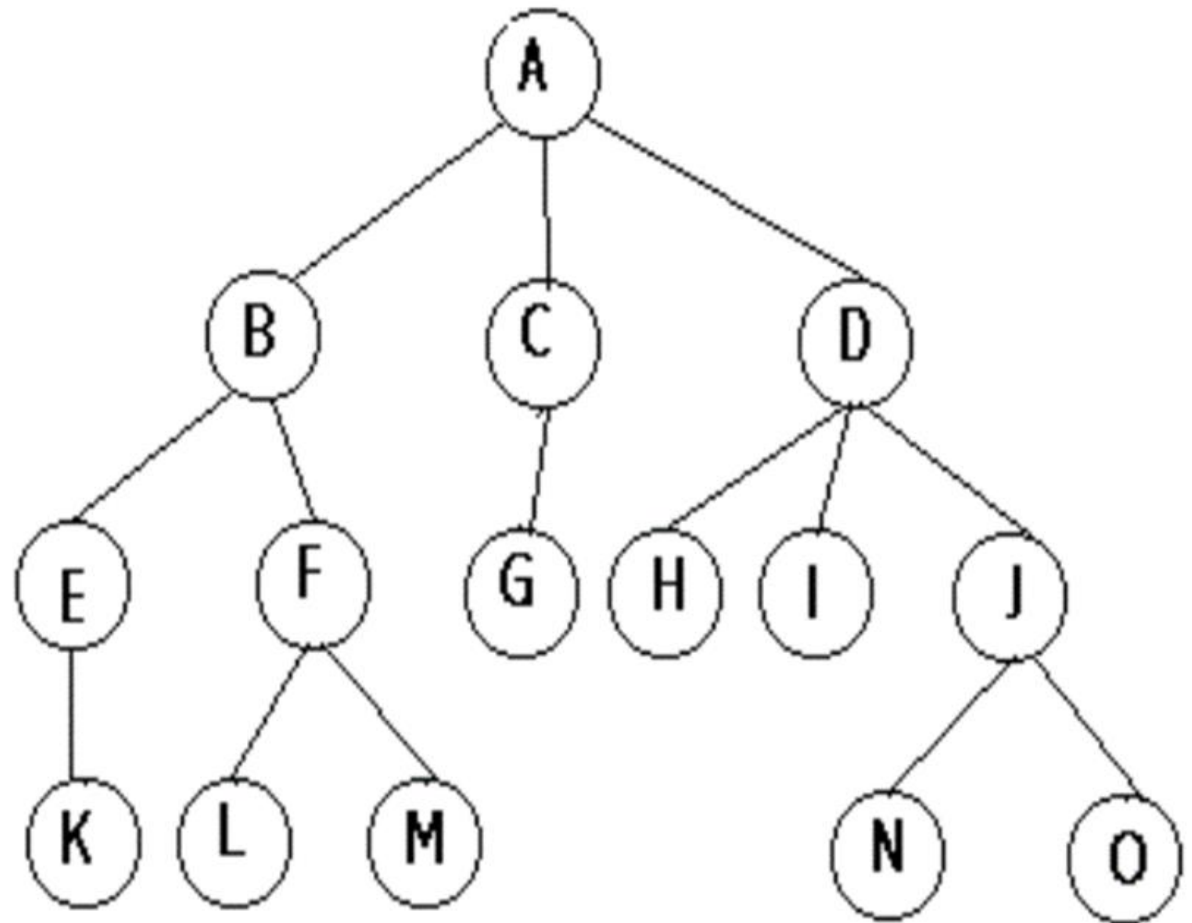
2) NODO PADRE

Nodo padre: nodo que contiene un puntero al nodo actual. En el ejemplo, el nodo 'A' es padre de 'B', 'C' y 'D'.



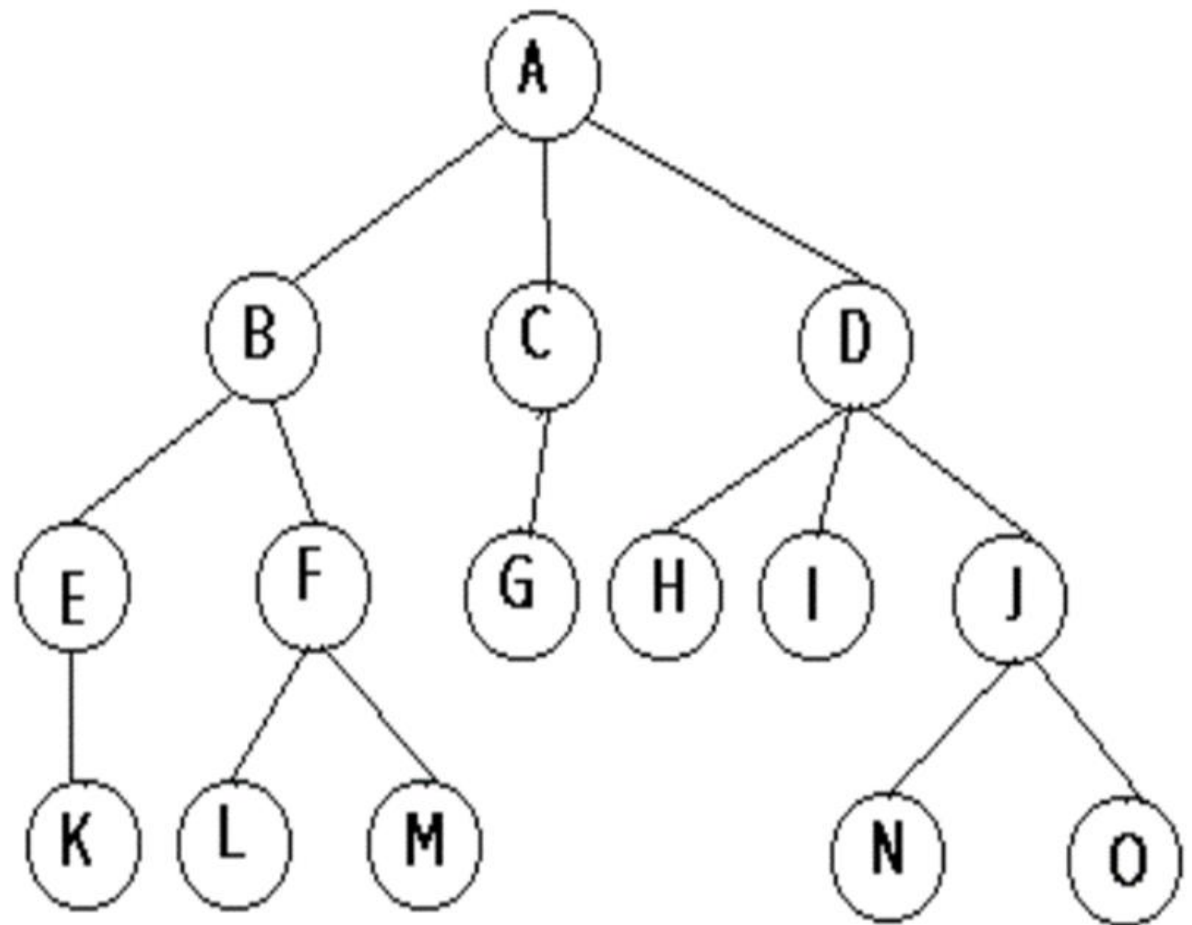
3) NODO RAIZ

Nodo Raíz: nodo que no tiene padre. Este es el nodo que usaremos para referirnos al árbol. En el ejemplo, ese nodo es el 'A'.



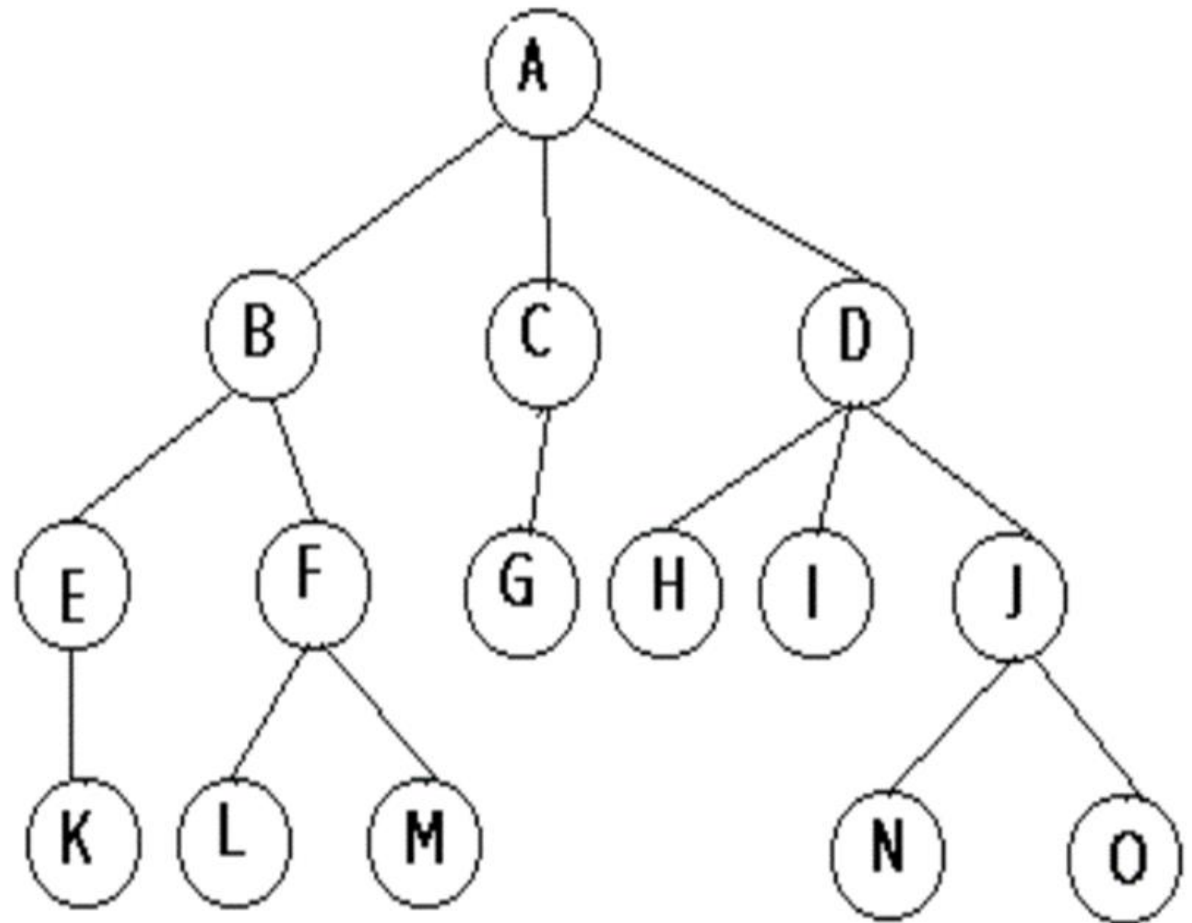
4) NODO TERMINAL

Nodo Hoja (Terminal):
nodo que no tiene
hijos. En el ejemplo
hay varios: 'G', 'H', 'I',
'K', 'L', 'M', 'N' y 'O'.



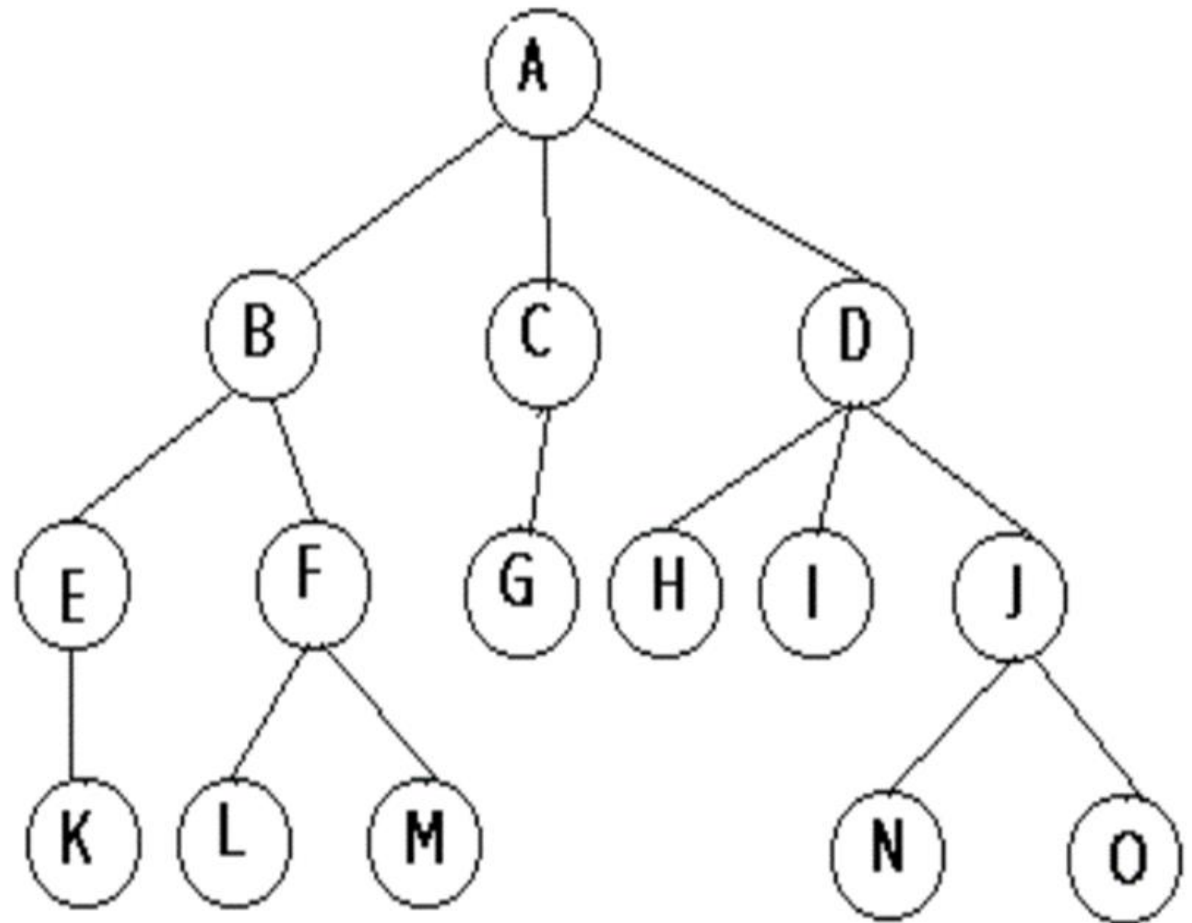
5) NODO NO TERMINAL

Nodo No Terminal
(Terminal): Son
aquellos nodos que
tienen descendientes



6) NODO RAMA

Nodo Rama: aunque esta definición apenas la usaremos, esos son los nodos que no pertenecen a ninguna de las dos categorías anteriores. En el ejemplo: 'B', 'C', 'D', 'E', 'F' y 'J'.

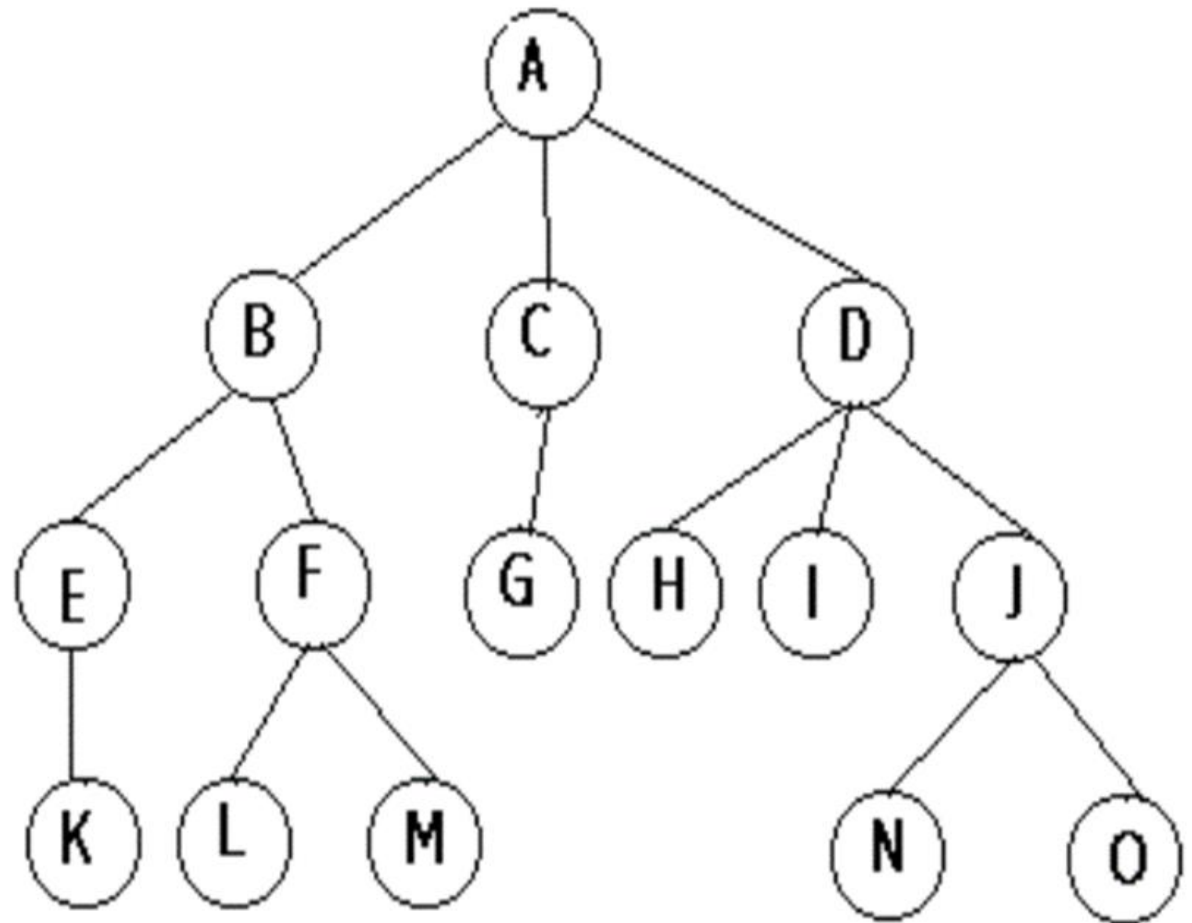


7) ORDEN

Orden: es el número potencial de hijos que puede tener cada elemento de árbol. De este modo, diremos que un árbol en el que cada nodo puede apuntar a otros dos es de orden dos, si puede apuntar a tres será de orden tres, etc.

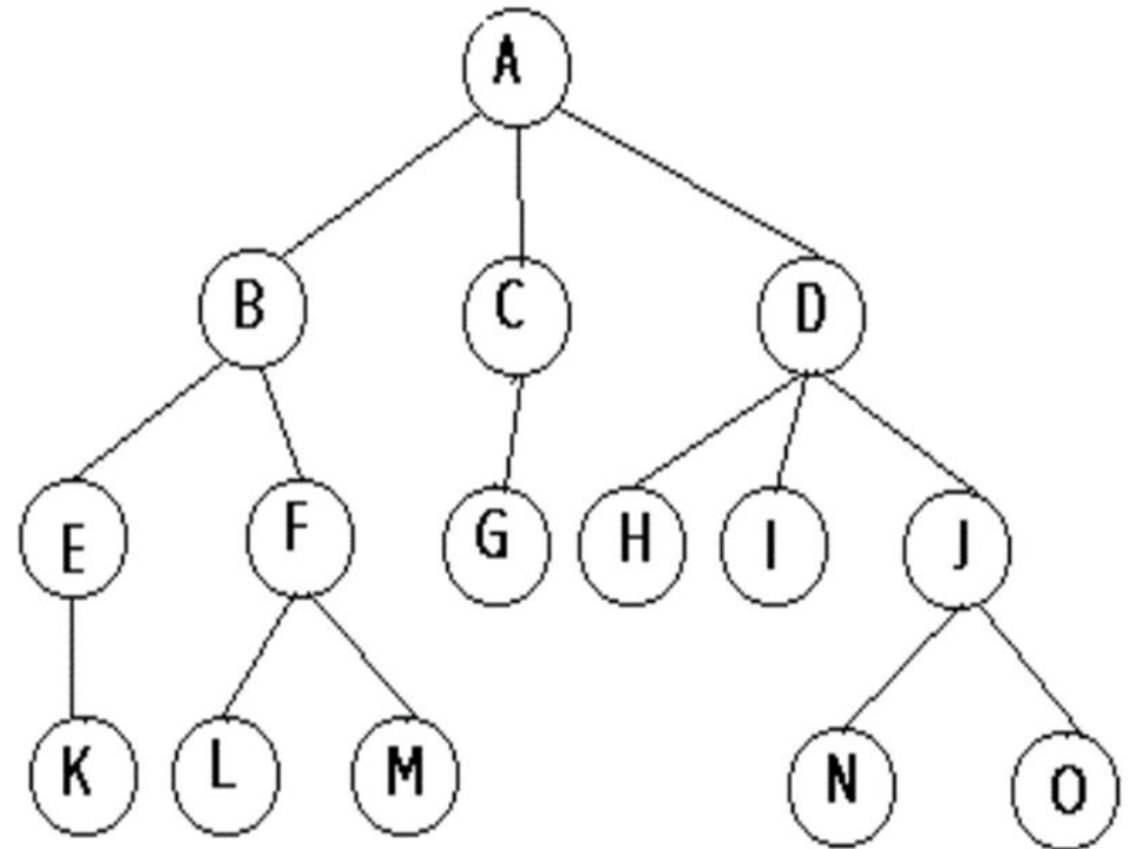
8) GRADO DE ARBOL

El número de hijos que tiene el elemento con más hijos dentro del árbol. En el árbol del ejemplo, el grado es tres, ya que tanto 'A' como 'D' tienen tres hijos, y no existen elementos con más de tres hijos.



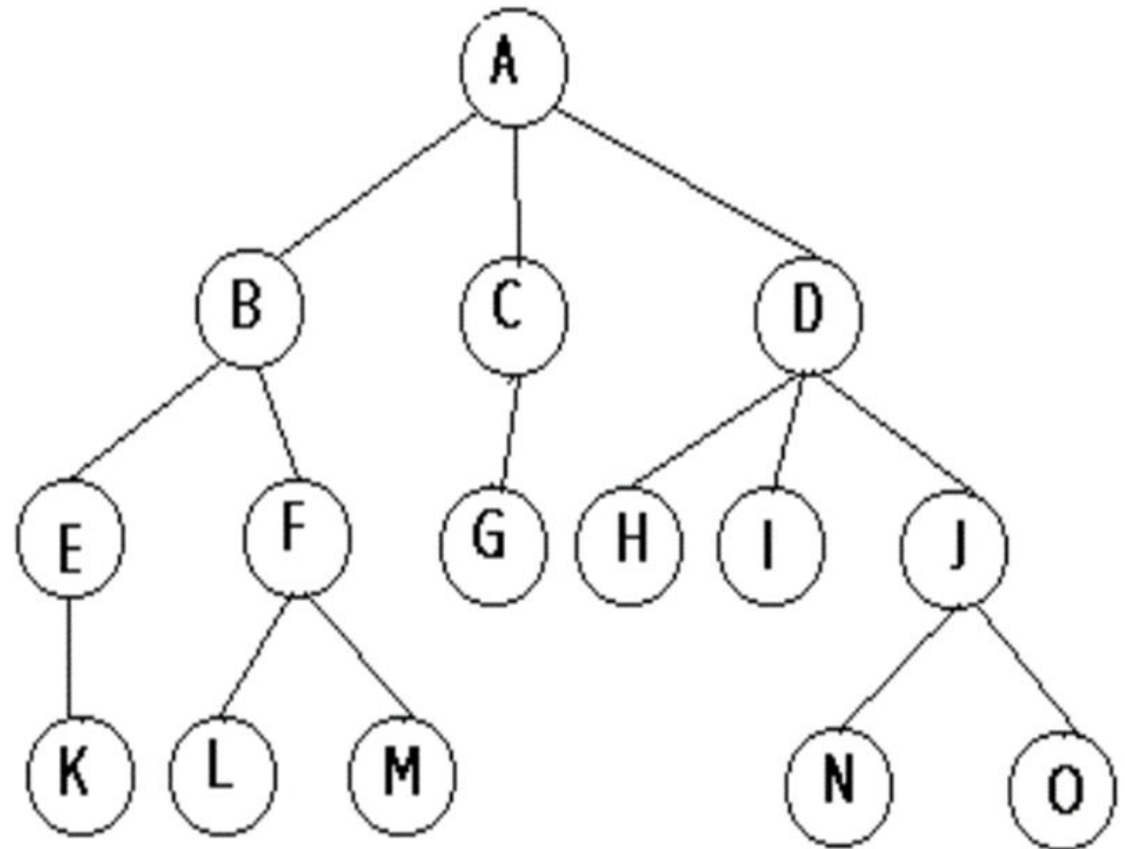
9) NIVEL

Se define para cada elemento del árbol como la distancia a la raíz, medida en nodos. El nivel de la raíz es cero y el de sus hijos uno. Así sucesivamente. En el ejemplo, el nodo 'D' tiene nivel 1, el nodo 'G' tiene nivel 2, y el nodo 'N', nivel 3.



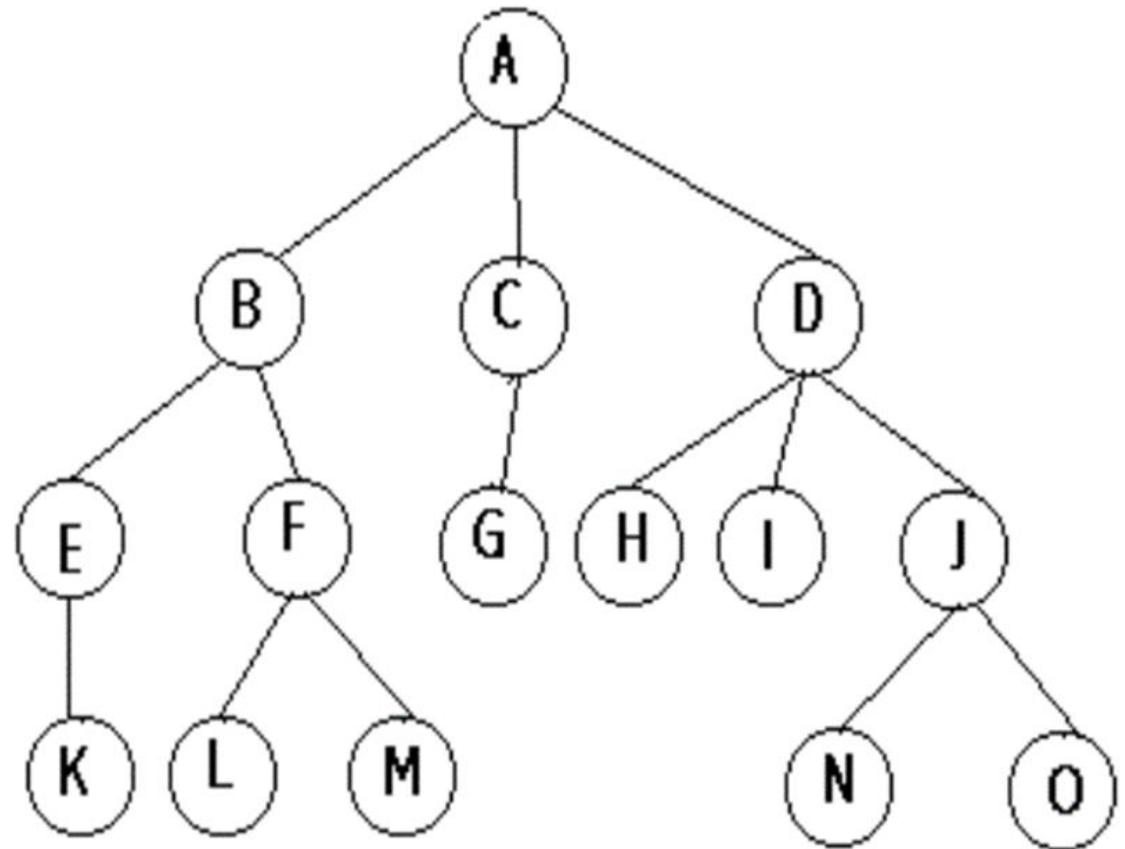
10) ALTURA DE NODO

Es la longitud del camino más largo desde ese nodo hasta un nodo terminal.



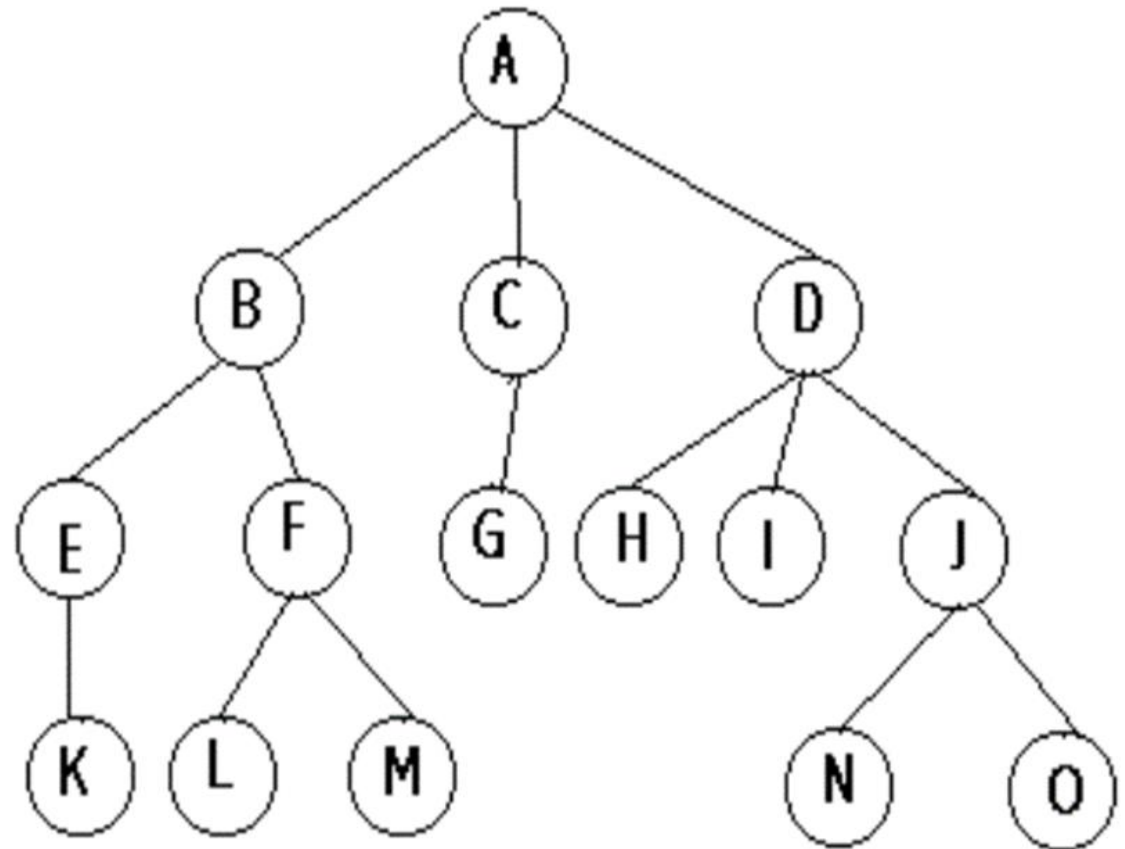
11) ALTURA DE ÁRBOL

Es la altura de la raíz, es la longitud del camino más largo del nodo raíz.



12) PROFUNDIDAD DE UN NODO

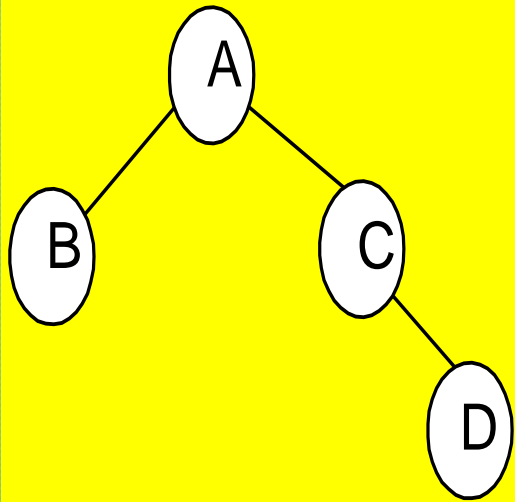
Es la longitud del camino desde la raíz hasta ese nodo.



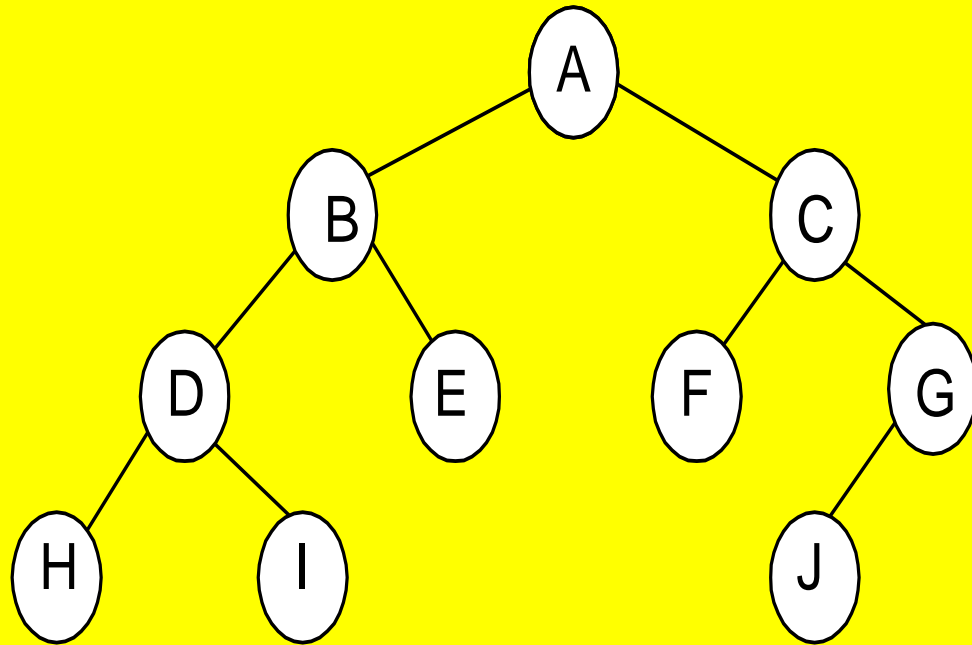
4. ARBOL BINARIO

Es un conjunto de nodos el cual puede ser vacío, o puede contener una raíz y dos sub-árboles denominados sub-árbol izquierdo y derecho. Cada uno de estos sub-árboles es, a su vez, un árbol binario.

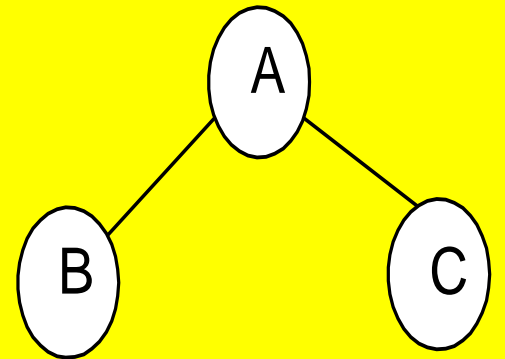
4. ARBOL BINARIO (2)



(a)



(b)



(c)

Figura 1. Diversos tipos de árboles binarios

4. ARBOL BINARIO (2)

Un **árbol binario lleno** es aquel en el que cada nodo tiene dos hijos o ninguno si es una hoja.

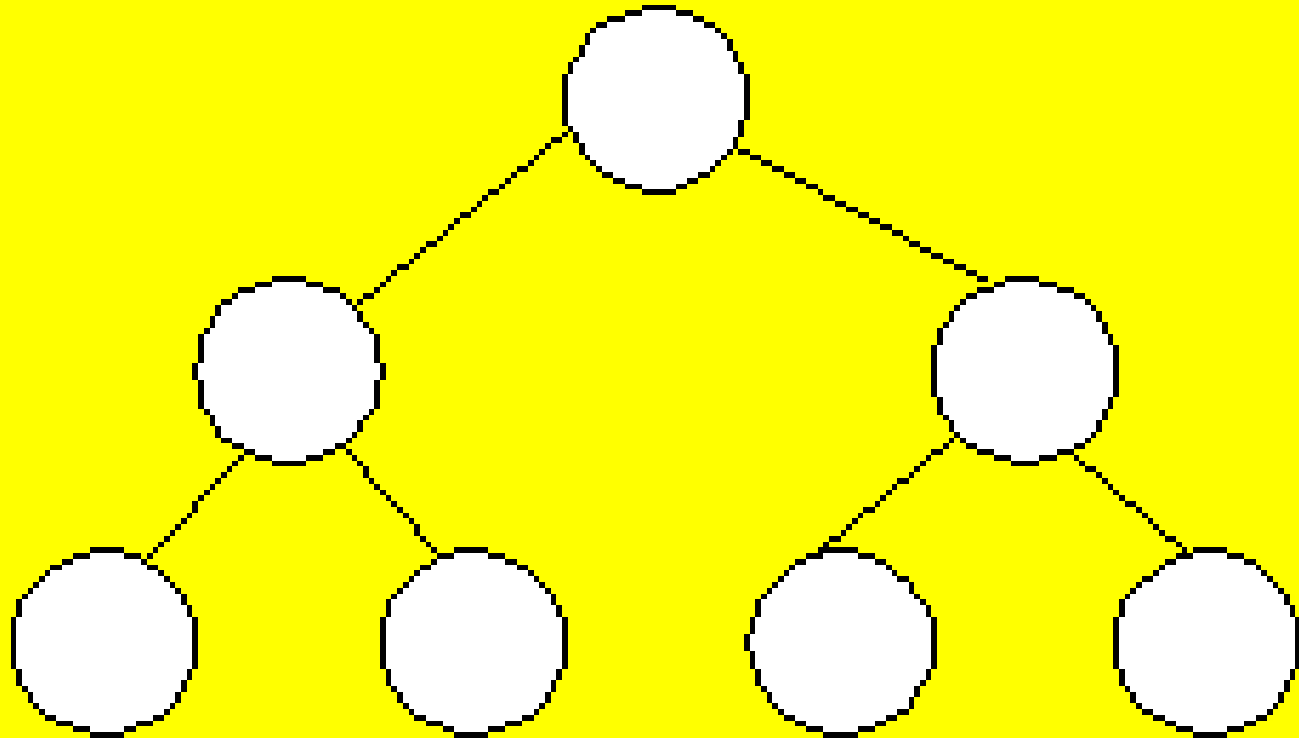


Figura 2. Arbol lleno de altura 3

4. ARBOL BINARIO (2)

Un **árbol binario completo** es aquel en el que cada nodo tiene, uno o dos hijos o ninguno si es una hoja.

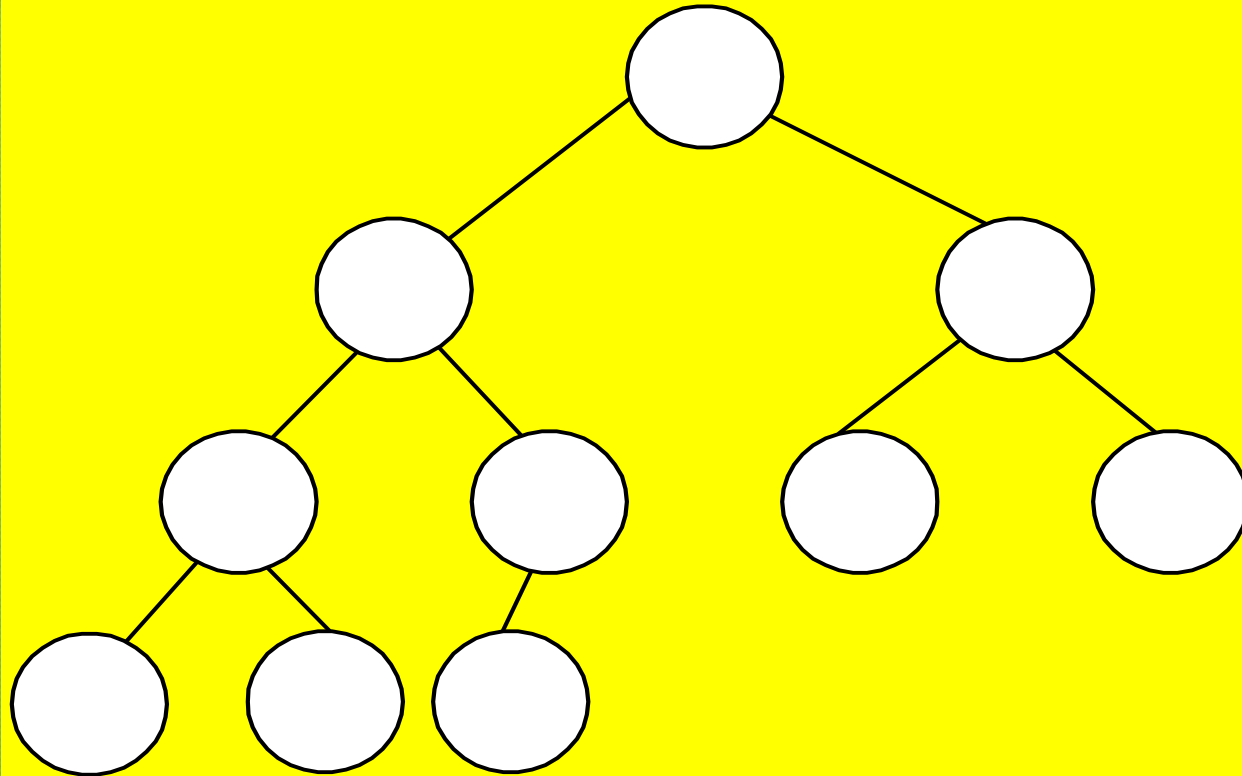


Figura 3. Arbol Binario Completo

4. ARBOL BINARIO (2)

Un árbol binario completo es equilibrado, mientras que un árbol binario lleno es totalmente equilibrado.

Si un árbol binario es lleno, es necesariamente completo.

Un árbol binario es completamente (totalmente) equilibrado si los subárboles izquierdo y derecho de cada nodo tienen la misma altura.

4. ARBOL BINARIO (2)

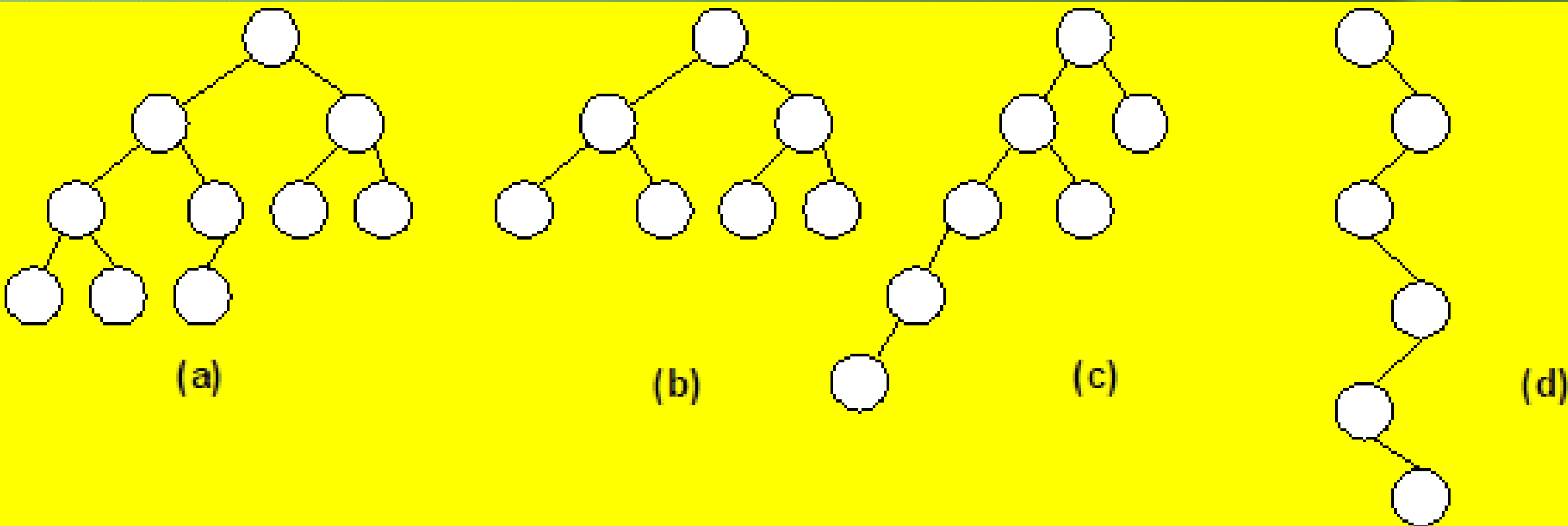


Figura 4. Arbol Binario (a) Equilibrado, (b) Completamente Equilibrado, (c) y (d) Arboles no Equilibrados

5. REPRESENTACIÓN DE UNA ARBOL EN MEMORIA

Hay dos formas tradicionales de representar un árbol binario en memoria:

- Por medio de datos tipo punteros también conocidos como variables dinámicas o listas.
- Por medio de arreglos.

Sin embargo, la más utilizada es la primera, puesto que es la más natural para tratar este tipo de estructuras.

5. REPRESENTACIÓN DE UNA ARBOL EN MEMORIA (2)

Cada nodo del árbol binario se representa gráficamente de la siguiente manera:



Nodo

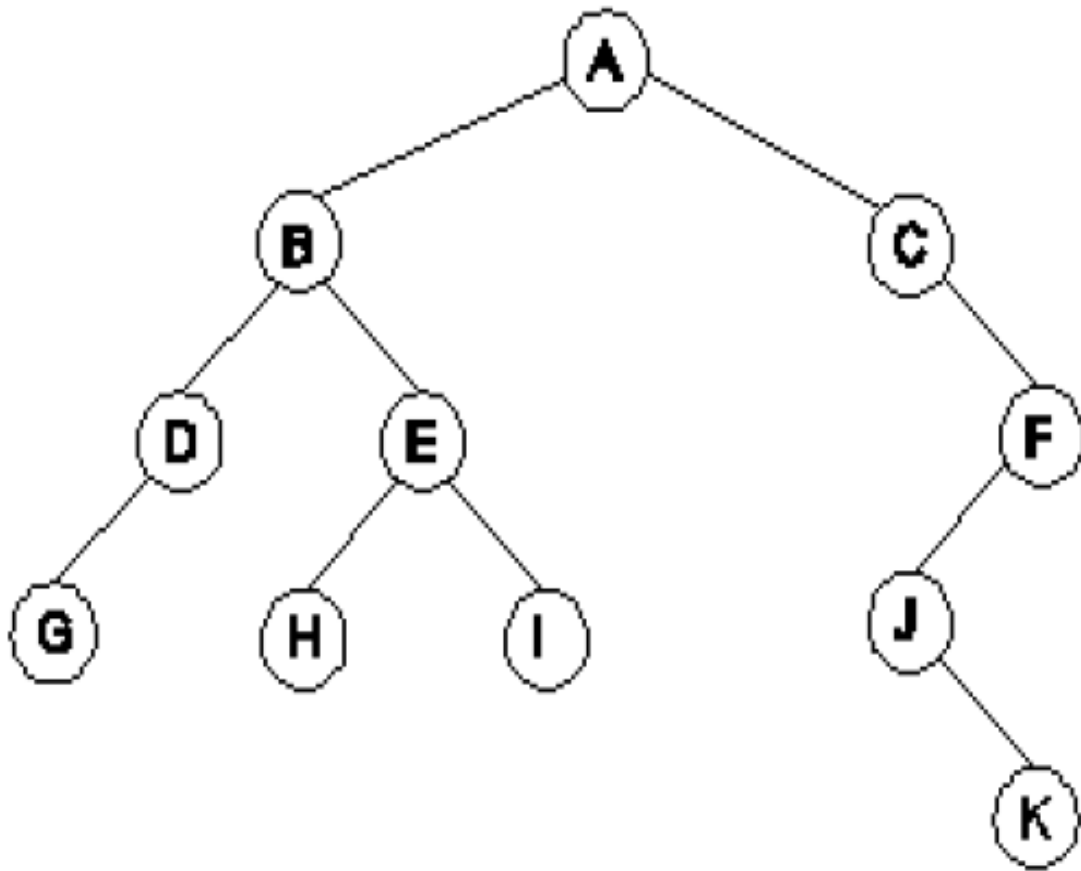
6. RECORRIDO DE UNA ARBOL BINARIO

Hay tres manera de recorrer un árbol: en inorden, preorden y postorden. Cada una de ellas tiene una secuencia distinta para analizar el árbol como se puede ver a continuación:

6. RECORRIDO DE UNA ARBOL BINARIO (2)

Pre-orden	In-orden	Post-orden
1. Visitar la raíz	1. Ir a bus-árbol izquierdo	1. Ir a sub-árbol izquierdo
2. ir a sub-árbol izquierdo	2. Visitar la raíz	2. Ir a sub-árbol derecho
3. Ir a sub-árbol derecho	3. ir a subárbol derecho	3. Visitar la raíz

6. RECORRIDO DE UNA ARBOL BINARIO (3)



Inorden:
GDBHEIACJKF

Preorden:
ABDGEHICFJK

Postorden:
GDHIEBKJFCA

7. CLASIFICACIÓN DE ÁRBOLES BINARIOS

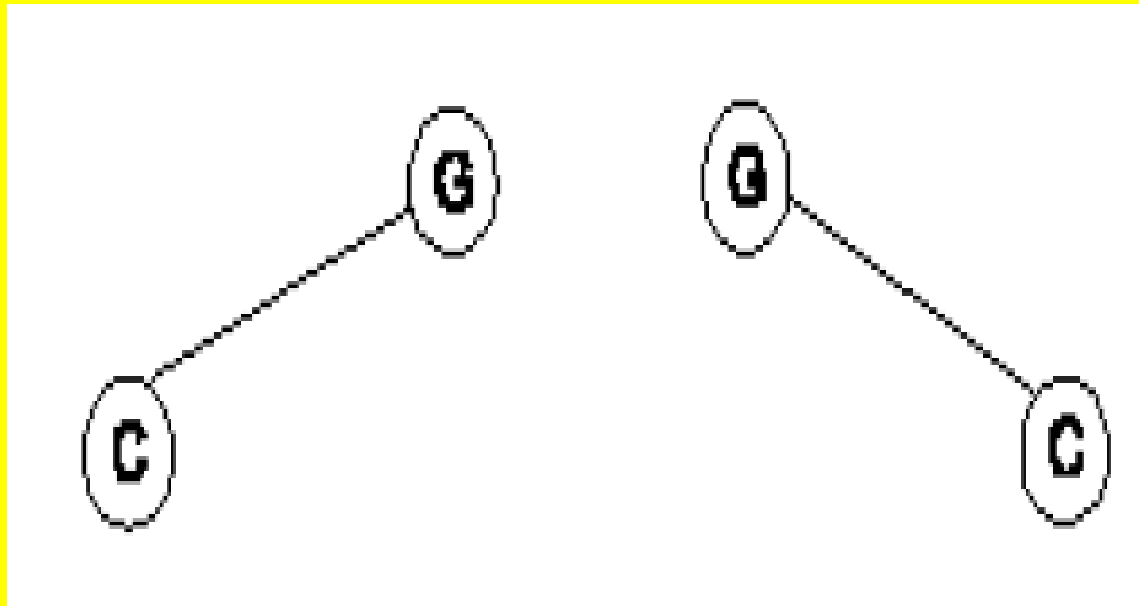
Existen cuatro tipos de árbol binario:.

- A. B. Distinto.
- A. B. Similares.
- A. B. Equivalentes.
- A. B. Completos.

A) ARBOLES BINARIOS DISTINTOS

Se dice que dos árboles binarios son distintos cuando sus estructuras son diferentes.

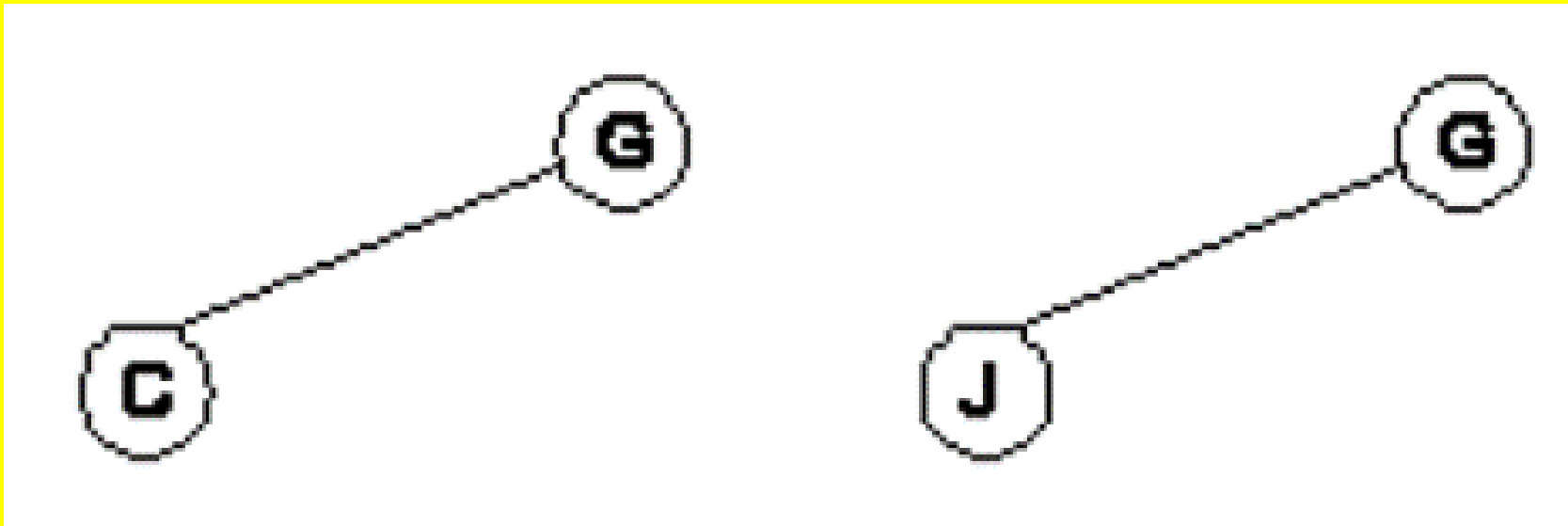
Ejemplo:



B) ARBOLES BINARIOS SIMILARES

Dos arboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos es diferente.

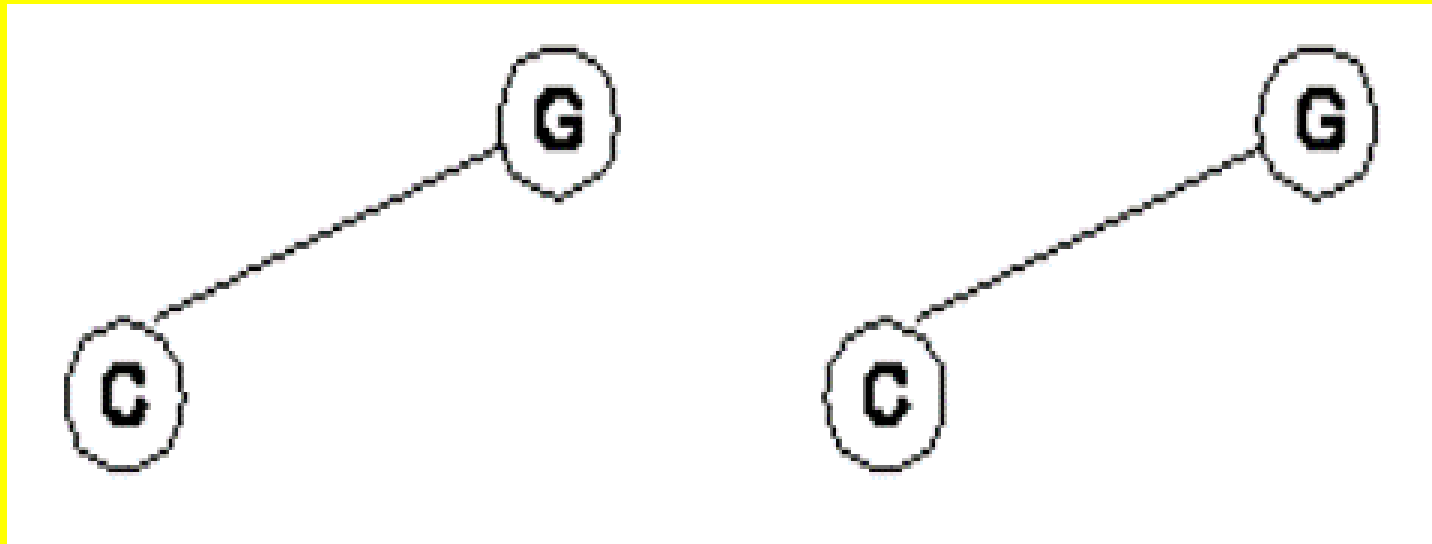
Ejemplo:



C) ARBOLES BINARIOS EQUIVALENTES

Son aquellos árboles que son similares y que además los nodos contienen la misma información.

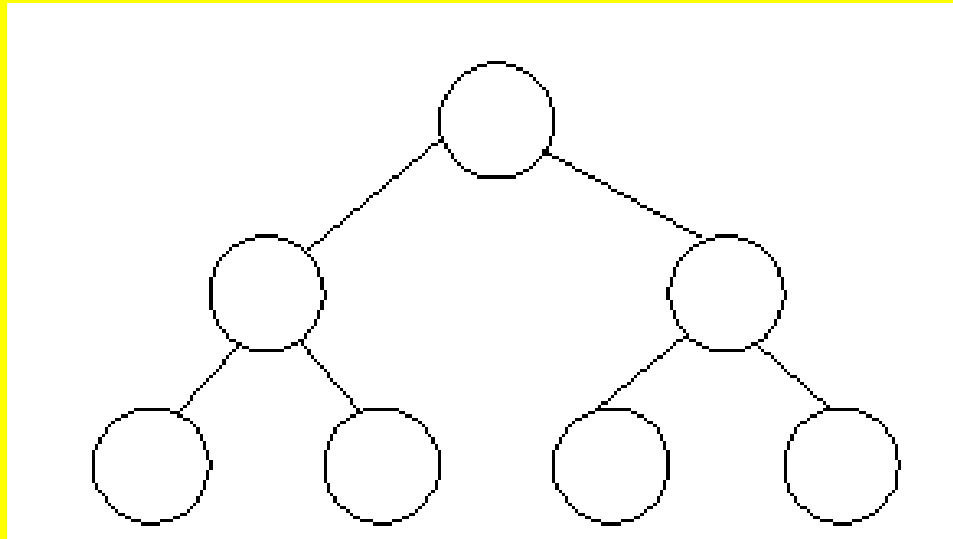
Ejemplo:



D) ARBOLES BINARIOS LLENOS

Son aquellos arboles en los que todos sus nodos excepto los del ultimo nivel, tiene dos hijos; el subarbol izquierdo y el subárbol derecho.

Ejemplo:

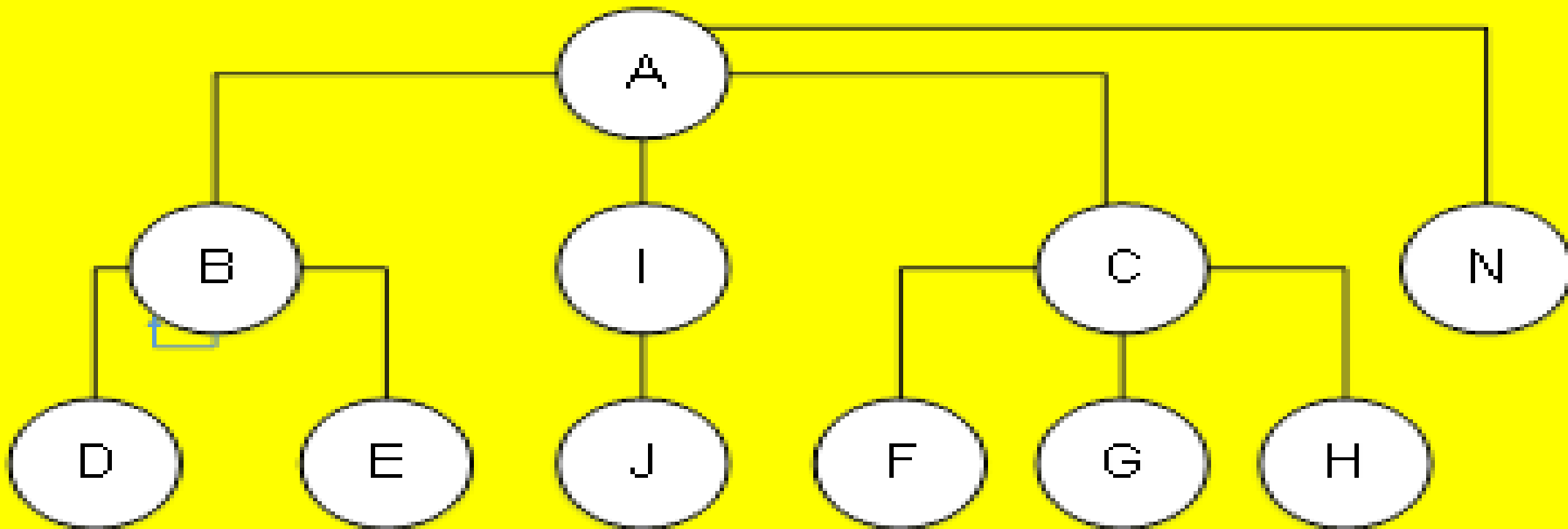


8. TRANSFORMACIÓN DE UN ÁRBOL EN UN ARBOL BINARIO

- 1ro. El **primer hijo** en aparecer de un nodo x en G será hijo izquierdo del nodo x en B .
- 2do. Los nodos x que tengan un **siguiente hermano** en G tendrán como hijo derecho a el siguiente hermano en B .
- 3ro. Si un **nodo no tiene hijo** en G , entonces no tiene hijo izquierdo en B .
- 4to. Si un **nodo no tiene un siguiente hermano** en G , entonces no tiene hijo derecho en B .

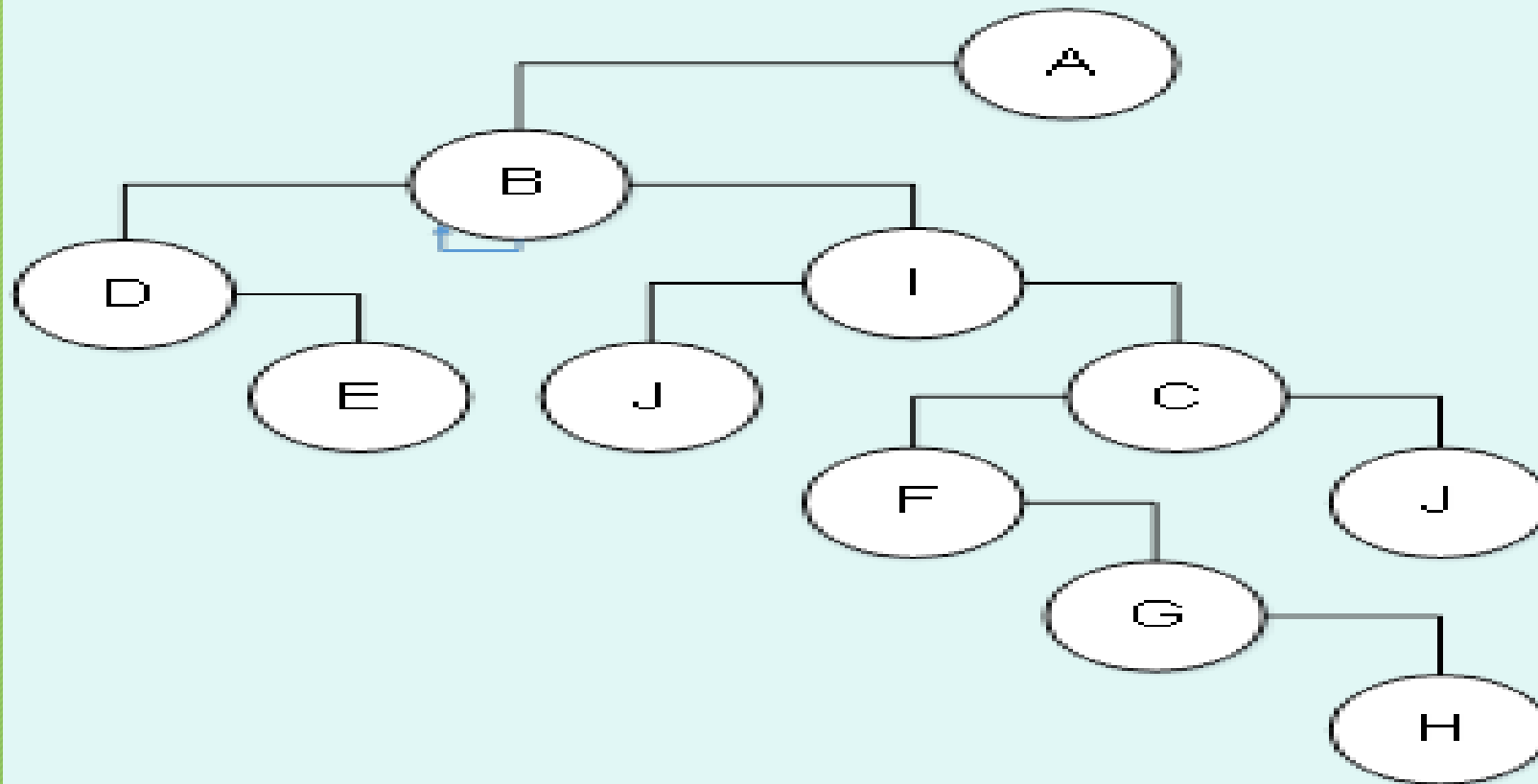
EJEMPLO: TRANSFORMACIÓN A UN ARBOL BINARIO

ARBOL N-ARIO G



EJEMPLO: TRANSFORMACIÓN A UN ARBOL BINARIO

ARBOL BINARIO B



9. ÁRBOL DE EXPRESIONES

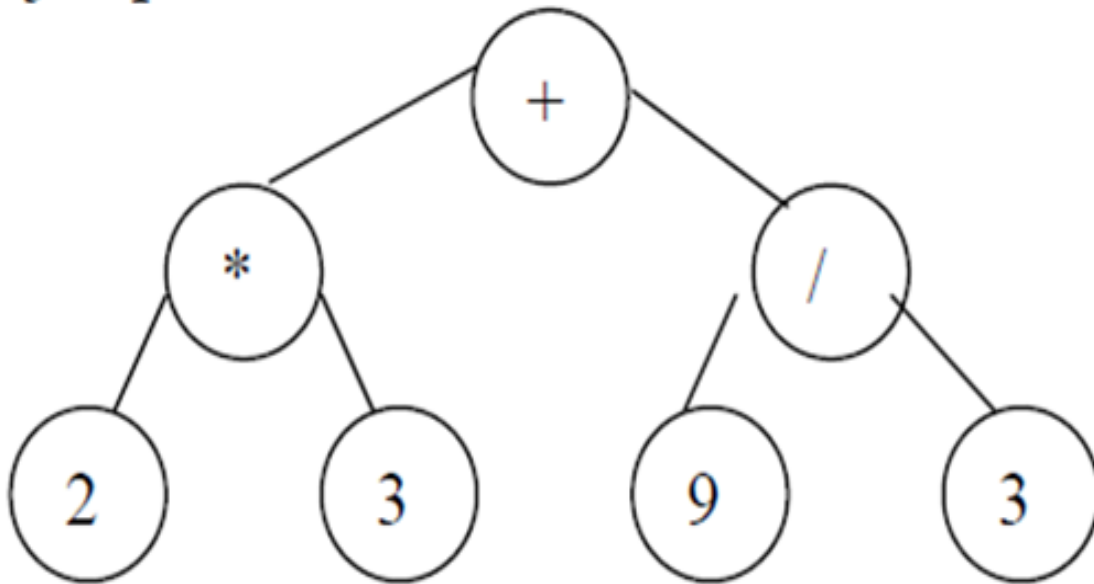
Los árboles binarios se utilizan para representar expresiones en memoria; esencialmente, en compiladores de lenguajes de programación.

Ejemplo:

La Figura siguiente muestra un árbol binario de expresiones para la expresión aritmética $(a+b*c)$.

9. ÁRBOL DE EXPRESIONES - EJEMPLO

Ejemplo: $2 * 3 + 9 / 3$



Preorden
(Notación prefija)

+ * 2 3 / 9 3

Infija
(Notación infija)

2 * 3 + 9 / 3

Postorden
(Notación Postfija)

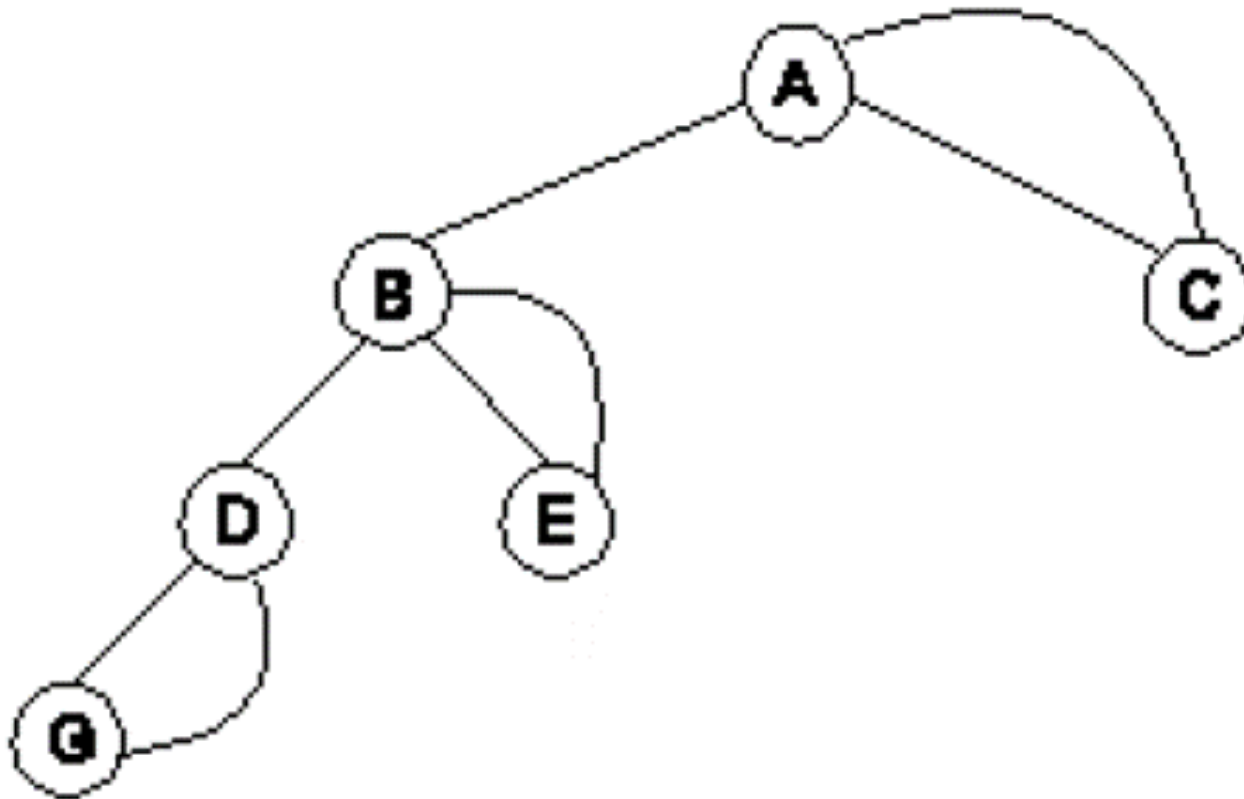
2 3 * 9 3 / +

10. ÁRBOLES ENHEBRADOS (1)

Existe un tipo especial de árbol binario llamado enhebrado, el cual contiene **hebras que pueden estar a la derecha o a la izquierda.**

El siguiente ejemplo es un árbol binario enhebrado a la derecha.

10. ÁRBOLES ENHERBRADOS - EJEMPLO (2)



10. ÁRBOLES ENHEBRADOS (3)

A) ARBOL ENHEBRADO A LA DERECHA

Este tipo de árbol tiene un apuntador a la derecha que apunta a un nodo antecesor.

B) ARBOL ENHEBRADO A LA IZQUIERDA

Estos arboles tienen un apuntador a la izquierda que apunta al nodo antecesor en orden.

11. PROCEDIMIENTO PARA CREAR UN ARBOL BINARIO

Procedimiento crear(q:nodo)

inicio

```
mensaje("Rama izquierda?")
lee(respuesta)
si respuesta = "si" entonces
    new(p)
    q(li) <-- null
    crear(p)
en caso contrario
    q(li) <-- null
mensaje("Rama derecha?")
lee(respuesta)
si respuesta="si" entonces
    new(p)
    q(ld) <-- p
    crear(p)
en caso contrario
    q(ld) <-- null
```

fin

//Programa Función Principal

Inicio

```
new(p)
raiz <-- p
crear(p)
```

Fin

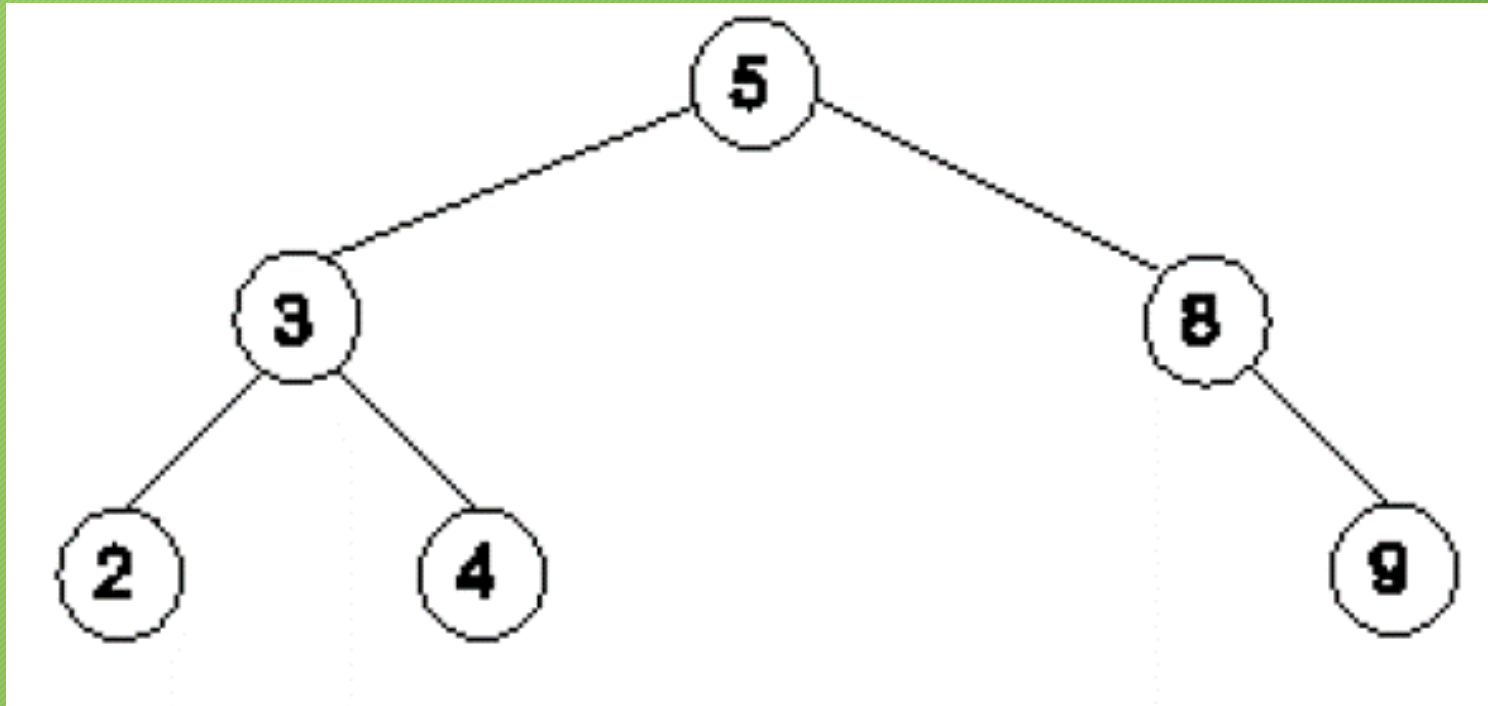
12. ÁRBOLES BINARIOS DE BÚSQUEDA (1)

Un árbol de búsqueda binaria es una estructura apropiada para muchas de las aplicaciones que se han discutido anteriormente con listas. La ventaja especial de utilizar un árbol es que se facilita la búsqueda.

12. ÁRBOLES BINARIOS DE BÚSQUEDA (2)

Un árbol binario de búsqueda es aquel en el que el hijo de la izquierda (si existe) de cualquier nodo contiene un valor más pequeño que el nodo padre, y el hijo de la derecha (si existe) contiene un valor más grande que el nodo padre.

12. ÁRBOLES BINARIOS DE BÚSQUEDA (3) - EJEMPLO



ÁRBOLES BINARIOS DE BÚSQUEDA

(3) - IMPLEMENTACIÓN

En el documento correspondiente al Cap. VI se incluye la implementación de un Arbol Binario de Búsqueda.

Se pide ejecutar en java el programa incluido y probarlo con datos de prueba adecuados.

FIN DEL TEMA

GRACIAS POR SU ATENCIÓN



**La tecnología es un
sirviente útil, pero un
jefe peligroso.**

Christian Lous
Lange



EJERCICIOS PROPUESTOS PARA LA PRÁCTICA NRO. 6

En cada uno de los ejercicios se pide implementar en el lenguaje Java las Funciones Recursivas correspondientes invocándolas desde el programa principal:

- 1) Hallar el máximo común divisor

Programa en Java:

Programa en Java:

```
package E1;
public class Main {
    public static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return gcd(b, a % b);
        }
    }
    public static void main(String[] args) {
        int a = 15;
        int b = 25;
        int result = gcd(a, b);
        System.out.println("El MCD de "+a+" y "+b+" es: "+result);
    }
}
```

Salida del programa

```
<terminated> Main (10) [Java Application] C:\Program Files\Java\jre1.
El MCD de 15 y 25 es: 5
```

- 2) Hallar el mínimo común múltiplo

Programa en Java:

Programa en Java:

```
package E2;
public class Main {
    public static int gcd(int a, int b) {
        if (b == 0) {
            return a;
        } else {
            return gcd(b, a % b);
        }
    }
    public static int lcm(int a, int b) {
        int gcdResult = gcd(a, b);
        return (a * b) / gcdResult;
    }
    public static void main(String[] args) {
        int a = 12;
        int b = 18;
        int result = lcm(a, b);
        System.out.println("El MCM de " + a + " y " + b + " es: " + result);
    }
}
```

Salida del programa

```
<terminated> Main (11) [Java Application] C:\Program Files\Java\jre
El MCM de 12 y 18 es: 36
```

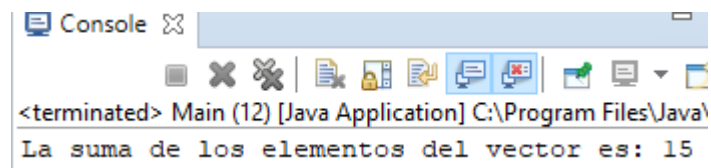
- 3) Sumar los elementos de un vector V de dimensión N.

Programa en Java:

```
package E3;
public class Main {
    public static int sumVector(int[] vector, int n) {
        if (n <= 0) {
            return 0;
        } else {
            return vector[n - 1] + sumVector(vector, n - 1);
        }
    }

    public static void main(String[] args) {
        int[] vector = {1, 2, 3, 4, 5};
        int n = vector.length;
        int result = sumVector(vector, n);
        System.out.println("La suma de los elementos del vector es: " + result);
    }
}
```

Salida del programa



The screenshot shows a Java IDE console window titled "Console". The output text is:
<terminated> Main (12) [Java Application] C:\Program Files\Java\
La suma de los elementos del vector es: 15

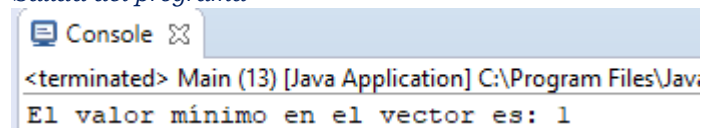
- 4) Hallar el valor mínimo en un vector V de dimensión N.

Programa en Java:

```
public class Main {
    public static int findMin(int[] vector, int n) {
        if (n == 1) {
            return vector[0];
        } else {
            return Math.min(vector[n - 1], findMin(vector, n - 1));
        }
    }

    public static void main(String[] args) {
        int[] vector = {5, 2, 8, 1, 6};
        int n = vector.length;
        int result = findMin(vector, n);
        System.out.println("El valor mínimo en el vector es: " + result);
    }
}
```

Salida del programa



The screenshot shows a Java IDE console window titled "Console". The output text is:
<terminated> Main (13) [Java Application] C:\Program Files\Java\
El valor mínimo en el vector es: 1

- 5) Sumar los elementos de una matriz A de dimensión NxM.

Programa en Java:

Programa en Java:

```
package E5;
public class Main {
    public static int sumMatrix(int[][] matrix, int rows, int cols) {
        if (rows == 0) {
            return 0;
        } else {
            int rowSum = 0;
            for (int i = 0; i < cols; i++) {
                rowSum += matrix[rows - 1][i];
            }
            return rowSum + sumMatrix(matrix, rows - 1, cols);
        }
    }
    public static void main(String[] args) {
        int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int rows = matrix.length;
        int cols = matrix[0].length;
        int result = sumMatrix(matrix, rows, cols);
        System.out.println("La suma de los elementos de la matriz es: " + result);
    }
}
```

Salida del programa

```
<terminated> Main (14) [Java Application] C:\Program Files\Java\jre1.8.0
La suma de los elementos de la matriz es: 45
|
```

- 6) Realizar la búsqueda binaria recursiva en un Vector V de dimensión N.

Programa en Java:

Programa en Java:

```
package E6;
public class Main {
    public static int binariob(int[] vector, int left, int right, int target) {
        if (right >= left) {
            int mid = left + (right - left) / 2;
            if (vector[mid] == target) {
                return mid;
            }
            if (vector[mid] > target) {
                return binariob(vector, left, mid - 1, target);
            }
            return binariob(vector, mid + 1, right, target);
        }
        return -1;
    }
    public static void main(String[] args) {
        int[] vec = {1, 3, 5, 7, 9};
        int target = 5;
        int res = binariob(vec, 0, vec.length - 1, target);
        System.out.println("El elemento "+target+" se encuentra en la posición: "+res);
    }
}
```

Salida del programa

```
<terminated> Main (15) [Java Application] C:\Program Files\Java\jre1.8.0_341\bin\j
El elemento 5 se encuentra en la posición: 2
```

- 7) Realizar la ordenación de elementos de un Vector V de dimensión N con el método Quick Sort.

Programa en Java:

Programa en Java:

```
package E7;
public class Main {
    public static void quickSort(int[] vector, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(vector, low, high);

            quickSort(vector, low, pivotIndex - 1);
            quickSort(vector, pivotIndex + 1, high);
        }
    }

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

        for (int j = low; j < high; j++) {
            if (vector[j] <= pivot) {
                i++;
                swap(vector, i, j);
            }
        }

        swap(vector, i + 1, high);
        return i + 1;
    }

    public static void swap(int[] vector, int i, int j) {
        int temp = vector[i];
        vector[i] = vector[j];
        vector[j] = temp;
    }

    public static void main(String[] args) {
        int[] vector = {5, 2, 8, 1, 6};
        int n = vector.length;

        System.out.println("Vector antes de ordenar:");
        for (int num : vector) {
            System.out.print(num + " ");
        }

        quickSort(vector, 0, n - 1);

        System.out.println("\nVector después de ordenar:");
        for (int num : vector) {
            System.out.print(num + " ");
        }
    }
}
```

Salida del programa

```
<terminated> Main (16) [Java Application] C:\
Vector antes de ordenar:
5 2 8 1 6
Vector después de ordenar:
1 2 5 6 8
```

- 8) Realizar en forma recursiva la función de Stirling

Programa en Java:

Programa en Java:

```
package E8;
public class Main {
    public static int stirling(int n, int k) {
        if (k == 0 || k > n) {
            return 0;
        } else if (n == k || k == 1) {
            return 1;
        } else {
            return k * stirling(n - 1, k) + stirling(n - 1, k - 1);
        }
    }

    public static void main(String[] args) {
        int n = 5;
        int k = 2;
        int resultado = stirling(n, k);
        System.out.println("El num de Stirling S("+n+", "+k+" ) es: "+resultado);
    }
}
```

Salida del programa

```
<terminated> Main (17) [Java Application] C:\Program F
El num de Stirling S(5, 2) es: 15
```

- 9) Implementar en forma recursiva la función de ackerman

Programa en Java:

Programa en Java:

```
package E9;
public class Main {
    public static int ackerman(int m, int n) {
        if (m == 0) {
            return n + 1;
        } else if (n == 0) {
            return ackerman(m - 1, 1);
        } else {
            return ackerman(m - 1, ackerman(m, n - 1));
        }
    }

    public static void main(String[] args) {
        int m = 2;
        int n = 3;
        int resultado = ackerman(m, n);
        System.out.println("El resultado de la función de Ackerman para");
        System.out.print("m = " + m + " y n = " + n + " es: " + resultado);
    }
}
```

Salida del programa

```
<terminated> Main (18) [Java Application] C:\Program Files\Java\jre1.8.0_341\l
El resultado de la función de Ackerman para
m = 2 y n = 3 es: 9|
```

10) Implementar en forma recursiva el problema de las torres de Hanoi

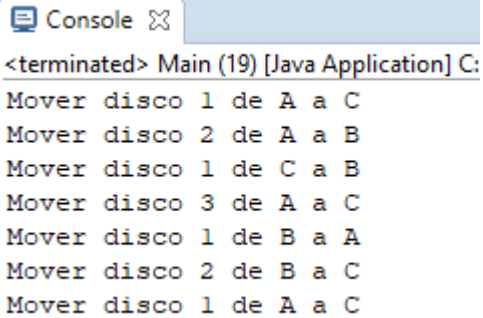
Programa en Java:

Programa en Java:

```
package E10;
public class Main {
    public static void torresH(int disks, char s, char aux, char dest) {
        if (disks == 1) {
            System.out.println("Mover disco 1 de " + s + " a " + dest);
        } else {
            torresH(disks - 1, s, dest, aux);
            System.out.println("Mover disco " + disks + " de " + s + " a " + dest);
            torresH(disks - 1, aux, s, dest);
        }
    }

    public static void main(String[] args) {
        int disks = 3;
        torresH(disks, 'A', 'B', 'C');
    }
}
```

Salida del programa



The screenshot shows a console window titled "Console" with a close button. The output text is as follows:

```
<terminated> Main (19) [Java Application] C:
Mover disco 1 de A a C
Mover disco 2 de A a B
Mover disco 1 de C a B
Mover disco 3 de A a C
Mover disco 1 de B a A
Mover disco 2 de B a C
Mover disco 1 de A a C
```



EJERCICIOS PROPUESTOS PARA LA PRÁCTICA NRO. 8

Realizar la implementación en Java de lo siguiente:

- 1) Dado un árbol binario de búsqueda, que contiene el nombre, el apellido y la edad de un grupo de personas, ordenados por edades.

Se pide: Hacer un algoritmo que visualice los datos de las personas de mayor a menor edad.

Programa en Java:

Programa en Java:

```
public void visualizarPersonasMayorAMenor() {
    ayudanteVisualizarMayorAMenor(raiz);
}

private void ayudanteVisualizarMayorAMenor(NodoArbol nodo) {
    if (nodo == null)
        return;
    ayudanteVisualizarMayorAMenor(nodo.nododerecho);
    System.out.println(nodo.datos.nombre + " " + nodo.datos.apellido + " - Edad: " +
nodo.datos.edad);
    ayudanteVisualizarMayorAMenor(nodo.nodoizquierdo);
}

public static void main(String args[]) {
    // Pregunta 1: Árbol binario de búsqueda con personas ordenadas por edades
    Arbol arbolPersonas = new Arbol();
    // Datos de prueba por defecto (nombre, apellido, edad)
    arbolPersonas(new Persona("Maria", "Gutierrez", 31));
    arbolPersonas(new Persona("Sara", "Peres", 27));
    arbolPersonas(new Persona("Angel", "Lopez", 35));
    arbolPersonas(new Persona("Diana", "Williams", 20));
    arbolPersonas(new Persona("Eva", "Miller", 40));
    arbolPersonas(new Persona("Severa", "Mamani", 31));
    arbolPersonas(new Persona("Pedro", "Picapiedra", 36));
    arbolPersonas(new Persona("Pablo", "Marbol", 38));
    arbolPersonas(new Persona("Ana", "Zeballos", 40));
    arbolPersonas(new Persona("Eva", "Quispe", 10));
    System.out.println("Datos de personas de mayor a menor edad:");
    arbolPersonas.visualizarPersonasMayorAMenor();
}
```


- 2) Dado un Árbol T, se pide invertir sus elementos:

Programa en Java:

Programa en Java:

```
public static Arbol invertirArbol(Arbol arbol) {
    Arbol arbolInvertido = new Arbol();
    invertirArbolRecursivo(arbol.raiz, arbolInvertido);
    return arbolInvertido;
}

private static void invertirArbolRecursivo(NodoArbol nodo, Arbol arbolInvertido)
{
    if (nodo != null) {
        arbolInvertido.insertarNodo(nodo.datos);
        invertirArbolRecursivo(nodo.nodoizquierdo, arbolInvertido);
        invertirArbolRecursivo(nodo.nododerecho, arbolInvertido);
    }
}

public static void main(String args[]) {
    // Pregunta 2: Invertir elementos del árbol
    Arbol arbolInvertido = invertirArbol(arbolPersonas);
    System.out.println("\nDatos del árbol invertido:");
    arbolInvertido.recorridoInorden();
}
```

- 3) Dado un Árbol Binario de Búsqueda (ABB) que contiene números enteros. Se pide: Hacer un procedimiento que reciba como parámetros: el puntero a raíz del árbol, y un número entero. El procedimiento debe buscar el elemento NUM y una vez encontrado visualizar todos sus antecesores (los anteriores).

Programa en Java:

Programa en Java:

```
public void visualizarPersonasMayorAMenor() {
    ayudanteVisualizarMayorAMenor(raiz);
}

private void ayudanteVisualizarMayorAMenor(NodoArbol nodo) {
    if (nodo == null)
        return;
    ayudanteVisualizarMayorAMenor(nodo.nododerecho);
    System.out.println(nodo.datos.nombre + " " + nodo.datos.apellido + " - Edad: " +
nodo.datos.edad);
    ayudanteVisualizarMayorAMenor(nodo.nodoizquierdo);
}

public static void main(String args[]) {
    // Pregunta 3: Buscar y visualizar antecesores de un número en un ABB
    int numeroBuscado = 15;
    System.out.println("\nAntecesores de " + numeroBuscado + ":");
    arbolABB.visualizarAntecesores(numeroBuscado);
}
```

- 4) Hacer una función que guarde en un archivo secuencial un árbol binario simétrico. Definir las estructuras necesarias.

Programa en Java:

```
Programa en Java:
public static void guardarArbolEnArchivo(Arbol arbol, String nombreArchivo) {
    try {
        FileWriter fw = new FileWriter(nombreArchivo);
        guardarArbolRecursivo(arbol.raiz, fw);
        fw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void guardarArbolRecursivo(NodoArbol nodo, FileWriter fw) throws
IOException {
    if (nodo != null) {
        guardarArbolRecursivo(nodo.nodoizquierdo, fw);
        fw.write(nodo.datos.edad + "\n");
        guardarArbolRecursivo(nodo.nododerecho, fw);
    }
}

public static void main(String args[]) {
    // Pregunta 4:
    guardarArbolEnArchivo(arbolSimetrico, "arbol_simetrico.txt");
}
}
```

- 5) Hacer una función que genere un árbol binario simétrico leyendo los datos desde un archivo secuencial. Definir las estructuras necesarias.

Programa en Java:

```
Programa en Java:
public static Arbol generarArbolDesdeArchivo(String nombreArchivo) {
    Arbol arbolDesdeArchivo = new Arbol();
    try {
        BufferedReader br = new BufferedReader(new FileReader(nombreArchivo));
        String linea;
        while ((linea = br.readLine()) != null) {
            int edad = Integer.parseInt(linea);
            arbolDesdeArchivo.insertarNodo(new Persona("", "", edad));
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return arbolDesdeArchivo;
}

public static void main(String args[]) {
    // Pregunta 5: Generar árbol binario simétrico leyendo datos desde un archivo
    Arbol arbolDesdeArchivo = generarArbolDesdeArchivo("arbol_simetrico.txt");
    System.out.println("\nDatos del árbol desde archivo:");
    arbolDesdeArchivo.recorridoInorden();
}
}
```

Salida del programa

```
Recorrido Inorden Descendentemente
Nombre      Apellido      Edad:
Ana      Zaballos      40
Eva      Miller      40
Pablo    Marbol      38
Pedro    Picapiedra      36
Angel    Lopez      35
Severa   Mamani      31
Maria    Gutierrez      31
Sara     Peres      27
Diana    Williams      20
Eva      Quispe      10
```

```
Invertiendo el árbol binario
Nombre      Apellido      Edad:
Eva      Quispe      10
Diana    Williams      20
Sara     Peres      27
Maria    Gutierrez      31
Severa   Mamani      31
Angel    Lopez      35
Pedro    Picapiedra      36
Pablo    Marbol      38
Eva      Miller      40
Ana      Zaballos      40
```

Ingrese la edad para buscar los antecesores 15
Buscar antecesores de la Personaa con edad 15:

Guardar árbol simétrico en formato de cadena:
31 35 40 40 36 38 31 27 20 10

LABORATORIO NRO. 7

NOMBRE COMPLETO: MAYTA CUEVAS ANDY BORIS

DOCENTE: M. Sc. ZARA YUJRA CAMA **GESTION:**2023

MATERIA: ESTRUCTURA DE DATOS (INF-131)

En cada uno de los ejercicios se pide implementar en el lenguaje Java las Funciones Recursivas correspondientes invocándolas desde el programa principal:

- 1) Hallar el máximo común divisor
- 2) Hallar el mínimo común múltiplo
- 3) Sumar los elementos de un vector V de dimensión N.
- 4) Hallar el valor mínimo en un vector V de dimensión N.
- 5) Sumar los elementos de una matriz A de dimensión NxM.
- 6) Realizar la búsqueda binaria recursiva en un Vector V de dimensión N.
- 7) Realizar la ordenación de elementos de un Vector V de dimensión N con el método Quick Sort.
- 8) Realizar en forma recursiva la función de Stirling
- 9) Implementar en forma recursiva la función de ackerman
- 10) Implementar en forma recursiva el problema de las torres de Hanoi

```
package practica7;
public class Main {
    public static void main(String[] args) {
        // 1) Hallar el máximo común divisor
        int a = 150;
        int b = 250;
        int gcdResult = mcd(a, b);
        System.out.println("\n----- 1 -----");
        System.out.println("El máximo común divisor de " + a + " y " + b
+ " es: " + gcdResult);

        // 2) Hallar el mínimo común múltiplo
        int lcmResult = mcm(a, b);
        System.out.println("\n----- 2 -----");
        System.out.println("El mínimo común múltiplo de " + a + " y " + b
+ " es: " + lcmResult);

        // 3) Sumar los elementos de un vector V de dimensión N
        int[] vector = {1, 2, 3, 4, 5};
        int vectorSum = sumVector(vector, vector.length);
        System.out.println("\n----- 3 -----");
        System.out.println("La suma de los elementos del vector " +
listaCadenas(vector) + " es: " + vectorSum);

        // 4) Hallar el valor mínimo en un vector V de dimensión N
        int vectorMin = buscarMenor(vector, vector.length);
        System.out.println("\n----- 4 -----");
        System.out.println("El valor mínimo en el vector " +
listaCadenas(vector) + " es: " + vectorMin);
    }
}
```

```

// 5) Sumar los elementos de una matriz A de dimensión NxM
int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int matrixSum = sumaMatriz(matrix, matrix.length - 1,
matrix[0].length - 1);
System.out.println("\n-----      5 -----");
System.out.println("La suma de los elementos de la matriz " +
matrizCadena(matrix) + " es: " + matrixSum);

// 6) Realizar la búsqueda binaria recursiva en un Vector V de
dimensión N
int target = 5;
int binarySearchResult = binarioBusqueda(vector, 0, vector.length
- 1, target);
System.out.println("\n-----      6 -----");
System.out.println("El elemento " + target + " se encuentra en la
posición: " + binarySearchResult);

// 7) Realizar la ordenación de elementos de un Vector V de
dimensión N con el método Quick Sort
System.out.println("\n-----      7 -----");
System.out.println("Vector antes de ordenar: " +
listaCadenas(vector));
quickSort(vector, 0, vector.length - 1);
System.out.println("Vector después de ordenar: " +
listaCadenas(vector));

// 8) Realizar en forma recursiva la función de Stirling
int n = 5;
int k = 2;
int stirlingResult = stirling(n, k);
System.out.println("\n-----      8 -----");
System.out.println("El número de Stirling S(" + n + ", " + k + ")
es: " + stirlingResult);

// 9) Implementar en forma recursiva la función de Ackerman
int m = 2;
int ackermanResult = ackerman(m, n);
System.out.println("\n-----      9 -----");
System.out.println("El resultado de la función de Ackerman para m
= " + m + " y n = " + n + " es: " + ackermanResult);

// 10) Implementar en forma recursiva el problema de las torres
de Hanoi
int disks = 3;
System.out.println("\n-----      10 -----");
System.out.println("Movimientos de las Torres de Hanoi para " +
disks + " discos:");
torresH(disks, 'A', 'B', 'C');
}

// Funciones recursivas
// 1) Hallar el máximo común divisor
public static int mcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return mcd(b, a % b);
}

```

```

    }

    // 2) Hallar el mínimo común múltiplo
    public static int mcm(int a, int b) {
        return (a * b) / mcd(a, b);
    }

    // 3) Sumar los elementos de un vector V de dimensión N
    public static int sumVector(int[] vector, int n) {
        if (n == 0) {
            return 0;
        }
        return vector[n - 1] + sumVector(vector, n - 1);
    }

    // 4) Hallar el valor mínimo en un vector V de dimensión N
    public static int buscarMenor(int[] vector, int n) {
        if (n == 1) {
            return vector[0];
        }
        return Math.min(vector[n - 1], buscarMenor(vector, n - 1));
    }

    // 5) Sumar los elementos de una matriz A de dimensión NxM
    public static int sumaMatriz(int[][] matrix, int row, int col) {
        if (row < 0) {
            return 0;
        }
        if (col < 0) {
            return sumaMatriz(matrix, row - 1, matrix[0].length - 1);
        }
        return matrix[row][col] + sumaMatriz(matrix, row, col - 1);
    }

    // 6) Realizar la búsqueda binaria recursiva en un Vector V de dimensión N
    public static int binarioBusqueda(int[] vector, int left, int right,
    int target) {
        if (left <= right) {
            int mid = left + (right - left) / 2;
            if (vector[mid] == target) {
                return mid;
            }
            if (vector[mid] < target) {
                return binarioBusqueda(vector, mid + 1, right, target);
            }
            return binarioBusqueda(vector, left, mid - 1, target);
        }
        return -1;
    }

    // 7) Realizar la ordenación de elementos de un Vector V de dimensión N con el método Quick Sort
    public static void quickSort(int[] vector, int a, int b) {
        if (a < b) {
            int pivotIndex = particion(vector, a, b);
            quickSort(vector, a, pivotIndex - 1);

```

```

        quickSort(vector, pivotIndex + 1, b);
    }
}

public static int particion(int[] vector, int low, int high) {
    int pivot = vector[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (vector[j] < pivot) {
            i++;
            swap(vector, i, j);
        }
    }

    swap(vector, i + 1, high);
    return i + 1;
}

public static void swap(int[] vector, int i, int j) {
    int temp = vector[i];
    vector[i] = vector[j];
    vector[j] = temp;
}

// 8) Realizar en forma recursiva la función de Stirling
public static int stirling(int n, int k) {
    if (k == 0 || k > n) {
        return 0;
    }
    if (n == k || k == 1) {
        return 1;
    }
    return k * stirling(n - 1, k) + stirling(n - 1, k - 1);
}

// 9) Implementar en forma recursiva la función de Ackerman
public static int ackerman(int m, int n) {
    if (m == 0) {
        return n + 1;
    }
    if (n == 0) {
        return ackerman(m - 1, 1);
    }
    return ackerman(m - 1, ackerman(m, n - 1));
}

// 10) Implementar en forma recursiva el problema de las torres de Hanoi
public static void torresH(int disks, char source, char aux, char
dest) {
    if (disks == 1) {
        System.out.println("Mover disco 1 de " + source + " a " +
dest);
    } else {
        torresH(disks - 1, source, dest, aux);
    }
}

```

```

        System.out.println("Mover disco " + disks + " de " + source +
" a " + dest);
        torresH(disks - 1, aux, source, dest);
    }
}

// Funciones auxiliares

public static String listaCadenas(int[] array) {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    for (int i = 0; i < array.length; i++) {
        sb.append(array[i]);
        if (i < array.length - 1) {
            sb.append(", ");
        }
    }
    sb.append("]");
    return sb.toString();
}

public static String matrizCadena(int[][] matrix) {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    for (int i = 0; i < matrix.length; i++) {
        sb.append(listaCadenas(matrix[i]));
        if (i < matrix.length - 1) {
            sb.append(", ");
        }
    }
    sb.append("]");
    return sb.toString();
}
}

```



```

----- 1 -----
El máximo común divisor de 150 y 250 es: 50

----- 2 -----
El mínimo común múltiplo de 150 y 250 es: 750

----- 3 -----
La suma de los elementos del vector [1, 2, 3, 4, 5] es: 15

----- 4 -----
El valor mínimo en el vector [1, 2, 3, 4, 5] es: 1

----- 5 -----
La suma de los elementos de la matriz [[1, 2, 3], [4, 5, 6], [7, 8, 9]] es: 45

----- 6 -----
El elemento 5 se encuentra en la posición: 4

----- 7 -----
Vector antes de ordenar: [1, 2, 3, 4, 5]
Vector después de ordenar: [1, 2, 3, 4, 5]

----- 8 -----
El número de Stirling  $S(5, 2)$  es: 15

----- 9 -----
El resultado de la función de Ackerman para  $m = 2$  y  $n = 5$  es: 13

----- 10 -----
Movimientos de las Torres de Hanoi para 3 discos:
Mover disco 1 de A a C
Mover disco 2 de A a B
Mover disco 1 de C a B
Mover disco 3 de A a C
Mover disco 1 de B a A
Mover disco 2 de B a C
Mover disco 1 de A a C

```

Activar \

LABORATORIO NRO. 8

NOMBRE COMPLETO: MAYTA CUEVAS ANDY BORIS

DOCENTE: M. Sc. ZARA YUJRA CAMA **GESTION:**2023

MATERIA: ESTRUCTURA DE DATOS (INF-131)

Realizar la implementación en Java de lo siguiente:

- 1) Dado un árbol binario de búsqueda, que contiene el nombre, el apellido y la edad de un grupo de personas, ordenados por edades. Se pide: Hacer un algoritmo que visualice los datos de las personas de mayor a menor edad.
- 2) Dado un Árbol T, se pide invertir sus elementos:
- 3) Dado un Árbol Binario de Búsqueda (ABB) que contiene números enteros. Se pide: Hacer un procedimiento que reciba como parámetros: el puntero a raíz del árbol, y un número entero. El procedimiento debe buscar el elemento NUM y una vez encontrado visualizar todos sus antecesores (los anteriores).
- 4) Hacer una función que guarde en un archivo secuencial un árbol binario simétrico. Definir las estructuras necesarias.
- 5) Hacer una función que genere un árbol binario simétrico

```
package practica8;
import java.util.Scanner;
public class Persona {
    String nombre;
    String apellido;
    int edad;

    public Persona(String nombre, String apellido, int edad) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}
```

```
package practica8;
import java.util.Scanner;
public class BinarioArbol {
    private static class NodoArbol {
        NodoArbol nodoizquierdo;
        Persona data;
        NodoArbol nododerecho;

        public NodoArbol(Persona data) {
            this.data = data;
            nodoizquierdo = nododerecho = null;
        }

        public synchronized void insertar(Persona Persona) {
            if (Persona.edad < data.edad) {
                if (nodoizquierdo == null)
                    nodoizquierdo = new NodoArbol(Persona);
                else
```

```

        nodoizquierdo.insertar(Persona);
    } else {
        if (nododerecho == null)
            nododerecho = new NodoArbol(Persona);
        else
            nododerecho.insertar(Persona);
    }
}

private NodoArbol raiz;

public BinarioArbol() {
    raiz = null;
}

public synchronized void insertarNodo(Persona Persona) {
    if (raiz == null)
        raiz = new NodoArbol(Persona);
    else
        raiz.insertar(Persona);
}

private void ayudanteInordenDesc(NodoArbol nodo) {
    if (nodo == null)
        return;
    ayudanteInordenDesc(nodo.nododerecho);
    System.out.println("Nombre: " + nodo.data.nombre + " Apellido: "
+ nodo.data.apellido + " Edad: " + nodo.data.edad);
    ayudanteInordenDesc(nodo.nodoizquierdo);
}

public synchronized void recorridoInordenDesc() {
    ayudanteInordenDesc(raiz);
}

private void invertArbol(NodoArbol nodo) {
    if (nodo == null)
        return;

    NodoArbol temp = nodo.nodoizquierdo;
    nodo.nodoizquierdo = nodo.nododerecho;
    nodo.nododerecho = temp;

    invertArbol(nodo.nodoizquierdo);
    invertArbol(nodo.nododerecho);
}

public synchronized void invertArbol() {
    invertArbol(raiz);
}

private boolean findAndPrintAncestors(NodoArbol nodo, int target) {
    if (nodo == null)
        return false;

    if (nodo.data.edad == target)

```

```

        return true;

    if (findAndPrintAncestors(nodo.nodoizquierdo, target) ||
        findAndPrintAncestors(nodo.nododerecho, target))
    {
        System.out.println("Nombre: " + nodo.data.nombre + "
Apellido: " + nodo.data.apellido + " Edad: " + nodo.data.edad);
        return true;
    }

    return false;
}

public synchronized void findAndPrintAncestors(int target) {
    findAndPrintAncestors(raiz, target);
}

private void saveSymmetricArbolToString(StringBuilder sb, NodoArbol
nodo) {
    if (nodo == null)
        return;

    sb.append(nodo.data.edad).append(" ");
    saveSymmetricArbolToString(sb, nodo.nodoizquierdo);
    saveSymmetricArbolToString(sb, nodo.nododerecho);
}

public synchronized String getSymmetricArbolAsString() {
    StringBuilder sb = new StringBuilder();
    saveSymmetricArbolToString(sb, raiz);
    return sb.toString();
}

public static void main(String[] args) {
    BinarioArbol Arbol = new BinarioArbol();

    Arbol.insertarNodo(new Persona("Alice", "Johnson", 30));
    Arbol.insertarNodo(new Persona("Bob", "Smith", 25));
    Arbol.insertarNodo(new Persona("Charlie", "Brown", 35));
    Arbol.insertarNodo(new Persona("Diana", "Williams", 20));
    Arbol.insertarNodo(new Persona("Eva", "Miller", 40));

    System.out.println("Recorrido Inorden Descendente:");
    Arbol.recorridoInordenDesc();

    System.out.println("\nInvertir el árbol:");
    Arbol.invertArbol();
    Arbol.recorridoInordenDesc();

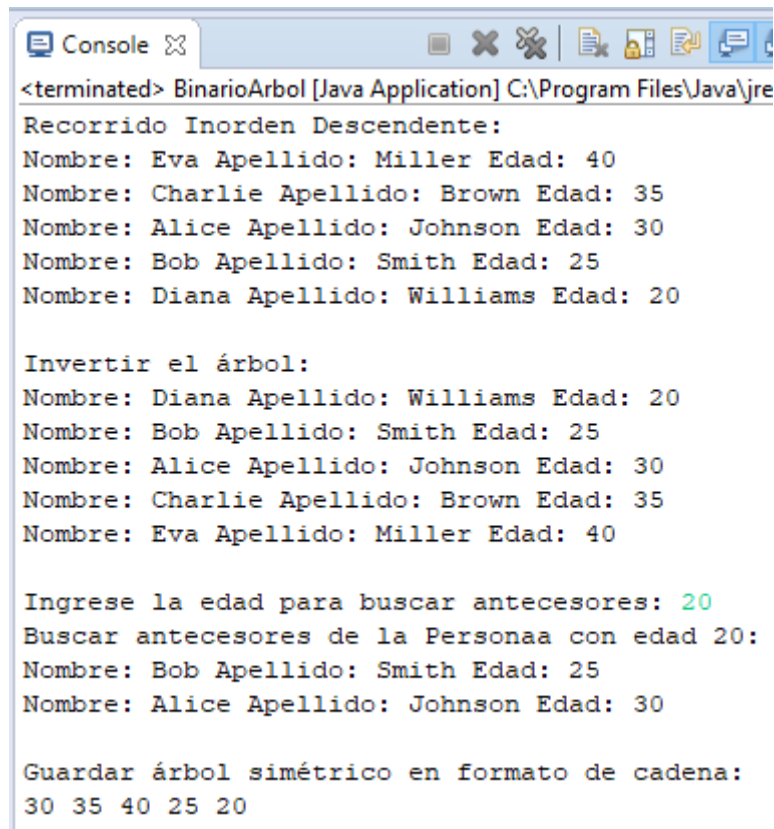
    Scanner scanner = new Scanner(System.in);
    System.out.print("\nIngrese la edad para buscar antecesores: ");
    int targetAge = scanner.nextInt();
    System.out.println("Buscar antecesores de la Personaa con edad "
+ targetAge + ":");
    Arbol.findAndPrintAncestors(targetAge);
}

```

```

        System.out.println("\nGuardar árbol simétrico en formato de
cadena:");
        String symmetricArbolString = Arbol.getSymmetricArbolAsString();
        System.out.println(symmetricArbolString);
    }
}

```



The screenshot shows a Java application window titled "BinarioArbol [Java Application] C:\Program Files\Java\jre". The console output is as follows:

```

<terminated> BinarioArbol [Java Application] C:\Program Files\Java\jre
Recorrido Inorden Descendente:
Nombre: Eva Apellido: Miller Edad: 40
Nombre: Charlie Apellido: Brown Edad: 35
Nombre: Alice Apellido: Johnson Edad: 30
Nombre: Bob Apellido: Smith Edad: 25
Nombre: Diana Apellido: Williams Edad: 20

Invertir el árbol:
Nombre: Diana Apellido: Williams Edad: 20
Nombre: Bob Apellido: Smith Edad: 25
Nombre: Alice Apellido: Johnson Edad: 30
Nombre: Charlie Apellido: Brown Edad: 35
Nombre: Eva Apellido: Miller Edad: 40

Ingrese la edad para buscar antecesores: 20
Buscar antecesores de la Personaa con edad 20:
Nombre: Bob Apellido: Smith Edad: 25
Nombre: Alice Apellido: Johnson Edad: 30

Guardar árbol simétrico en formato de cadena:
30 35 40 25 20

```