

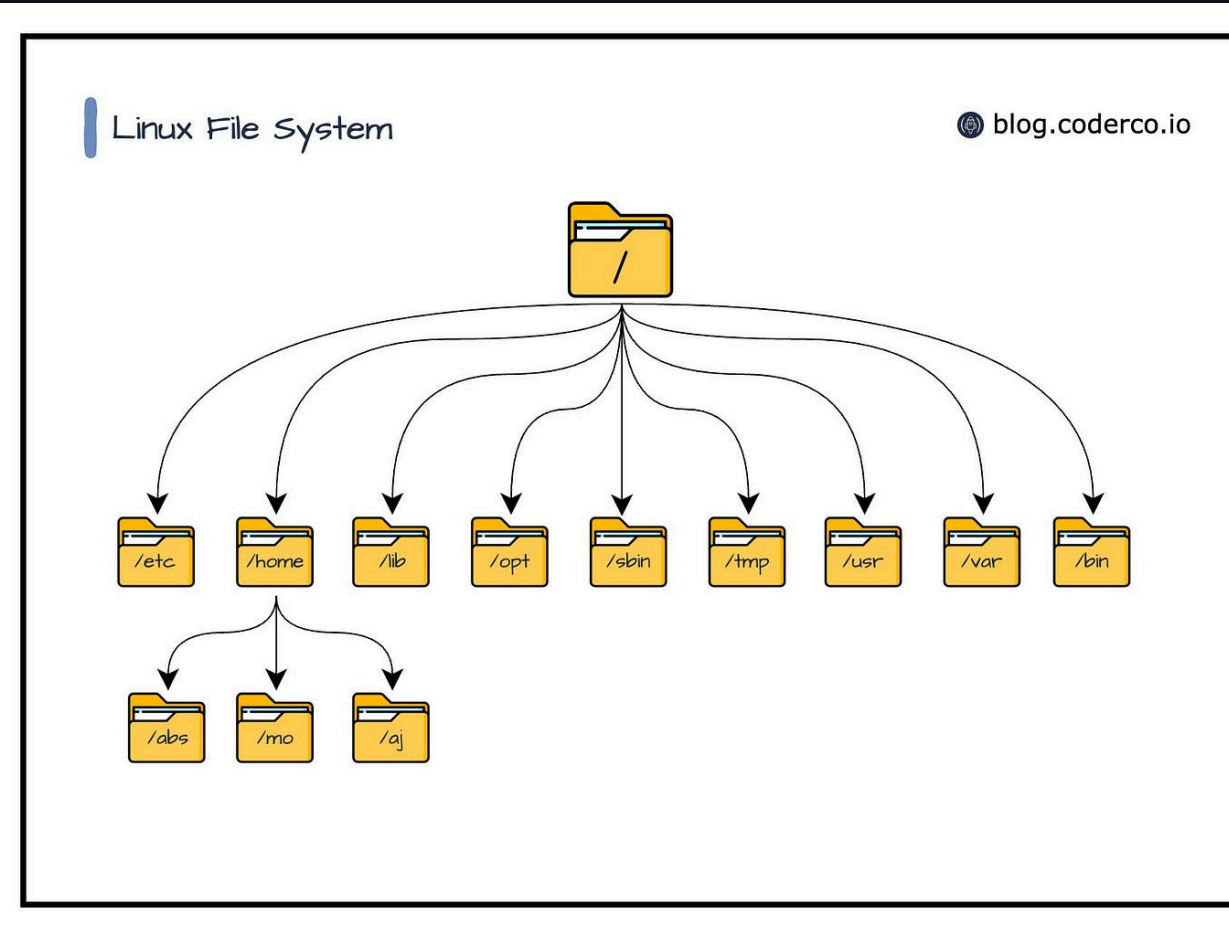
# Estructura de datos y algoritmos

Rodrigo Alvarez

[rodrigo.alvarez2@mail.udp.cl](mailto:rodrigo.alvarez2@mail.udp.cl)

# Árboles

- Son una estructura de datos no lineal, llamadas estructuras jerárquicas
- Son las estructuras no lineales más utilizadas para resolver problemas de software como:
  - Árboles de directorios
  - Toma de decisiones
  - Organización de información en bases de datos
  - etc

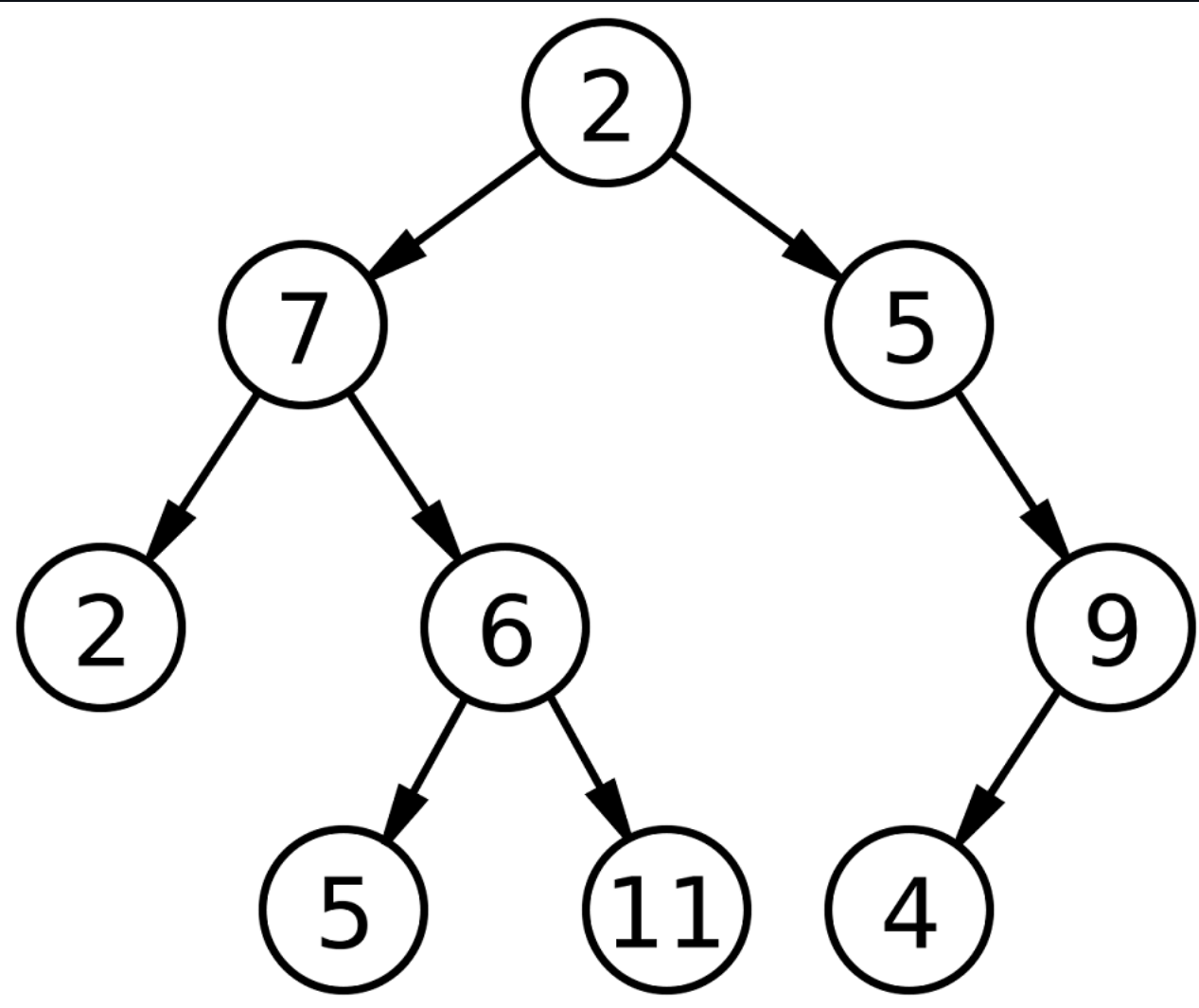


# Árboles: definiciones

- Un Árbol consiste en un nodo **r** denominado nodo raíz y una lista o conjunto de subárboles ( $A_1, A_2, A_3, \dots A_n$ )
- Se definen como nodos hijos de **r** a los nodos raíces de los subárboles  $A_1, A_2, A_3, \dots A_n$
- Si **b** es un nodo hijo de **a** entonces **a** es el nodo padre de **b**
- Un nodo puede tener cero o más hijos, y uno o ningún padre. El único nodo que no tiene padre es el nodo raíz del árbol

## Árboles: definiciones

- **Subárbol:** Un árbol que es parte de otro árbol
- **Profundidad de un nodo:** Número de aristas que hay desde la raíz hasta el nodo
- **Altura de un nodo:** Número de aristas que hay desde el nodo hasta el nodo más lejano



## Árbol binario

- Es un árbol en el que cada nodo tiene a lo más dos hijos
- Cada nodo tiene un nodo padre, excepto la raíz

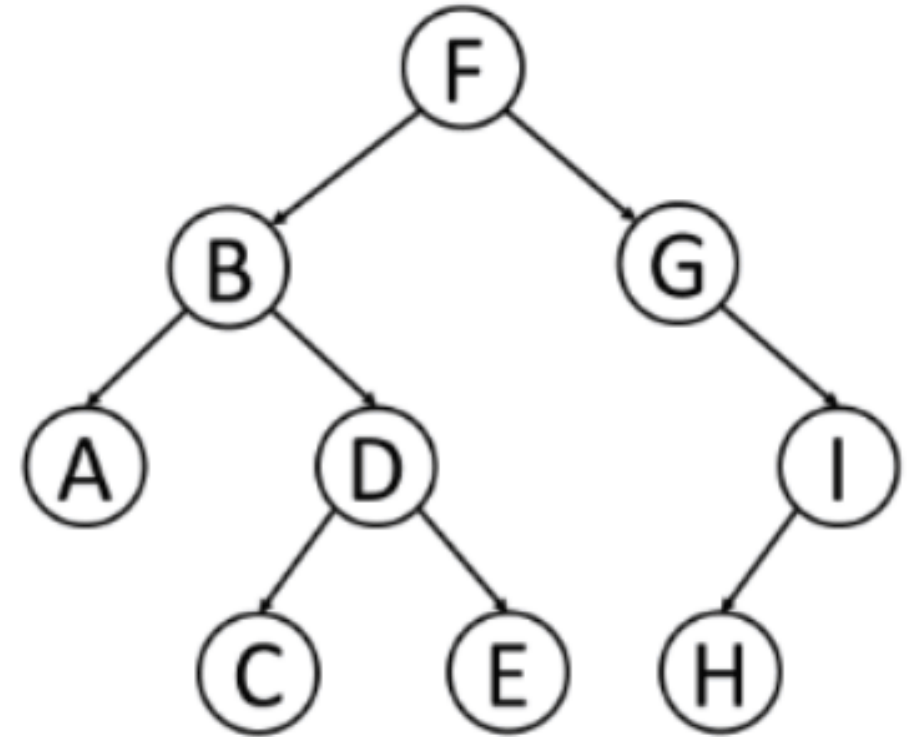
# Nodo

```
class Node {  
    int data;  
    Node left, right;  
    Node(int item) {  
        data = item;  
        left = right = null;  
    }  
}
```

# Recorridos (tree traversals)

## Recorrido Inorder

En este recorrido, de manera **recursiva**, primero se visita el **subárbol izquierdo**, luego la **raíz** y finalmente el **subárbol derecho**



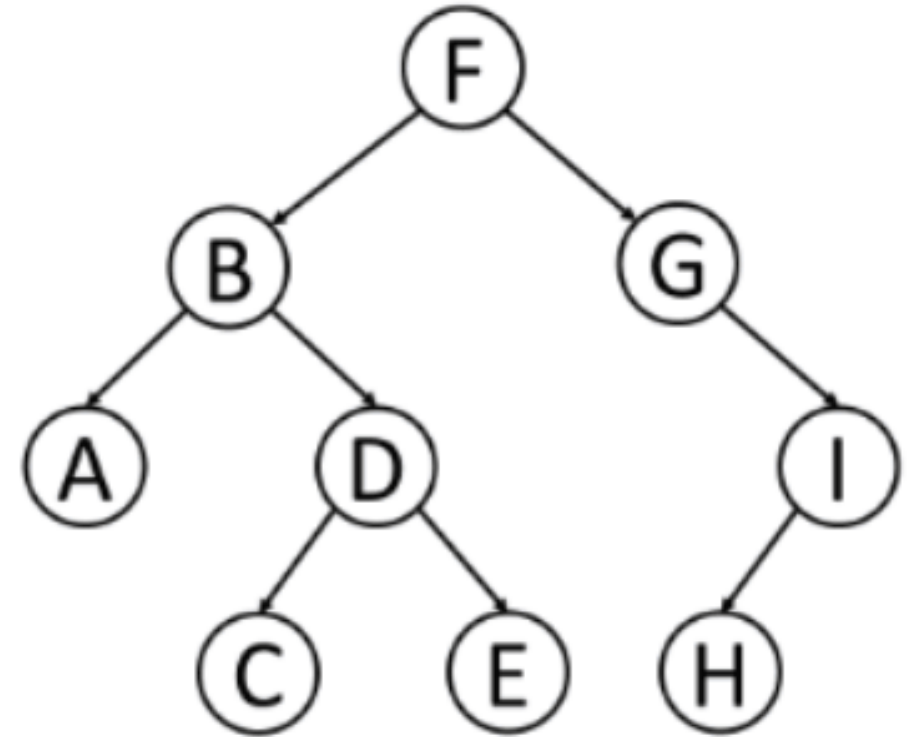
Inorder:

|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|



## Recorrido Inorder

En este recorrido primero se visita el subárbol izquierdo, luego la raíz y finalmente el subárbol derecho



Inorder:

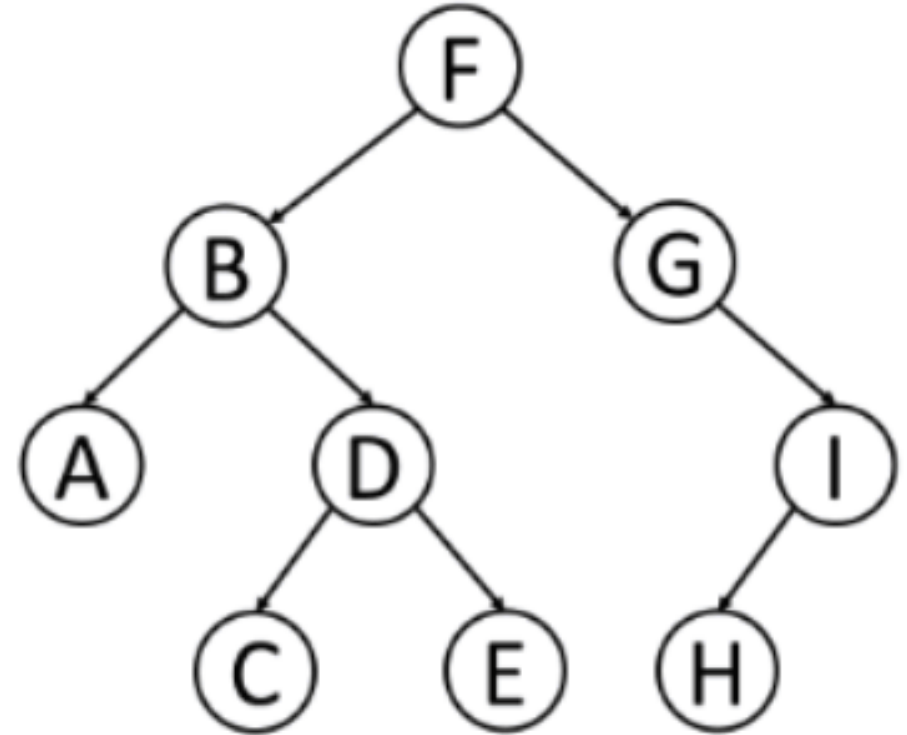
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|

## Recorrido Inorder

```
void inorder(Node node) {  
    if (node == null) return;  
    inorder(node.left);  
    System.out.print(node.data + " ");  
    inorder(node.right);  
}
```

## Recorrido Preorder

En este recorrido primero se visita la raíz, luego el subárbol izquierdo y finalmente el subárbol derecho

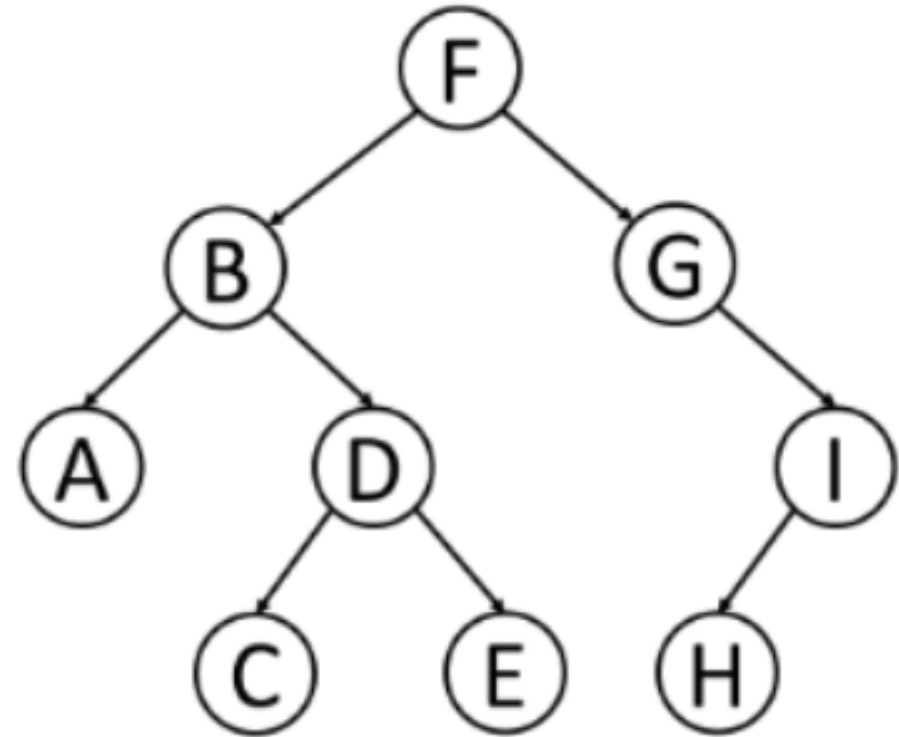


Preorder:

|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|

## Recorrido Preorder

En este recorrido primero se visita la raíz, luego el subárbol izquierdo y finalmente el subárbol derecho



Preorder:

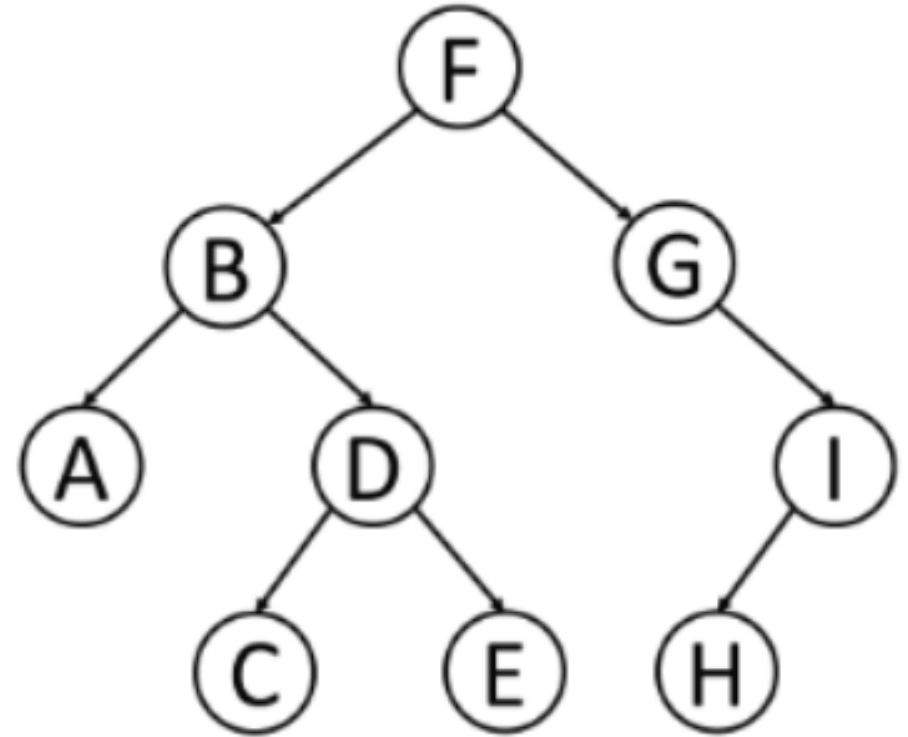
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|

## Recorrido Preorder

```
void preorder(Node node) {  
    if (node == null) return;  
    System.out.print(node.data + " ");  
    preorder(node.left);  
    preorder(node.right);  
}
```

## Recorrido Postorder

En este recorrido primero se visita el subárbol izquierdo, luego el subárbol derecho y finalmente la raíz

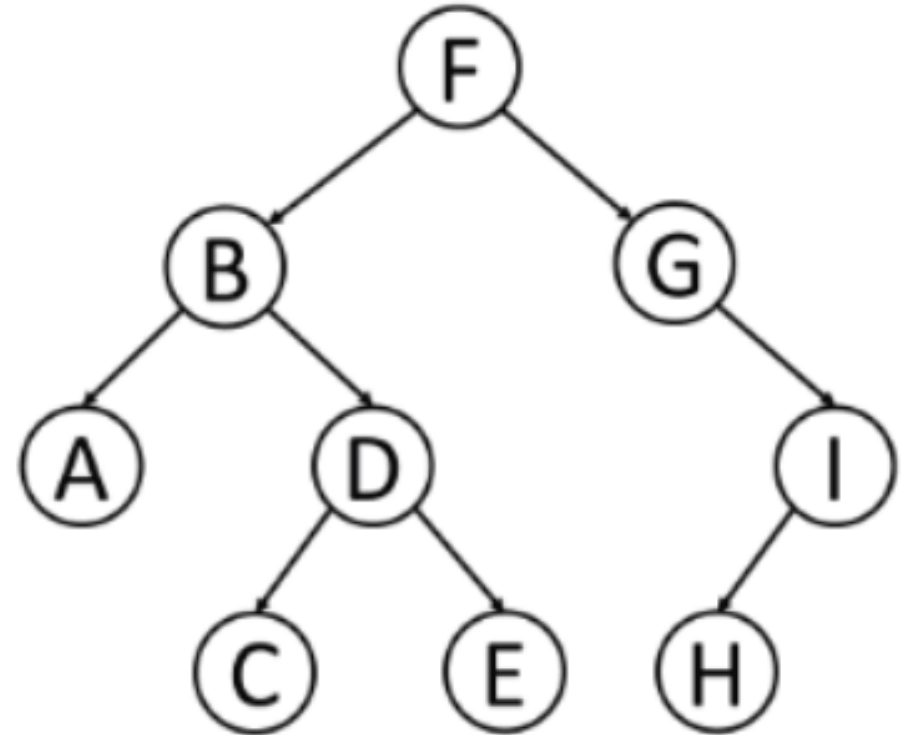


Postorder:

|  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|

## Recorrido Postorder

En este recorrido primero se visita el subárbol izquierdo, luego el subárbol derecho y finalmente la raíz



Postorder:

|  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|
|  |  |  |  |  |  |  |  |  |  |
|--|--|--|--|--|--|--|--|--|--|

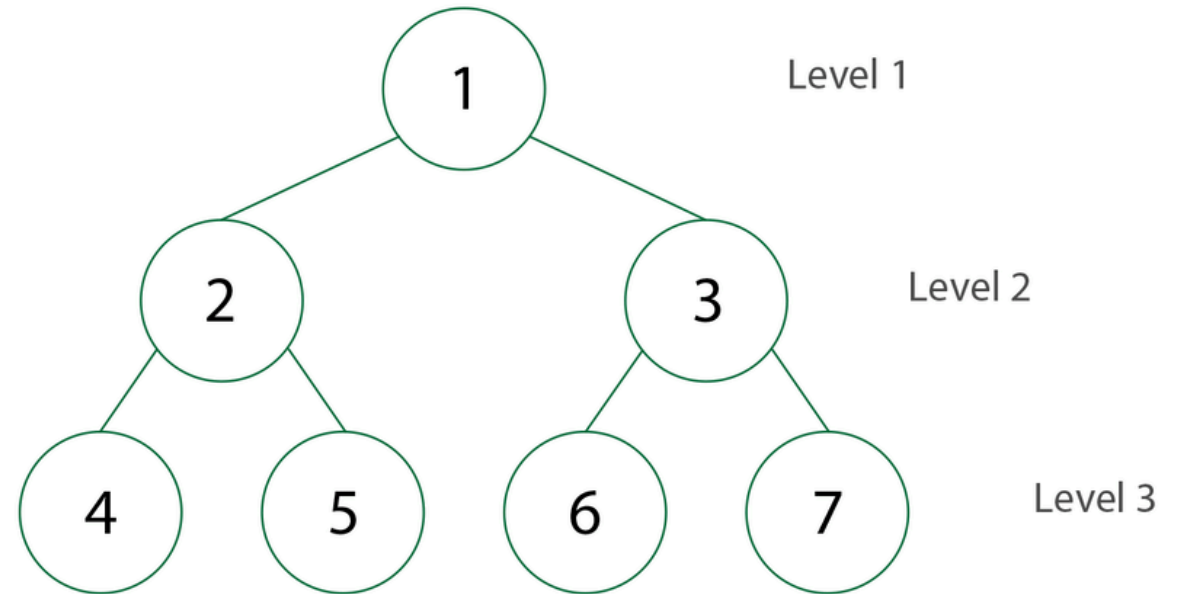
## Recorrido Postorder

```
void postorder(Node node) {  
    if (node == null) return;  
    postorder(node.left);  
    postorder(node.right);  
    System.out.print(node.data + " ");  
}
```



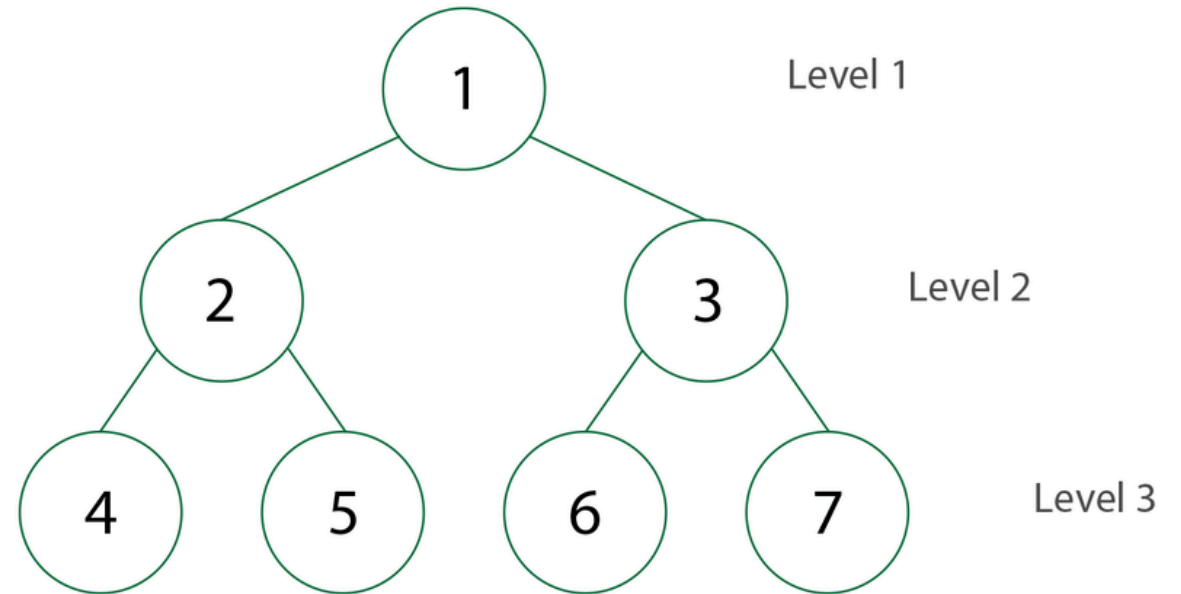
# Recorrido por niveles

En este recorrido se recorre el árbol por niveles



# Recorrido por niveles

En este recorrido se recorre el árbol por niveles



## Recorrido por niveles

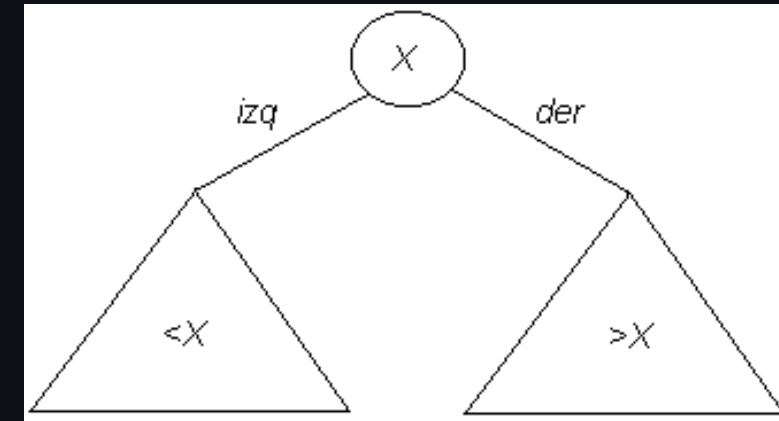
```
void byLevelTraversal(Node root) {  
    if (root == null) return;  
    Queue<Node> q = new LinkedList<>();  
    q.add(root);  
    while (!q.isEmpty()) {  
        Node node = q.poll();  
        System.out.print(node.data + " ");  
        if (node.left != null) q.add(node.left);  
        if (node.right != null) q.add(node.right);  
    }  
}
```

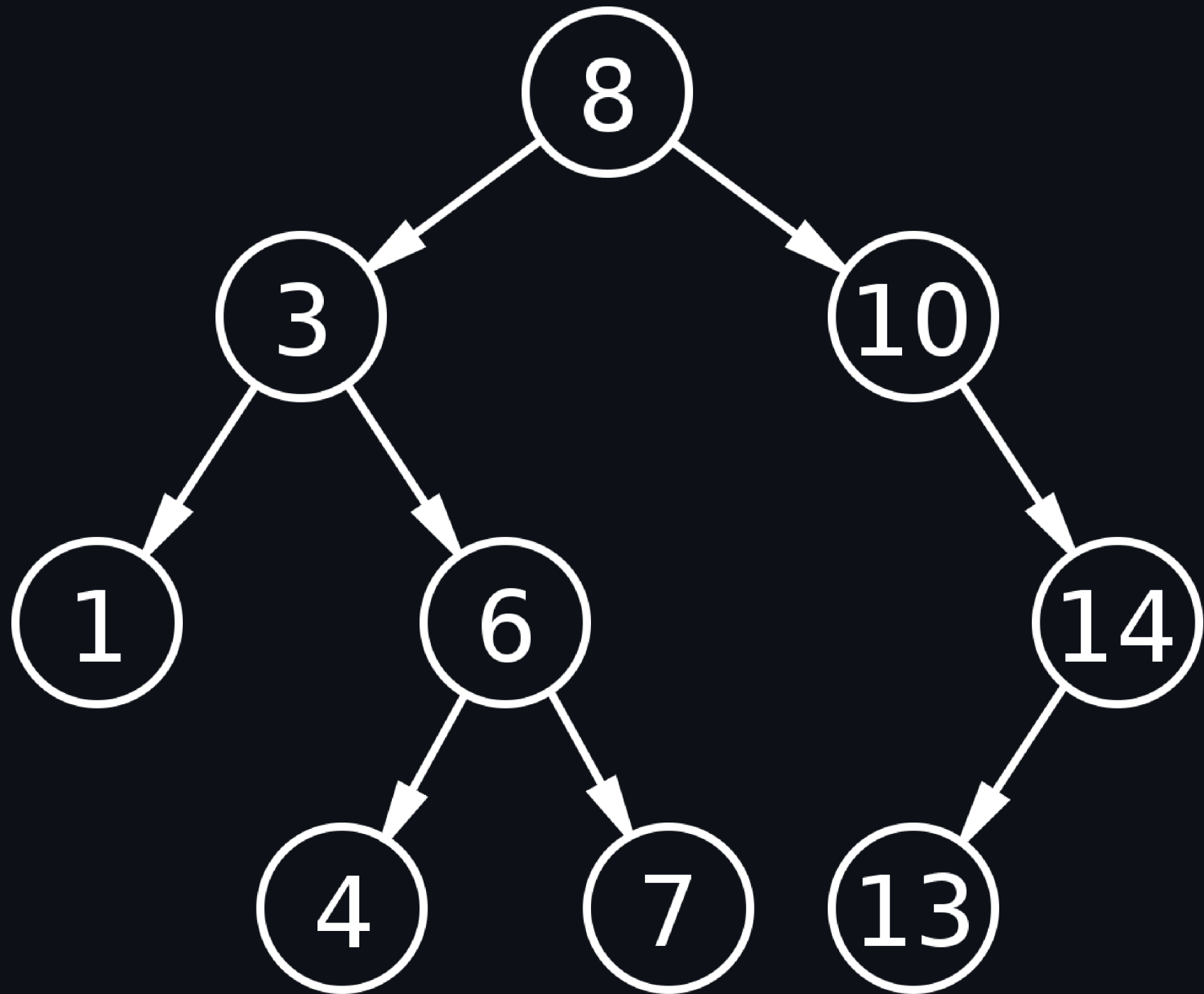
# Árbol binario de búsqueda (BST)

Sea  $A$  un árbol binario de raíz  $R$  e hijos izquierdo y derecho (posiblemente nulos)  $H_I$  y  $H_D$ , respectivamente.

Decimos que  $A$  es un árbol binario de búsqueda (BST en inglés) si y solo si se satisfacen las dos condiciones al mismo tiempo:

- $H_I$  es un BST con todos los elementos menores que  $R \vee H_I$  es un árbol vacío
- $H_D$  es un BST con todos los elementos mayores que  $R \vee H_D$  es un árbol vacío





# Árbol binario de búsqueda (BST): operaciones

- **Búsqueda:** Buscar un elemento en el árbol
- **Inserción:** Insertar un elemento en el árbol
- **Eliminación:** Eliminar un elemento del árbol

- implementación de un BST
- MIT Binary trees
  - parte 1
  - parte 2
- árboles binarios
- árboles binarios de búsqueda (bst)