

# **Estructura de datos avanzadas**

# **ARBOLES BINARIOS**



Por: Ing. Edwin Calle Terrazas

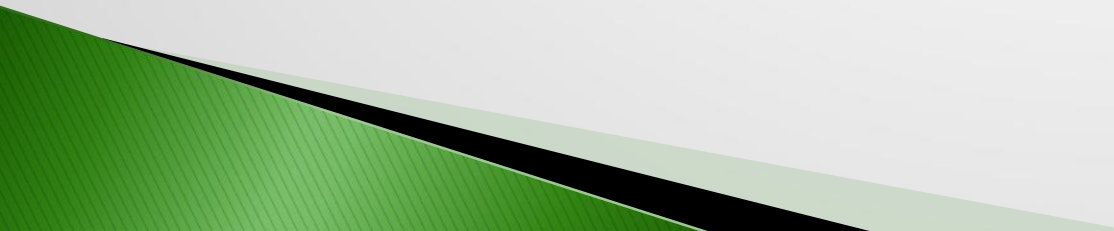
## **INTRODUCCIÓN**

Hasta el momento hemos estudiando estructuras de datos lineales, en el cual para el caso de las listas, las inserciones y búsquedas requieren navegar las listas secuencialmente.

Los arboles son estructuras no lineales, donde se pueden obtener métodos para manejar elementos ordenados de manera rápida como también realizar búsquedas rápidas de sus elementos.

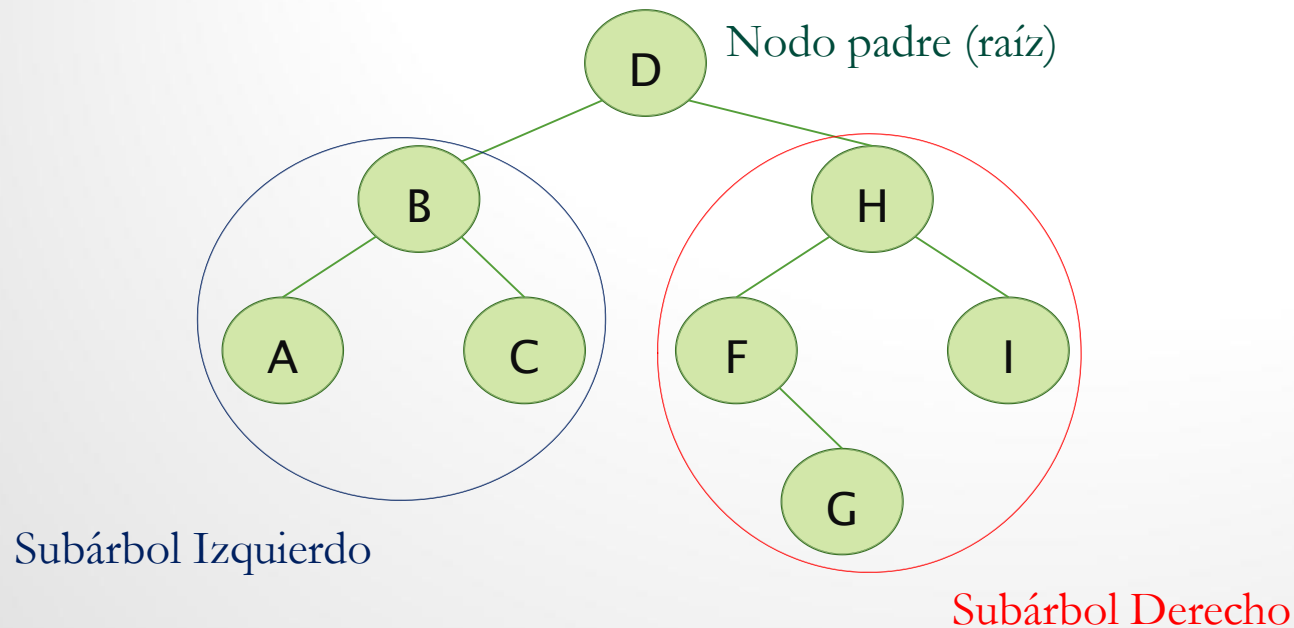
## **TERMINOLOGÍA**

La terminología es sacada de la botánica y de la genealogía y los términos se aplican indistintamente.



# ÁRBOLES BINARIOS

Un árbol binario se define, como un conjunto de nodos en el cual existe un nodo especial llamado raíz y tiene 2 subconjuntos disjuntos, llamados subárbol izquierdo y subárbol derecho.



# RECORRIDO SOBRE ÁRBOLES BINARIOS

Existen 3 recorridos principales:

## **PreOrden**

Se procesa el elemento del nodo padre (raíz)

Se recorre el subárbol izquierdo en preOrden

Se recorre el subárbol derecho en preOrden

## **InOrden**

Se recorre el subárbol izquierdo en InOrden

Se procesa el elemento del nodo padre (raíz)

Se recorre el subárbol derecho en InOrden

## **PostOrden**

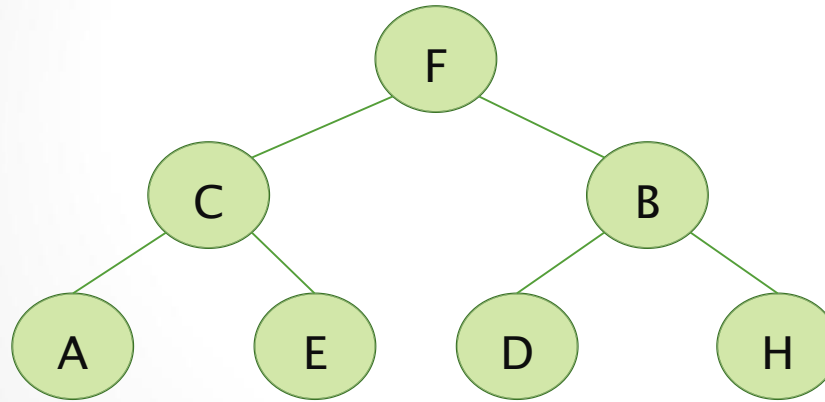
Se recorre el subárbol izquierdo en postOrden

Se recorre el subárbol derecho en postOrden

Se procesa el elemento del nodo padre (raíz)



Dado el siguiente árbol binario (no ordenado), encontrar sus recorridos.

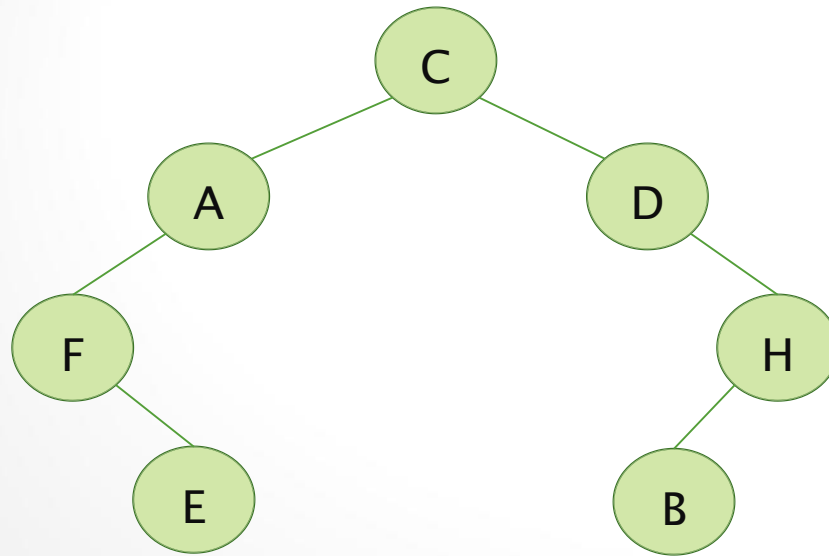


PreOrden: F C A E B D H

InOrden: A C E F D B H

PostOrden: A E C D H B F

Dado el siguiente árbol binario (no ordenado), encontrar sus recorridos.



PreOrden: C A F E D H B

InOrden: F E A C D B H

PostOrden: E F A B H D C

# RECONSTRUCCIÓN DE UN ARBOL A PARTIR DE SUS RECORRIDOS

Es posible reconstruir un árbol a partir de dos de sus recorridos. Uno de los recorridos debe ser el InOrden.

## EJEMPLO

Reconstruir el árbol conociendo los recorridos:

InOrden: B D A C E

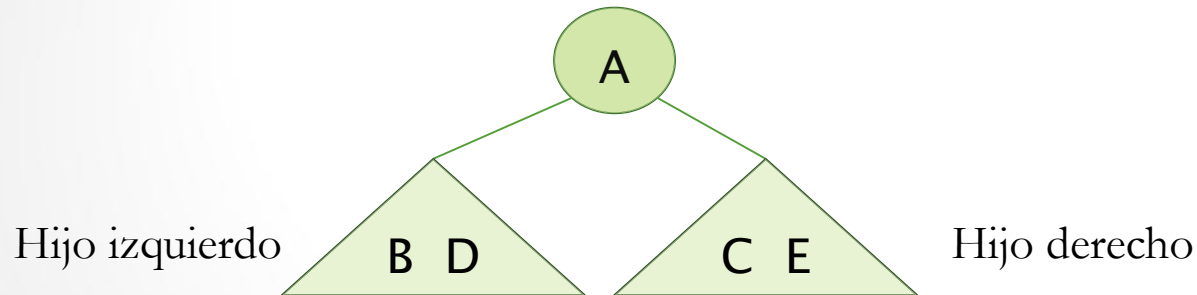
PreOrden: A B D C E

### Solución

Con el recorrido preOrden, podemos conocer la raíz, pues es el primero de la lista (A)

Luego identificamos la A, en el recorrido inOrden y los elementos que están a su izquierda son los elementos del hijo izquierdo y los que están a la derecha de la A, son los elementos del hijo derecho.

InOrden : B D A C E  
PreOrden : A B D C E



Para resolver el hijo izquierdo. Se identifica su secuencia inOrden (B,D), en el recorrido preOrden

PreOrden : A B D C E

De aquí se determina que B es la raíz en el hijo izquierdo

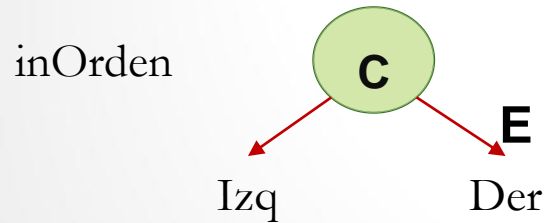




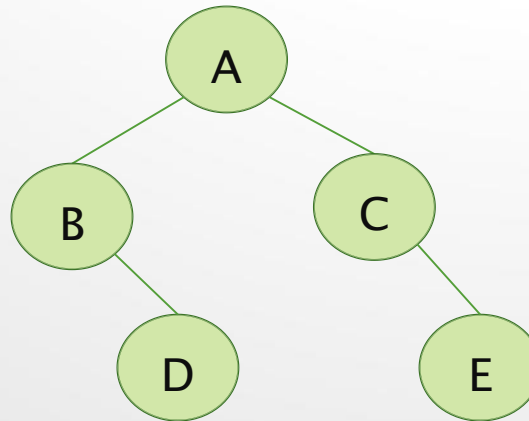
Procedemos de la misma forma para el hijo derecho

PreOrden : A B D **C E**

De aquí se determina que C es la raíz, del hijo derecho



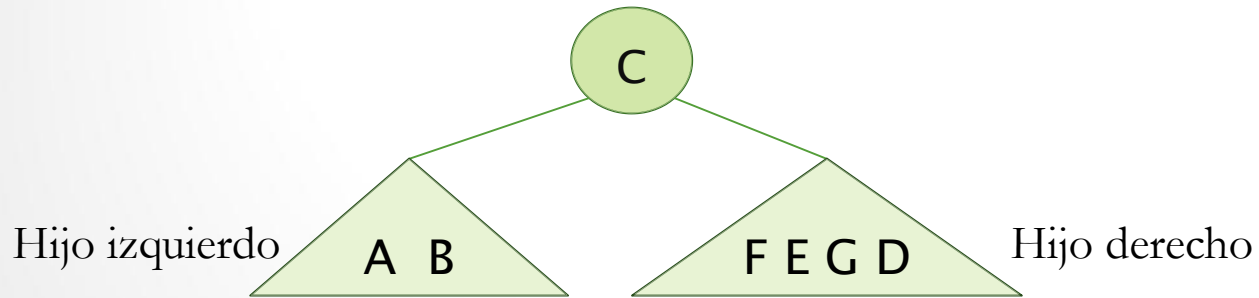
El árbol reconstruido es:



Reconstruir el árbol conociendo los recorridos:

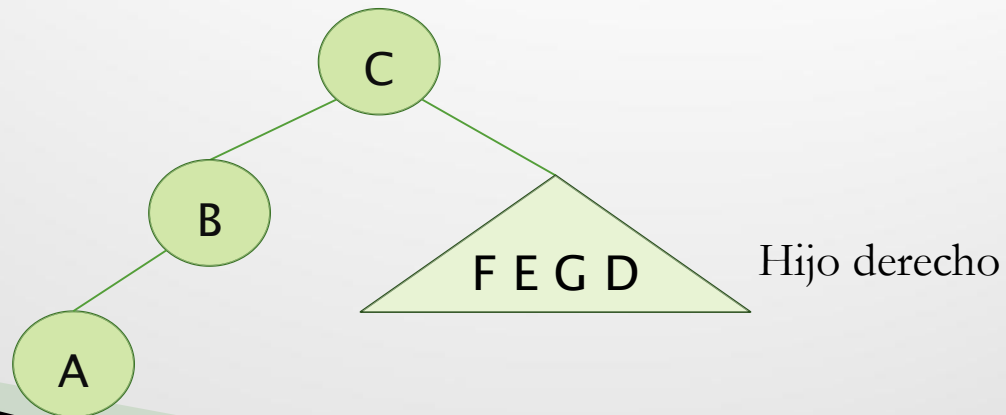
InOrden : A B C F E G D

PreOrden : C B A D E F G



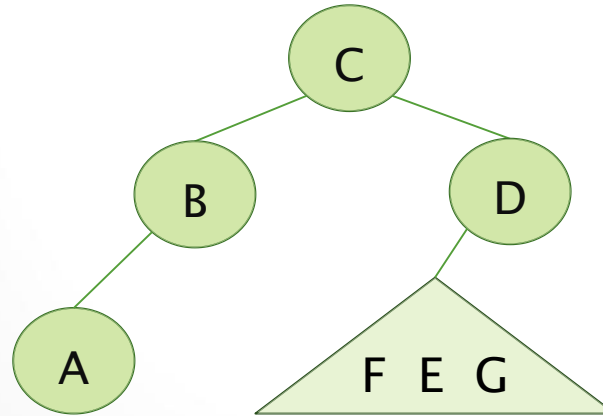
Identifico su secuencia inOrden A B en el recorrido preOrden

PreOrden : C B A D E F G



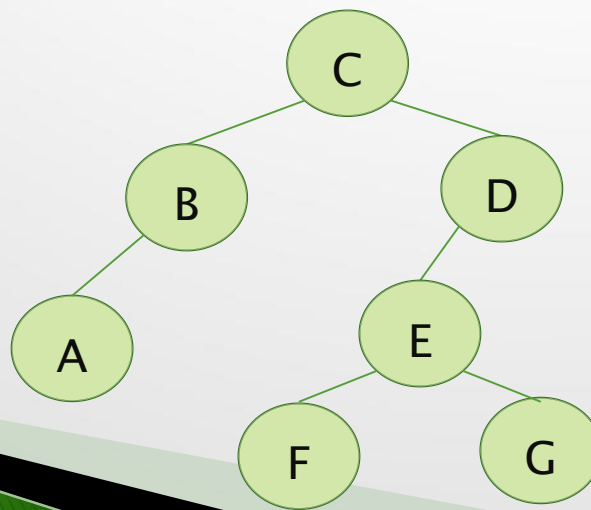
Identifico su secuencia inOrden F E G D en el recorrido preOrden

PreOrden : C B A **D E F G**



Identifico su secuencia inOrden F E G en el recorrido preOrden

PreOrden : C B A D **E F G**

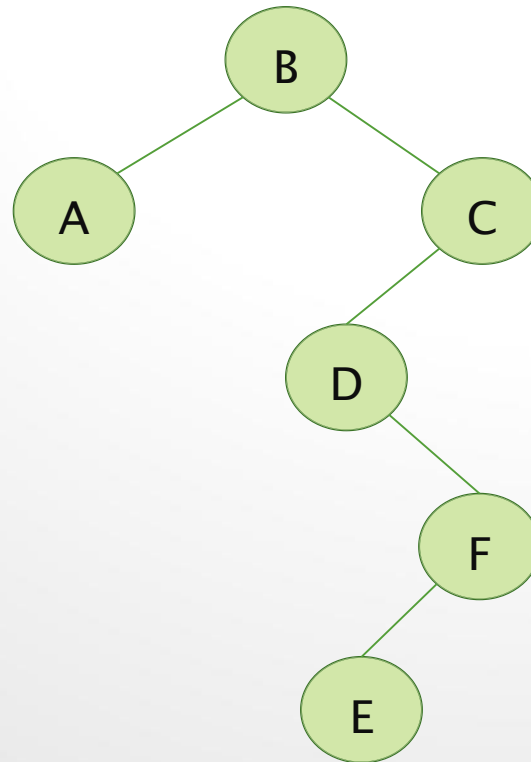


Sean los siguientes recorridos:

PreOrden: B A C D F E

InOrden: A B D E F C

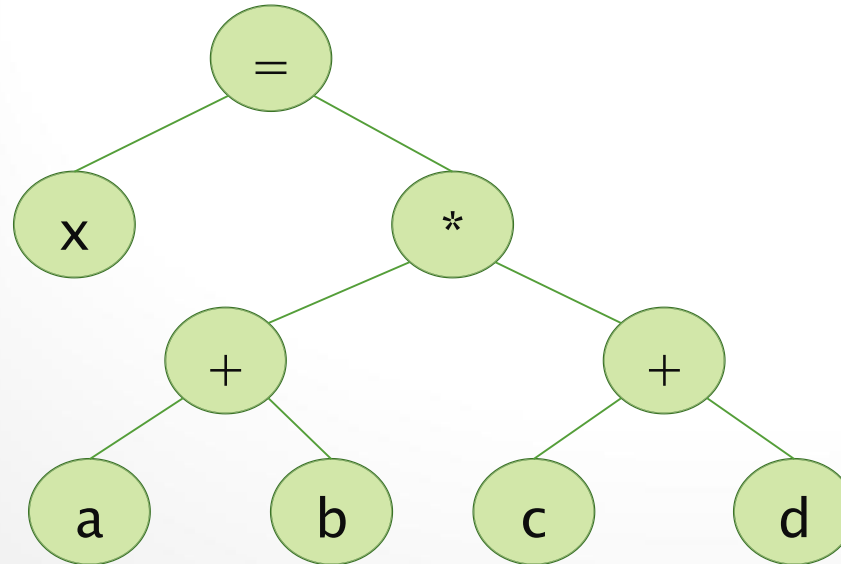
Encontrar el árbol binario único que satisfaga estos recorridos.



# APLICACIONES DE LOS ÁRBOLES

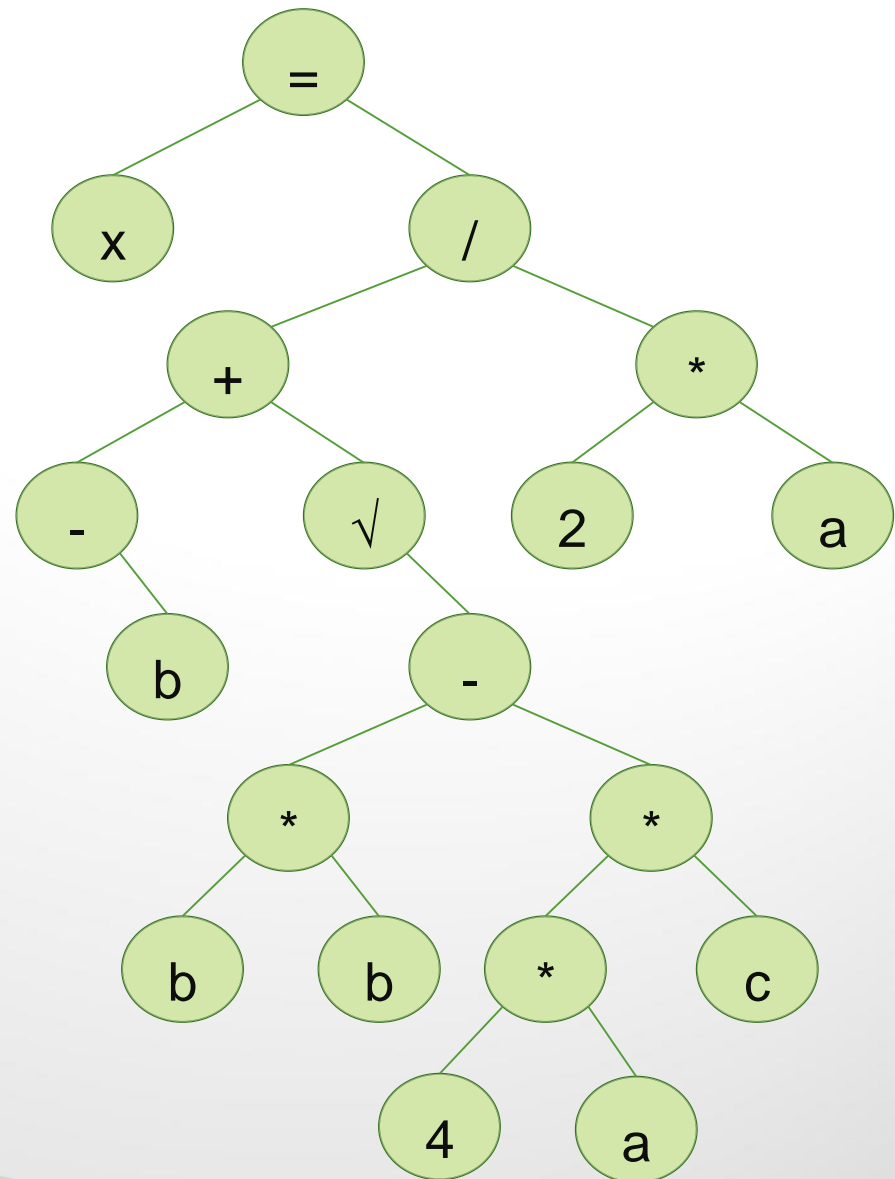
Los árboles binarios se pueden utilizar par evaluar expresiones aritméticas:

$$x=(a+b)*(c+d)$$



Generar el árbol de la siguiente fórmula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

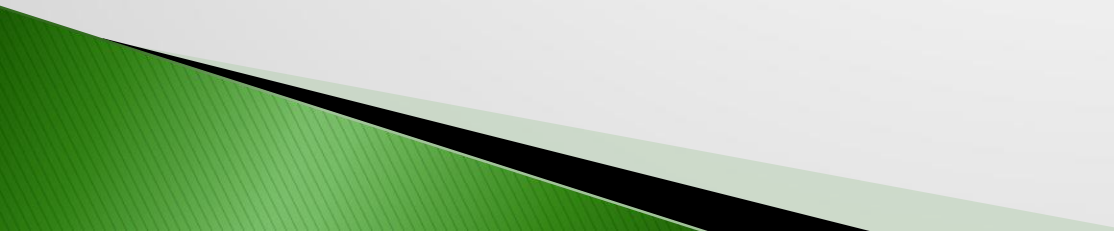


## **ARBOLES BINARIOS DE BÚSQUEDA (ABB)**

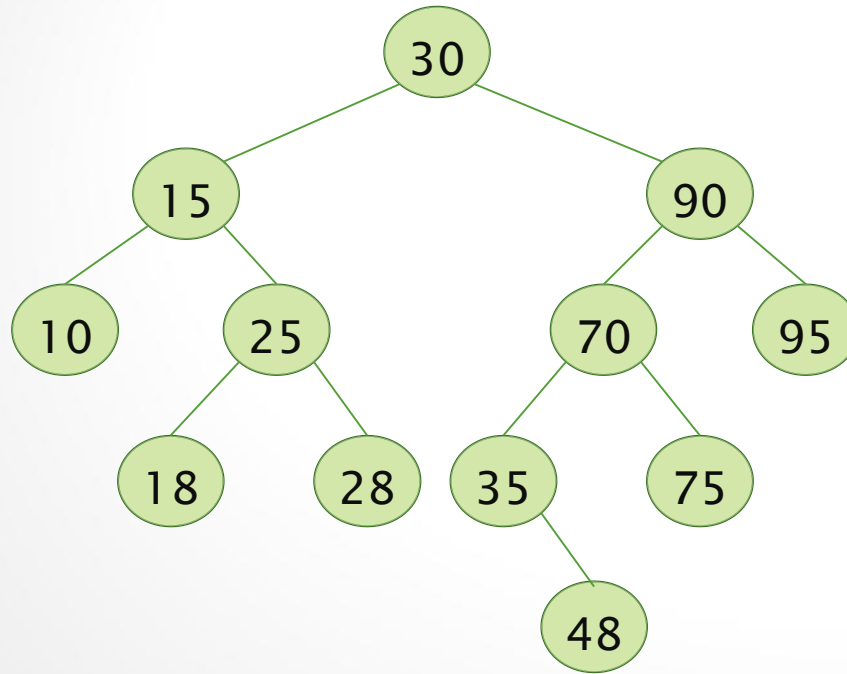
Un árbol binario de búsqueda, es una estructura de datos, cuyas operaciones de inserción y eliminación mantienen el siguiente principio:

Para todo nodo  $N$ , el nodo  $q$  se encuentra a su izquierda es menor que  $N$  (si este existe) y el nodo de derecha si es que existe es mayor que  $N$ .

Un ABB, se crea a partir de una lista de elementos, su utilidad es el hecho de que realizando el recorrido inOrden, los elementos aparecen ordenados de menor a mayor.



Dado el siguiente árbol, encontrar sus recorridos.



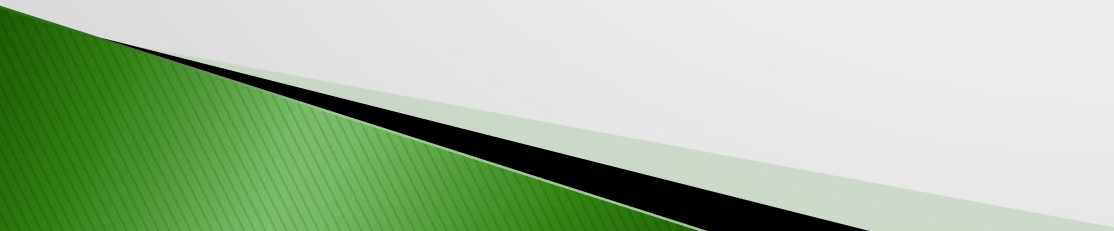


## IMPLEMENTACIÓN

Un árbol binario de búsqueda, es una estructura de datos, cuyas operaciones de inserción y eliminación mantienen el siguiente principio:

Para todo nodo  $N$ , el nodo  $q$  se encuentra a su izquierda es menor que  $N$  (si este existe) y el nodo de derecha si es que existe es mayor que  $N$ .

Un ABB, se crea a partir de una lista de elementos, su utilidad es el hecho de que realizando el recorrido inOrden, los elementos aparecen ordenados de menor a mayor.



```

class Nodo {
    private Nodo hijolzq;
    private int elem;
    private Nodo hijoDer;

    public Nodo(int ele) {
        hijolzq=null;
        elem = ele;
        hijoDer=null;
    }
    public void setHI(Nodo izq){
        hijolzq = izq;
    }
    public void setElem(int e) {
        elem = e;
    }
    public void setHD(Nodo der) {
        hijoDer = der;
    }
    public Nodo getHI() {
        return hijolzq;
    }
    public int getElem() {
        return elem;
    }
    public Nodo getHD() {
        return hijoDer;
    }
}
//end class

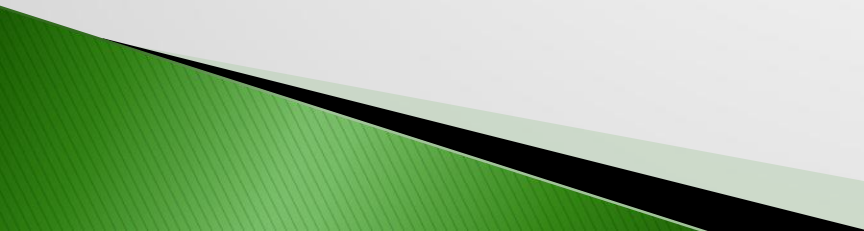
```

hijolzq	elem	hijoDer
---------	------	---------

```
class Arbol
{
    private Nodo raiz;

    public Arbol() {
        raiz = null;
    }
    private boolean esHoja(Nodo pr) {
        return pr.getHI() == null && pr.getHD() == null;
    }
    .....
    .....

}
```

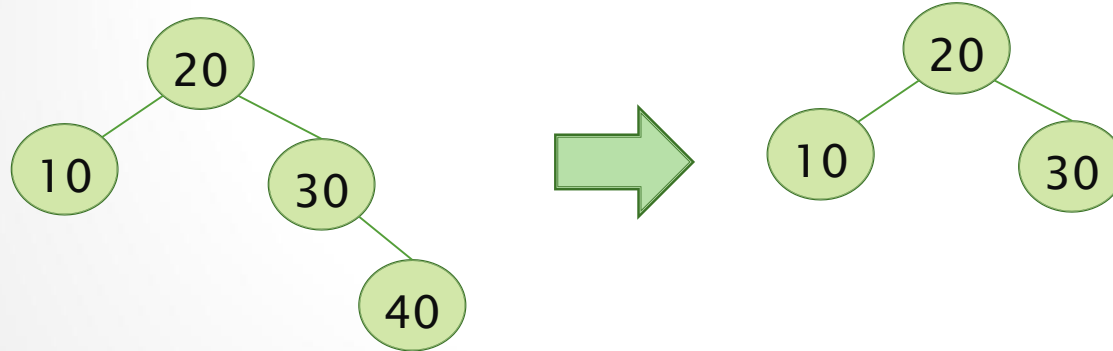


# EL METODO ELIMINAR

Para eliminar un elemento de un árbol ordenado, se toma 3 casos:

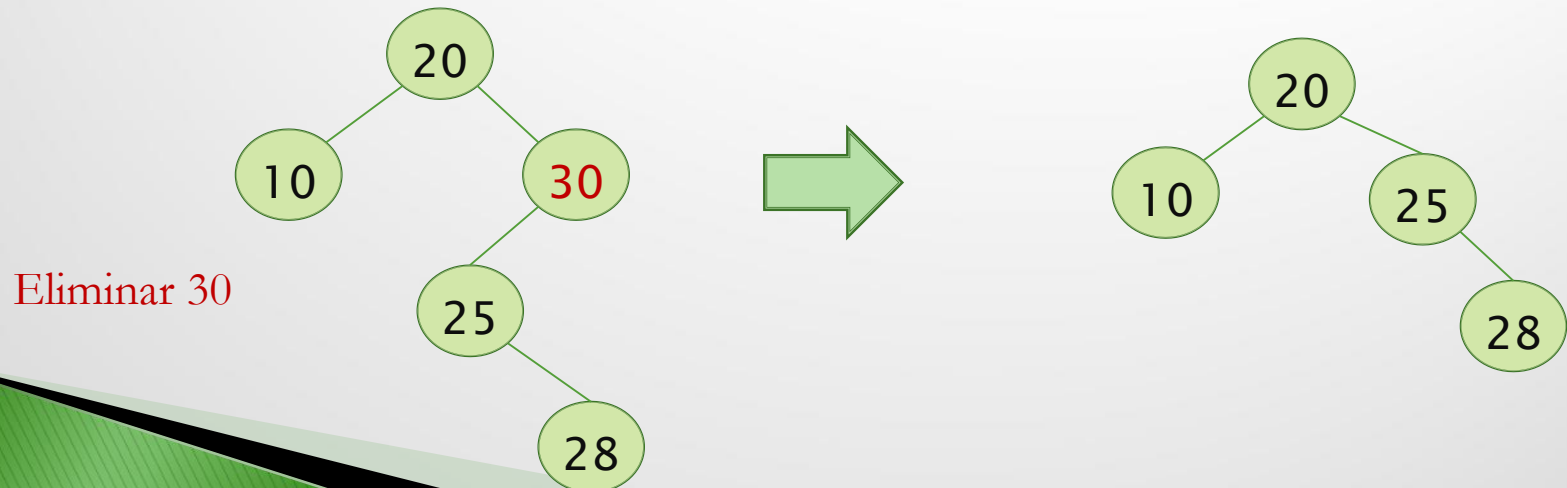
1. Si el nodo N a eliminar es una hoja

Simplemente se quita la hoja N



2. Si N tiene exactamente un hijo

Ese único hijo se sustituye a N.

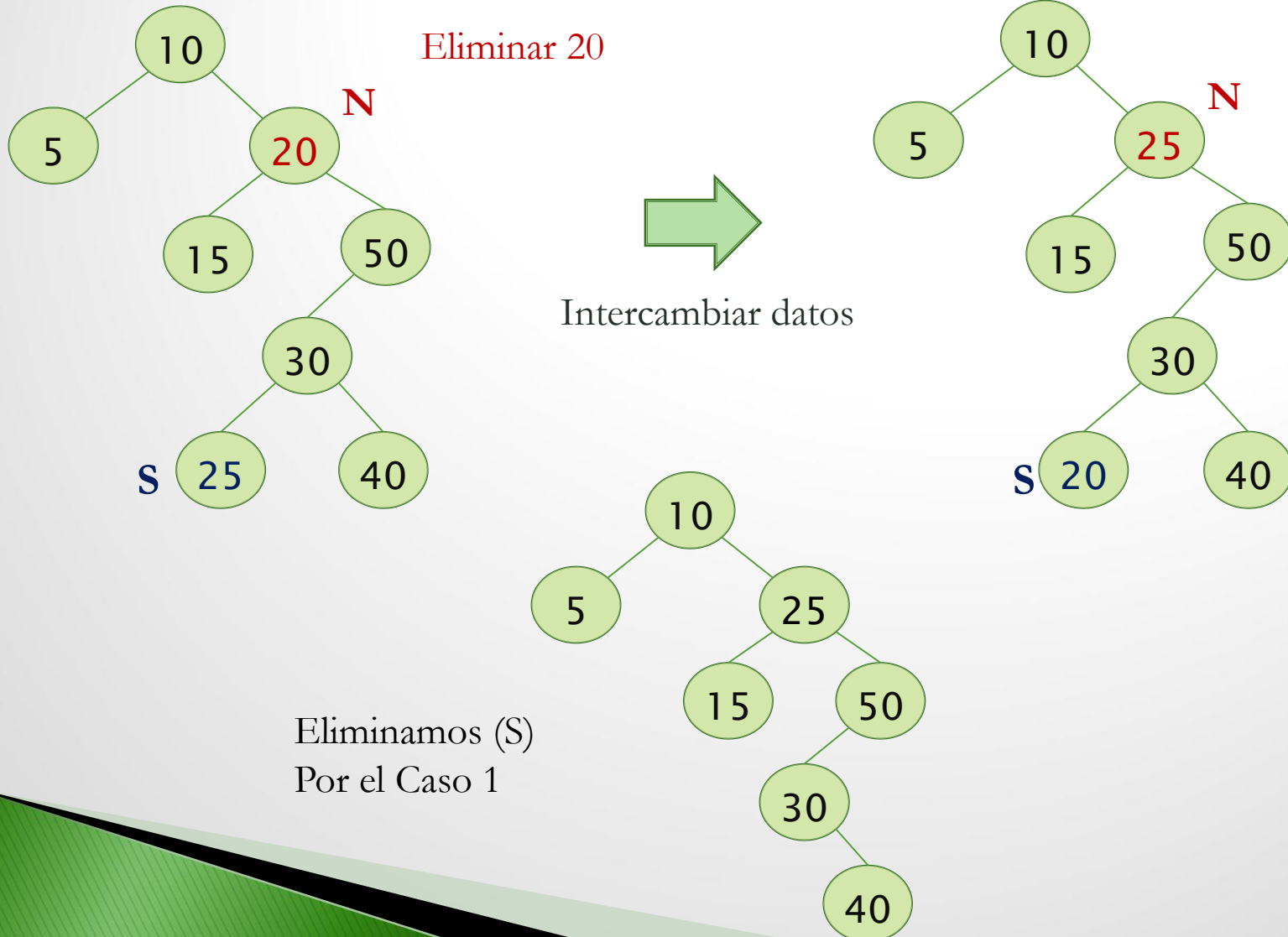


3. Si N tiene dos hijos

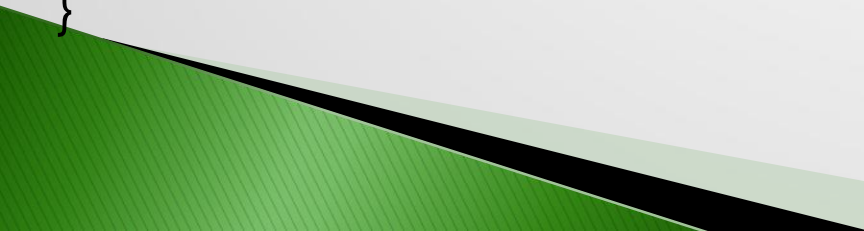
Buscar el nodo sucesor inorden de N (digamos S)

Intercambiar el dato del nodo N con el dato del nodo S

Luego eliminar el nodo S, por el caso 1 ó 2

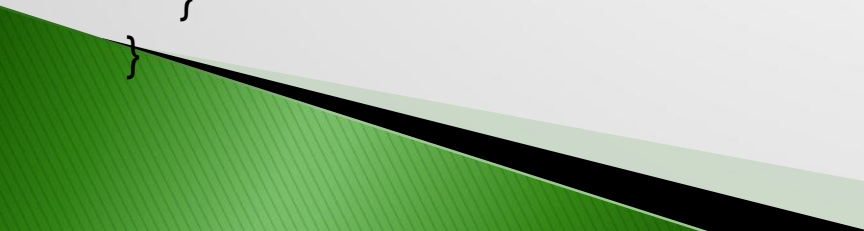


```
public void eliminar(int x) {  
    Nodo p = raiz;  Nodo ap = null;  
    while (p != null && p.getElem() != x) {  
        ap = p;  
        if (x > p.getElem())  
            p = p.getHD();  
        else  
            p = p.getHI();  
    }  
    if (p == null)  
        return;  
    if (esHoja(p))  
        elimCaso1(ap, p);  
    else  
        if ((p.getHD() != null && p.getHI() == null) || (p.getHD() == null && p.getHI() != null))  
            elimCaso2(ap, p);  
        else  
            elimCaso3(p);  
}
```



```
private void elimCaso1(Nodo ap, Nodo p) {  
    if (ap == null){  
        raiz = null;  
    }  
    else  
    {  
        if (p == ap.getHD())  
            ap.setHD(null);  
        else  
            ap.setHI(null);  
    }  
}
```

```
private void elimCaso2(Nodo ap, Nodo p) {  
    if (ap == null) {  
        if (p.getHD() != null)  
            raiz = p.getHD();  
        else  
            raiz = p.getHI();  
    }  
    else  
    {  
        if (p == ap.getHD())  
            if (p.getHD() != null)  
                ap.setHD(p.getHD());  
            else  
                ap.setHD(p.getHI());  
        else  
            if (p.getHD() != null)  
                ap.setHD(p.getHD());  
            else  
                ap.setHI(p.getHI());  
    }  
}
```





```
private void elimCaso3(Nodo p) {  
    Nodo s = p.getHD();  
    Nodo as = null;  
    while (s.getHI() != null) {  
        as = s;  
        s = s.getHI();  
    }  
    int aux = p.getElem();  
    p.setElem(s.getElem());  
    s.setElem(aux);  
    if (as == null)  
        p.setHD(null);  
    else  
        as.setHI(null);  
}
```

