



Universidad Nacional Mayor de San Marcos  
Universidad del Perú. Decana de América

# Estructura de Datos

---

Semana 9

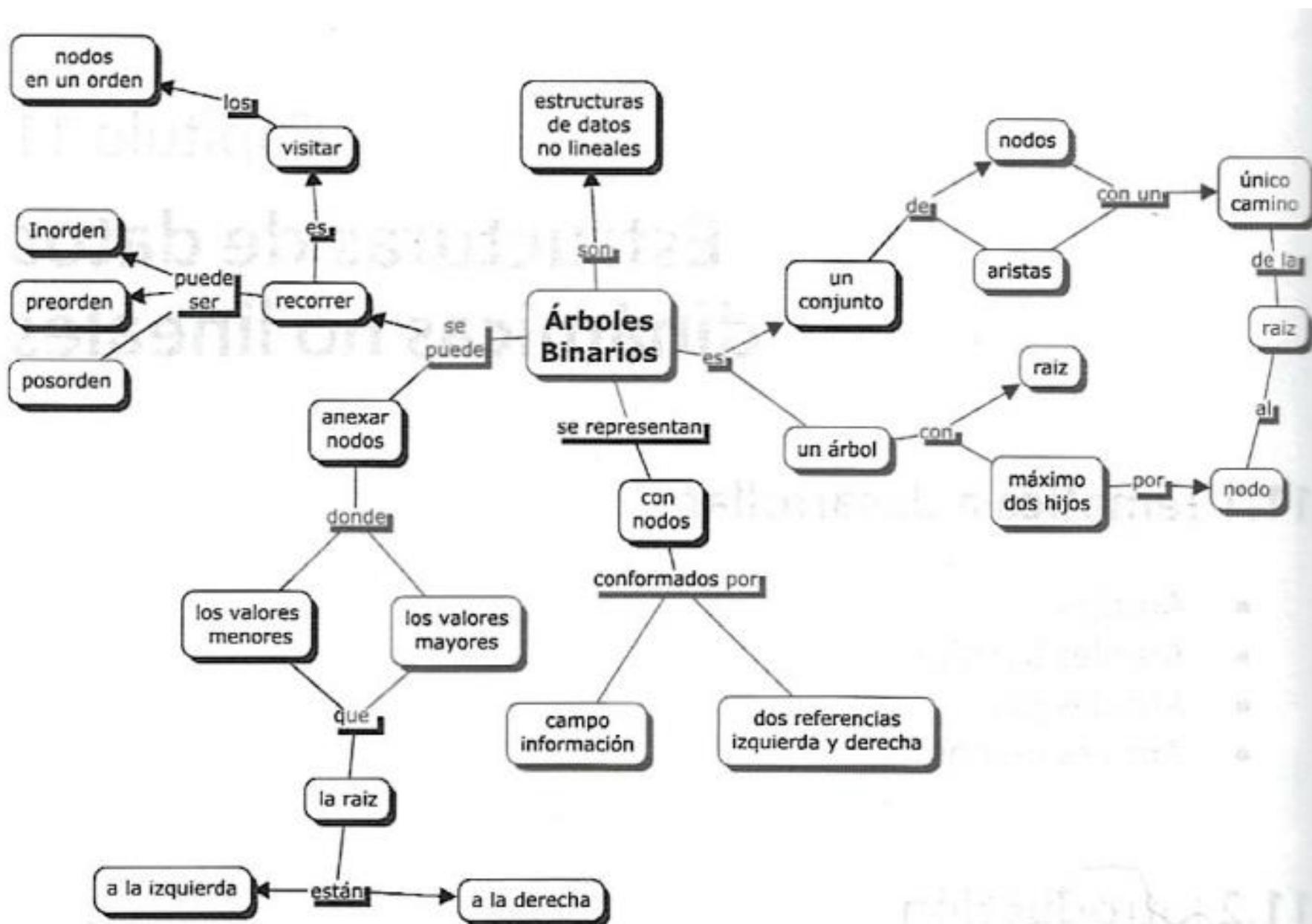


# Logro de la sesión

Al finalizar la sesión, el estudiante:

- Identifica, analiza y resuelve problemas algorítmicos que usan el tipo abstracto de datos: árboles, desarrollando funciones/métodos que usen esas estructuras.

# DEFINICIÓN DE ARBOL



## DEFINICION DE ARBOL

Los árboles son estructuras de datos jerarquizadas, organizadas y dinámicas. Formada por un conjunto de nodos y un conjunto de punteros que conectan pares de nodos.

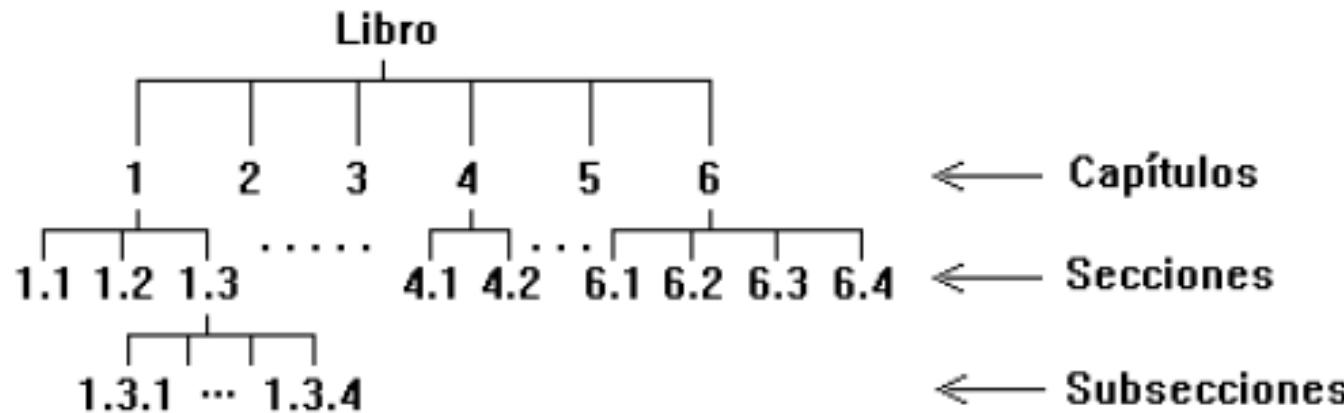
- *Jerárquica* porque los componentes están a distinto nivel.
- *Organizada* porque importa la forma en que este dispuesto el contenido.
- *Dinámica* porque su forma, tamaño y contenido pueden variar durante la ejecución.

Para que se utilizan

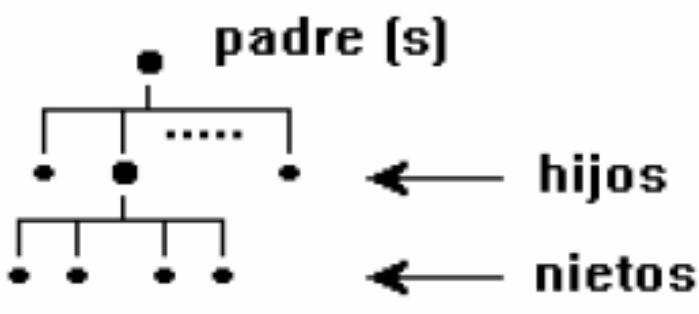
Son Estructuras de Datos “No Lineales”

- ✓ Representar Fórmulas Algebraicas.
- ✓ Organizar Objetos.
- ✓ Inteligencia Artificial.
- ✓ Algoritmos de Cifrado.

# Ejemplos de Estructuras Tipo Árbol



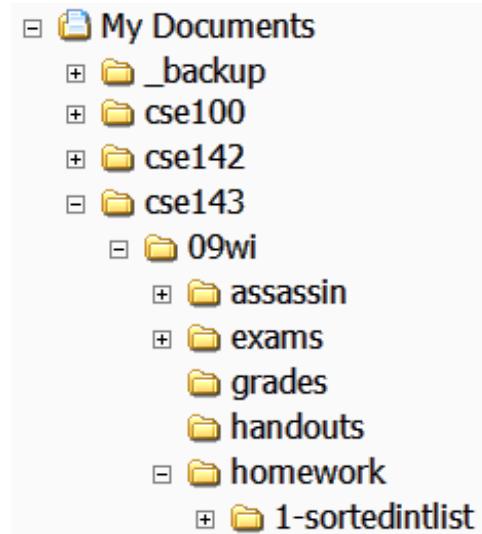
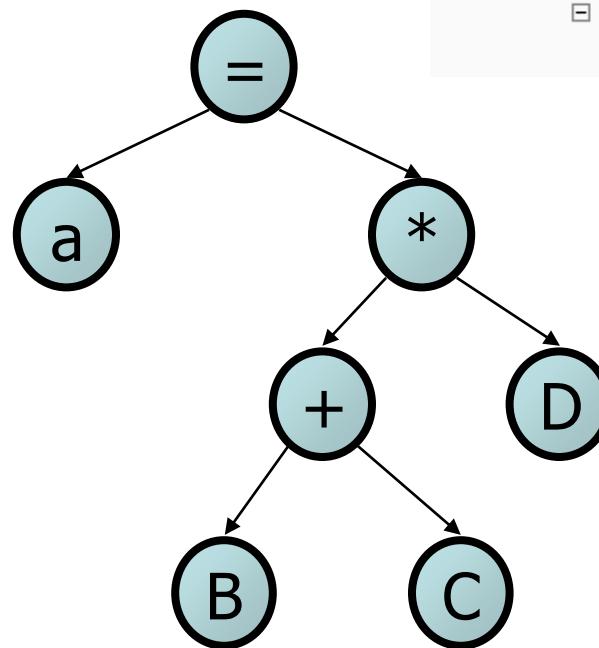
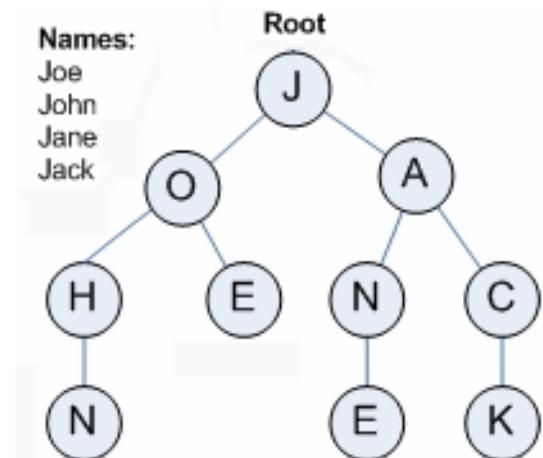
## ► Arboles genealógicos



Al primer nodo se le conoce con el nombre de RAIZ

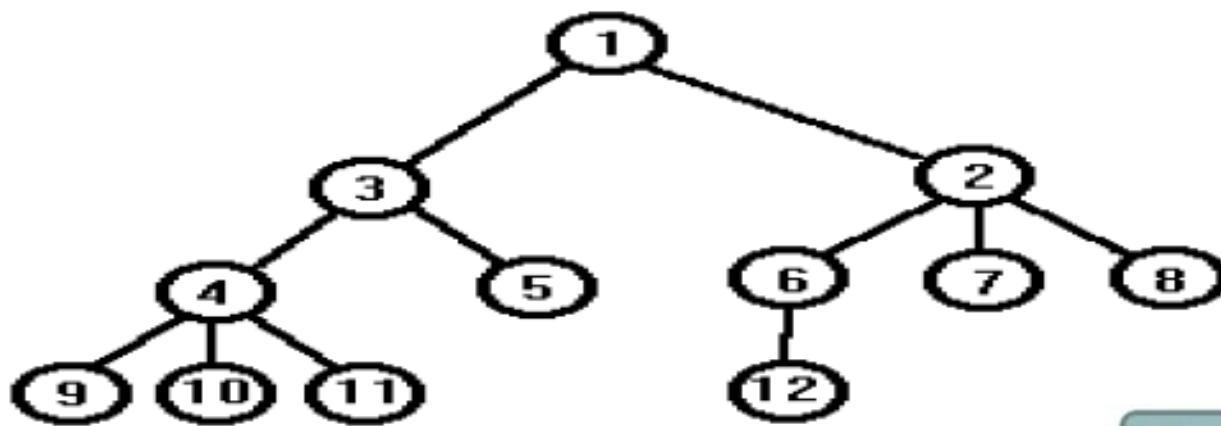
# Árboles en informática

- carpetas/archivos en una computadora
- genealogía familiar; organigramas
- IA: árboles de decisión
- compiladores: árbol de análisis
  - $a = (b + c) * d;$
- Tries (reTrieval)



# Representación gráfica

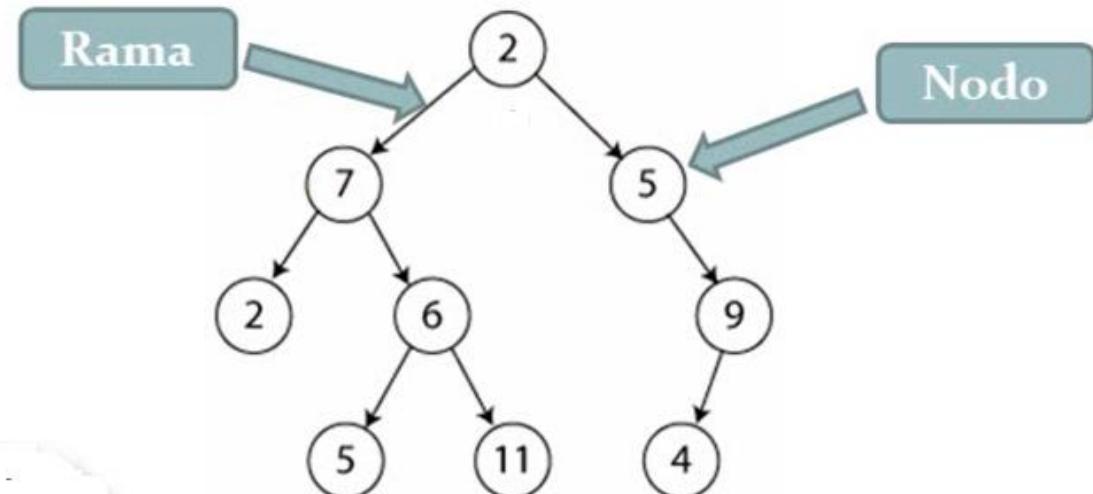
- La forma mas común, Grafos con la raíz hacia arriba:



¿De qué esta compuesto un Árbol?

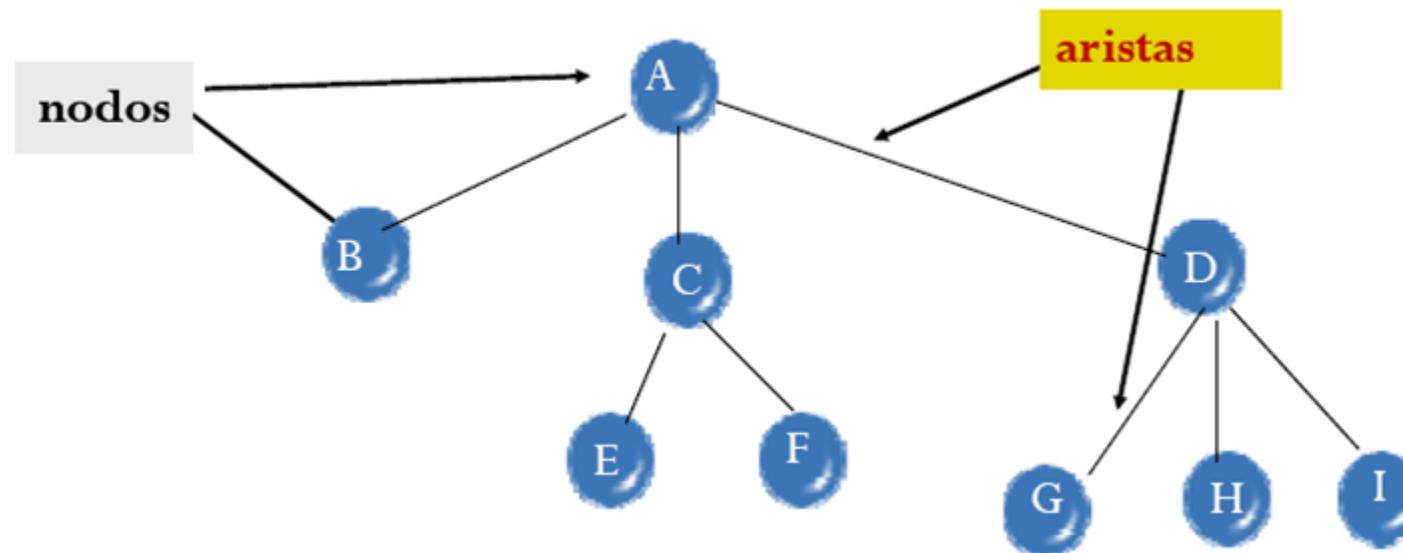
Consta de un conjunto finito de elementos llamados "Nodos"

Consta de un conjunto finito de líneas dirigidas llamadas "Ramas"



## ¿Qué es un árbol?

- Un árbol es una estructura de datos no lineal formada por nodos los cuales están conectados por aristas.
- Un árbol es un conjunto finito  $T$  de uno o mas nodos tal que, existe un nodo especial llamado raíz



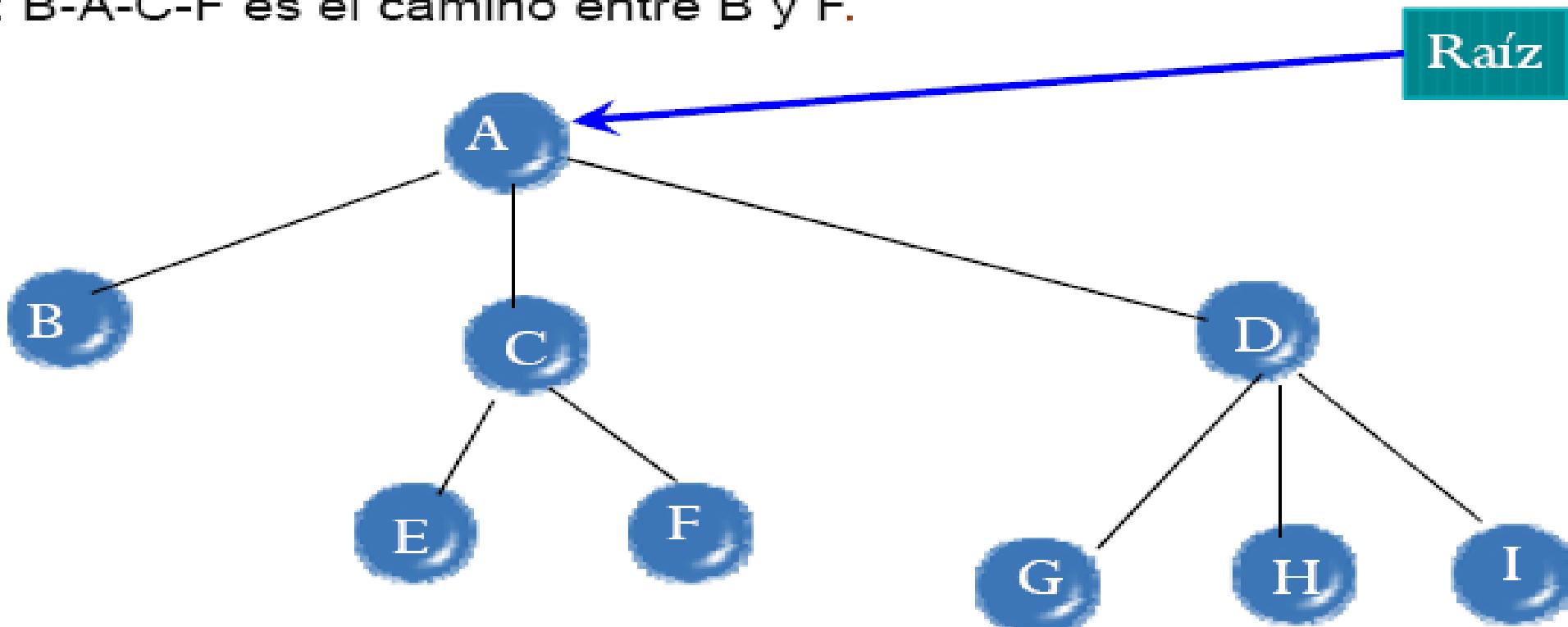
# Conceptos Básicos

**ARBOL VACIO:** No contiene ningún nodo.

**RAÍZ:** Es el nodo que está al tope del árbol. Un árbol solo tiene una raíz.

**CAMINO:** Es la secuencia de nodos que hay que visitar para llegar de un nodo a otro de un árbol.

Ejemplo: B-A-C-F es el camino entre B y F.

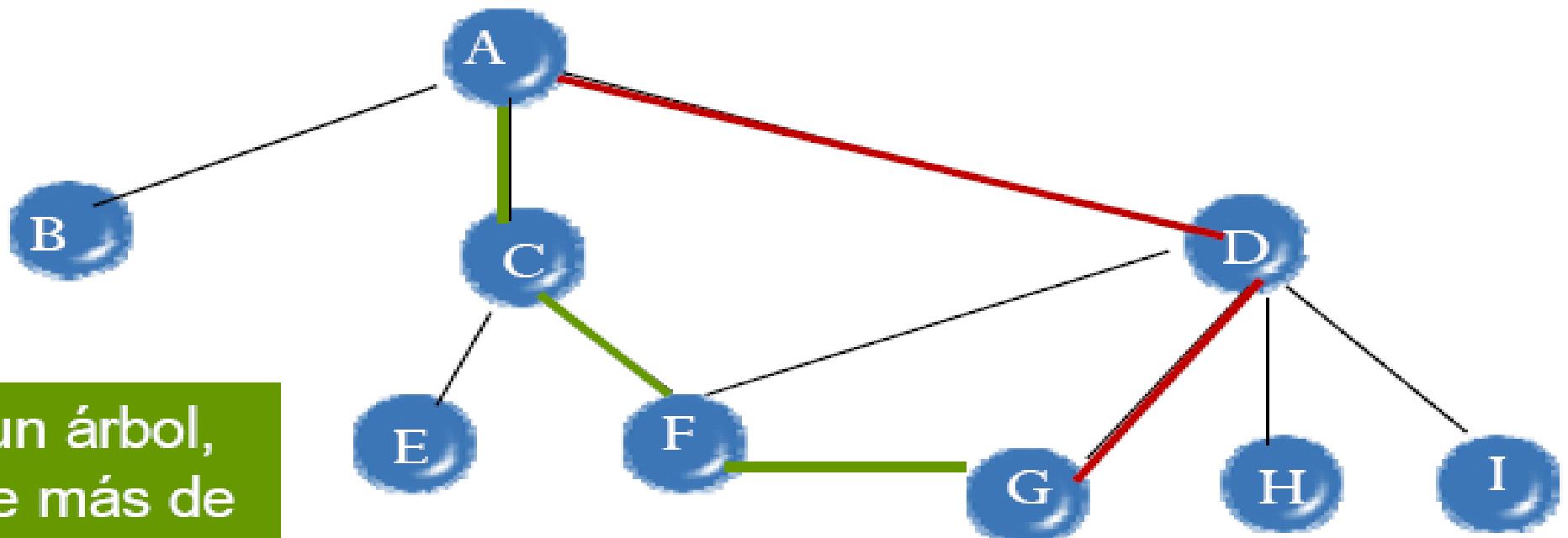


## Conceptos Básicos

Se define como un árbol, **al conjunto de nodos y aristas:**

Si y solo **si existe un único camino desde la raíz hasta cada uno de sus nodos.**

Esto no es un árbol,  
porque existe más de  
un camino desde la  
raíz para llegar a  
algunos nodos



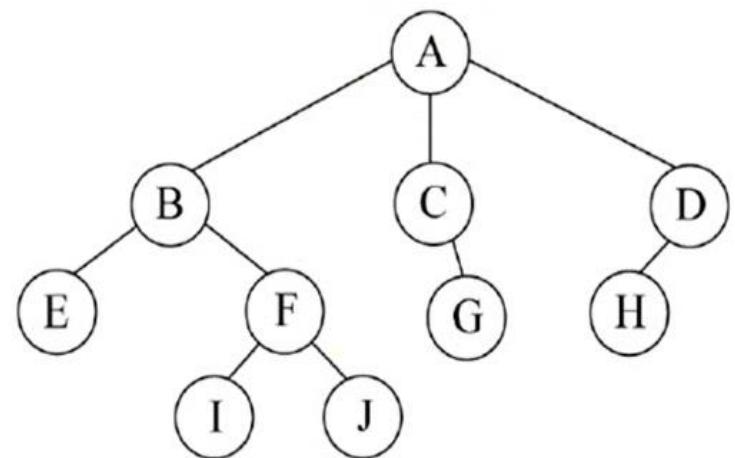
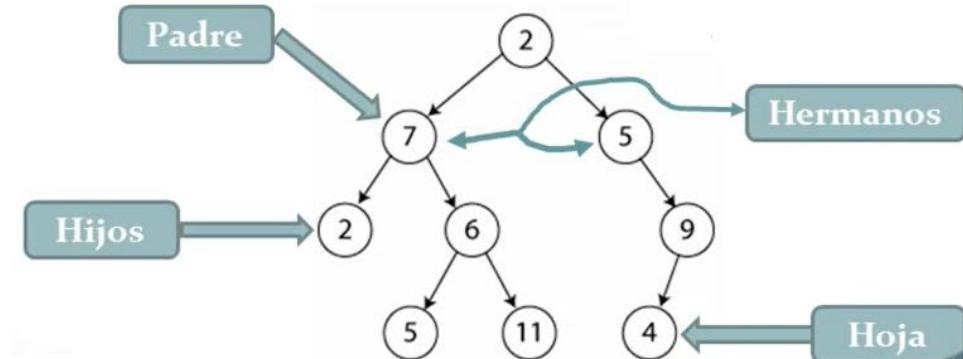
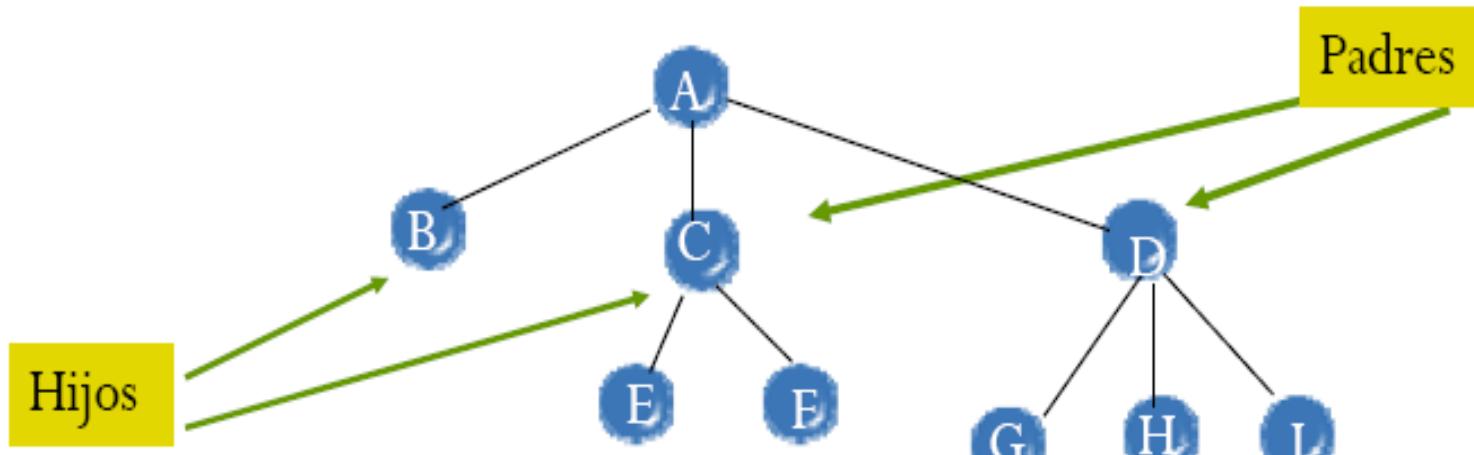
# Conceptos Básicos

**PADRE:** En un árbol toda rama va de un nodo n1 a un nodo n2, se dice que n1 es padre de n2.

Ejemplo: C es padre de E y de F, D es padre de G, de H y de I.

**HIJO:** todo nodo puede tener mas de una arista que lo lleva a otro nodo por debajo de él. Estos nodos que se encuentran por debajo de un nodo dado se llaman hijos.

Ejemplo: E es hijo de C, B es hijo de A, H es hijo de D



Padres: A, B, C, D, F  
Hijos: B, C, D, E, F, G, H, I, J  
Hermanos: {B, C, D}, {E, F}, {I, J}  
Hojas: G, H, I, J

# Conceptos Básicos

**Hojas:** Nodos que no tienen hijos. En un árbol solo puede haber una raíz pero pueden haber muchas hojas.

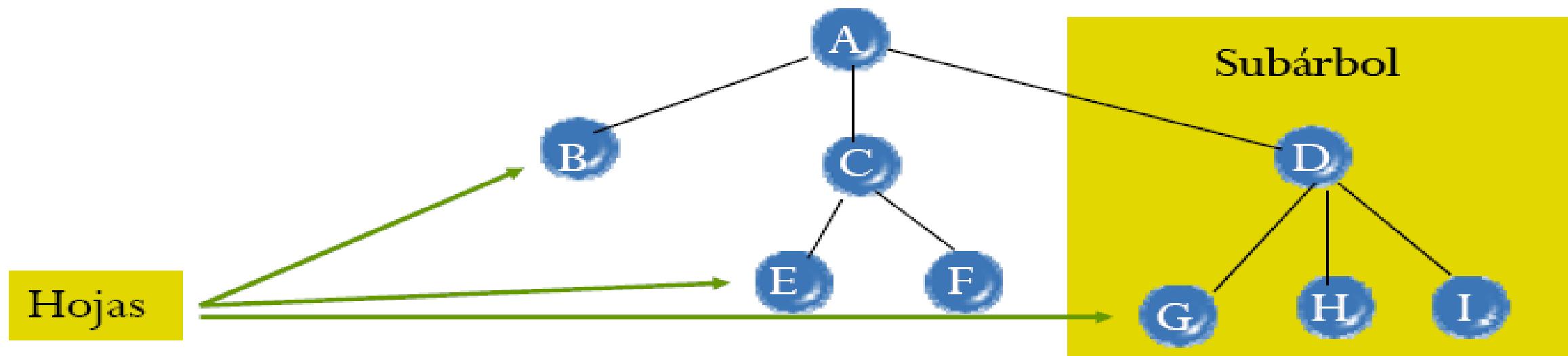
Ejemplo: B,E,F,G,H,I son hojas.

**Subárbol:** Cualquier nodo es la raíz de un subárbol.

**Visitar:** Un nodo es visitado cuando el control del programa se detiene en él para *hacer alguna operación* tal como: acceder, imprimir, modificar su data.

**Recorrer:** Significa visitar todos sus nodos *en un orden específico*.

Ejemplo: B-A-C-E-F-D-G-H-I



## Conceptos Básicos

**Nivel:** De un nodo es:

El número de generaciones que hay desde la raíz hasta él.

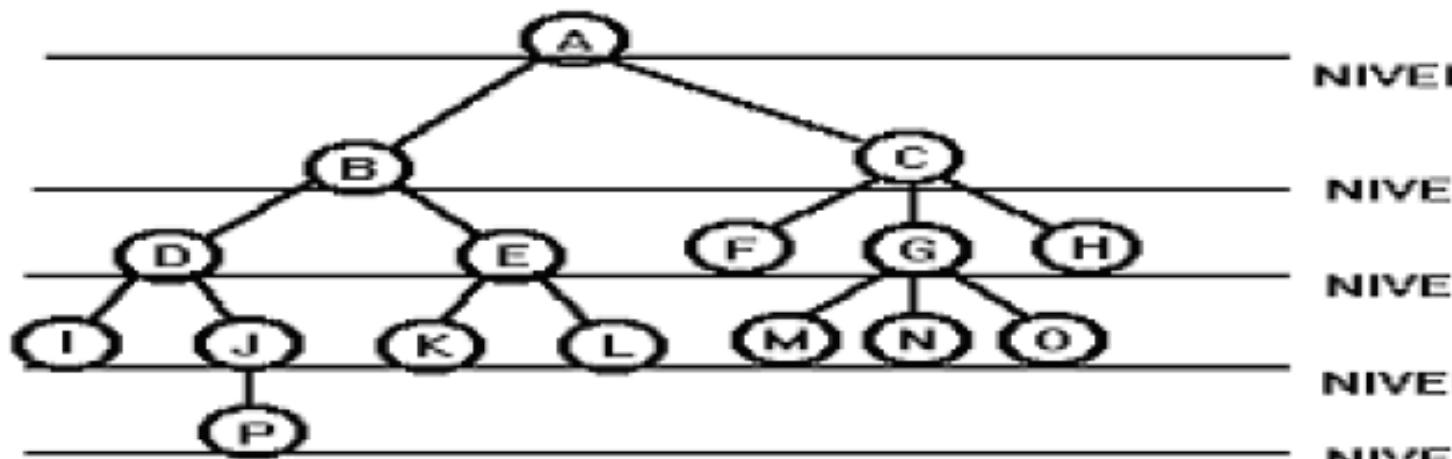
El número de nodos en el camino de este hacia la raíz

**Profundidad o altura:**

Es la longitud del camino más largo desde la raíz hasta una hoja.

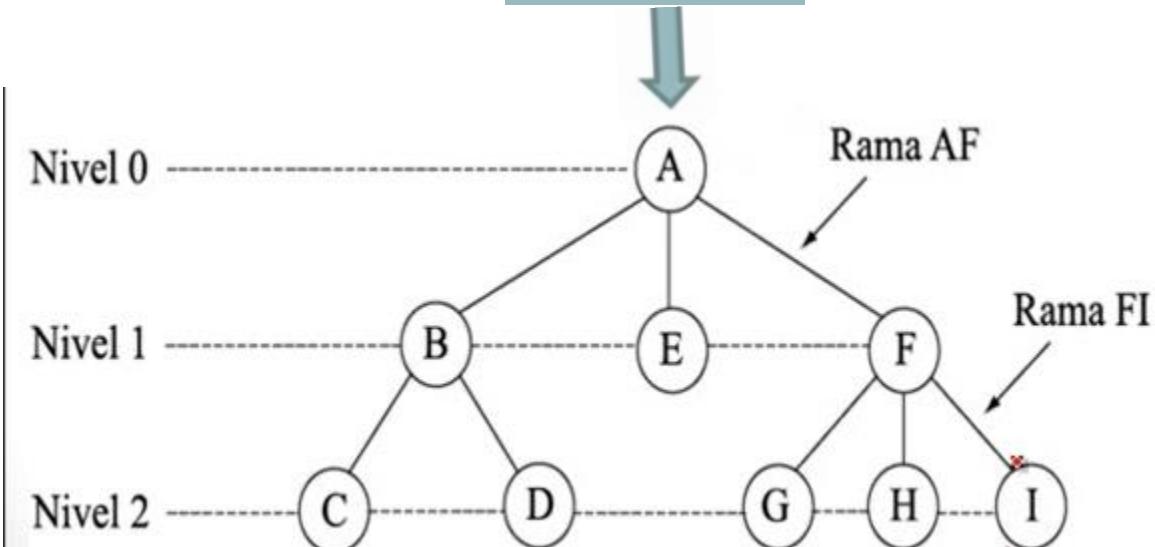
La profundidad de este árbol es 5 La raíz tiene profundidad 0.

El nivel máximo de todos los nodos en el árbol



Profundidad: 5

Árbol con  
Altura 3



La “Altura o Profundidad” de un Árbol es el nivel de la hoja del camino más largo desde la raíz más uno. Por definición , la altura de un árbol vacío es 0.

## Propiedades

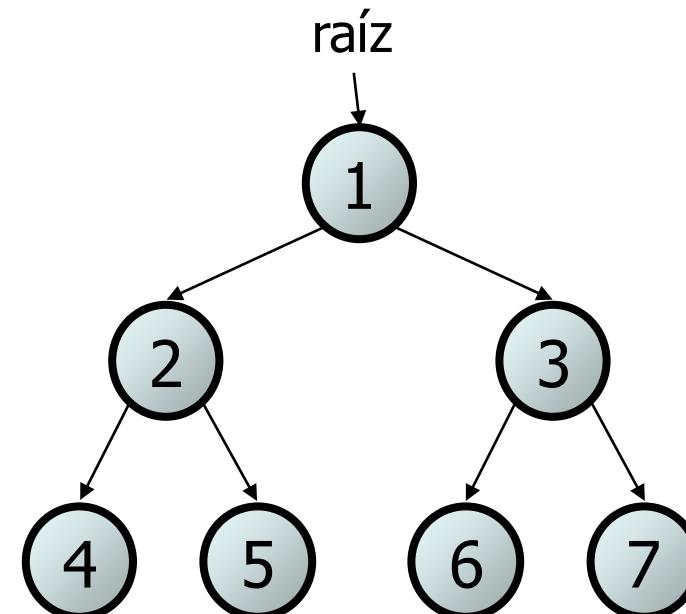
- Cada vértice o nodo puede tener un nombre y una información asociada
- Un camino es una lista de vértices diferentes en los que vértices sucesivos están conectados por arcos en el árbol
- La longitud de un camino es el número de nodos del camino menos uno. Por tanto, hay un camino de longitud cero de cualquier nodo a si mismo.
- Un nodo Y está abajo de un nodo X, si X está en el camino de Y a la raíz
- Cada nodo, excepto la raíz, tiene exactamente un nodo arriba de él, que es llamado su padre
- Los nodos que están exactamente abajo de él son llamados sus hijos

## Propiedades

- **El grado del nodo** es el número de hijos que cuelgan de un nodo.
- **El grado de un árbol** es el grado máximo de los nodos del árbol.
- **Un nodo de grado cero** es llamado hoja, es decir, no tiene hijos.

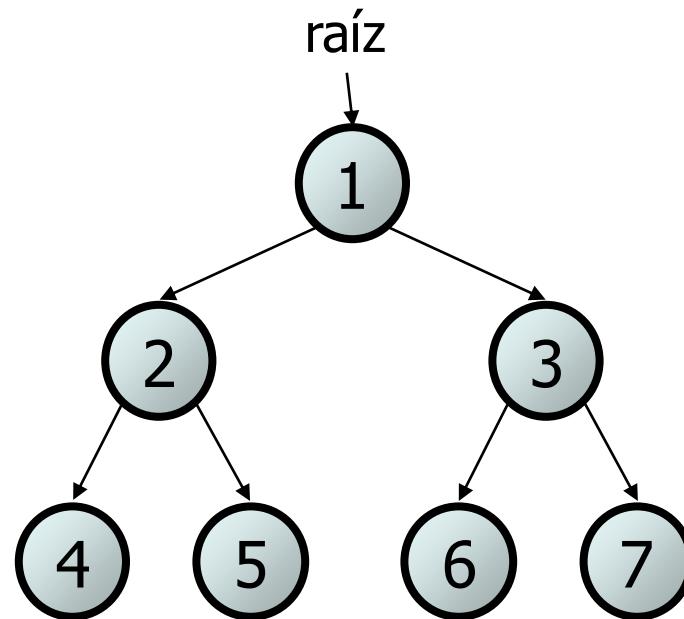
# Árboles

- **árbol** : Una estructura acíclica dirigida de nodos enlazados.
  - *Dirigido* : Tiene enlaces unidireccionales entre nodos.
  - *acíclico* : ningún camino vuelve al mismo nodo dos veces.
  - **Árbol binario** : Aquel en el que cada nodo tiene como máximo dos hijos.
- Un árbol se puede definir como:
  - vacío ( nulo ), o
  - un nodo **raíz que contiene**:
    - **datos** ,
    - un subárbol **izquierdo** , y
    - un subárbol **derecho** .
      - (El subárbol izquierdo y/o derecho podría estar vacío).



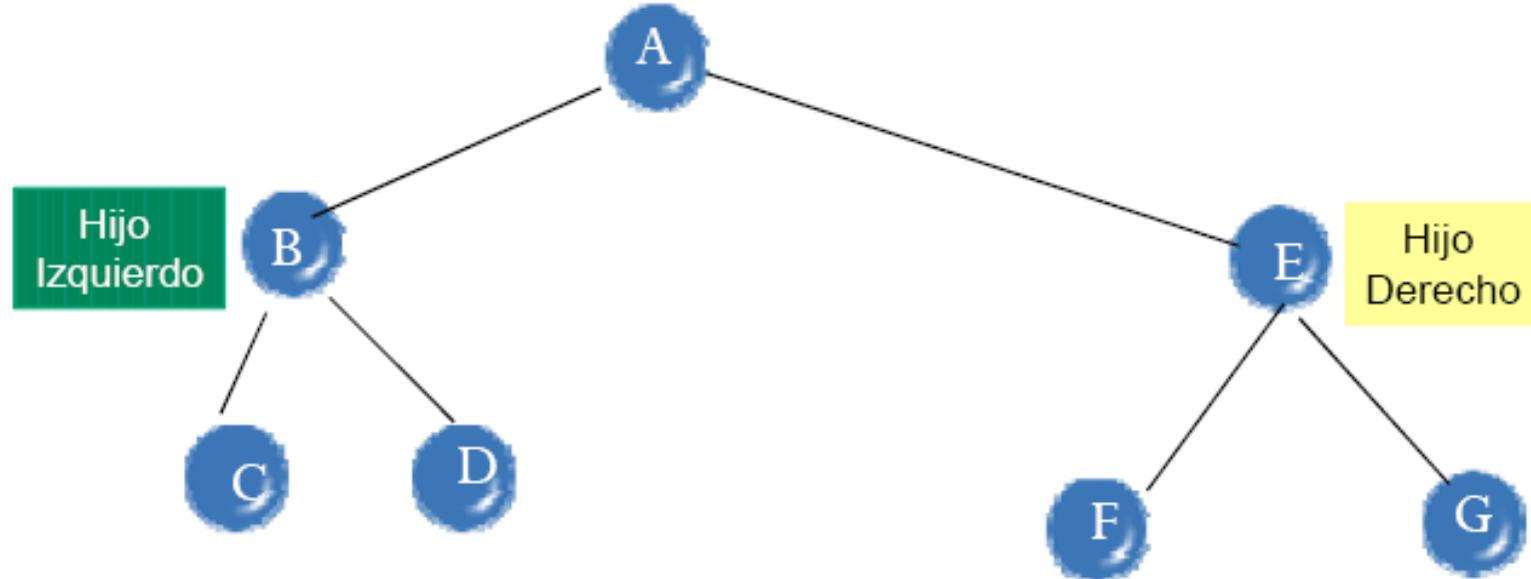
# Terminología

- **nodo** : un objeto que contiene un valor de dato e hijos izquierdo/derecho
  - **raíz** : nodo superior de un árbol
  - **hoja** : un nodo que no tiene hijos
  - **rama** : cualquier nodo interno; ni la raíz ni una hoja
- 
- **padre** : un nodo que se refiere a este
  - **hijo** : un nodo al que se refiere este nodo
  - **hermano** : un nodo con un común



## Árboles Binarios

- Cuando cada nodo puede tener un máximo de dos hijos.
- Los hijos de un nodo cualquiera se denotarán :  
hijo **derecho** e hijo **izquierdo**, de acuerdo a su posición en el gráfico del árbol.



# Árboles Binarios

Definición (continuación)

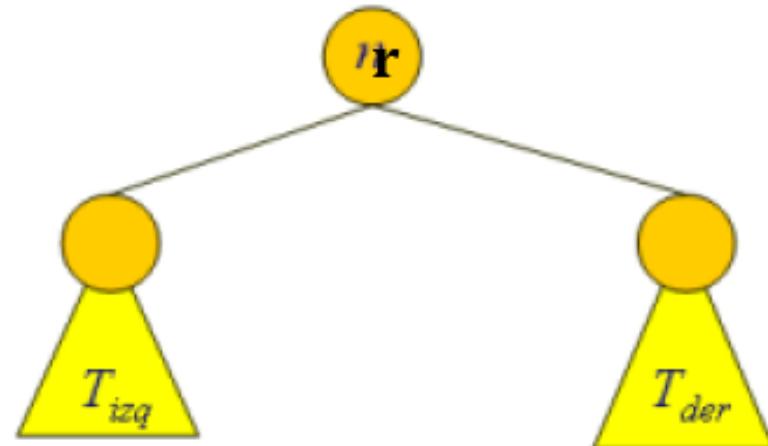
El conjunto consiste de una raíz  $r$  y exactamente dos árboles binarios distintos

$T_{izq}$  ,  $T_{Der}$

$T=\{r, T_{izq} , T_{Der} \}$

El árbol  $T_{izq}$  es llamado subárbol izquierdo de  $T$  y

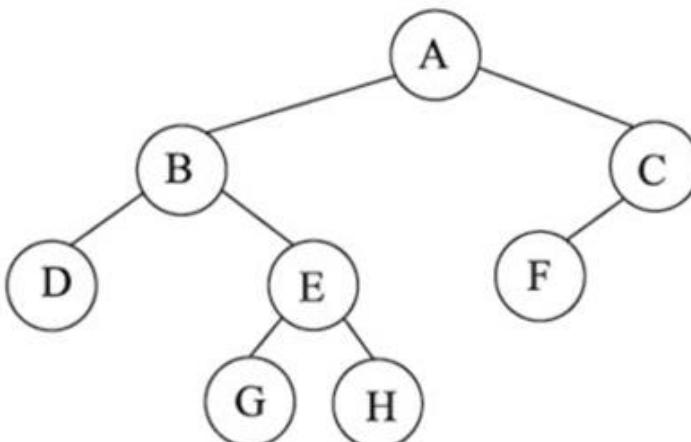
El árbol  $T_{Der}$  es denominado subárbol derecho de  $T$ .



Son Árboles cuyos Nodos no pueden tener más de 2 Subárboles.

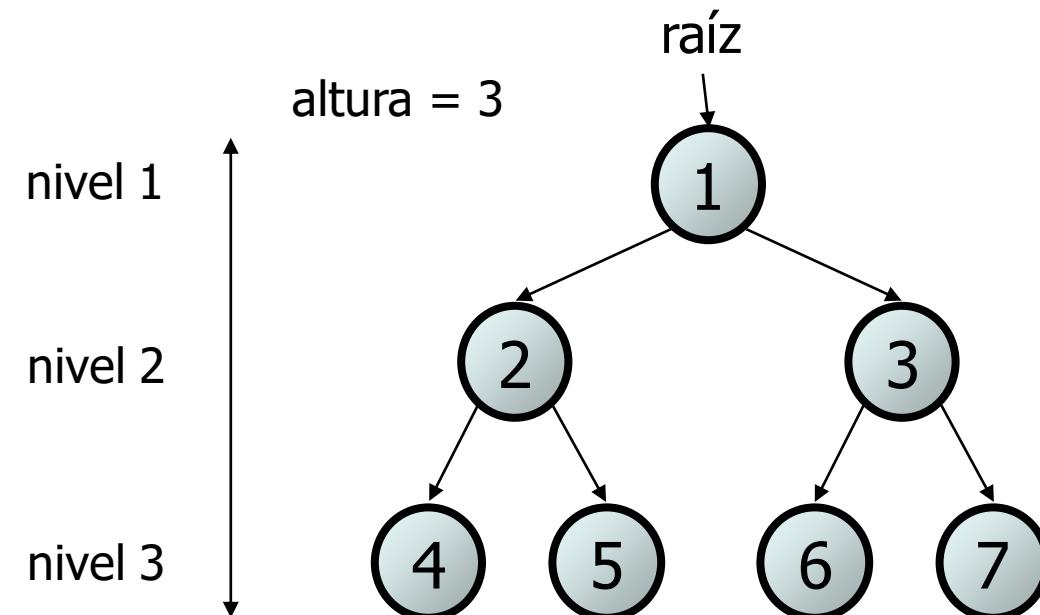
En un árbol binario, cada nodo puede tener cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.

Un Árbol binario es una estructura recursiva. Cada nodo es la raíz de su propio subárbol y tiene hijos, que son raíces de árboles, llamados subárboles derecho e izquierdo del nodo, respectivamente.

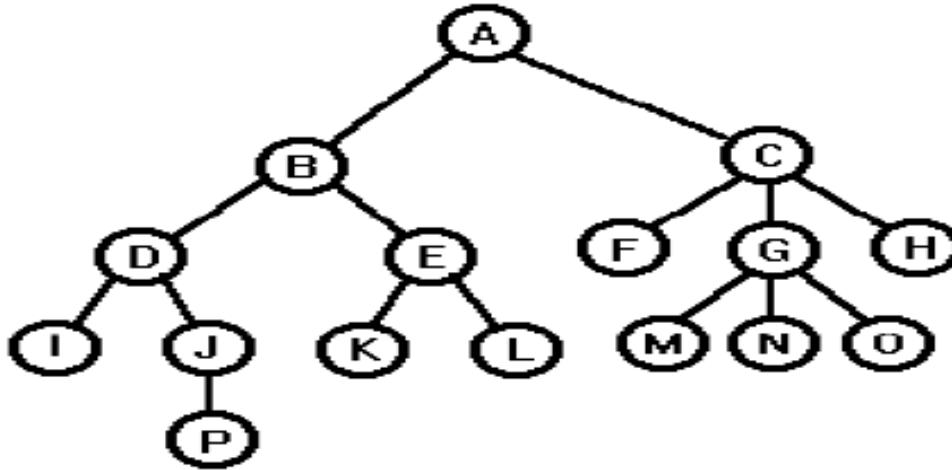


# Terminología 2

- **subárbol** : el árbol de nodos accesibles a la izquierda/derecha desde el nodo actual
- **altura** : longitud del camino más largo desde la raíz hasta cualquier nodo
- **nivel o profundidad** : longitud del camino desde una raíz hasta un nodo dado
- **árbol completo** : uno donde cada rama tiene 2 hijos

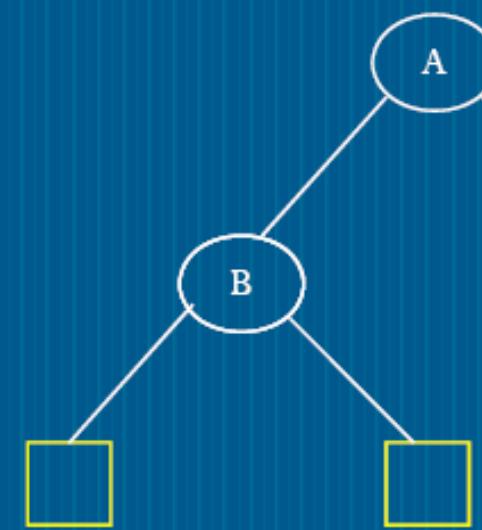


## Ejemplo



- El camino del nodo A al nodo P es  $(A, B, D, J, P)$ , cuya longitud de camino es 4
- Las hojas son: I, P, K, L, F, M, N, O, P y H
- El Único nodo de grado 1 es J.
- Los nodos de grado 2 son A, B, D y E.
- Los nodos de grado 3 son C y G.
- No existen nodos de mayor grado, por tanto, el grado del árbol es 3.

## Ejemplos de árboles binarios

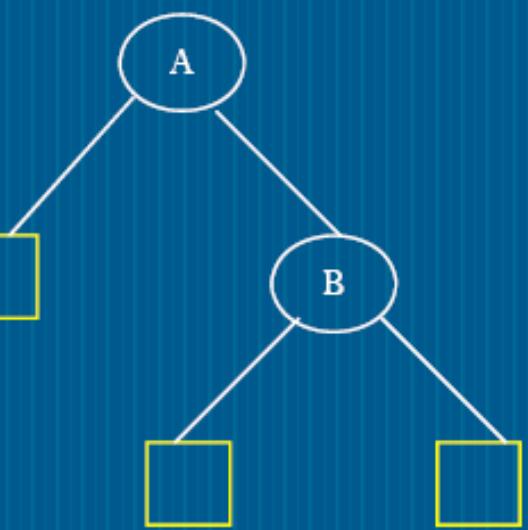


$$T_A = \{A, T_{izq}, T_{der}\}$$

$$T_{izq} : \{B, \emptyset, \emptyset\}$$

$$T_{der} : \emptyset$$

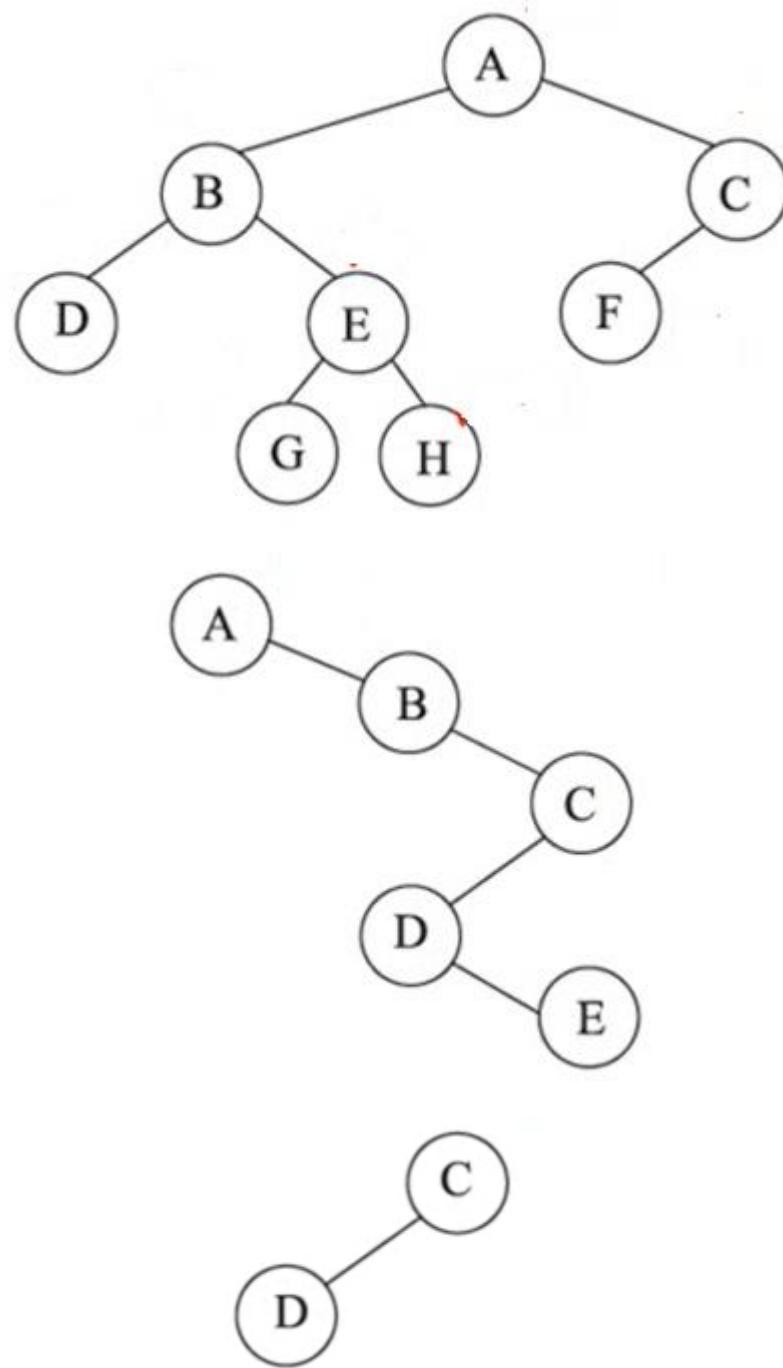
$$T_A \neq T_B$$



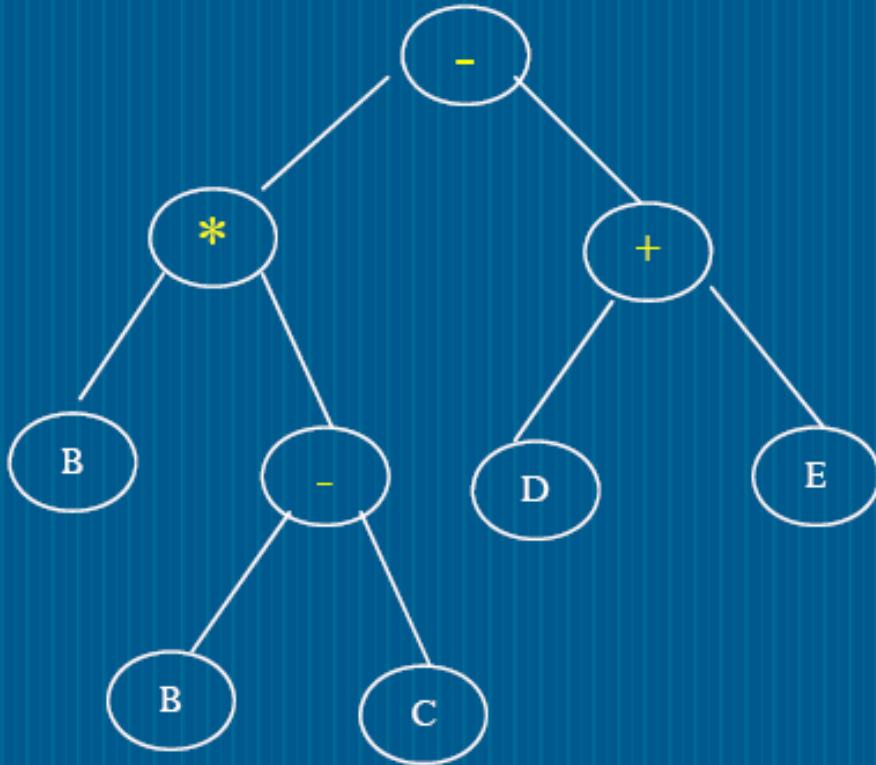
$$T_B = \{A, T_{izq}, T_{der}\}$$

$$T_{izq} : \emptyset$$

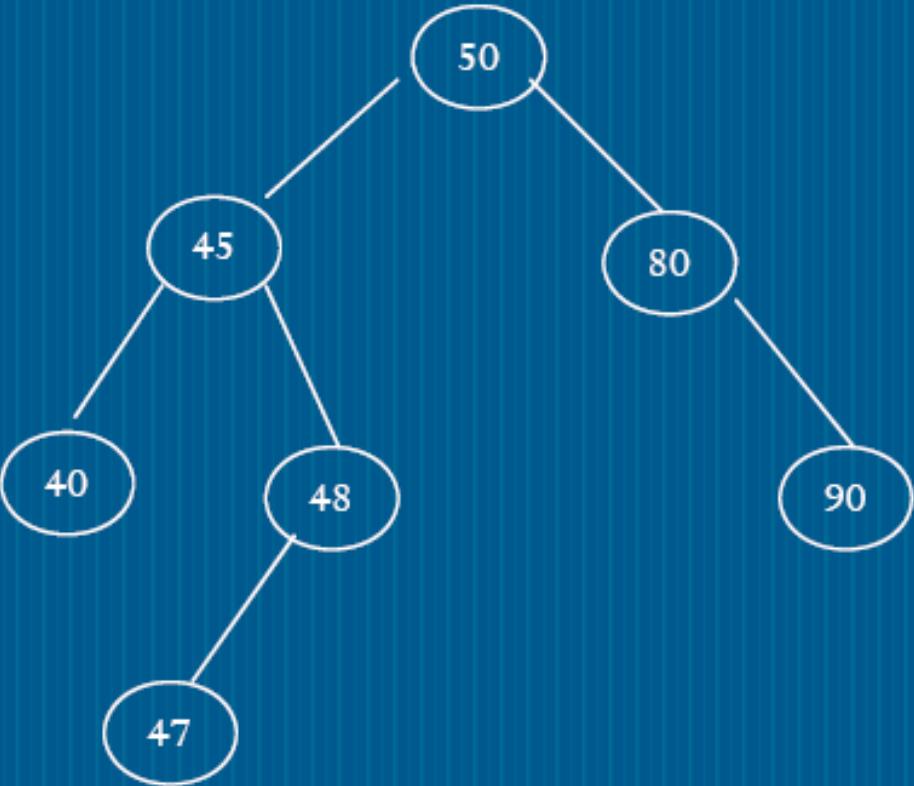
$$T_{der} : \{B, \emptyset, \emptyset\}$$



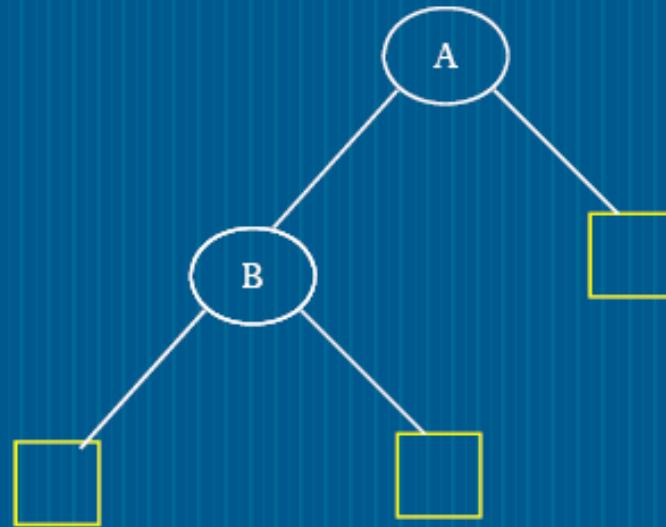
## Ejemplos de árboles binarios



## Ejemplos de árboles binarios

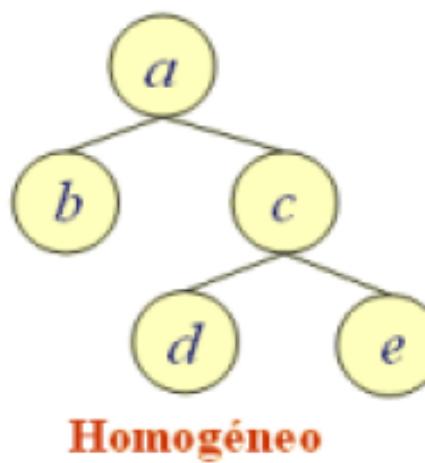
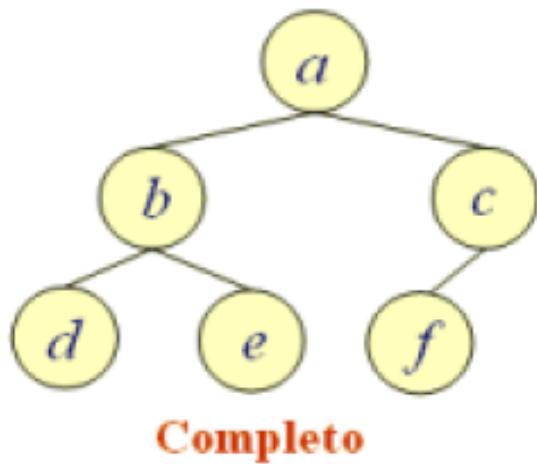


- Grado ( El grado de todos los nodos de un árbol binario es **dos**)
- Hoja. (Nodo en que ambos hijos son el árbol vacío)
- Hijo
- Padre
- Hermano



# AB HOMOGENEO Y COMPLETO

- **Árbol binario homogéneo**. Es aquel cuyos nodos tienen grado 0 ó 2 (no hay ninguno de grado 1).
- **Árbol binario completo**. Es aquel que tiene todos los niveles llenos excepto, quizás, el último en cuyo caso los huecos deben quedar a la derecha.



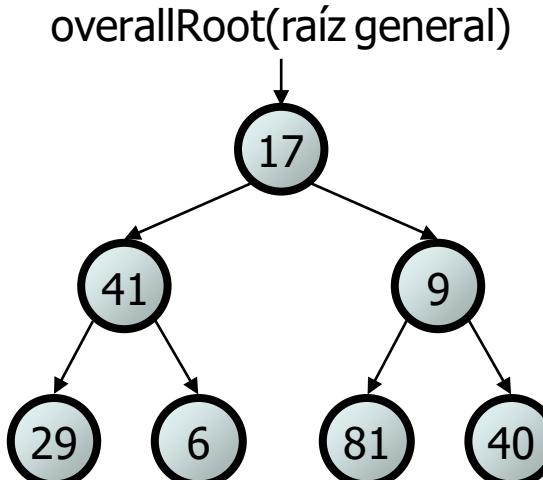
# RECORRIDOS DE ARBOLES BINARIOS

Un árbol binario se puede recorrer de tres maneras diferentes:

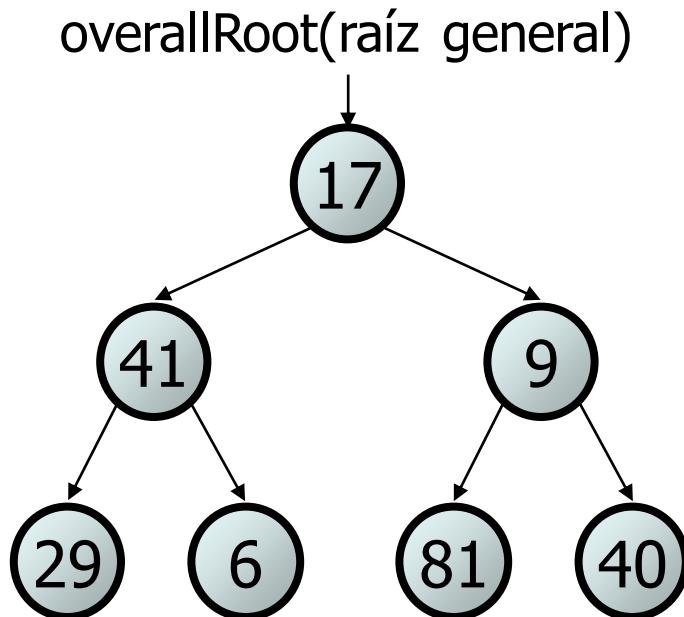
1. Recorrido en PreOrden
2. Recorrido en EnOrden
3. Recorrido en PostOrden

# Recorridos (traversals)

- **Recorrido** : Un examen de los elementos de un árbol.
  - Un patrón utilizado en muchos métodos y algoritmos de árbol.
- Órdenes comunes para recorridos:
  - **pre-order**: procesar el nodo raíz, luego sus subárboles izquierdo/derecho
  - **in-order**: procesar el subárbol izquierdo, luego el nodo raíz, luego el derecho
  - **post-order**: procesa los subárboles izquierdo/derecho, luego el nodo raíz



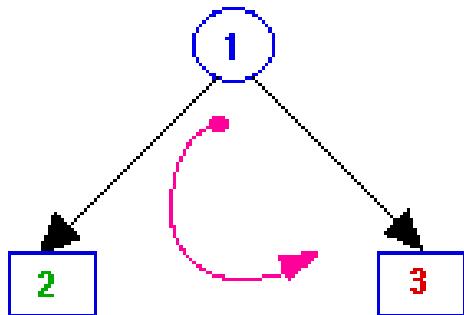
# Ejemplo recorrido



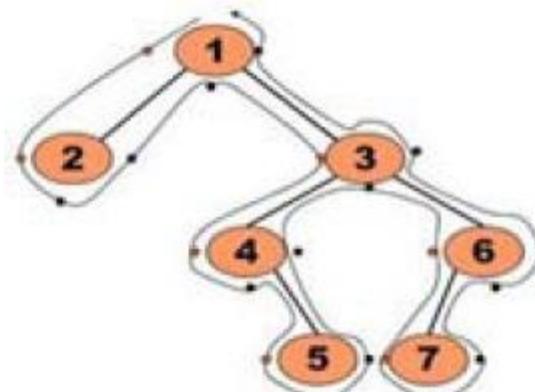
- **pre-order:** 17 41 29 6 9 81 40
- **in-order:** 29 41 6 17 81 9 40
- **post-order:** 29 6 41 81 40 9 17

## En PREORDEN de un árbol binario diferente del vacío ( $R$ , $T_{IZQ}$ , $T_{DER}$ )

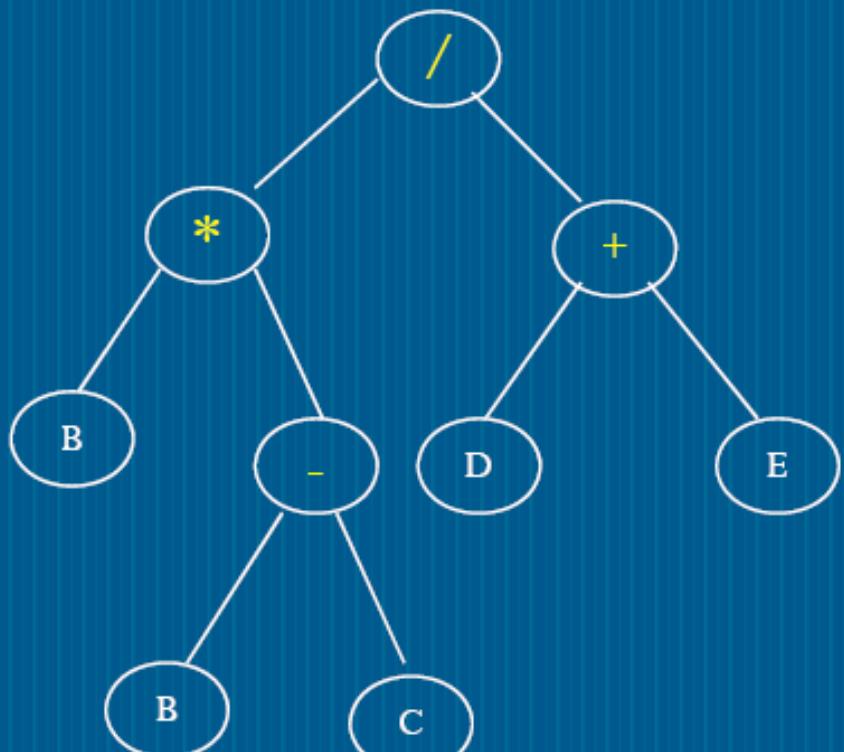
1. Se visita la raíz del árbol
2. Se recorre en preorden el subárbol izquierdo de la raíz.
3. Se recorre en preorden el subárbol Derecho de la raíz.



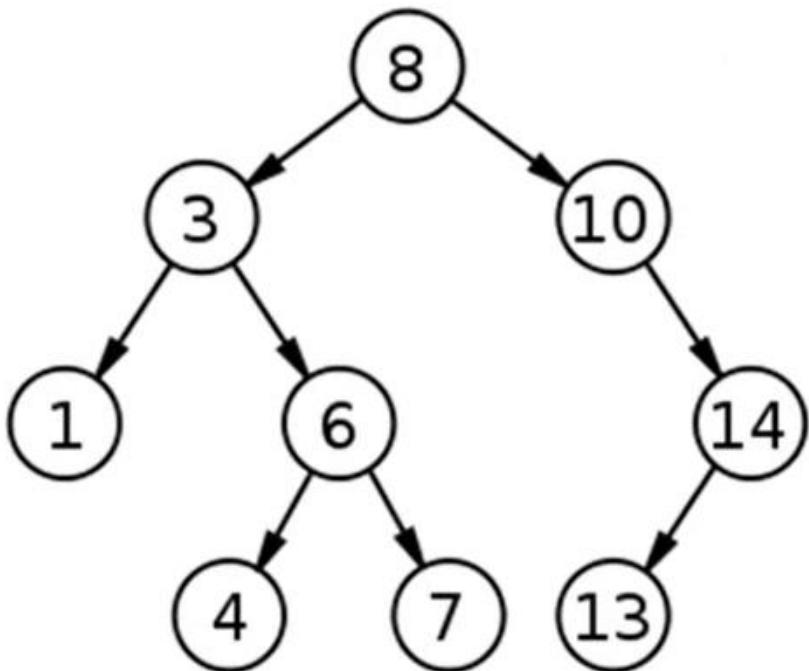
a) Preorden



## Recorrido en preorden

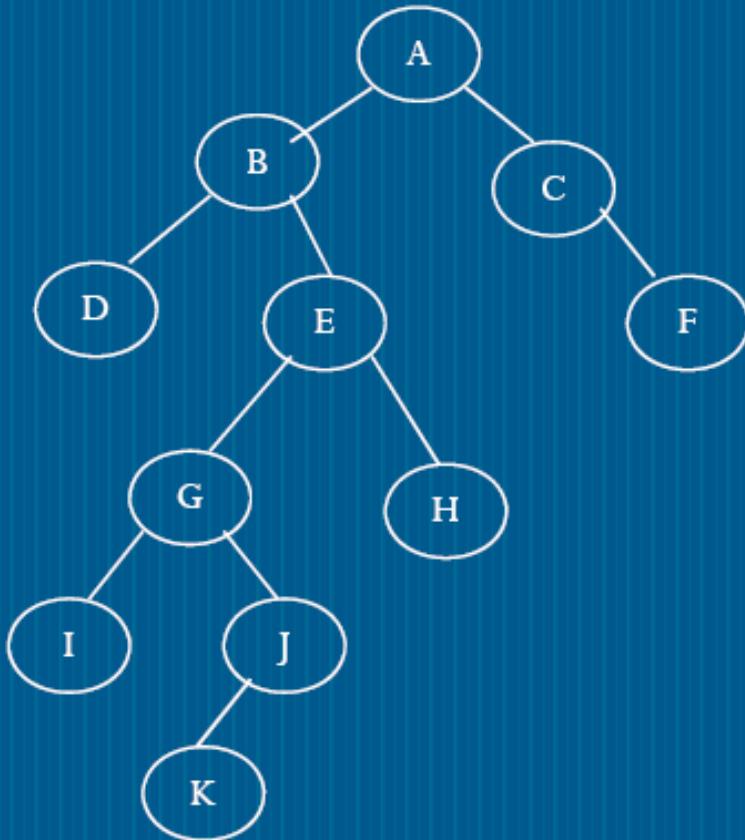


/, \*, B, -, B, C, +, D, E



8, 3, 1, 6, 4, 7, 10, 14, 13

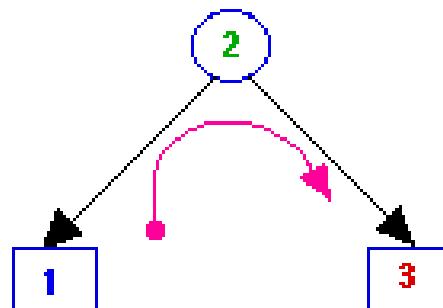
## Recorrido en preorden



A, B, D, E, G, I, J, K, H, C, F

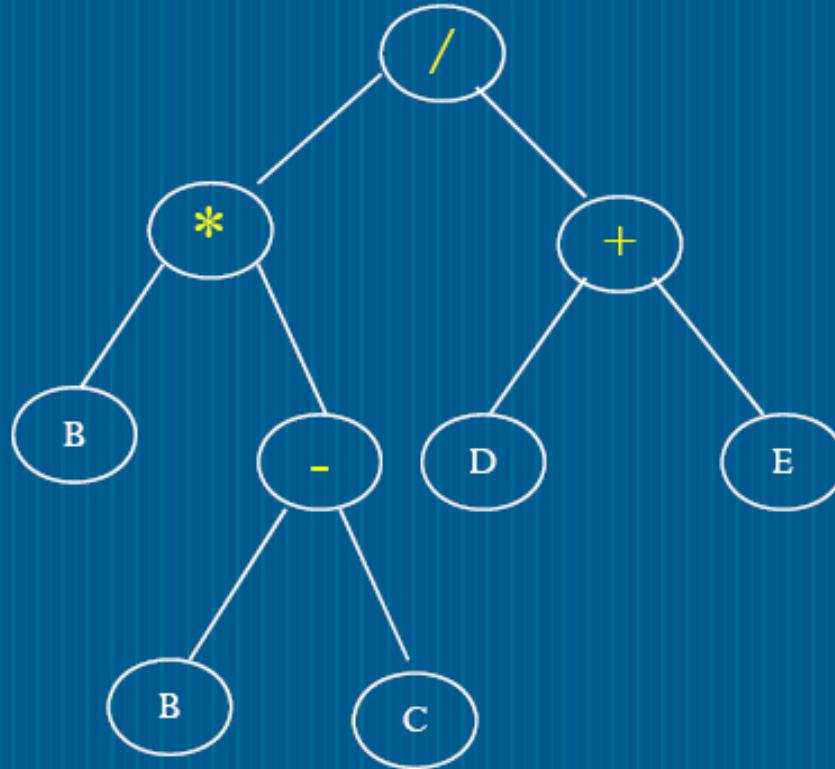
## En ENTREORDEN de un árbol binario diferente del vacío ( $T_{IZQ}$ , R, $T_{DER}$ )

1. Se recorre en entreorden el subárbol izquierdo de la raíz
2. Se visita la raíz del árbol
3. Se recorre en entreorden el subárbol Derecho de la raíz.



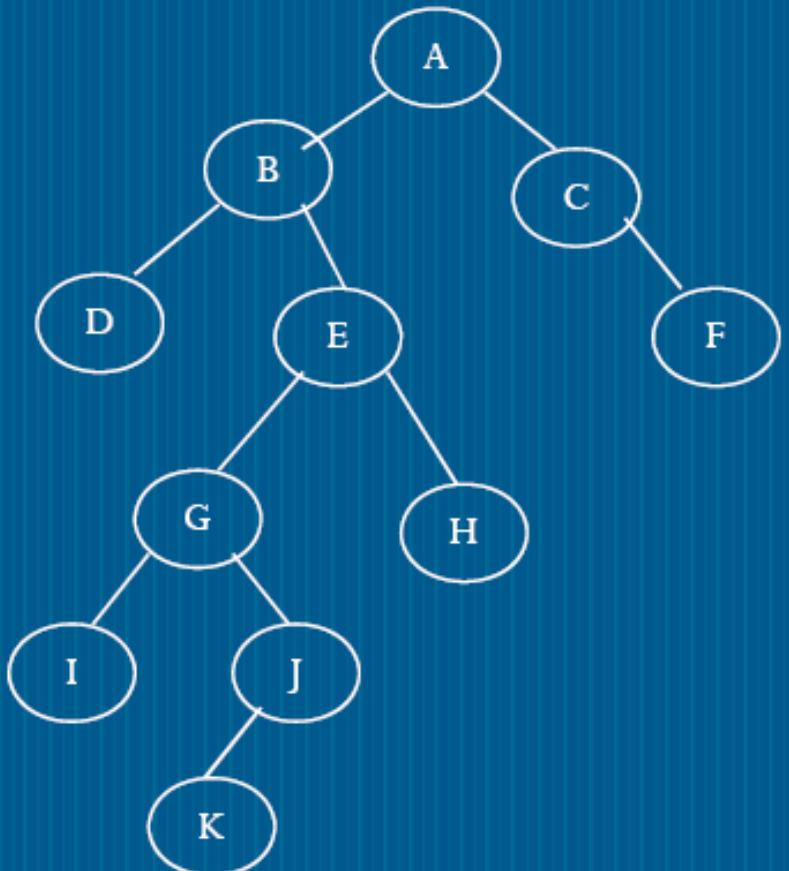
b) Inorden

## Recorrido en entreorden

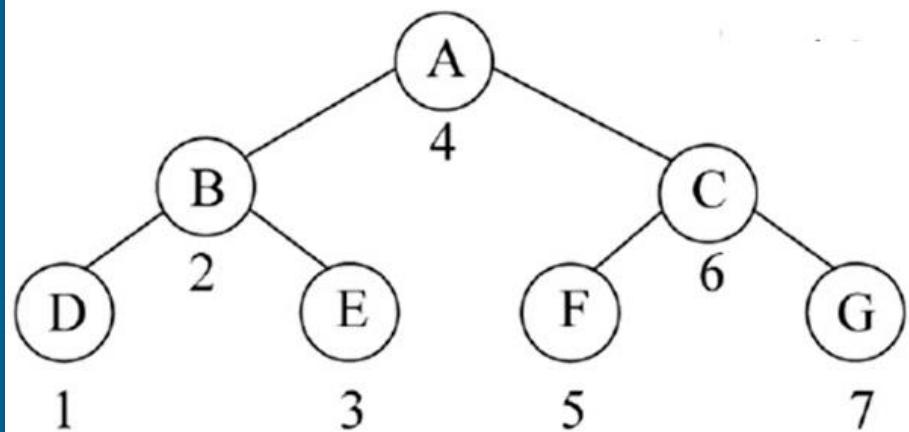


B, \*, B, -, C, /, D, +, E

## Recorrido en entreorden



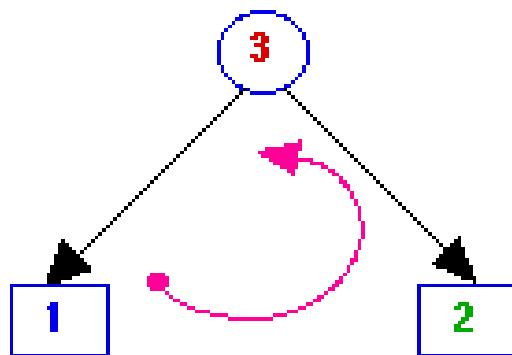
D, B, I, G, K, J, E, H, A, C, F



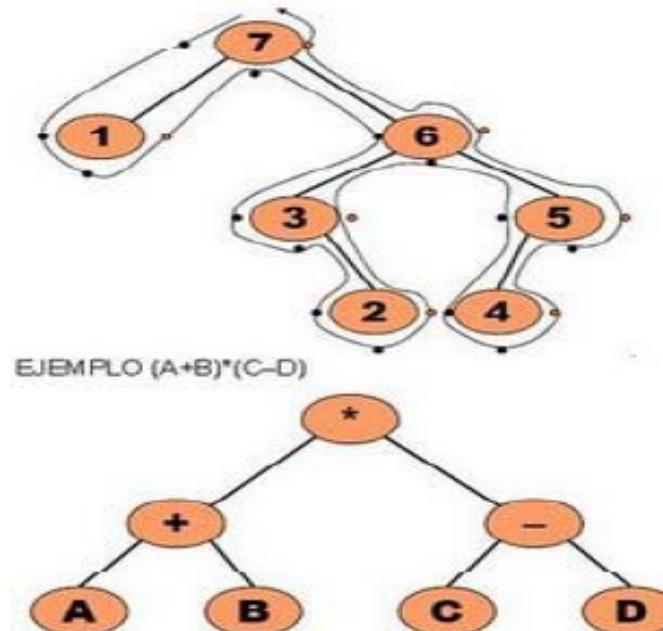
recorrido: D, B, E, A, F, C, G

## En POSTORDEN de un árbol binario diferente del vacío ( $T_{IZQ}$ , $T_{DER}$ , R)

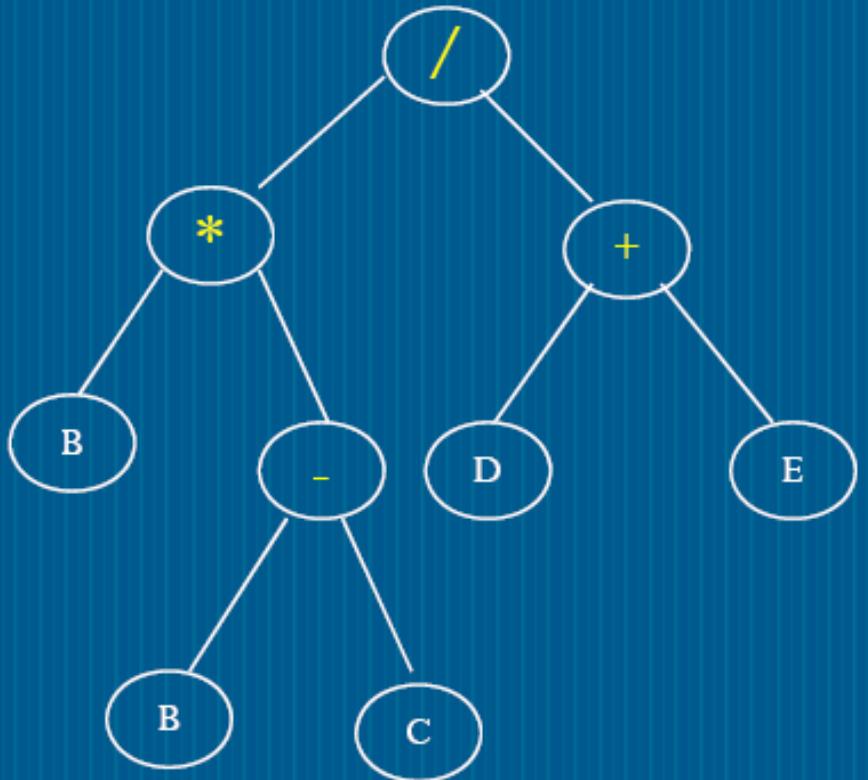
1. Se recorre en postorden el subárbol izquierdo de la raíz.
2. Se recorre en postorden el subárbol derecho de la raíz.
3. Se visita la raíz del árbol.



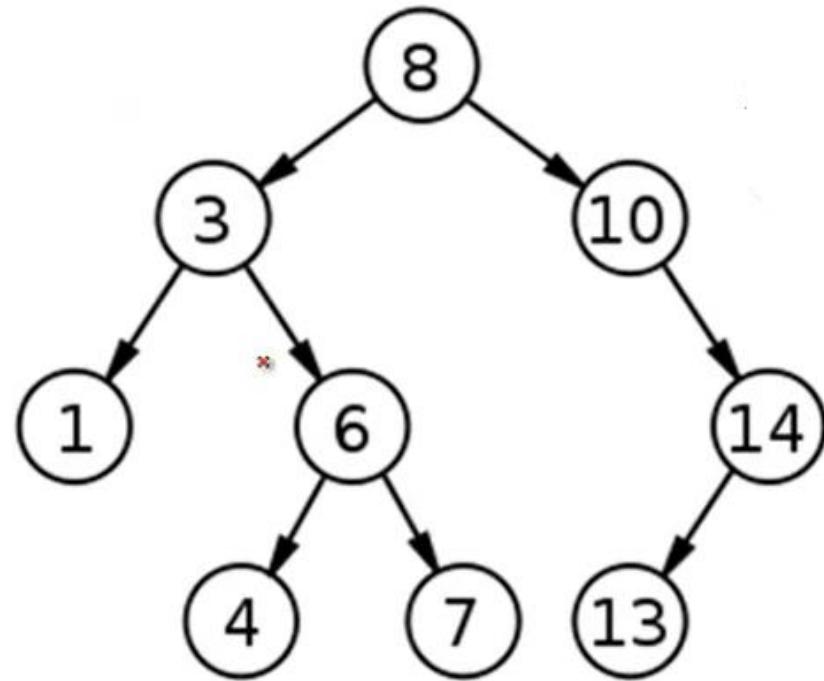
c) Postorden



## Recorrido en postorden

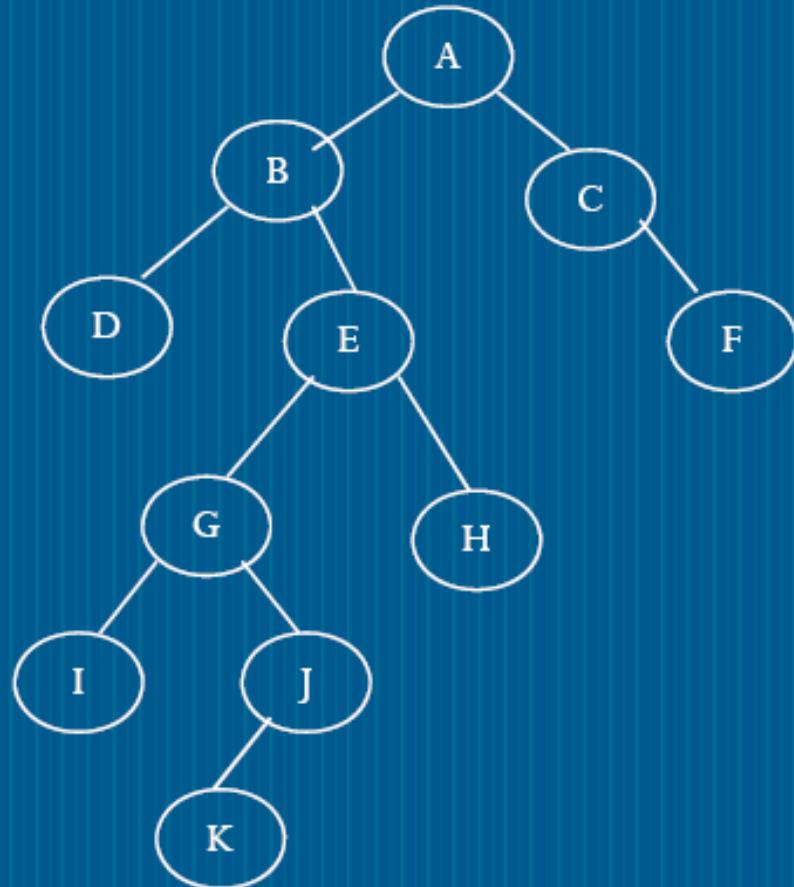


B, B, C, -, \*, D, E, +, /



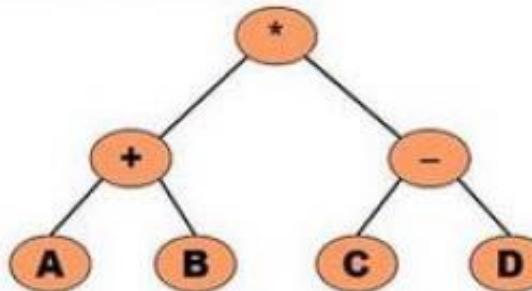
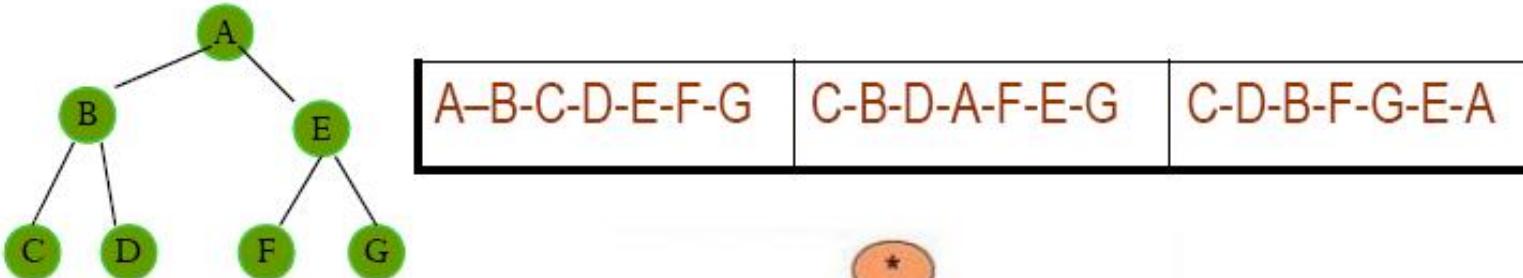
1, 4, 7, 6, 3, 13, 14, 10, 8

## Recorrido en postorden



D, I, K, J, G, H, E, B, F, C, A

## Recorrido de Árboles Binarios



**Recorridos Preorden:** Se recorrido es Raiz - Izquierda - Derecha

[\*, + ,A , B , - , C , D]

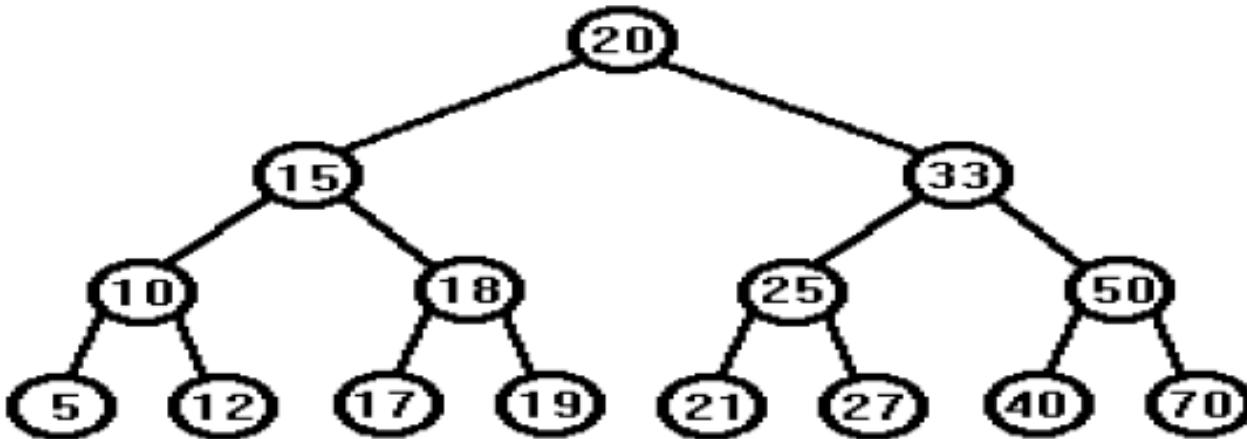
**Recorridos Inorden:** Se recorrido es Izquierda - Raiz - Derecha

[A , + , B, \* , C , - , D ]

**Recorridos Postorden:** Su recorrido es Izquierda - Derecha - Raiz

[A , B , + , C , D , - , \* , ]

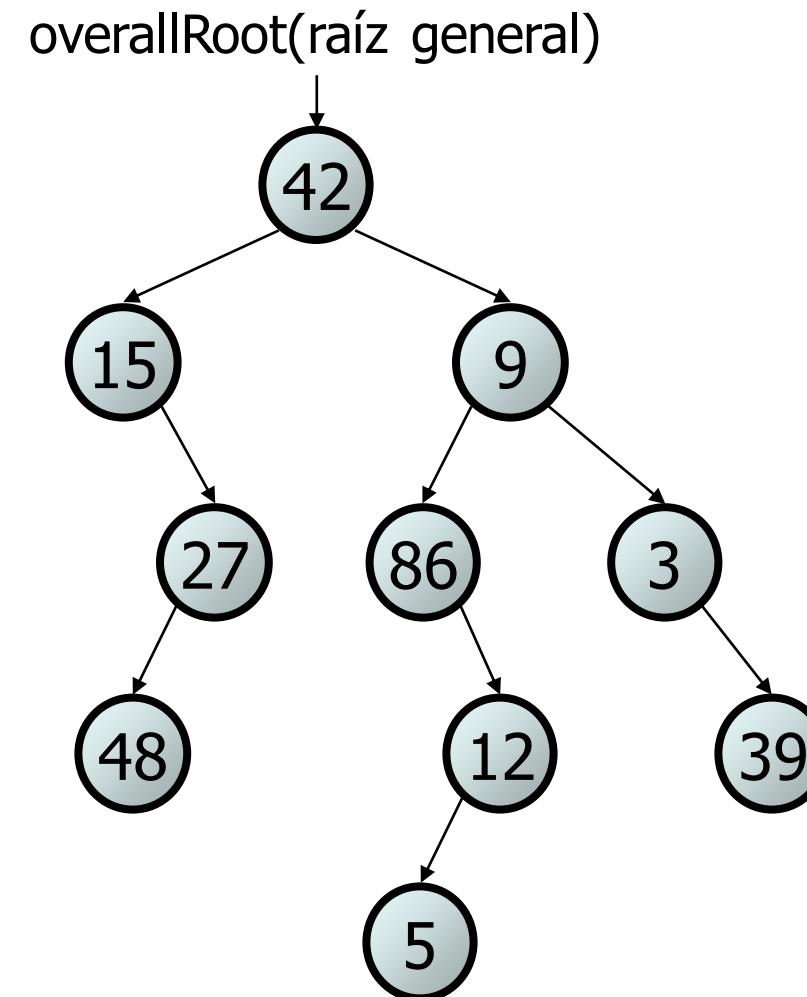
# Recorridos, Ejemplo



- **preorden:**
  - 20, 15 10, 5, 12, 18, 17, 19, 33, 25, 21, 27, 50, 40, 70
- **inorden:**
  - 5, 10, 12, 15, 17, 18, 19, 20, 21, 25, 27, 33, 40, 50, 70
- **postorden:**
  - 5, 12, 10, 17, 19, 18, 15, 21, 27, 25, 40, 70, 50, 33, 20

# Ejercicio

- Proporcione recorridos pre-, in- y post-order para el siguiente árbol:



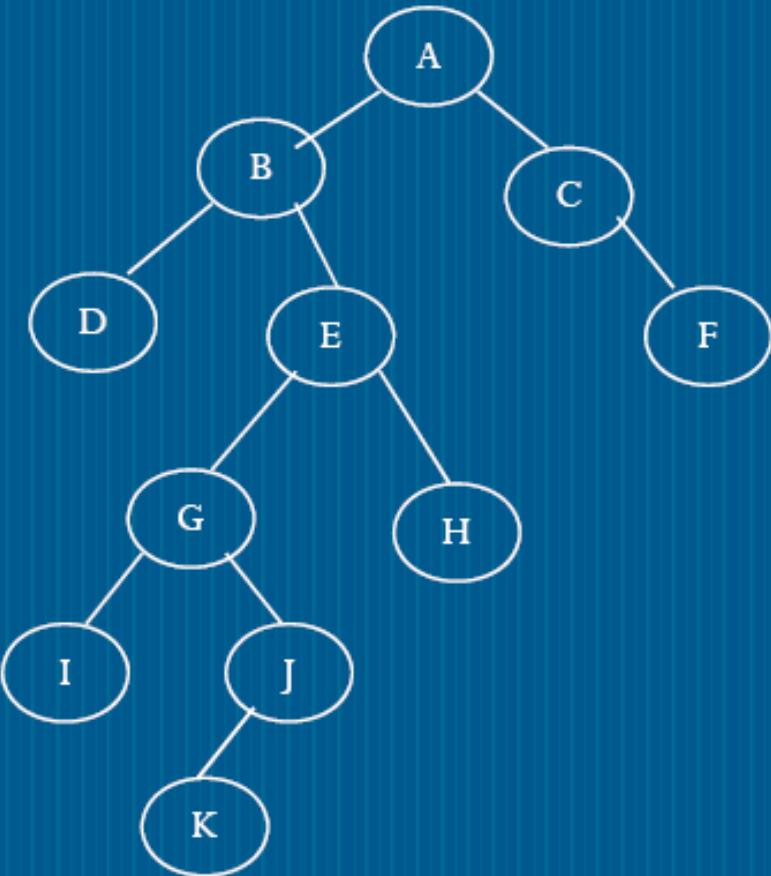
- pre: 42 15 27 48 9 86 12 5 3 39
- in: 15 48 27 42 86 5 12 9 3 39
- post: 48 27 15 5 12 86 39 3 42

## Recorrido a lo ancho

El procedimiento es:

- Se visita el nodo del nivel 0 (esto es la raíz),
- A continuación todos los nodos del nivel 1,
- Después todos los nodos del nivel 2 y así sucesivamente.

## Recorrido a lo ancho



A, B, C, D, E, F, G, H, I, J, K

## Árboles Binarios

1. Es una estructura recursiva.
2. Cada nodo es la raíz de su propio subárbol
3. Tiene hijos que son raíces de árboles llamados subárbol izquierdo y subárbol derecho.
  - Cómo trabajaremos esta estructura ?
  - Cómo serán nuestros nodos ?

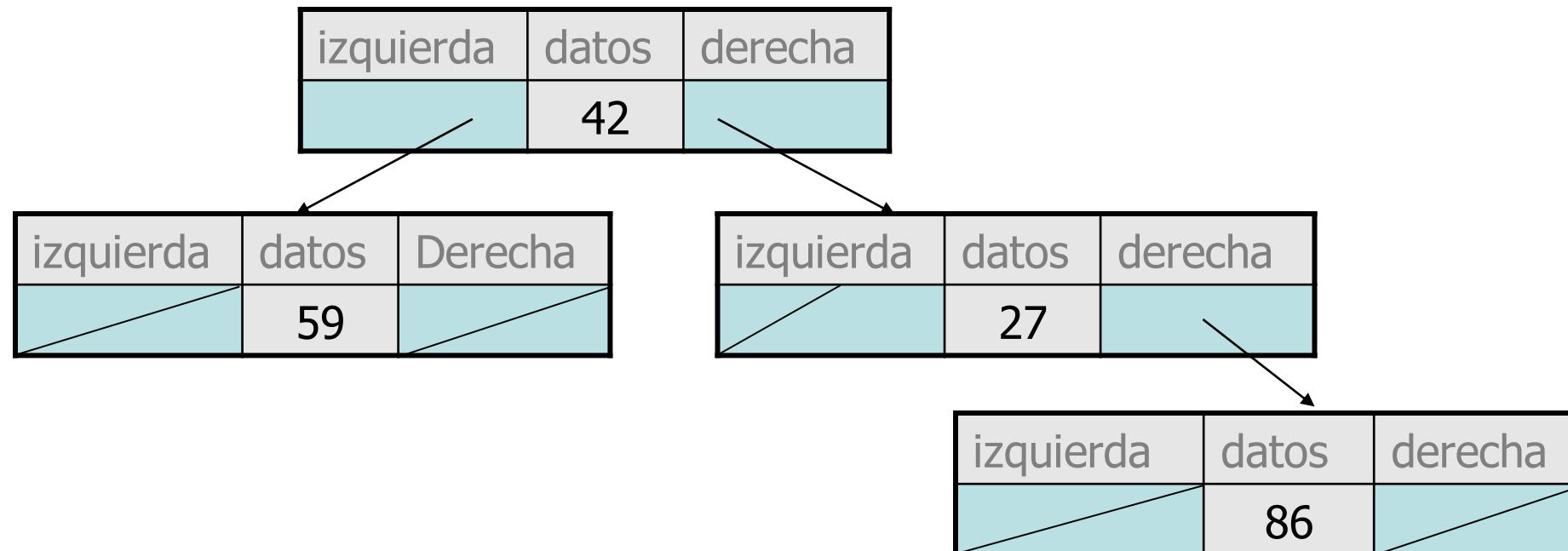
```
class CnodoArb {  
    private Cobjeto objeto;  
    private CnodoArb hijolqzq;  
    private CnodoArb hijoDer;  
  
    public CnodoArb( ) {  
        objeto = new Cobjeto( );  
        hijoDer = null;  
        hijolqzq = null; }  
.....  
    public void ImprimeNodo( ){  
        objeto.ImprimeObjeto( ); } }
```

# Un nodo de árbol para enteros

- **Un objeto nodo de árbol** básico almacena datos y se refiere a izquierda/derecha



- Se pueden vincular múltiples nodos en un árbol más grande



# clase IntTreeNode

```
// Un objeto IntTreeNode es un nodo en un árbol binario de enteros.

public class IntTreeNode {
    public int data;                      // datos almacenados en este nodo
    public IntTreeNode left;               // referencia al subárbol izquierdo
    public IntTreeNode right;              // referencia al subárbol derecho

    // Construye un nodo hoja con los datos dados.
    public IntTreeNode(int data) {
        this(data, null, null);
    }

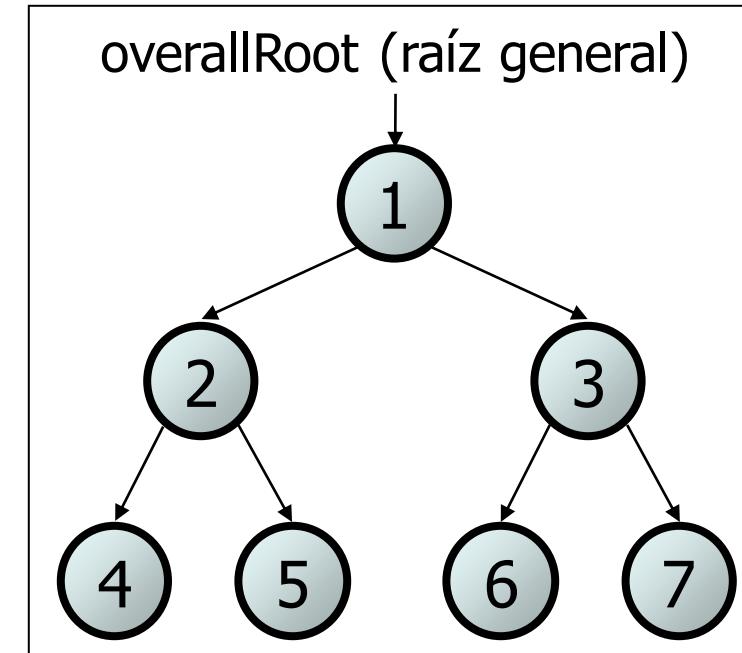
    // Construye un nodo de rama con los datos y enlaces
    // proporcionados.
    public IntTreeNode(int data, IntTreeNode left,
                       IntTreeNode right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}
```

# clase IntTree

```
// Un objeto IntTree representa un árbol binario completo de
// enteros.
public class IntTree {
    private IntTreeNode overallRoot; // nulo para un árbol vacío

    métodos
}
```

- El código del cliente habla con el IntTree, no con los objetos nodo dentro de él.
- Los métodos de IntTree crean y manipulan los nodos, sus datos y enlaces entre ellos.



# constructor IntTree

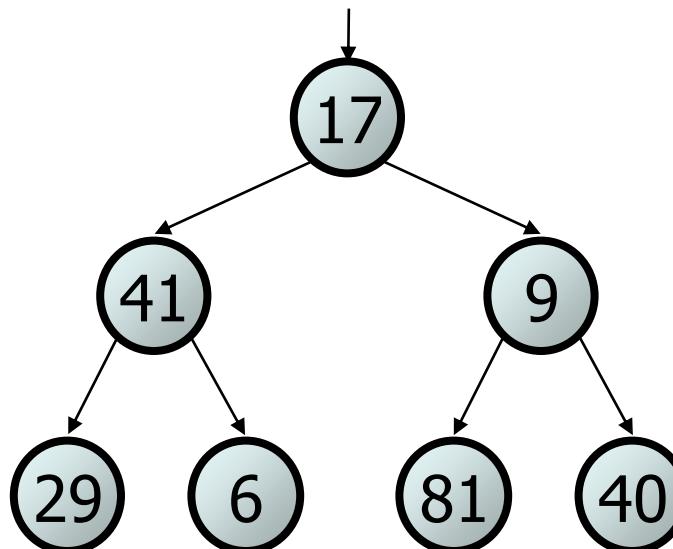
- Supongamos que tenemos los siguientes constructores:

```
public IntTree ( IntTreeNode overallRoot )
public IntTree ( int height )
```

- El segundo constructor creará un árbol y lo llenará con nodos con valores de datos aleatorios del 1 al 100 hasta que esté lleno a la altura dada.

```
IntTree tree = new IntTree( 3 );
```

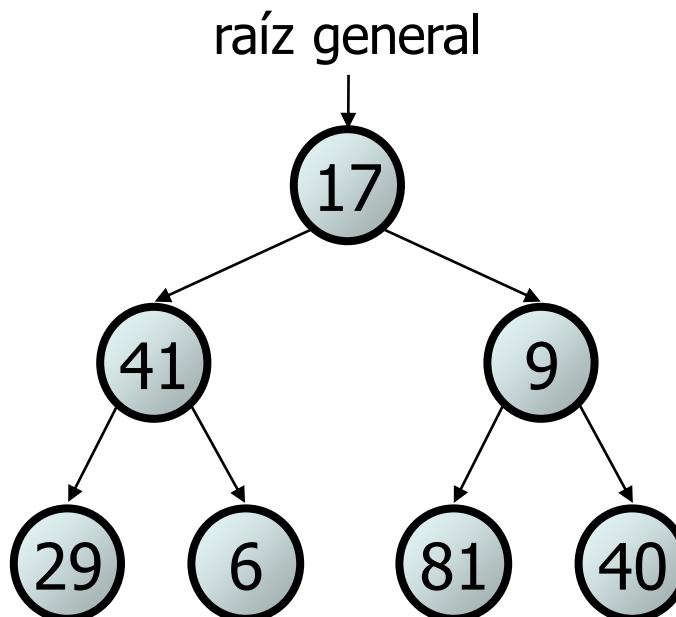
overallRoot



# Ejercicio

- Agregue un método `print` a la clase `IntTree` que imprima los elementos del árbol, separados por espacios.
  - El subárbol izquierdo de un nodo debe imprimirse antes que él, y su subárbol derecho debe imprimirse después.
  - Ejemplo: `árbol.imprimir();`

29 41 6 17 81 9 40



# Solución de ejercicio

```
// Un objeto IntTree representa un árbol binario completo de
// enteros.

public class IntTree {
    private IntTreeNode overallRoot; // nulo para un árbol vacío
    ...

    public void print() {
        print(overallRoot);
        System.out.println(); // fin de línea en la salida
    }

    private void print(IntTreeNode root) {
        // (el caso base es implicitamente no hacer nada en nulo)
        if (root != null) {
            // caso recursivo: imprimir izquierda, raíz, derecha
            print(overallRoot.left);
            System.out.print(overallRoot.data + " ");
            print(overallRoot.right);
        }
    }
}
```

# Plantilla para métodos de árbol

```
public class IntTree {  
    private IntTreeNode overallRoot;  
    ...  
  
    public type name(parameters) {  
        name(overallRoot, parameters);  
    }  
  
    private type name(IntTreeNode root, parameters) {  
        ...  
    }  
}
```

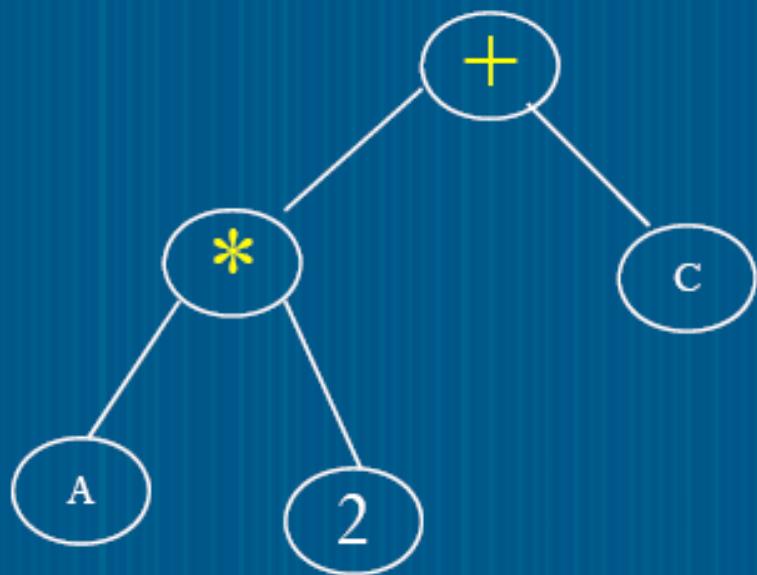
- Los métodos de árbol a menudo se implementan recursivamente
  - con un par público/privado
  - la versión privada acepta el nodo raíz para procesar

# Árbol de Expresiones Binarias

Un árbol de expresión binaria es un árbol binario en el cual:

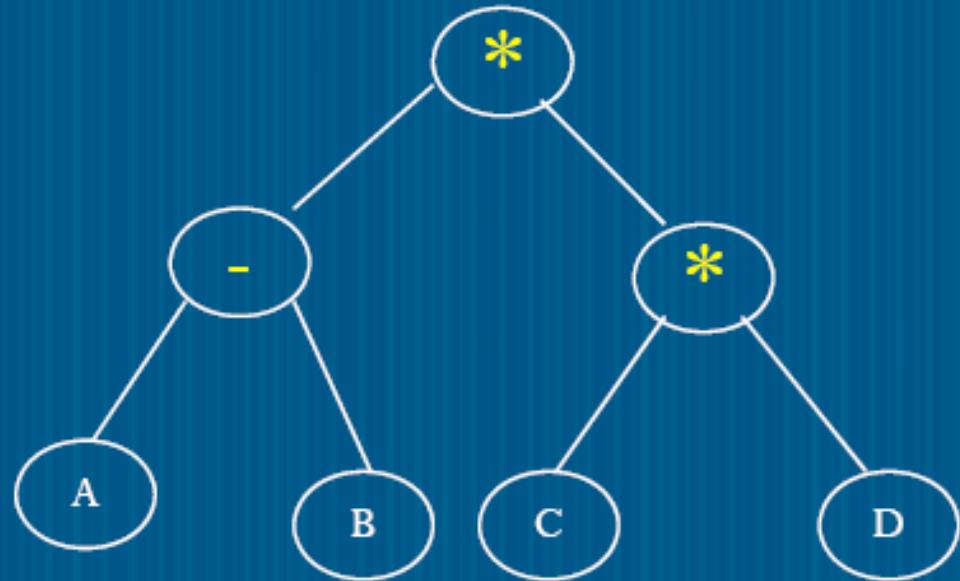
- Cada nodo hoja es un operando  
(un valor o un identificador)
- Cada nodo padre representa un operador binario

# Árbol de expresión binaria



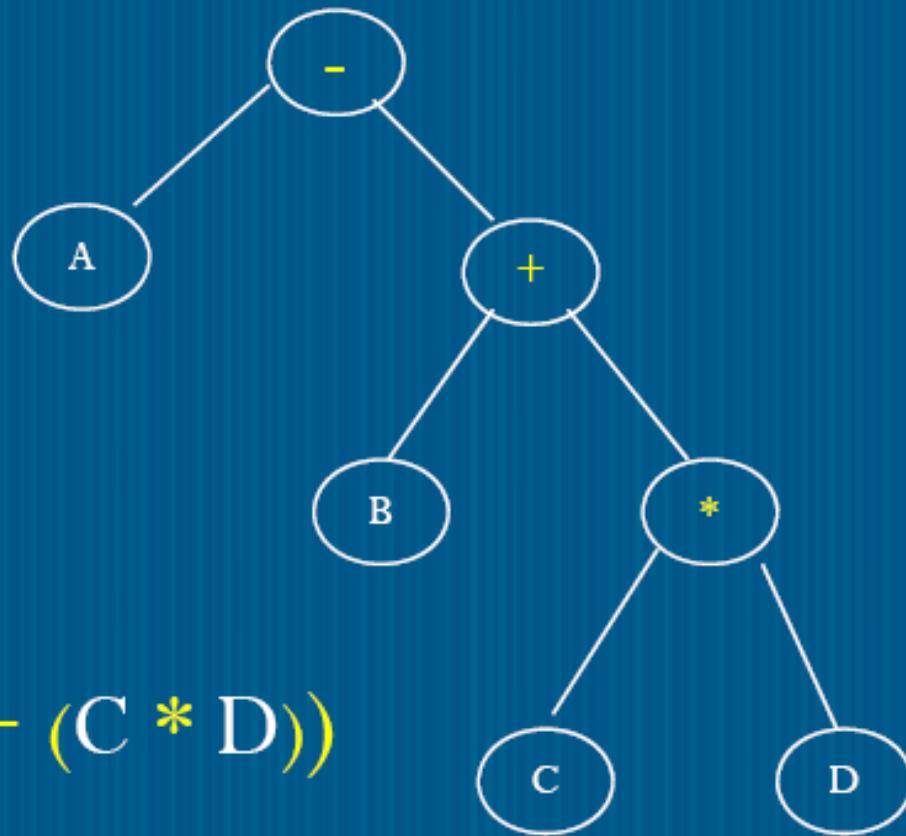
A \* 2 + C

# Árbol de expresión binaria



$(A - B) * (C * D)$

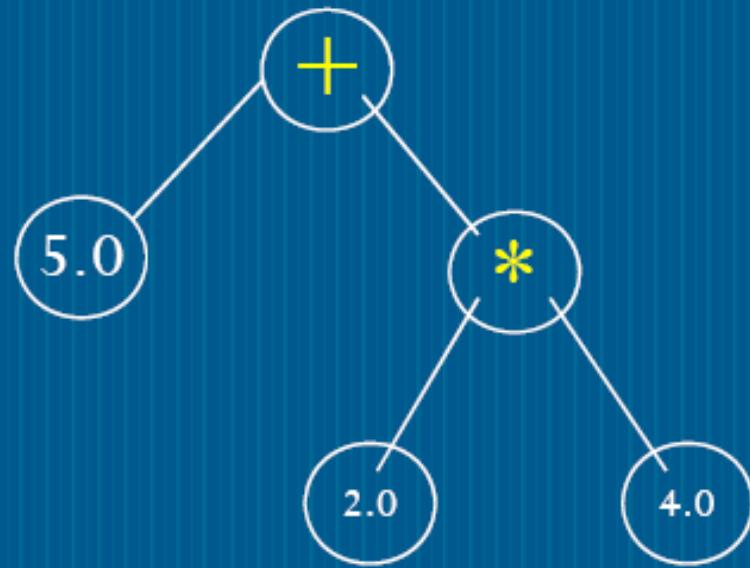
# Árbol de expresión binaria



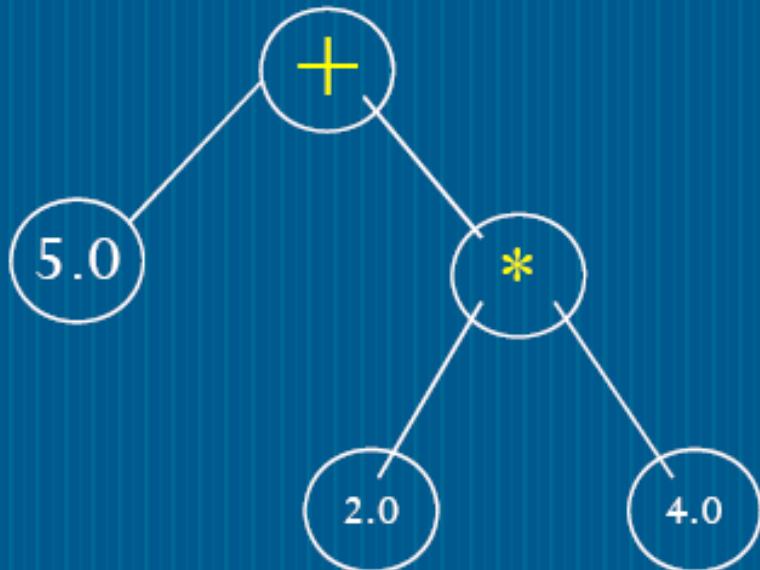
$$A - (B + (C * D))$$

# Ejemplo

ConstruirArbol (+, 5.0, \*, 2.0, 4.0)



## Ejemplo Evaluar()



T

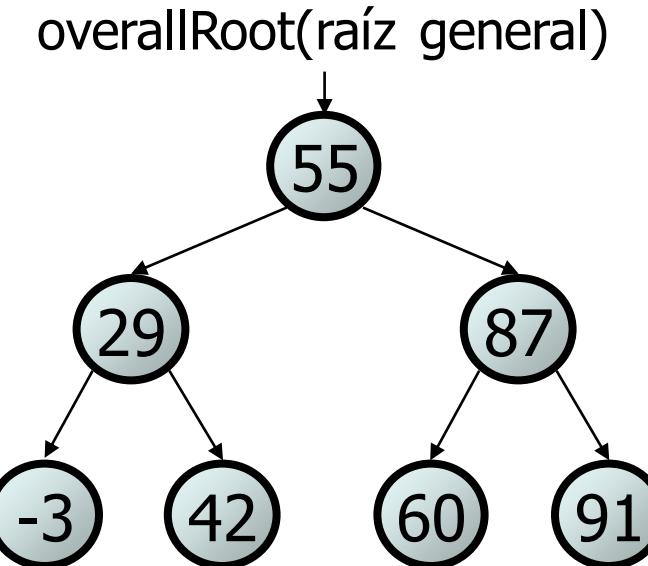
se retorna el valor 13.0

# Árboles Binarios de Búsqueda (ABB)

# Árboles Binarios de Búsqueda

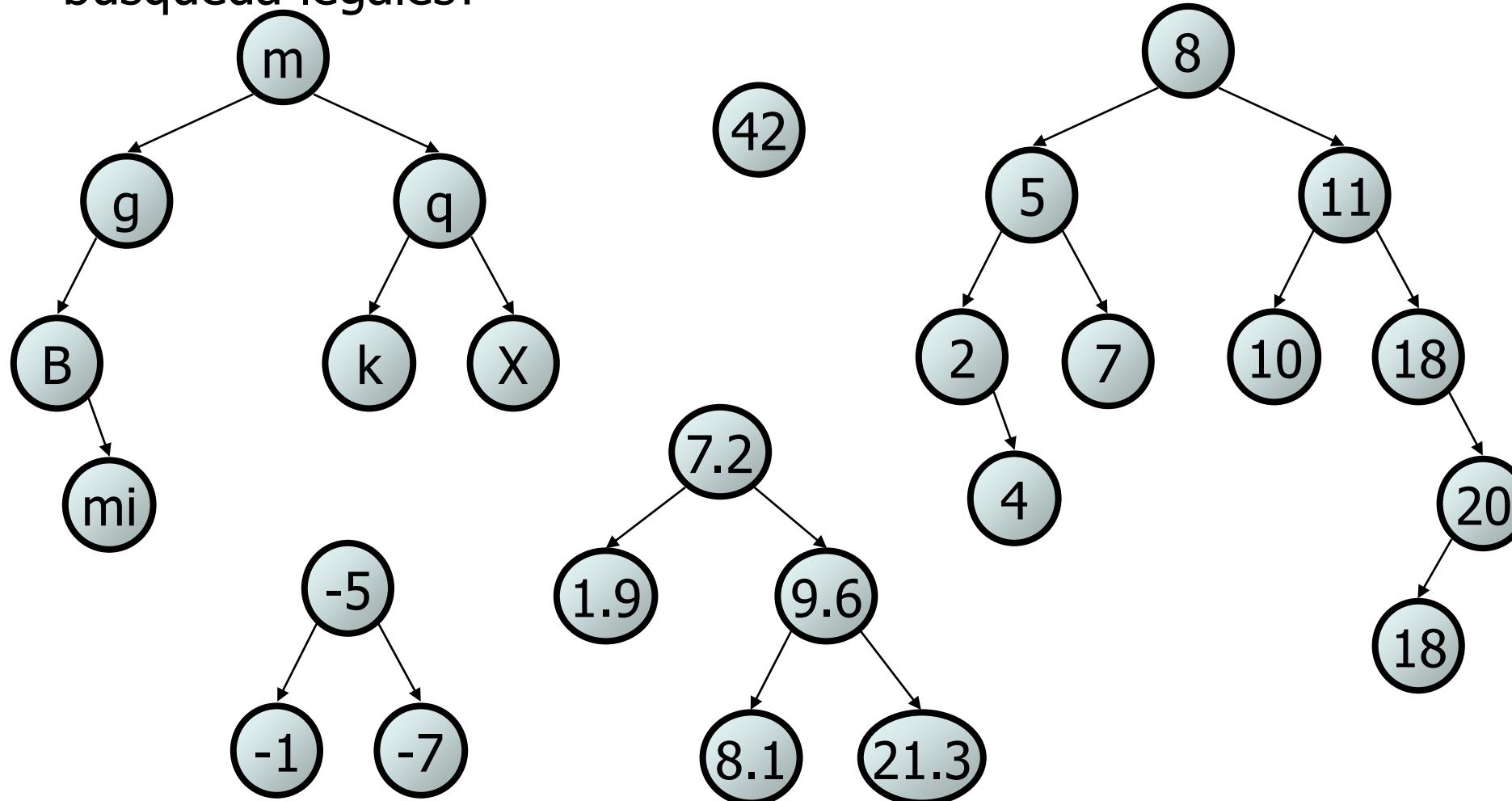
- **árbol binario de búsqueda ("BST"):** un árbol binario que es:
  - vacío (null), o
  - un nodo raíz R tal que:
    - cada elemento del subárbol izquierdo de R contiene datos "menores que" los datos de R,
    - cada elemento del subárbol derecho de R contiene datos "mayores que" R,
    - Los subárboles izquierdo y derecho de R también son árboles de búsqueda binarios.

- Los BST almacenan sus elementos ordenados, lo que es útil para tareas de búsqueda/clasificación.



# Ejercicio

- ¿Cuáles de los árboles que se muestran son árboles binarios de búsqueda legales?

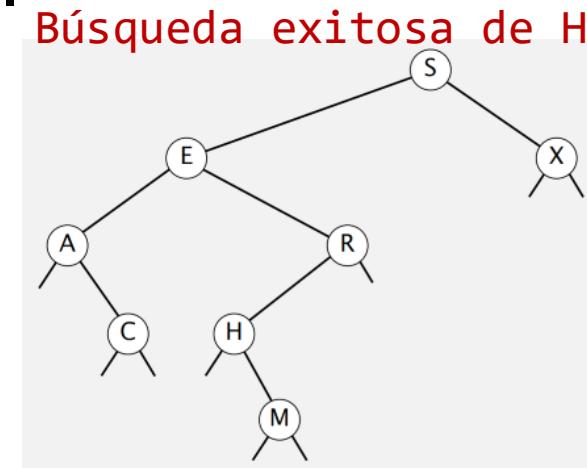


# Buscando en un BST

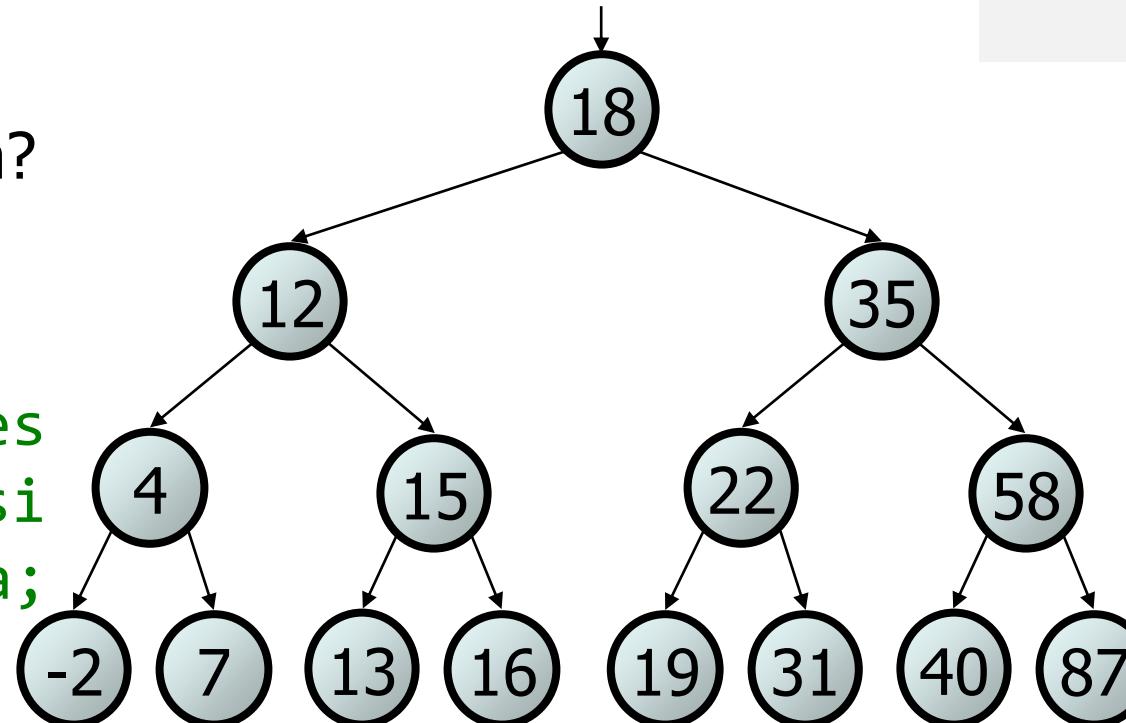
- Describa un algoritmo para buscar en el árbol de abajo el valor 31.

- Luego busque el valor 6.

- ¿Cuál es el número máximo de nodos que necesitaría examinar para realizar cualquier búsqueda?



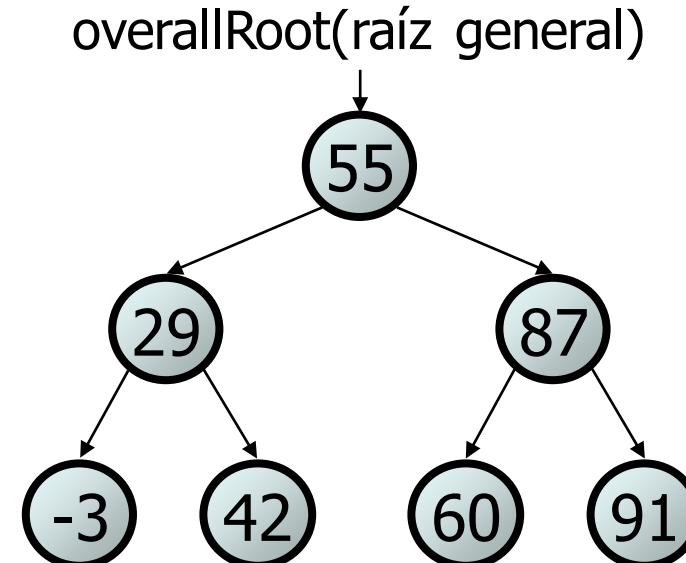
overallRoot(raíz general)



Buscar (search): Si es menor, ve a la izquierda; si es mayor, ve a la derecha; si es igual, acertaste.

# Ejercicio

- Convierta la clase IntTree en una clase SearchTree .
  - Los elementos del árbol constituirán un árbol de búsqueda binaria legítimo.
- Agregue un método contains a la clase SearchTree que busca en el árbol un número entero determinado y devuelve true si lo encuentra.
  - Si una variable SearchTree denominada Tree referencia al siguiente árbol, las siguientes llamadas tendrían estos resultados:
    - tree.contains(29) → true
    - tree.contains(55) → true
    - tree.contains(63) → false
    - tree.contains(35) → false



# Solución del ejercicio

```
// Devuelve si este árbol contiene el entero dado.
public boolean contains(int value) {
    return contains(overallRoot, value);
}

private boolean contains(IntTreeNode root, int value) {
    if (root == null) {
        return false;
    } else if (root.data == value) {
        return true;
    } else if (root.data > value) {
        return contains(root.left, value);
    } else { // root.data < value
        return contains(root.right, value);
    }
}
```

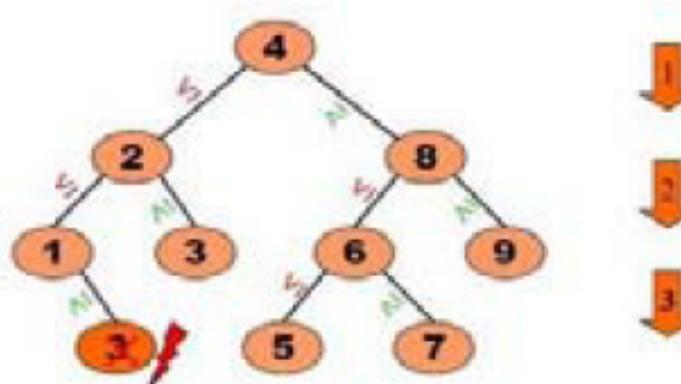
## Árboles binarios de búsqueda

Es un árbol binario formado por elementos en los cuales se asume un orden lineal donde:

1. Los elementos almacenados en el subárbol izquierdo de cualquier nodo: son menores que el elemento almacenado en dicho nodo.
2. Los elementos almacenados en el subárbol derecho de cualquier nodo: son mayores que el elemento almacenado en ese nodo.

Operaciones: Insertar

Eliminar Miembro



# Definiciones

- Son el caso particular mas simple
- Definición:
- Un conjunto  $T$  de elementos es un árbol binario de búsqueda si:
  - $T$  es vacío
  - $T$  esta particionado en 3 conjuntos disjuntos
    1. **Un solo elemento llamado la raíz**
    2. **2 conjuntos que son ABB, llamados subárbol izquierdo y subárbol derecho**

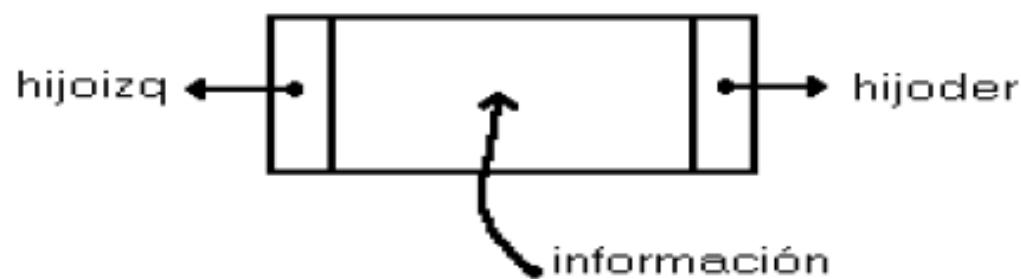


## Definiciones

- Donde  $n$  es un nodo y  $TD$  y  $TI$  son árboles binarios, además de cumplir con las siguientes propiedades:
  - La llave de búsqueda (el valor) de  $n$  es MAYOR que todas las llaves de búsqueda de  $TI$
  - La llave de búsqueda (el valor) de  $n$  es MENOR que todas las llaves de búsqueda de  $TN$

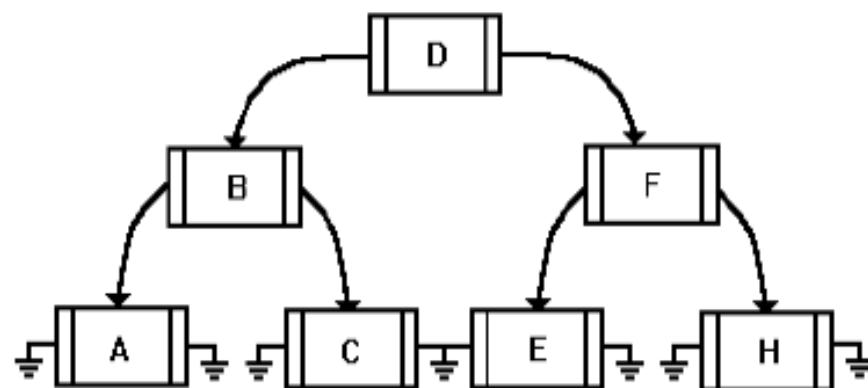
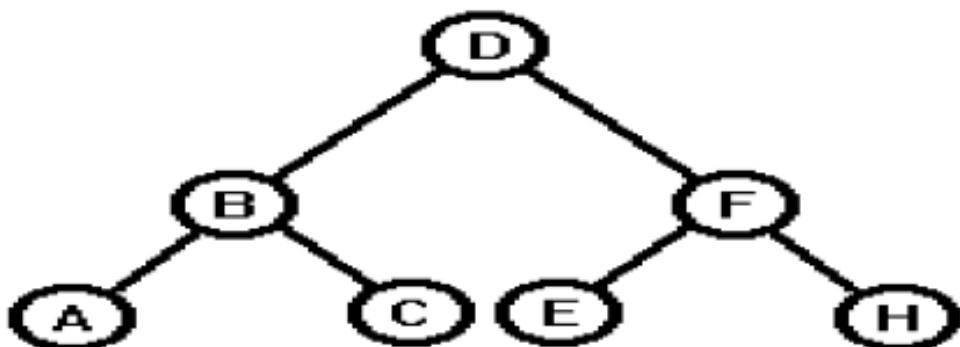
# Representación

- La forma mas simple de representar un árbol binario es la lista ligada:



# Ejemplo

- Representación ligada



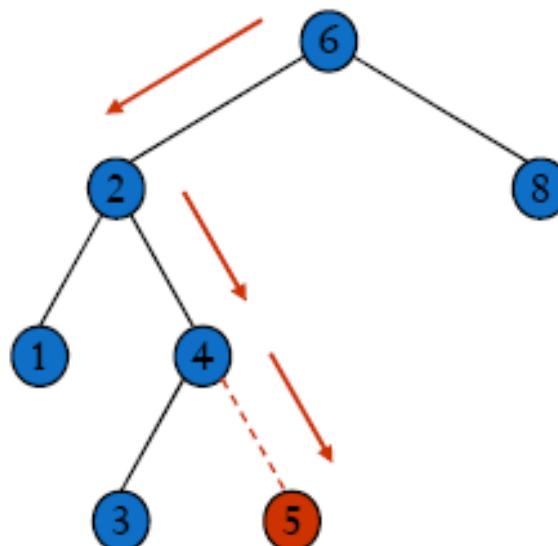
# Operaciones con un ABB

- Inserción
- Búsqueda
- Eliminación
- Recorridos

# Inserción

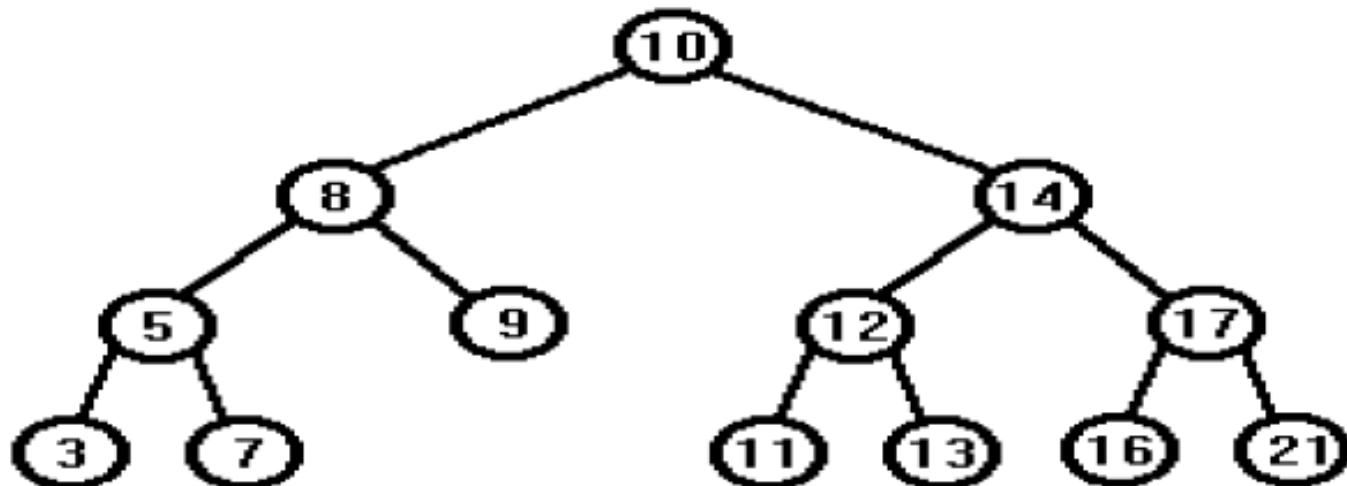
- Insertar el elemento X
- Si el árbol esta vacío, X será la raíz del árbol
- Si no esta vacío se compara el valor de X con el del nodo padre:
  - Si es **menor** se inserta como un *hijo izquierdo*
  - Si es **mayor** se inserta como *hijo derecho*

Insertando 5



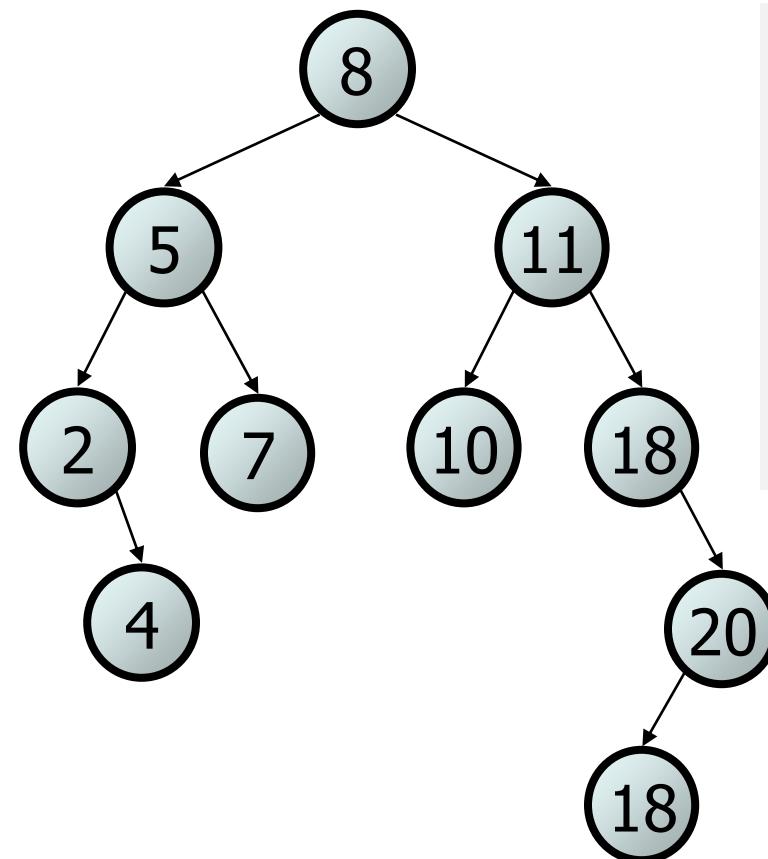
## Inserción, Ejemplo

- Crear un ABB insertando los siguientes elementos: 10, 8, 14, 12, 9, 17, 5, 7, 11, 16, 13, 3, 21

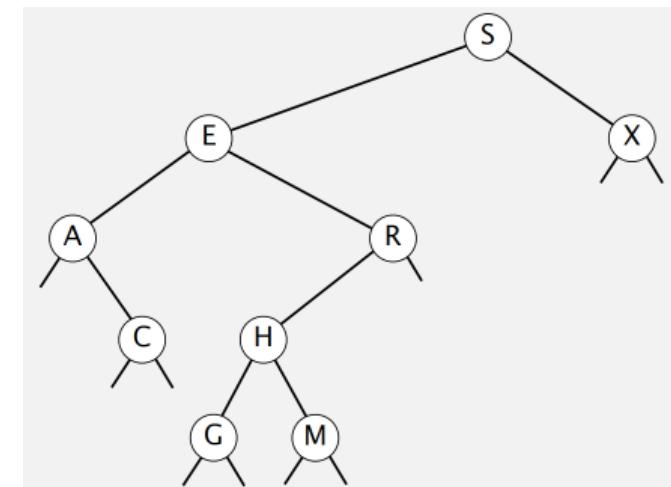


# Insertar a un BST

- Supongamos que queremos insertar el valor 14 al BST a continuación.
  - ¿Dónde se debe agregar el nuevo nodo?
- ¿Dónde adicionaríamos el valor 3?
- ¿Dónde adicionaríamos 7?
- Si el árbol está vacío, ¿dónde se debe agregar un nuevo valor?
- ¿Cuál es el algoritmo general?

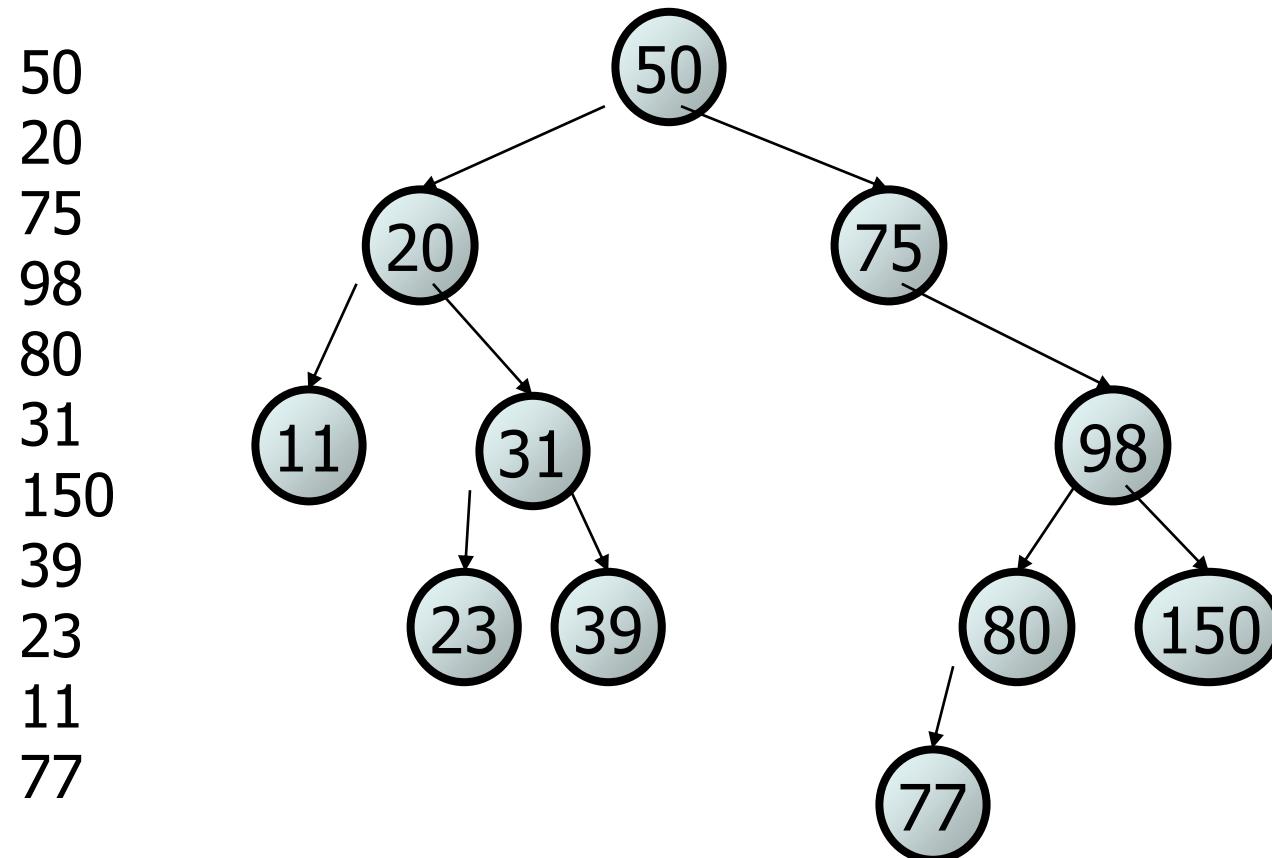


Insertar G



# Ejercicio para Insertar

- Dibuje cómo se vería un árbol de búsqueda binaria si se agregaran los siguientes valores a un árbol inicialmente vacío en este orden:

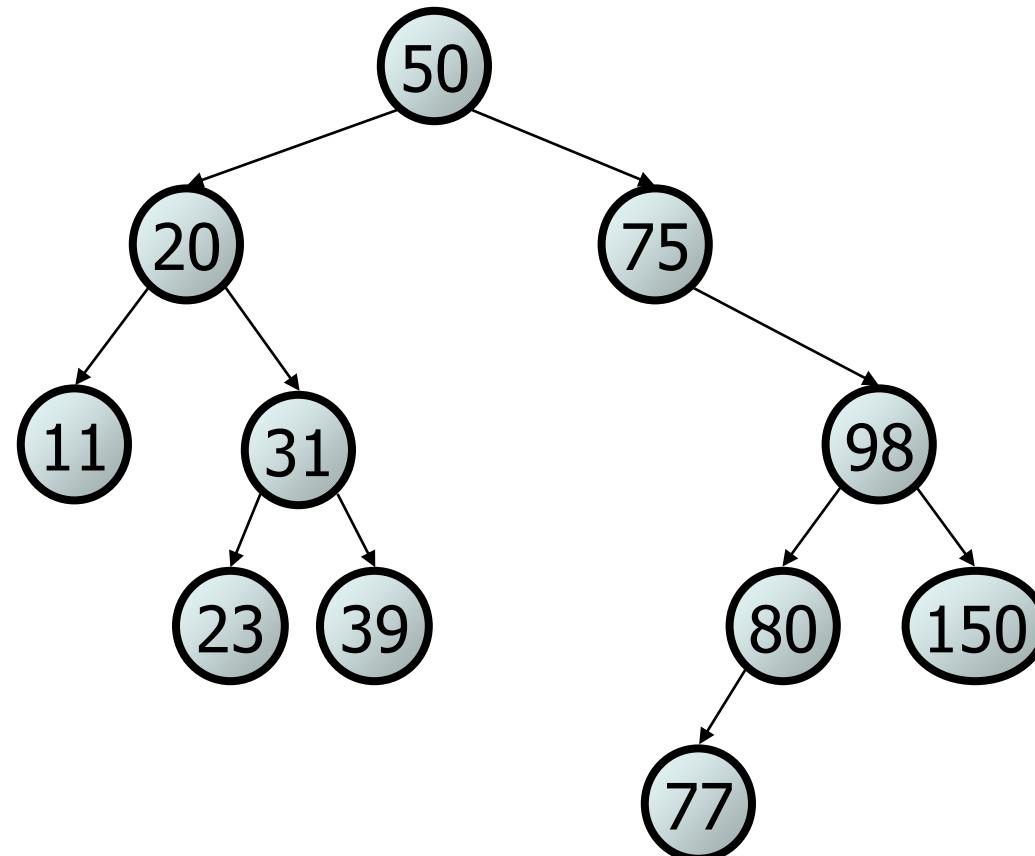


**Insertar:** Si es menor, ve a la izquierda; si es mayor, ve a la derecha; si es nulo, inserte.

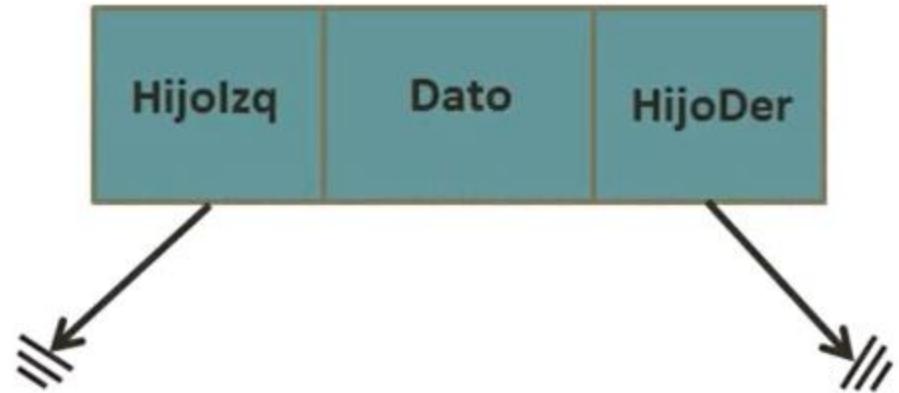
# Ejercicio para Adicionar

- Dibuje cómo se vería un árbol de búsqueda binaria si se agregaran los siguientes valores a un árbol inicialmente vacío en este orden:

50  
20  
75  
98  
80  
31  
150  
39  
23  
11  
77



# Estructura de un Nodo de un árbol binario



Clase NodoArbol y sus operaciones

```
public class NodoArbol {  
    int dato;  
    NodoArbol hijoIzquierdo;  
    NodoArbol hijoDerecho;  
    public NodoArbol(int d) {  
        this.dato=d;  
        this.hijoIzquierdo=null;  
        this.hijoDerecho=null;  
    }  
}
```

1. Crear el Árbol (Vacío)
2. Insertar un Nodo
3. Borrar un Nodo
4. Saber si está Vacío
5. Obtener la Raíz
6. Determinar su Altura
7. Determinar su número de elementos

# Clase Arbol Binario con el método agregarNodo (Insertar)

```
2 public class ArbolBinario {  
3     NodoArbol raiz;  
4     public ArbolBinario() {  
5         raiz = null;  
6     }  
7     //Método para insertar un Nodo en el arbol  
8     public void agregarNodo(int d, String nom) {  
9         NodoArbol nuevo = new NodoArbol(d, nom);  
10        if (raiz == null) {  
11            raiz = nuevo;  
12        } else {  
13            NodoArbol auxiliar = raiz;  
14            NodoArbol padre;  
15            while (true) {  
16                padre = auxiliar;  
17                if (d < auxiliar.dato) {  
18                    auxiliar = auxiliar.hijoIzquierdo;  
19                    if (auxiliar == null) {  
20                        padre.hijoIzquierdo = nuevo;  
21                        return;  
22                    }  
23                } else {  
24                    auxiliar = auxiliar.hijoDerecho;  
25                    if (auxiliar == null) {  
26                        padre.hijoDerecho = nuevo;  
27                        return;  
28                    }  
29                }  
30            }  
31        }  
32    }  
33}
```

```
2 public class NodoArbol {  
3     int dato;  
4     String nombre;  
5     NodoArbol hijoIzquierdo, hijoDerecho;  
6     public NodoArbol(int d, String nom) {  
7         this.dato = d;  
8         this.nombre = nom;  
9         this.hijoIzquierdo = null;  
10        this.hijoDerecho = null;  
11    }  
12    @Override  
13    public String toString(){  
14        return nombre + "Su Dato es " + dato;  
15    }  
16}
```

# Clase Arbol Binario con diversos métodos para recorrerlo

```
33     //Metodo para saber cuando el arbol está vacio
34     public boolean estaVacio(){
35         return raiz==null;
36     }
37     //Metodo para recorrer el árbol inOrden
38     public void entreOrden(NodoArbol r){
39         if(r!=null){
40             entreOrden(r.hijoIzquierdo);
41             System.out.println(r.dato);
42             entreOrden(r.hijoDerecho);
43         }
44     }
45 }
```

```
52     public void preOrden(NodoArbol r) {
53         if (r != null) {
54             System.out.println(r.dato);
55             preOrden(r.hijoIzquierdo);
56             preOrden(r.hijoDerecho);
57         }
58     }
59
60     public void postOrden(NodoArbol r) {
61         if (r != null) {
62             postOrden(r.hijoIzquierdo);
63             postOrden(r.hijoDerecho);
64             System.out.println(r.dato);
65         }
66     }
67 }
```

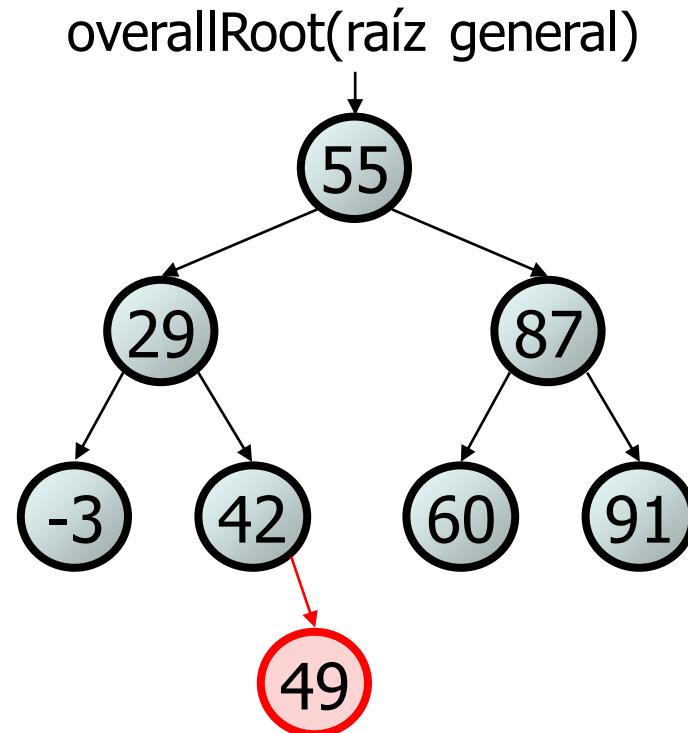
# Pruebas con la Clase Arbol Binario

```
3 public class TestArbol {
4     public static void main(String[] args) {
5         int opcion=0, elemento;
6         String nombre;
7         ArbolBinario arboll = new ArbolBinario();
8         do{
9             try{
10                 opcion=Integer.parseInt(JOptionPane.showInputDialog(null,
11                     "1. Agregar un nodo\n"+
12                     "2. Recorrer el arbol EntreOrden\n"+
13                     "3. Recorrer el arbol PreOrden\n"+
14                     "4. Recorrer el arbol PostOrden\n"+
15                     "5. Salir\n"+
16                     "Eligen una opción...", "Arboles Binarios"
17                     ,JOptionPane.QUESTION_MESSAGE));
18
19                 switch(opcion){
20                     case 1:
21                         elemento = Integer.parseInt(JOptionPane.showInputDialog(null,
22                             "Ingresa el número del nodo...", "Agregando el nodo"
23                             , JOptionPane.QUESTION_MESSAGE));
24                         nombre = JOptionPane.showInputDialog(null,
25                             "Ingresa el nombre del nodo...", "Agregando el nodo",
26                             JOptionPane.QUESTION_MESSAGE);
27                         arboll.agregarNodo(elemento, nombre);
28                         break;
29                     case 2:
30                         if(!arboll.estaVacio()){
31                             arboll.entreOrden(arboll.raiz);
32                         }else{
33                             JOptionPane.showMessageDialog(null, "El árbol está vacío",
34                             ";Cuidado!", JOptionPane.INFORMATION_MESSAGE);
35                         }
36                         break;
37
38                     case 3:
39                         if(!arboll.estaVacio()){
40                             arboll.preOrden(arboll.raiz);
41                         }else{
42                             JOptionPane.showMessageDialog(null, "El árbol está vacío",
43                             ";Cuidado!", JOptionPane.INFORMATION_MESSAGE);
44                         }
45                         break;
46                     case 4:
47                         if(!arboll.estaVacio()){
48                             arboll.postOrden(arboll.raiz);
49                         }else{
50                             JOptionPane.showMessageDialog(null, "El árbol está vacío",
51                             ";Cuidado!", JOptionPane.INFORMATION_MESSAGE);
52                         }
53                         break;
54
55                     case 5:
56                         JOptionPane.showMessageDialog(null, "Aplicacion Finalizada",
57                             "Fin", JOptionPane.INFORMATION_MESSAGE);
58                         break;
59                     default:
60                         JOptionPane.showMessageDialog(null, "Opción incorrecta",
61                             ";Cuidado!", JOptionPane.INFORMATION_MESSAGE);
62                 }
63             }catch(NumberFormatException n){
64                 JOptionPane.showMessageDialog(null,"Error "+ n.getMessage());
65             }
66         }while(opcion!=5);
67     }
68 }
```

# Ejercicio

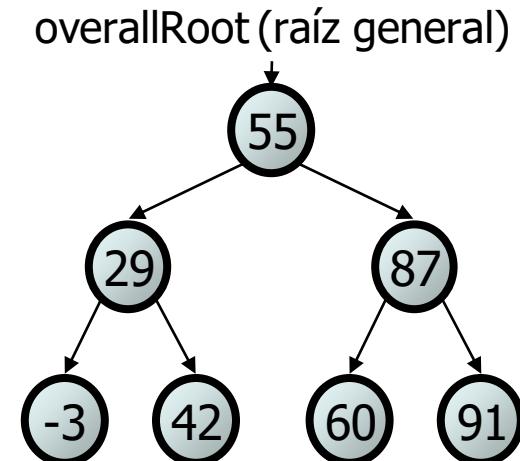
- Agregue un método `add` a la clase `SearchTree` que agregue un valor entero dado al árbol. Suponga que los elementos de `SearchTree` constituyen un árbol de búsqueda binaria legítimo y agregue el nuevo valor en el lugar apropiado para mantener el orden.

– `árbol.add(49);`



# Una solución incorrecta

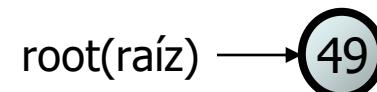
```
// Agrega el valor dado a este BST de manera ordenada.  
public void add(int value) {  
    add(overallRoot, value);  
}  
  
private void add(IntTreeNode root, int value) {  
    if (root == null) {  
        root = new IntTreeNode(value);  
    } else if (root.data > value) {  
        add(root.left, value);  
    } else if (root.data < value) {  
        add(root.right, value);  
    }  
    // else root.data == value;  
    // un duplicado (no agregar)  
}
```



- ¿Por qué no funciona esta solución?

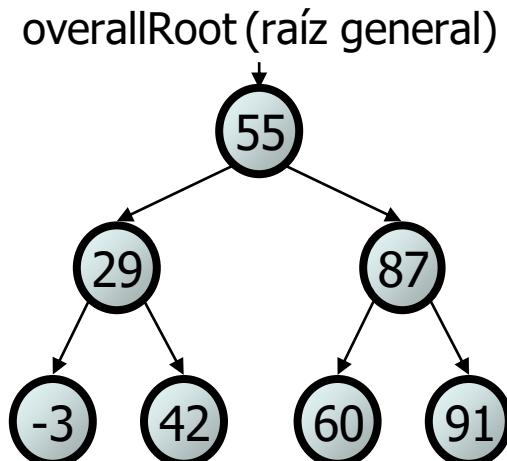
# El problema

- Al igual que con las listas enlazadas, si solo modificamos a qué se refiere una variable local, no cambiará la colección.



```
private void add(IntTreeNode root, int value) {  
    if (root == null) {  
        root = new IntTreeNode(value);  
    }  
}
```

- En el caso de la lista enlazada, ¿cómo modificamos realmente la lista?
  - cambiando el frente (front)
  - cambiando el campo siguiente (next) de un nodo



# Una pobre solución correcta

```
// Agrega el valor dado a este BST de manera ordenada. (mal estilo)
public void add(int value) {
    if (overallRoot == null) {
        overallRoot = new IntTreeNode(value);
    } else if (overallRoot.data > value) {
        add(overallRoot.left, value);
    } else if (overallRoot.data < value) {
        add(overallRoot.right, value);
    }
    // else overallRoot.data == value; un duplicado (no agregar)
}

private void add(IntTreeNode root, int value) {
    if (root.data > value) {
        if (root.left == null) {
            root.left = new IntTreeNode(value);
        } else {
            add(overallRoot.left, value);
        }
    } else if (root.data < value) {
        if (root.right == null) {
            root.right = new IntTreeNode(value);
        } else {
            add(overallRoot.right, value);
        }
    }
    // else root.data == value; un duplicado (no agregar)
}
```

# **x = change(x);**

- Los métodos de cadena que modifican una cadena en realidad devuelven una nueva.
  - Si queremos modificar una variable de cadena, debemos reasignarla.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);      // lil bow wow
s = s.toUpperCase();
System.out.println(s);      // LIL BOW WOW
```

- A este patrón algorítmico general lo llamamos **x = change(x);**
- Usaremos este enfoque cuando escribamos métodos que modifiquen la estructura de un árbol binario.

# Aplicando $x = \text{change}(x)$

- Los métodos que modifican un árbol deben tener el siguiente patrón:
  - entrada (parámetro): estado antiguo del nodo
  - salida (retorno): nuevo estado del nodo

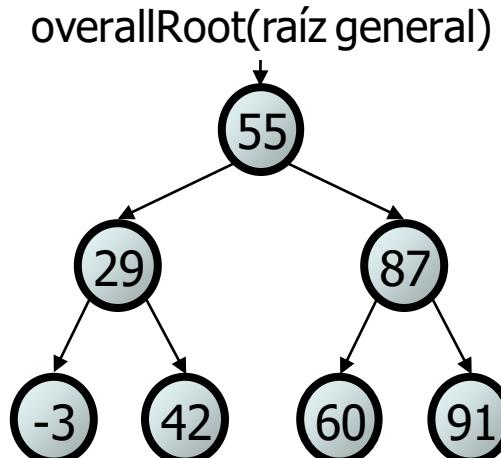


- Para cambiar realmente el árbol, debe reasignar:

```
root = change (root, parámetros) ;  
root.left = change (root.left, parámetros) ;  
root.right = change (root.right, parámetros) ;
```

# una solución correcta

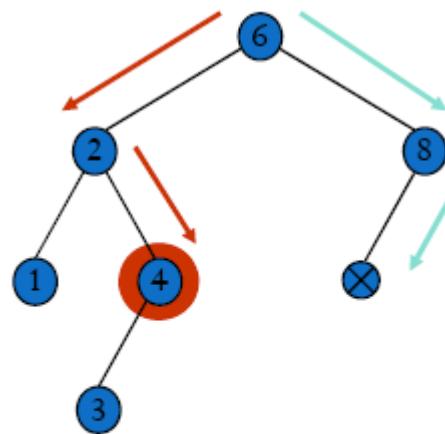
```
// Agrega el valor dado a este BST de manera ordenada.  
public void add(int value) {  
    overallRoot = add(overallRoot, value);  
}  
  
private IntTreeNode add(IntTreeNode root, int value) {  
    if (root == null) {  
        root = new IntTreeNode(value);  
    } else if (root.data > value) {  
        root.left = add(root.left, value);  
    } else if (root.data < value) {  
        root.right = add(root.right, value);  
    } // sino un duplicado  
  
    return root;  
}
```



- Piensa en el caso cuando `root` es una hoja...

# Búsqueda

- Para buscar al elemento X:
  - Si X es la llave de la raíz de un ABB se ha terminado.
  - Si X es **menor** que el padre, *se busca a la izquierda*
  - Si X es **mayor** que el padre, *se busca a la derecha*



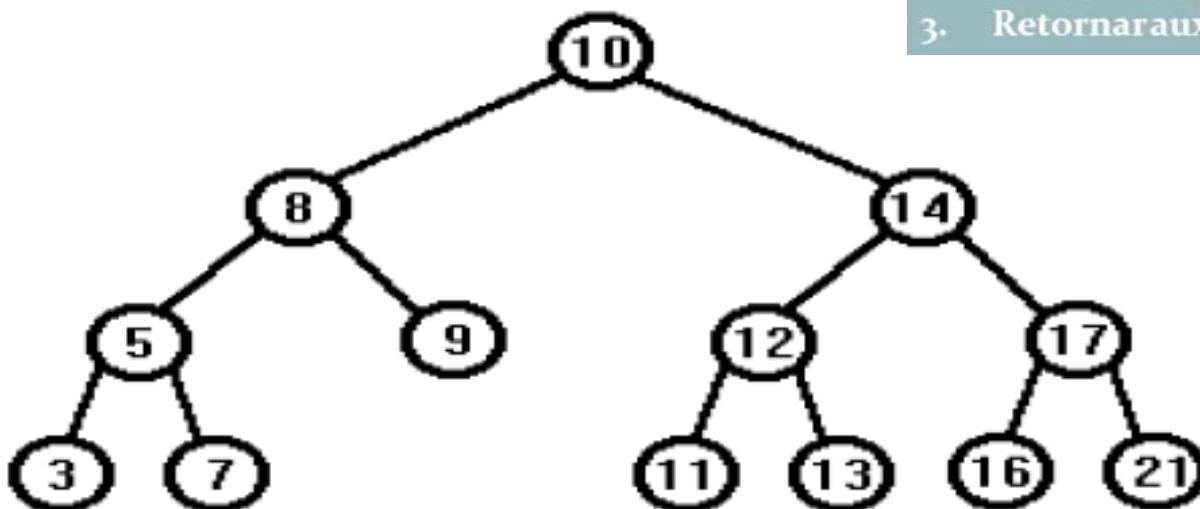
Buscando 4: VERDADERO

Buscando 7: FALSO

# Búsqueda, Ejemplo

- Si se busca el elemento “7”:
- Se compara 7 con 10 y se desciende por la izquierda
- Se compara 7 con 8 y se desciende por la izquierda
- Se compara 7 con 5 y se desciende por la derecha
- Se encuentra la llave deseada

1. Crear un puntero auxiliar de tipo NodoArbol y apuntarlo a la raíz.
2. Mientras auxiliar de dato sea diferente de dato buscado hacer
  1. Si dato buscado es menor que auxiliar de dato entonces
    1. Apuntar a auxiliar a auxiliar de hijo izquierdo
  2. Si No
    1. Apuntar a auxiliar a auxiliar de hijo derecho
  3. Si auxiliares idéntico a nulo
    1. Retornar nulo
  3. Retornar auxiliar



# Clase Arbol Binario con el método buscarNodo

```
65 //Método para buscar un nodo en el árbol
66 public NodoArbol buscarNodo(int d) {
67     NodoArbol aux = raiz;
68     while(aux.dato!=d) {
69         if(d<aux.dato) {
70             aux=aux.hijoIzquierdo;
71         }else{
72             aux=aux.hijoDerecho;
73         }
74         if(aux==null) {
75             return null;
76         }
77     }
78     return aux;
79 }
```

1. Crear un puntero auxiliar de tipo NodoArbol y apuntarlo a la raíz.
2. Mientras auxiliar de dato sea diferente de dato buscado hacer
  1. Si dato buscado es menor que auxiliar de dato entonces
    1. Apuntar a auxiliar a auxiliar de hijo izquierdo
  2. Si No
    1. Apuntar a auxiliar a auxiliar de hijo derecho
  3. Si auxiliares idéntico a nulo
    1. Retornar nulo
  3. Retornar auxiliar

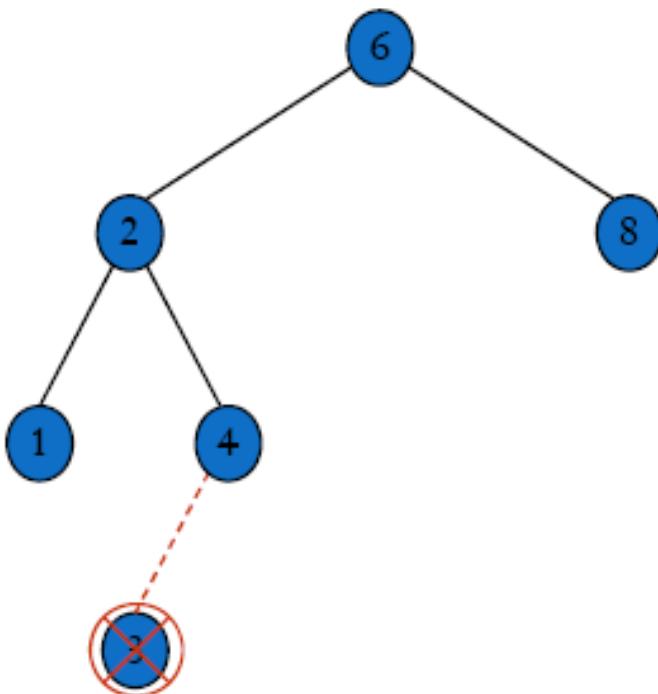
# Eliminación

- Existen cuatro distintos escenarios:
  1. Intentar eliminar un nodo que no existe.
    - No se hace nada, simplemente se regresa FALSE.
  2. Eliminar un nodo hoja.
    - Caso sencillo; se borra el nodo y se actualiza el apuntador del nodo padre a NULL.
  3. Eliminar un nodo con un solo hijo.
    - Caso sencillo; el nodo padre del nodo a borrar se convierte en el padre del único nodo hijo.
  4. Eliminar un nodo con dos hijos.
    - Caso complejo, es necesario mover más de un apuntador.

# ELIMINAR (casos sencillos)

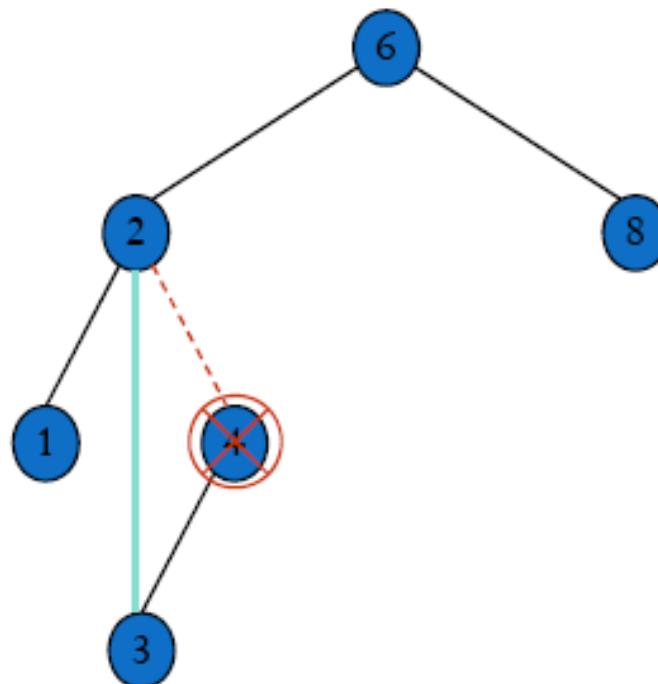
Eliminar nodo hoja

*Eliminar 3*



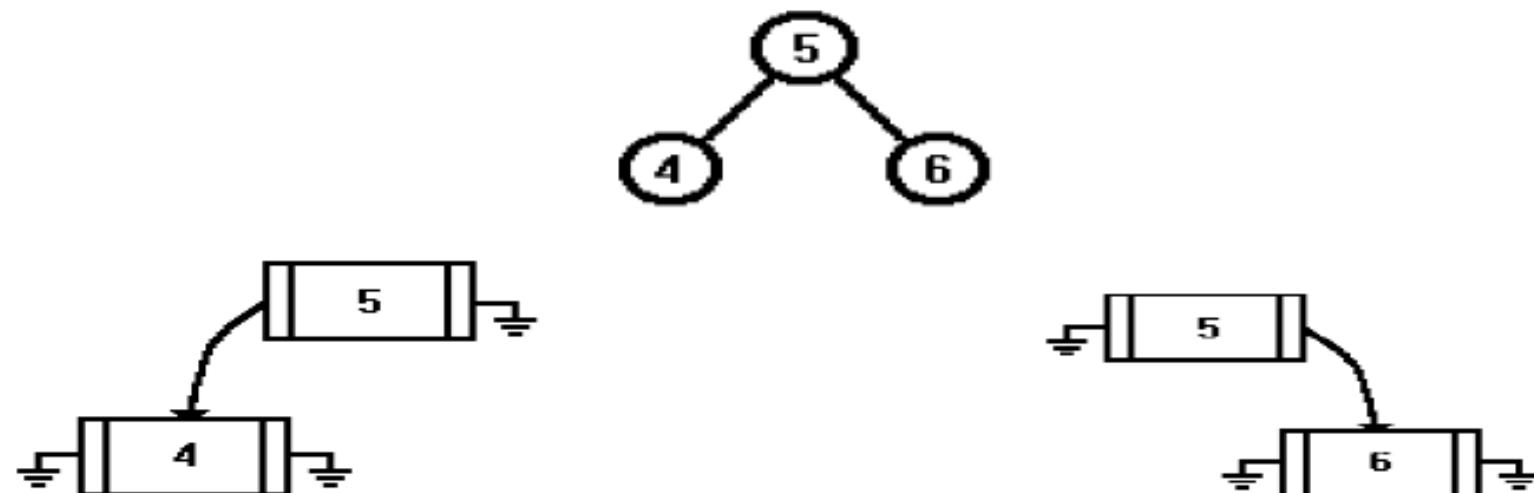
Eliminar nodo con un hijo

*Eliminar 4*



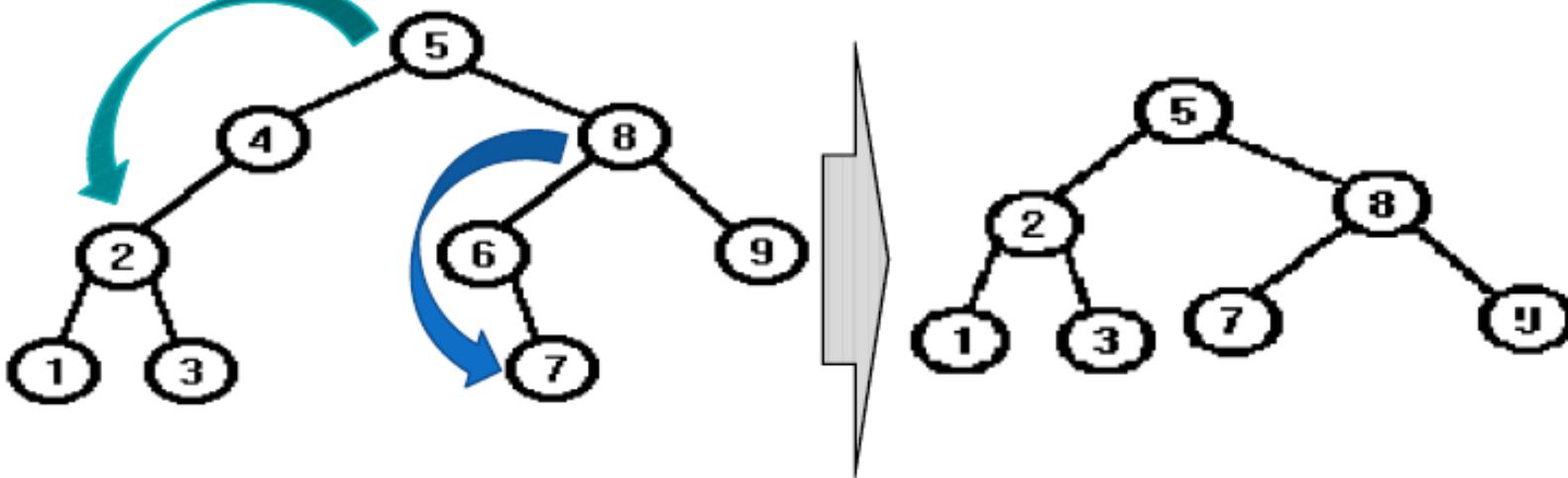
# Eliminación de una hoja

- Caso mas sencillo, en donde lo único que hay que hacer es que la liga que lo referencia debe ser *nil*



## Con un hijo único

- Para eliminar un nodo, su padre deberá referenciar a su hijo.

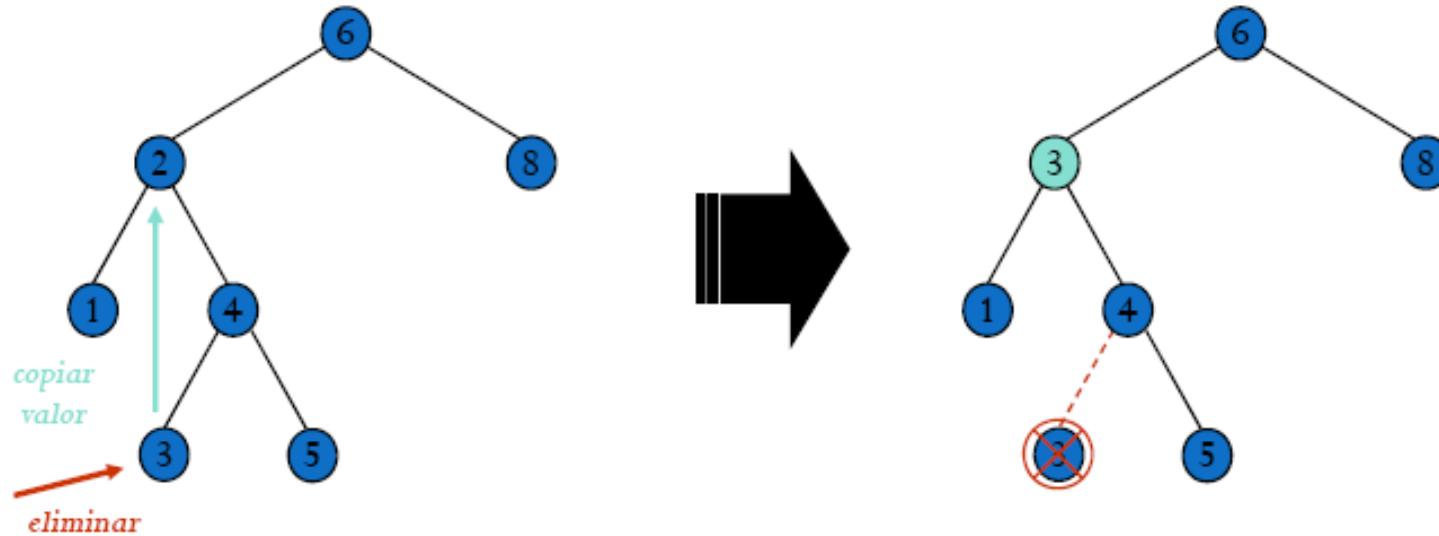


- Al eliminar la llave 4, se debe cambiar el subárbol izquierdo de 5 por el que contiene a 2 como raíz.
- Es el mismo procedimiento para eliminar un nodo con un único hijo izquierdo.
- Al eliminar el nodo cuya información es 6, el subárbol izquierdo de 8 referenciará al subárbol derecho del 6 (nodo 7)

# ELIMINAR (Caso complejo)

Eliminar nodo con dos hijos

Eliminar 2

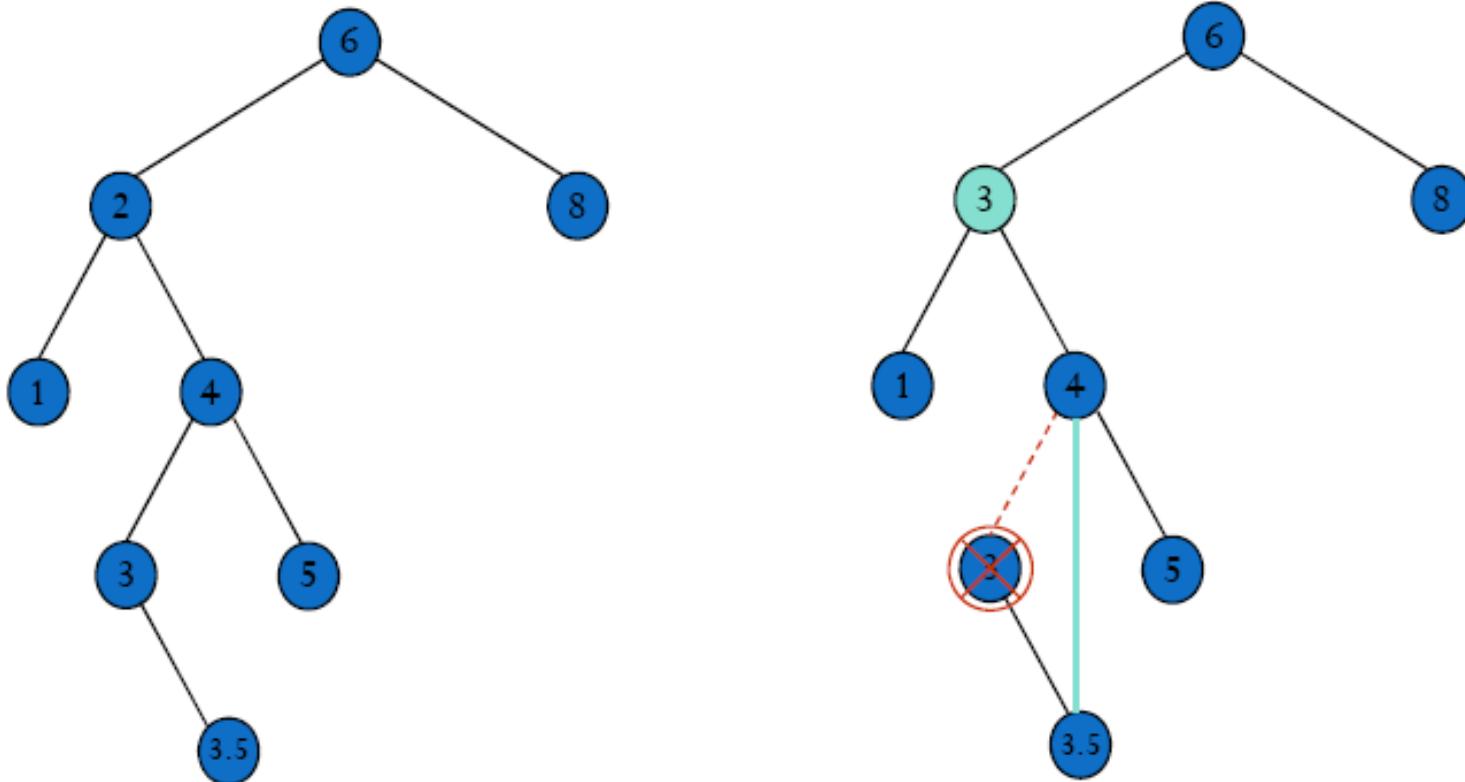


- Remplazar el dato del nodo que se desea eliminar con el dato del nodo más pequeño del subárbol derecho
- Después, eliminar el nodo más pequeño del subárbol derecho (caso fácil)

# Otro ejemplo (caso complejo)

Eliminar nodo con dos hijos

*Eliminar 2*



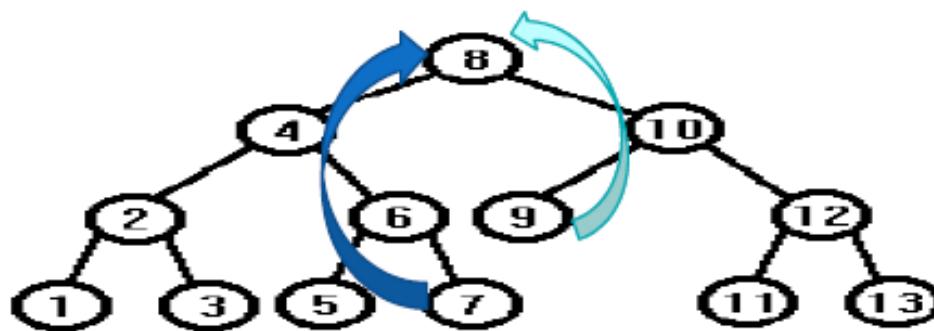
## Con dos hijos

- Existen dos opciones:
  - Sustituir el valor de la información del nodo a eliminar por el nodo que contiene la menor llave del subárbol derecho (**Menor de los mayores**)
  - Sustituir el valor de la información del nodo a eliminar por el nodo que contiene la mayor llave del subárbol izquierdo (**Mayor de los menores**)

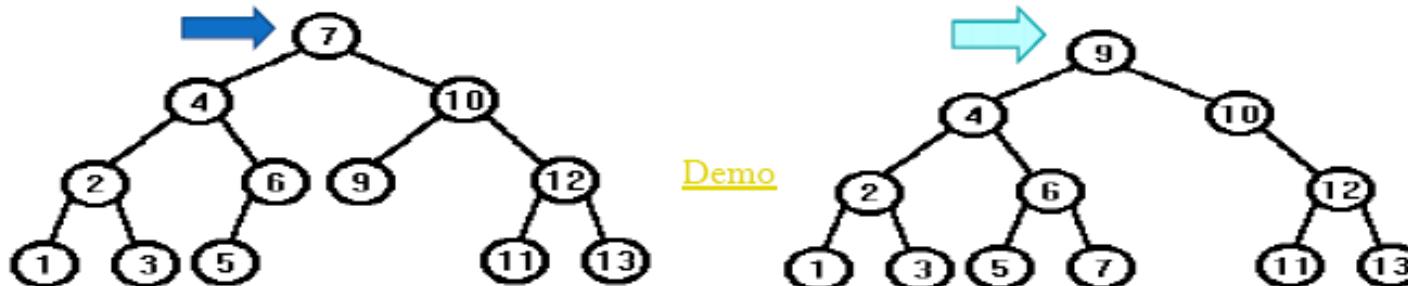
# Eliminación con dos hijos

Si se desea eliminar la raíz,

- Sustituir el valor por la llave con el mayor de los menores y luego eliminar ese valor
- O sustituir el valor por la llave con el menor de los mayores y luego eliminar ese valor



**Utilizando el mayor de los menores** • **Utilizando el menor de los mayores**



## Eliminación de un Nodo

```
public void eliminar(int cod) throws Exception
{ eliminar(raiz,cod);
}

private void eliminar(Nodo rz,int cod) throws Exception
{ if(rz==null) throw new Exception("Arbol de búsqueda binaria, no borrado");
  else if(cod<rz.codigo) eliminar(rz.izq,cod);
  else if(cod>rz.codigo) eliminar(rz.der,cod);
  else // nodo encontrado
  { Nodo t=rz;
    // se enlaza directamente si alguna rama esta vacia
    if(rz.izq==null) rz=rz.der;
    else if(rz.der==null) rz=rz.izq;
    else // tiene dos ramos descendientes
    { Nodo dr=rz.izq;
      Nodo anter=null;
      t=rz;
      while(dr.der!=null)
      { anter=dr;
        dr=dr.der;
      }
      //reemplaza los datos del mayorde los menores
      rz.codigo=dr.codigo;
      rz.nombre=dr.nombre;
      //enlaza nodo ascendente
      if(anter==null) // anterior a rz
        rz.izq=dr.izq;
      else if(anter.izq==dr) anter.izq=dr.izq;
      else if(anter.der==dr) anter.der=dr.izq;
    }
    t=null;
  }
}
```

# Clase Arbol Binario con el método eliminar

```
80     //Método para eliminar un nodo del árbol
81     public boolean eliminar(int d){
82         NodoArbol auxiliar=raiz;
83         NodoArbol padre=raiz;
84         boolean esHijoIzq=true;
85         while(auxiliar.datos!=d){
86             padre=auxiliar;
87             if(d<auxiliar.datos){
88                 esHijoIzq=true;
89                 auxiliar=auxiliar.hijoIzquierdo;
90             }else{
91                 esHijoIzq=false;
92                 auxiliar=auxiliar.hijoDerecho;
93             }
94             if(auxiliar==null){
95                 return false;
96             }
97         }//Fin del while
98         if(auxiliar.hijoIzquierdo==null && auxiliar.hijoDerecho==null){
99             if(auxiliar==raiz){
100                 raiz=null;
101             }else if(esHijoIzq){
102                 padre.hijoIzquierdo=null;
103             }else{
104                 padre.hijoDerecho=null;
105             }
106         }
```

## Los Pasos son:

1. Buscar en el Árbol para encontrar la posición del nodo a eliminar.
2. Si el nodo a suprimir tiene menos de dos hijos, reajustar los enlaces de su antecesor.
3. Si el nodo tiene dos hijos (rama izquierda y derecha), es necesario subir a la posición que éste ocupa el dato más próximo de sus subárboles (el inmediatamente superior o el inmediatamente inferior) con el fin de mantener la estructura de árbol binario de búsqueda.

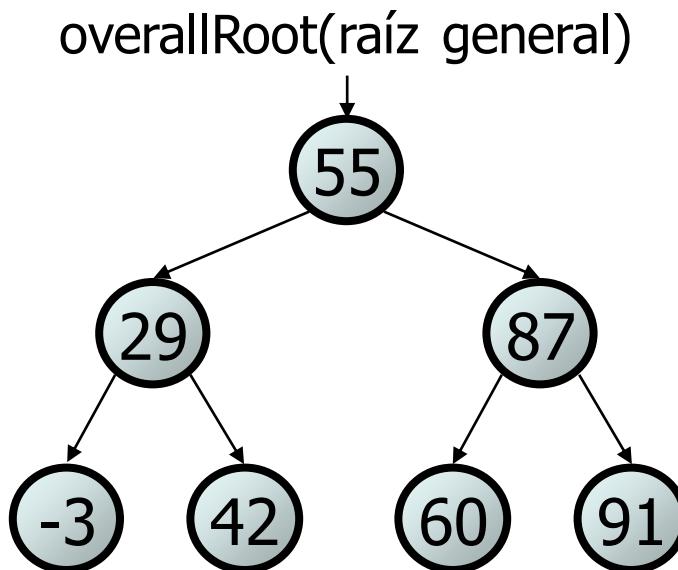
```
106     }else if(auxiliar.hijoDerecho==null){
107         if(auxiliar==raiz){
108             raiz=auxiliar.hijoIzquierdo;
109         }else if(esHijoIzq){
110             padre.hijoIzquierdo=auxiliar.hijoIzquierdo;
111         }else{
112             padre.hijoDerecho=auxiliar.hijoIzquierdo;
113         }
114     }else if(auxiliar.hijoIzquierdo==null){
115         if(auxiliar==raiz){
116             raiz=auxiliar.hijoDerecho;
117         }else if(esHijoIzq){
118             padre.hijoIzquierdo=auxiliar.hijoDerecho;
119         }else{
120             padre.hijoDerecho=auxiliar.hijoIzquierdo;
121         }
122     }else{
123         NodoArbol reemplazo=obtenerNodoReemplazo(auxiliar);
124         if(auxiliar==raiz){
125             raiz=reemplazo;
126         }else if(esHijoIzq){
127             padre.hijoIzquierdo=reemplazo;
128         }else{
129             padre.hijoDerecho=reemplazo;
130         }
131         reemplazo.hijoIzquierdo=auxiliar.hijoIzquierdo;
132     }
133     return true;
134 }

135 //Método encargado de devolver el nodo reemplazo
136 public NodoArbol obtenerNodoReemplazo(NodoArbol nodoReemp){
137     NodoArbol reemplazarPadre=nodoReemp;
138     NodoArbol reemplazo=nodoReemp;
139     NodoArbol auxiliar=nodoReemp.hijoDerecho;
140     while(auxiliar!=null){
141         reemplazarPadre=reemplazo;
142         reemplazo=auxiliar;
143         auxiliar=auxiliar.hijoIzquierdo;
144     }
145     if(reemplazo!=nodoReemp.hijoDerecho){
146         reemplazarPadre.hijoIzquierdo=reemplazo.hijoDerecho;
147         reemplazo.hijoDerecho=nodoReemp.hijoDerecho;
148     }
149     System.out.println("El nodo reemplazo es "+reemplazo);
150 }
151 }
```

# Ejercicio

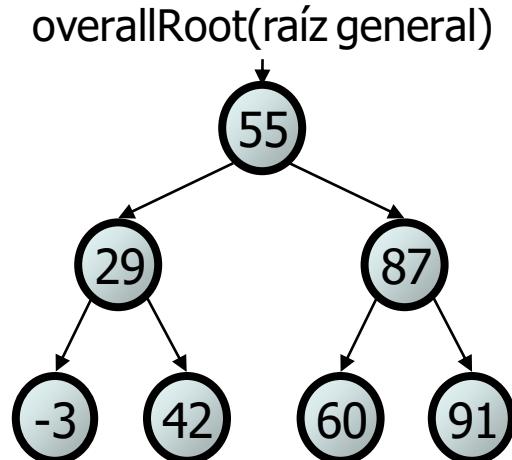
- Agregue un método `getMin` a la clase `IntTree` que devuelva el valor entero mínimo del árbol. Suponga que los elementos de `IntTree` constituyen un árbol de búsqueda binaria legítimo. Lanza una `NoSuchElementException` si el árbol está vacío.

```
int min = tree.getMin(); // -3
```



# Solución del ejercicio

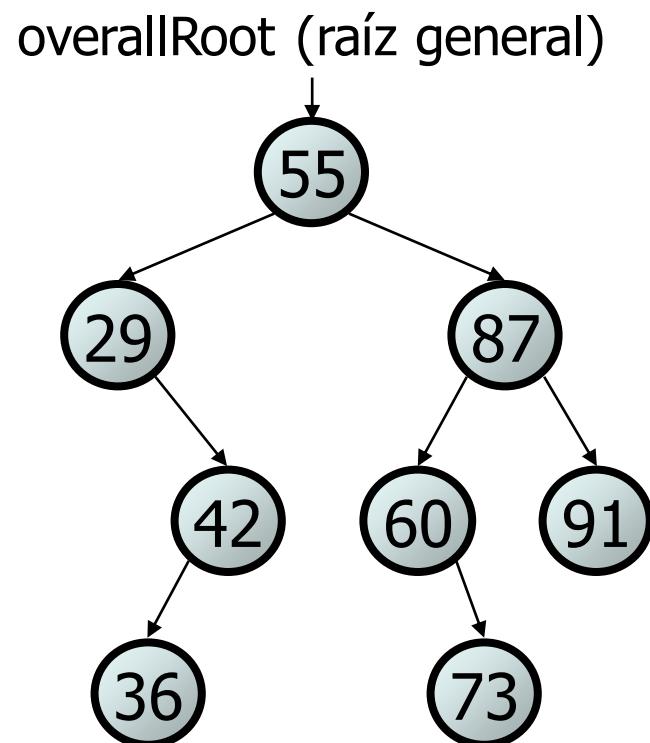
```
// Devuelve el valor mínimo de este BST.  
// Lanza una excepción NoSuchElementException si el árbol  
// está vacío.  
public int getMin() {  
    if (overallRoot == null) {  
        throw new NoSuchElementException();  
    }  
    return getMin(overallRoot);  
}  
  
private int getMin(IntTreeNode root) {  
    if (root.left == null) {  
        return root.data;  
    } else {  
        return getMin(root.left);  
    }  
}
```



# Ejercicio

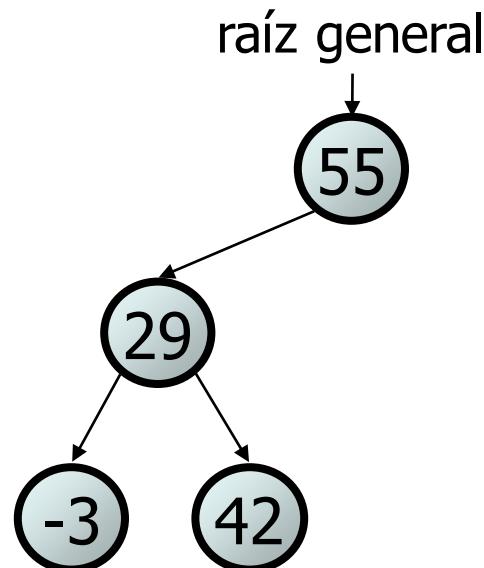
- Agregue un método `remove` a la clase `IntTree` que elimine un valor entero dado del árbol, si está presente. Suponga que los elementos de `IntTree` constituyen un árbol de búsqueda binaria legítimo y elimine el valor de tal manera que se mantenga el orden.

- `tree.remove(73);`
- `tree.remove(29);`
- `tree.remove(87);`
- `tree.remove(55);`

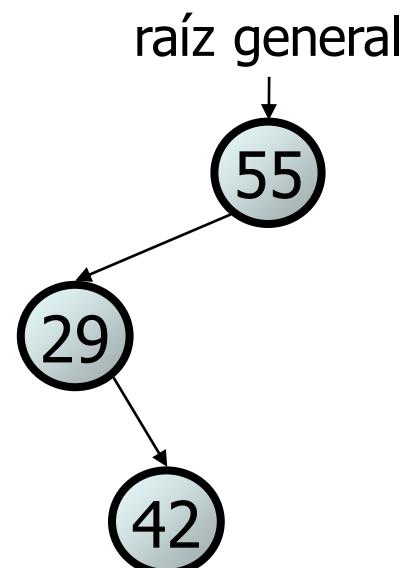


# Casos de eliminación 1

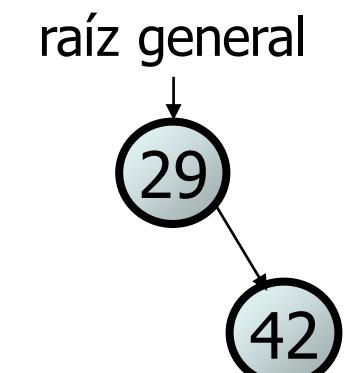
1. una hoja: reemplazar con null
2. un nodo con un hijo izquierdo solamente: reemplazar con hijo izquierdo
3. un nodo con un hijo derecho solamente: reemplazar con hijo derecho



`tree.remove(-3);`



`tree.remove(55);`

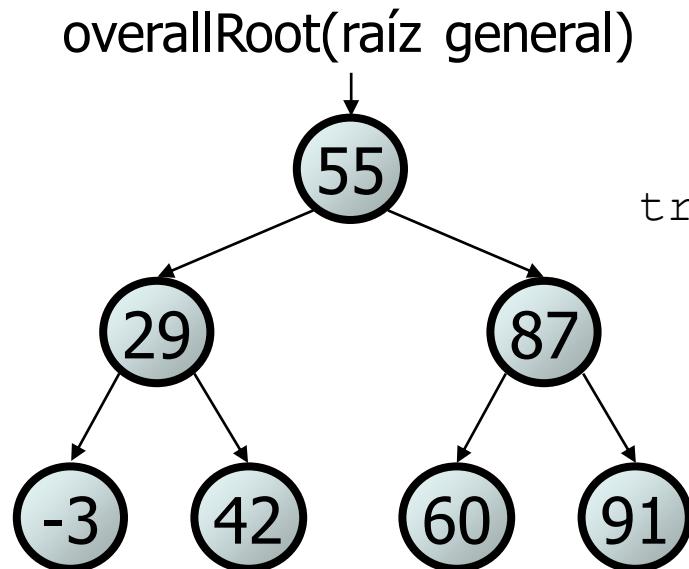


`tree.remove(29);`

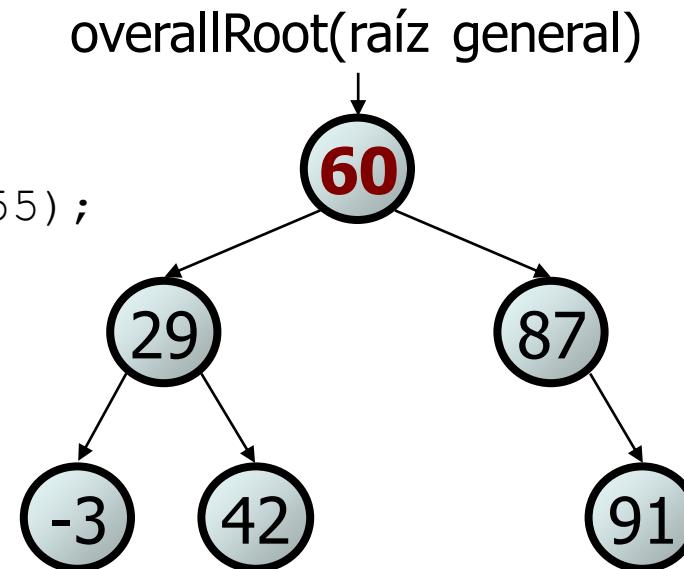


# Casos de eliminación 2

4. un nodo con **ambos** hijos: reemplazar con **min desde la derecha**



tree.remove(55);



# Solución del ejercicio

```
// Elimina el valor dado de este BST, si existe.
public void remove(int value) {
    overallRoot = remove(overallRoot, value);
}

private IntTreeNode remove(IntTreeNode root, int value) {
    if (root == null) {
        return null;
    } else if (root.data > value) {
        root.left = remove(root.left, value);
    } else if (root.data < value) {
        root.right = remove(root.right, value);
    } else { // root.data == value; eliminar este nodo
        if (root.right == null) {
            return root.left; // sin hijo R; reemplazar con L
        } else if (root.left == null) {
            return root.right; // sin hijo L; reemplazar con R
        } else {
            // ambos hijos; reemplazar con min de R
            root.data = getMin(root.right);
            root.right = remove(root.right, root.data);
        }
    }
    return root;
}
```

# EJEMPLO

 REGISTRO DE EMPLEADOS: Uso de Arboles Binarios de Búsqueda

<b>Datos del empleado</b>	
CODIGO	<input type="text"/>
NOMBRE	<input type="text"/>
APELLID...	<input type="text"/>
SEXO	--Seleccione-- <input type="button" value="▼"/>
SUELDO	<input type="text"/>

<b>Guardar</b>	<b>Actualizar</b>
<b>Consultar</b>	<b>Reestaurar</b>
<b>Eliminar</b>	<b>Salir</b>

**Mostrar en Pre-Orden**   **Mostrar In-Orden**   **Mostrar en Post-Orden**

Nº	Codigo	Nombres	Apellidos	Sexo	Sueldo
1	1253	CELIA	FLORES	FEMENINO	324,00
2	5698	MILUSKA	PRIETO	FEMENINO	234,00
3	6543	CARLOS	GONZALEZ	MASCULINO	2345,00
4	7345	JORGE	LAZO	MASCULINO	7890,00

**Empleado con el mayor sueldo**   **Monto de sueldos acumulados**

JORGE LAZO	10793,00
------------	----------

# Definición de clase ARBOL

```
public class Nodo {  
    int codigo;  
    String nombre;  
    String apellidos;  
    String sexo;  
    float sueldo;  
    public Nodo izq;  
    public Nodo der;  
    public Nodo(int cod, String nom, String ape, String sex, float suel)  
    {  
        codigo=cod;  
        nombre=nom;  
        apellidos=ape;  
        sexo=sex;  
        sueldo=suel;  
        izq=null;  
        der=null;  
    }  
}
```

## Instrucciones en el botón Guardar

```
private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {  
    //Capturando la informacion de los objetos  
    int cod=Integer.parseInt(txtCodigo.getText());  
    String nom=txtNombre.getText().toUpperCase();  
    String ape=txtApellidos.getText().toUpperCase();  
    String sex=cbxSexo.getSelectedItem().toString();  
    float suel=Float.parseFloat(txtSueldo.getText());  
    //Creando el nodo de la Lista en memoria y colocando la informacion  
    agregar(cod,nom,ape,sex,suel);  
    LimpiarEntradas();  
    //Mostrando la informacion en PreOrden  
    vaciar_tabla();  
    mostrarListaArbolInOrden(raiz);  
    Resumen(raiz);  
}
```

# Método Agregar que inserta datos en un árbol: campo código (número entero)

```
//Metodo que agrega la informacion al arbol BB
private void agregar(int cod, String nom, String ape, String sex, float suel)
{
    Nodo aux=null,back=null;
    Nodo nuevo=new Nodo(cod,nom,ape,sex,suel);
    //verificando la raiz
    if(raiz==null)
        raiz=nuevo;
    else
        { aux=raiz;
        while(aux!=null)
            { back=aux;
            if(cod<aux.codigo)
                aux=aux.izq;
            else
                aux=aux.der;
            }
            if(cod<back.codigo)
                back.izq=nuevo;
            else
                back.der=nuevo;
        }
    }
```

Método agregar no recursivo

## Método que inserta datos en un árbol: campo código (cadena)

```
private void agregar(String cod, String nom, String suel)
{
    Nodo aux=null, back=null;
    Nodo nuevo = new Nodo(cod, nom, suel);

    if(raiz==null)
        raiz = nuevo;
    else
    {
        aux = raiz;
        while(aux!=null)
        {
            back = aux;
            if(cod.compareTo(aux.codigo)<0)
                aux = aux.izq;
            else
                aux = aux.der;
        }
        if(cod.compareTo(back.codigo)<0)
            back.izq = nuevo;
        else
            back.der = nuevo;
    }
}
```

**Método agregar no recursivo**

## Método que inserta datos en un árbol: campo código (cadena)

```
public Nodo agregar (Nodo p,  String cod, String nom, String suel)
{
    if(p == null)
        p = new Nodo(cod, nom, suel);
    else
        if(cod.compareTo(p.codigo)>0)
            p.der = agregar(p.der, cod, nom, suel);
        else
            p.izq = agregar(p.izq, cod, nom, suel);
    return p;
}
```

**Método agregar recursivo**

# ALGORITMOS DE RECORRIDO

```
public void mostrarListaArbolInOrden(Nodo p) {
    // verificar que hayan elementos en el arbol
    if(p != null) {
        // llamada recursiva avanzando por la izquierda
        mostrarListaArbolInOrden(p.izq);
        //Colocando los datos en la tabla
        agregar_fila(p);
        // llamada recursiva avanzando por derecha
        mostrarListaArbolInOrden(p.der);
    } // fin del if
}

public void mostrarListaArbolPostOrden(Nodo p) {
    // verificar que hayan elementos en el arbol
    if(p != null) {
        // avanza enlaces con llamadas recursivas
        mostrarListaArbolPostOrden(p.izq);
        mostrarListaArbolPostOrden(p.der);
        //Colocando los datos en la tabla
        agregar_fila(p);
    } // fin del if
}

public void mostrarListaArbolPreOrden(Nodo p) {
    // verificar que hayan elementos en el arbol
    if(p != null) {
        //Colocando los datos en la tabla
        agregar_fila(p);
        // avanza enlaces con llamada recursiva
        mostrarListaArbolPreOrden(p.izq);
        mostrarListaArbolPreOrden(p.der);
    } // fin del if
}
```

## Método resumen y calculos\_inOrden para los datos de salida

```
void Resumen(Nodo p)
{ mayor=-9999;
  acum=0;
  String acumm="";
  calculos_InOrden(raiz);
  //Colocando la Información en los objetos
  txtNomMay.setText(nombres);
  // Dandole formato al acumulado
  DecimalFormat df2 = new DecimalFormat("####.00");
  acumm = df2.format(acum);
  txtAcumulado.setText(acumm);
}
public void calculos_InOrden(Nodo p){
  // verificar que hayan elementos en el arbol
  if(p != null) {
    // llamada recursiva avanzando por la izquierda
    calculos_InOrden(p.izq);
    //Colocando los datosd e salida
    if(p.sueldo>mayor)
      { mayor=p.sueldo;
        nombres=p.nombre+" "+p.apellidos;
      }
    acum=acum+p.sueldo;
    // llamada recursiva     avanzando por derecha
    calculos_InOrden(p.der);
  } // fin del if
}
```

## Método **agregar** fila utilizada en los recorridos del árbol

```
void agregar_fila(Nodo p)
{
    String nom,ape,se,su;
    int cod=p.codigo;
    nom=p.nombre;
    ape=p.apellidos;
    se=p.sexo;
    //dando Formato al sueldo
    DecimalFormat df2 = new DecimalFormat ("####.00");
    su = df2.format(p.sueldo);
    Object[] fila=(miModelo.getRowCount ()+1,cod,nom,ape,se,su);
    miModelo.addRow(fila);
}
```

# Instrucciones en el botón Consultar

```
private void btnConsultarActionPerformed(java.awt.event.ActionEvent evt) {  
    int cod=Integer.parseInt(txtCodigo.getText());  
    if(txtCodigo.getText().equalsIgnoreCase("")) {  
        JOptionPane.showMessageDialog(this,"Ingrese un codigo por favor");  
    }  
    else {  
        //Llamada a la funcion que retorna la posicion del dato buscado  
        pFound=Buscar(cod);  
        //Verificando el puntero pFound para mostrar la inf. buscada  
        if(pFound!=null) {  
            txtNombre.setText(pFound.nombre);  
            txtApellidos.setText(pFound.apellidos);  
            if(pFound.sexo.equalsIgnoreCase("MASCULINO"))  
                cbxSexo.setSelectedIndex(2);  
            else  
                cbxSexo.setSelectedIndex(1);  
            txtSueldo.setText(String.valueOf(pFound.sueldo));  
            //Habilitamos los objetos para eliminar y actualizar  
            Habilitar();  
        } else {  
            JOptionPane.showMessageDialog(this,"El código: "+cod + ", no es  
            )  
        }  
    }  
}
```

## Método **buscar** fila utilizada en el botón Consultar

```
Nodo Buscar(int codigo)
{
    Nodo t;
    t = raiz; // empieza por la raiz
    // mientras t no sea nulo
    while(t != null) {
        if(t.codigo==codigo)
            return t;// retorna nodo encontrado
        else {
            if(codigo>t.codigo)
                t = t.der; // avanza por la derecha
            else
                t = t.izq; // avanza por la izquierda
        } // fin del else
    } // fin del while
    return null// terminó el arbol y no lo encontró
}
```

# Instrucciones en el botón Actualizar

```
private void btnActualizarActionPerformed(java.awt.event.ActionEvent evt) {
    //Colocando la informacion en el puntero pFound
    pFound.nombre=txtNombre.getText().toUpperCase();
    pFound.apellidos=txtApellidos.getText().toUpperCase();
    pFound.sexo=cbxSexo.getSelectedItem().toString();
    pFound.sueldo=Float.parseFloat(txtSueldo.getText());
    LimpiarEntradas();
    Deshabilitar();
    vaciar_tabla();
    mostrarListaArbolInOrden(raiz);
    Resumen(raiz);
}
```

# Instrucciones en el botón Eliminar

```
private void btnEliminarActionPerformed(java.awt.event.ActionEvent evt) {
    if(txtCodigo.getText().equalsIgnoreCase(""))
        JOptionPane.showConfirmDialog(this,"Ingrese un dato a eliminar por favor");
    else
    { try
        { eliminar(Integer.parseInt(txtCodigo.getText()));
            vaciar_tabla();
            mostrarListaArbolInOrden(raiz);
            Resumen(raiz);
        }
        catch (Exception e)
        { System.err.println(e);
        }
    }
}
```

## Método eliminar

```
public void eliminar(int cod) throws Exception
{ eliminar(raiz,cod);
}

private void eliminar(Nodo rz,int cod) throws Exception
{ if(rz==null) throw new Exception("Arbol de búsqueda binaria, no borrado");
  else if(cod<rz.codigo) eliminar(rz.izq,cod);
  else if(cod>rz.codigo) eliminar(rz.der,cod);
  else // nodo encontrado
  { Nodo t=rz;
    // se enlaza directamente si alguna rama esta vacia
    if(rz.izq==null) rz=rz.der;
    else if(rz.der==null) rz=rz.izq;
    else // tiene dos ramos descendientes
    { Nodo dr=rz.izq;
      Nodo anter=null;
      t=rz;
      while(dr.der!=null)
      { anter=dr;
        dr=dr.der;
      }
      //reemplaza los datos del mayorde los menores
      rz.codigo=dr.codigo;
      rz.nombre=dr.nombre;
      //enlaza nodo ascendente
      if(anter==null) // anterior a rz
        rz.izq=dr.izq;
      else if(anter.izq==dr) anter.izq=dr.izq;
      else if(anter.der==dr) anter.der=dr.izq;
    }
    t=null;
  }
}
```

## CONSTRUYA

Construya un programa que manipule una estructura de Árbol de Búsqueda Binaria (ABB) que permita registrar la información de los estudiantes como: numero de matricula, nombres, apellido paterno, apellido materno, nota de 3 practicas y su promedio, y los muestre ordenados por el numero de matricula. El programa debe tener las opciones de ver la información del ABB en-Orden, en pre-Orden y en Post-Orden, búsquedas, actualización y eliminación de nodos.

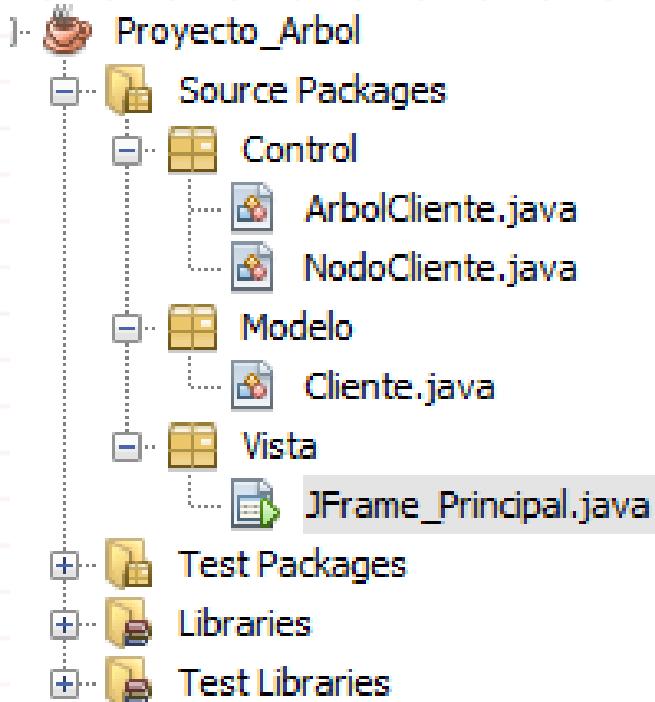
# Ejercicio N° 1

Elabore un proyecto que permita registrar dentro de una Árbol los datos (Apellidos, Nombres y Teléfono) de un grupo de clientes.  
Asimismo, deberá de permitir realizar diversas operaciones como:  
Aregar, listar, buscar y eliminar.

# SOLUCIÓN:



Paso 1: Realizar la siguiente arquitectura bajo el estándar MVC, creando un proyecto “Proyecto\_Arbol” con 3 package “Control”, “Modelo” y “Vista”



# SOLUCIÓN:

Paso 2: Dentro del package Modelo, crear la clase “Cliente”

```
1  package Modelo;  
2  
3  public class Cliente {  
4      private String Apellidos;  
5      private String Nombres;  
6      private String Telefono;  
7  
8      public Cliente(Object[] Registro){  
9          this.Apellidos = Registro[0].toString();  
10         this.Nombres = Registro[1].toString();  
11         this.Telefono = Registro[2].toString();  
12     }  
13  
14     public Object[] getRegistro(){  
15         Object[] Registro = {Apellidos, Nombres, Telefono};  
16         return Registro;  
17     }  
18  
19     public String getNomCompleto(){return Apellidos + " " + Nombres;}  
20  
21     public String getApellidos(){return Apellidos;}  
22     public void setApellidos(String Apellidos){this.Apellidos = Apellidos;}  
23  
24     public String getNombres(){return Nombres;}  
25     public void setNombres(String Nombres){this.Nombres = Nombres;}  
26  
27     public String getTelefono(){return Telefono;}  
28     public void setTelefono(String Telefono){this.Telefono = Telefono;}  
29 }  
30 }
```

# SOLUCIÓN:

Paso 3: Dentro del package Control, crear la clase “NodoCliente”

```
1  package Control;
2
3  import Modelo.Cliente;
4
5  public class NodoCliente {
6      private Cliente Elemento;
7      private NodoCliente Izq, Der;
8
9      public NodoCliente(Cliente Elemento){
10         this.Elemento = Elemento;
11         Izq = Der = null;
12     }
13
14     public NodoCliente getDer(){return Der;}
15     public void setDer(NodoCliente Der){this.Der = Der;}
16
17     public Cliente getElemento() {return Elemento;}
18
19     public void setElemento(Cliente Elemento) {this.Elemento = Elemento;}
20
21     public NodoCliente getIzq() {return Izq;}
22
23     public void setIzq(NodoCliente Izq) {this.Izq = Izq;}
24
25 }
26
```

# SOLUCIÓN:

Paso 4: Dentro del package Control, crear la clase ArbolCliente

```
1 package Control;
2
3 import Modelo.Cliente;
4 import javax.swing.table.DefaultTableModel;
5
6 public class ArbolCliente {
7     private NodoCliente Raiz;
8
9     public ArbolCliente(){
10         Raiz = null;
11     }
12
13     public NodoCliente getRaiz() {return Raiz;}
14
15     public void setRaiz(NodoCliente Raiz) {this.Raiz = Raiz;}
16
17     public NodoCliente Agregar(NodoCliente Nodo, Cliente Elemento){
18         if(Nodo == null){
19             NodoCliente Nuevo = new NodoCliente(Elemento);
20             return Nuevo;
21         }else{
22             if(Elemento.getNomCompleto().compareTo(Nodo.getElemento().getNomCompleto())> 0) Nodo.setDer(A
23                 else Nodo.setIzq(Agregar(Nodo.getIzq(), Elemento));
24         }
25         return Nodo;
26     }
27
28     public NodoCliente BuscarApeNom(String Dato){
29         NodoCliente Auxiliar = Raiz;
30         while(Auxiliar != null){
31             if(Auxiliar.getElemento().getNomCompleto().startsWith(Dato)) return Auxiliar;
32             else{
33                 if(Dato.compareTo(Auxiliar.getElemento().getNomCompleto()) > 0) Auxiliar = Auxiliar.getDe
```

# SOLUCIÓN:

Paso 5: Dentro del package Vista, crear la clase “JFrame\_Principal” que extienda javax.swing.JFrame

Title 1	Title 2	Title 3	Title 4

Apellidos:

Nombres:

Teléfono:

# SOLUCIÓN:

G1 ADA - Arbol de Clientes

Apellidos:	Ingrese sus apellidos	<input type="button" value="Nuevo"/>	
Nombres:	Ingrese sus nombres	<input type="button" value="Agregar"/>	
Telefono:	Ingrese su telefono	<input type="button" value="Cancelar"/>	
Apellido	Nombre	Telefono	<input type="button" value="Buscar"/>
			<input type="button" value="Eliminar"/>
			<input type="button" value="Salir"/>

**Pantalla inicial**

Ejecución del programa

G1 ADA - Arbol de Clientes

Apellidos:	Rojas Villanueva	<input type="button" value="Nuevo"/>	
Nombres:	Paula Elianne	<input type="button" value="Agregar"/>	
Telefono:	981600477	<input type="button" value="Cancelar"/>	
Apellido	Nombre	Telefono	<input type="button" value="Buscar"/>
			<input type="button" value="Eliminar"/>
			<input type="button" value="Salir"/>

**Agregar Cliente**

# SOLUCIÓN:

## Ejecución del programa

G1 ADA - Arbol de Clientes

Apellidos:	<input type="text"/>	<input type="button" value="Nuevo"/>	
Nombres:	<input type="text"/>	<input type="button" value="Agregar"/>	
Telefono:	<input type="text"/>	<input type="button" value="Cancelar"/>	
Apellido	Nombre	Telefono	
Rojas Villanueva	Paula Elianne	981600477	<input type="button" value="Buscar"/>
			<input type="button" value="Eliminar"/>
			<input type="button" value="Salir"/>

**Cliente Agregado**

G1 ADA - Arbol de Clientes

Apellidos:	<input type="text"/>	<input type="button" value="Nuevo"/>	
Nombres:	<input type="text"/>	<input type="button" value="Agregar"/>	
Telefono:	<input type="text"/>	<input type="button" value="Cancelar"/>	
Apellido	Nombre	Telefono	
Rojas Villanueva	Paula Elianne	981600477	<input type="button" value="Buscar"/>
Romero Angeles	Luis Alfredo	928471523	<input type="button" value="Eliminar"/>
Torres Talaver...	Luz Elena	987651332	<input type="button" value="Salir"/>
Zafra Moran	Rolando Jesús	946323538	
Zárate Villar	Jhennyfer	912354267	

**Clients Agregados**

# SOLUCIÓN:

## Ejecución del programa

**Buscar Cliente**

Entrada X

Apellido a buscar:

?

Aceptar Cancelar

G1 ADA - Arbol de Clientes - X

Apellidos: Rojas Villanueva Nuevo

Nombres: Paula Elianne Agregar

Teléfono: 981600477 Cancelar

Apellido	Nombre	Teléfono
Rojas Villanueva	Paula Elianne	981600477
Romero Angeles	Luis Alfredo	928471523
Torres Talaver...	Luz Elena	987651332
Zafra Moran	Rolando Jesús	946323538
Zárate Villar	Jhennyfer	912354267

Buscar Eliminar Salir

# SOLUCIÓN:

## Ejecución del programa

G1 ADA - Arbol de Clientes

Apellidos:	Rojas Villanueva	Nuevo
Nombres:	Paula Elianne	Agregar
Telefono:	981600477	Cancelar

Apellido	Nombre	Telefono
Rojas Villanueva	Paula Elianne	981600477
Romero Angeles	Luis Alfredo	928471523
Torres Talaver...	Luz Elena	987651332
Zafra Moran	Rolando Jesús	946323538
Zárate Villar	Jhennyfer	912354267

**Eliminar Cliente**

G1 ADA - Arbol de Clientes

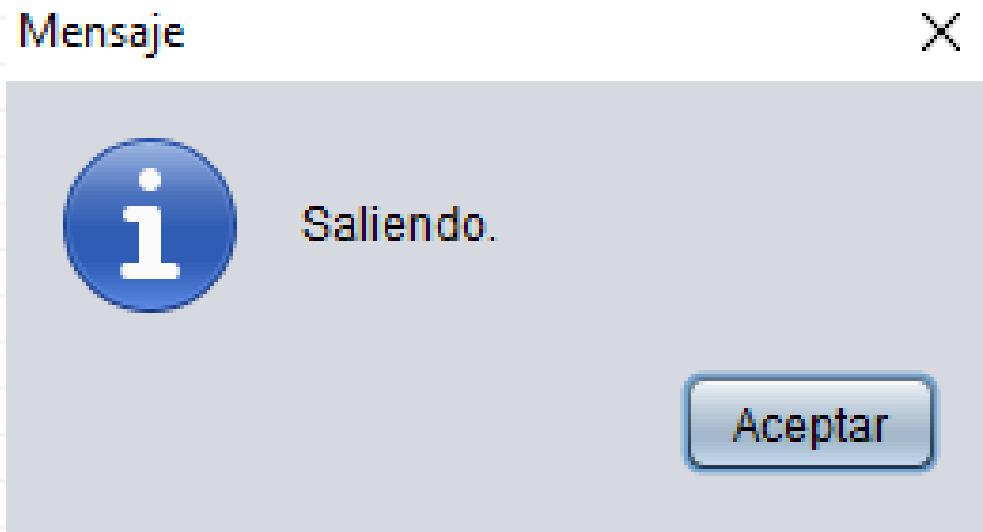
Apellidos:		Nuevo
Nombres:		Agregar
Telefono:		Cancelar

Apellido	Nombre	Telefono
Romero Angeles	Luis Alfredo	928471523
Torres Talaver...	Luz Elena	987651332
Zafra Moran	Rolando Jesús	946323538
Zárate Villar	Jhennyfer	912354267

**Cliente Eliminado**

# SOLUCIÓN:

Ejecución del programa



Salir programa

# Ejercicio N° 2

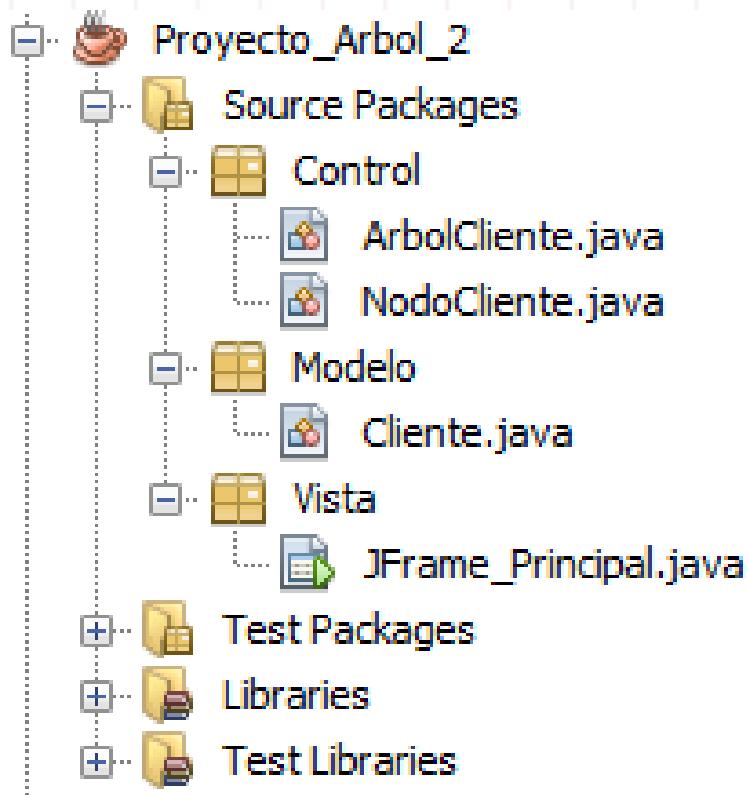
Elabore un proyecto que permita registrar dentro de una Árbol los datos (Código, Apellidos y Nombres) de un grupo de Contactos. El registro tomará como referencia al código del contacto y no deberá de permitir códigos repetidos.

Asimismo, deberá de permitir realizar diversas operaciones como:Buscar, Eliminar y Listar.

# SOLUCIÓN:



Paso 1: Realizar la siguiente arquitectura bajo el estándar MVC, creando un proyecto “Proyecto\_Arbol2” con 3 package “Control”, “Modelo” y “Vista”



# SOLUCIÓN:

Paso 2: Dentro del package Modelo, crear la clase “Cliente”

```
1 package Modelo;  
2  
3 public class Cliente {  
4     private String Código;  
5     private String Nombres;  
6     private String Apellidos;  
7  
8     public Cliente(Object[] Registro){  
9         this.Código = Registro[0].toString();  
10        this.Nombres = Registro[1].toString();  
11        this.Apellidos = Registro[2].toString();  
12    }  
13  
14    public Object[] getRegistro(){  
15        Object[] Registro = {Código, Nombres, Apellidos};  
16        return Registro;  
17    }  
18  
19    public String getApellidos(){return Apellidos;}  
20    public void setApellidos(String Apellidos){this.Apellidos = Apellidos;}  
21  
22    public String getNombres(){return Nombres;}  
23    public void setNombres(String Nombres){this.Nombres = Nombres;}  
24  
25    public String getCódigo(){return Código;}  
26    public void setCódigo(String Código){this.Código = Código;}  
27}  
28
```

# SOLUCIÓN:

Paso 3: Dentro del package Control, crear la clase “NodoCliente”

```
1  package Control;
2
3  import Modelo.Cliente;
4
5  public class NodoCliente {
6      private Cliente Elemento;
7      private NodoCliente Izq, Der;
8
9      public NodoCliente(Cliente Elemento) {
10         this.Elemento = Elemento;
11         Izq = Der = null;
12     }
13
14     public NodoCliente getDer(){return Der;}
15     public void setDer(NodoCliente Der){this.Der = Der;}
16
17     public Cliente getElemento() {return Elemento;}
18
19     public void setElemento(Cliente Elemento) {this.Elemento = Elemento;}
20
21     public NodoCliente getIzq() {return Izq;}
22
23     public void setIzq(NodoCliente Izq) {this.Izq = Izq;}
24
25 }
26 |
```

# SOLUCIÓN:

Paso 4: Dentro del package Control, crear la clase ArbolCliente

```
1  package Control;
2
3  import Modelo.Cliente;
4  import javax.swing.table.DefaultTableModel;
5
6  public class ArbolCliente {
7      private NodoCliente Raiz;
8
9      public ArbolCliente(){
10         Raiz = null;
11     }
12
13     public NodoCliente getRaiz() {return Raiz;}
14
15     public void setRaiz(NodoCliente Raiz) {this.Raiz = Raiz;}
16
17     public NodoCliente Agregar(NodoCliente Nodo, Cliente Elemento){
18         if(Nodo == null){
19             NodoCliente Nuevo = new NodoCliente(Elemento);
20             return Nuevo;
21         }else{
22             if(Elemento.getCodigo().compareTo(Nodo.getElemento().getCodigo())> 0) Nodo.setDer(Agregar(Nodo.getDer(), Elemento));
23             else Nodo.setIzq(Agregar(Nodo.getIzq(), Elemento));
24         }
25         return Nodo;
26     }
}
```

# SOLUCIÓN:

En la clase ArbolCliente, crear el metodo Buscar Código que recibe como parámetro el dato para comprobar si coincide con el código del cliente buscado

```
28 [-]     public NodoCliente BuscarCodigo(String Dato){  
29         NodoCliente Auxiliar = Raiz;  
30         while(Auxiliar != null){  
31             if(Auxiliar.getElemento().getCodigo().startsWith(Dato)) return Auxiliar;  
32             else{  
33                 if(Dato.compareTo(Auxiliar.getElemento().getCodigo()) > 0) Auxiliar = Auxiliar.getDer();  
34                 else Auxiliar = Auxiliar.getIzq();  
35             }  
36         }  
37         return null;  
38     }
```

# SOLUCIÓN

En la clase ArbolCliente, crear el metodo Buscar Código, ListarInOrden, BuscarMayorIzquierda, EliminarMayorIzquierda

```
40     public void ListarInOrden(NodoCliente Nodo, DefaultTableModel modTabla){  
41         if(Nodo != null){  
42             ListarInOrden(Nodo.getIzq(), modTabla);  
43             modTabla.addRow(Nodo.getElemento().getRegistro());  
44             ListarInOrden(Nodo.getDer(), modTabla);  
45         }  
46     }  
47  
48     public NodoCliente BuscarMayorIzquierda(NodoCliente Auxiliar){  
49         if(Auxiliar != null)  
50             while(Auxiliar.getDer() != null)  
51                 Auxiliar = Auxiliar.getDer();  
52             return Auxiliar;  
53     }  
54  
55     public NodoCliente EliminarMayorIzquierda(NodoCliente Auxiliar){  
56         if(Auxiliar == null) return null;  
57         else if(Auxiliar.getDer() != null){  
58             Auxiliar.setDer(EliminarMayorIzquierda(Auxiliar.getDer()));  
59             return Auxiliar;  
60         }  
61         return Auxiliar.getIzq();  
62     }
```

# SOLUCIÓN:

En la clase ArbolCliente, crear el metodo Eliminar que recibe como parametro el codigo y nodo que lo apunta para ser eliminado

```
64     public NodoCliente Eliminar(NodoCliente Auxiliar, String Dato){  
65         if(Auxiliar == null) return null;  
66         if(Dato.compareTo(Auxiliar.getElemento().getCodigo()) < 0) Auxiliar.setIzq(Eliminar(Auxiliar.getIzq(),  
67                                         Dato));  
68         else if(Dato.compareTo(Auxiliar.getElemento().getCodigo()) > 0) Auxiliar.setDer(Eliminar(Auxiliar.getDer(),  
69                                         Dato));  
70         else if (Auxiliar.getIzq() != null && Auxiliar.getDer() != null){  
71             Auxiliar.setElemento(BuscarMayorIzquierda(Auxiliar.getIzq()).getElemento());  
72             Auxiliar.setIzq(EliminarMayorIzquierda(Auxiliar.getIzq()));  
73         }  
74         else Auxiliar = (Auxiliar.getIzq() != null) ? Auxiliar.getIzq() : Auxiliar.getDer();  
75         return Auxiliar;  
76     }
```

# SOLUCIÓN:

Paso 5: Dentro del package Vista, crear la clase “JFrame\_Principal” que extiende javax.swing.JFrame

The diagram illustrates a Java Swing application window. At the top, there are three text input fields labeled "Codigo:", "Nombres:", and "Apellidos:". Below these is a table with four columns labeled "Title 1", "Title 2", "Title 3", and "Title 4". To the right of the table are four buttons: "Agregar", "Buscar", "Eliminar", and "Salir".

Title 1	Title 2	Title 3	Title 4

Codigo: \_\_\_\_\_

Nombres: \_\_\_\_\_

Apellidos: \_\_\_\_\_

Agregar

Buscar

Eliminar

Salir

# SOLUCIÓN:

G1 ADA - Registro

Codigo:	<input type="text"/>
Nombres:	<input type="text"/>
Apellidos:	<input type="text"/>

Codigo	Nombres	Apellidos

**Agregar**  
**Buscar**  
**Eliminar**

Pantalla inicial



## Ejecución del programa

G1 ADA - Registro

Codigo:	<input type="text" value="19200266"/>
Nombres:	<input type="text" value="Paula Elianne"/>
Apellidos:	<input type="text" value="Rojas Villanueva"/>

Codigo	Nombres	Apellidos

**Agregar**  
**Buscar**  
**Eliminar**

Registrar Cliente



# SOLUCIÓN:

## Ejecución del programa

G1 ADA - Registro

Codigo:	<input type="text"/>
Nombres:	<input type="text"/>
Apellidos:	<input type="text"/>

Codigo	Nombres	Apellidos
19200266	Paula Elianne	Rojas Villanu...

**Agregar**

**Buscar**

**Eliminar**

**Cliente Registrado**

G1 ADA - Registro

Codigo:	<input type="text"/>
Nombres:	<input type="text"/>
Apellidos:	<input type="text"/>

Codigo	Nombres	Apellidos
19200121	Rolando Jesus	Zafra Moran
19200213	Luz Elena	Torres Talaverano
19200231	Rodrigo Seba...	Vega Centeno
19200266	Paula Elianne	Rojas Villanueva
19200323	Jennyfer Nayeli	Zarate Villar
19200345	Luis Alfredo	Romero Angeles

**Agregar**

**Buscar**

**Eliminar**

**Clients Registrados**

# SOLUCIÓN:

## Ejecución del programa

G1 ADA - Registro

Codigo:	19200266
Nombres:	Paula
Apellidos:	Rojas

Codigo	Nombres	Apellidos
19200121	Rolando Jesus	Zafra Moran
19200213	Luz Elena	Torres Talaverano
19200231	Rodrigo Seba...	Vega Centeno
19200266	Paula Eianne	Rojas Villanueva
19200323	Jennyfer Nayeli	Zarate Villar
19200345	Luis Alfredo	Romero Angeles

**Registrar Cliente con mismo código**

Agregar      Buscar      Eliminar

G1 ADA - Registro

Codigo:	19200266
Nombres:	Paula
Apellidos:	Rojas

Mensaje

**Código duplicado**

i

Codigo
19200121
19200213
19200231
19200266
19200323
19200345

Luis Alfredo      Romero Angeles

Aceptar      Buscar      Eliminar

**Mensaje de código ya registrado**

# SOLUCIÓN:

G1 ADA - Registro

Codigo:	<input type="text"/>
Nombres:	<input type="text"/>
Apellidos:	<input type="text"/>

Codigo	Nombres	Apellidos
19200121	Rolando Jesus	Zafra Moran
19200213	Luz Elena	Torres Talaverano
19200231	Rodrigo Seba...	Vega Centeno
19200266	Paula Elianne	Rojas Villanueva
19200323	Jennyfer Nayeli	Zarate Villar
19200345	Luis Alfredo	Romero Angeles

**Agregar**    **Buscar**    **Eliminar**

**Eliminar Cliente**

Ejecución del programa

G1 ADA - Registro

Codigo:	<input type="text"/>
Nombres:	<input type="text"/>
Apellidos:	<input type="text"/>

Codigo	Nombres	Apellidos
19200121	Rolando Jesus	Zafra Moran
19200213	Luz Elena	Torres Talaverano
19200231	Rodrigo Seba...	Vega Centeno
19200323	Jennyfer Nayeli	Zarate Villar
19200345	Luis Alfredo	Romero Angeles

**Agregar**    **Buscar**    **Eliminar**

**Cliente Eliminado**

# SOLUCIÓN:

Ejecución del programa

Mensaje

X



Saliendo.

Aceptar

Salir programa



Universidad Nacional Mayor de San Marcos  
Universidad del Perú. Decana de América

# Resumiendo y Repasando...

- Un árbol binario es una estructura enlazada en la que cada nodo se refiere a otros dos nodos. Un árbol está vacío (nulo) o es un nodo con referencias a otros dos árboles, llamados subárboles.
- La raíz es el nodo superior de un árbol. Un nodo hoja es un nodo que no tiene hijos. Un nodo de rama es un nodo que tiene al menos un hijo. Cada nodo tiene un nivel relacionado con el número de enlaces que hay entre él y la raíz del árbol.
- Un objeto de nodo de árbol almacena un valor de datos y referencias izquierda/derecha a otros nodos. Un objeto de árbol general almacena una referencia a un solo nodo de árbol como su raíz general.

# Resumiendo y Repasando...

- Un recorrido es un examen de todos los elementos de un árbol binario. Los recorridos se realizan comúnmente en tres órdenes: Pre-order, In-order y Post-order. La diferencia se relaciona con la forma de examinar un nodo determinado en relación con sus subárboles izquierdo/derecho.
- Los métodos de árbol suelen ser recursivos y suelen implementarse con el uso de un par público/privado, en el que el método privado acepta una referencia a un nodo de árbol como parámetro. Esta técnica permite que el método opere recursivamente en una porción dada del árbol total.
- Los elementos de un árbol de búsqueda binaria se organizan en orden, con los elementos más pequeños a la izquierda y los elementos más grandes a la derecha. Los árboles de búsqueda son útiles para implementar colecciones que se pueden buscar rápidamente, como conjuntos y mapas.

# Resumiendo y Repasando...

- Para agregar un nodo a un árbol binario, recorra el árbol hacia la izquierda si el nuevo elemento es más pequeño que el nodo actual o hacia la derecha si es más grande. El final de este recorrido revelará el lugar adecuado para agregar el nuevo nodo.
- Para buscar un valor en un árbol binario, recorra el árbol hacia la izquierda si el valor objetivo es más pequeño que el nodo actual o hacia la derecha si es más grande. Debido a la forma en que se ordena el árbol, el final de este recorrido encontrará el nodo de destino o encontrará un callejón sin salida nulo, en cuyo caso el valor no está en el árbol.



Universidad Nacional Mayor de San Marcos  
Universidad del Perú. Decana de América