Estructura de datos II

ARBOLES M-VIAS

Por: Ing. Edwin Calle Terrazas

INTRODUCCIÓN

Un árbol se dice que es m vías, si es ordenado y posee m apuntadores (hijos).

Otra forma de llamarlos es: árboles (m-1) - m

Así el árbol de 3 vías puede ser llamado:

Árbol 2 – 3 (2 elementos y 3 hijos)

El árbol de 4 vías

Árbol 3 – 4 (3 elementos y 4 hijos)

El árbol de 5 vías

Árbol 4 – 5 (4 elementos y 5 hijos)

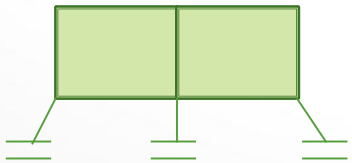
INSERCIÓN DE ELEMENTOS EN UN ÁRBOL M-VIAS

Puesto que el árbol tiene m apuntadores, entonces tendrá (m-1) elementos (datos).

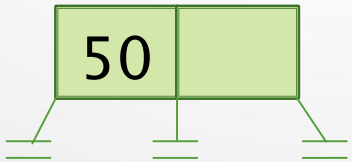
Para insertar elementos en este árbol, debemos primero llenar el nodo y luego ramificarlo.

En el siguiente ejemplo se muestra la inserción de los elementos en un árbol de 3 vías (árbol 2 - 3)

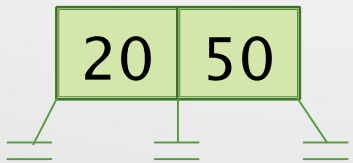
Árbol vacío



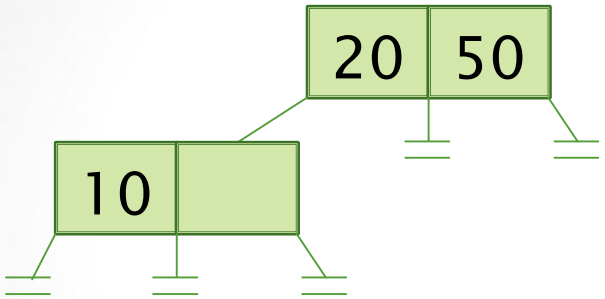
Insertar (50)



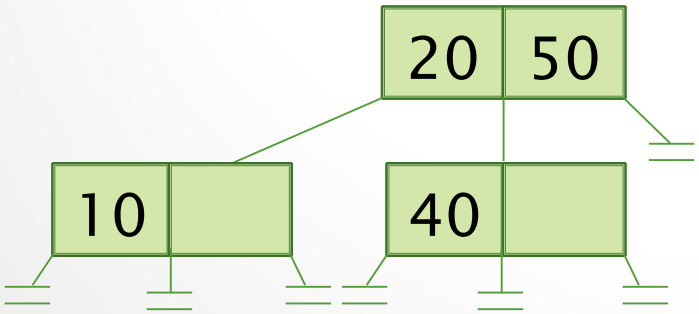
Insertar (20)



Insertar (10)

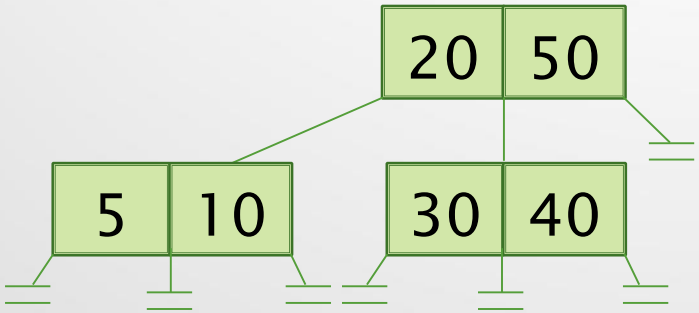


Insertar (40)



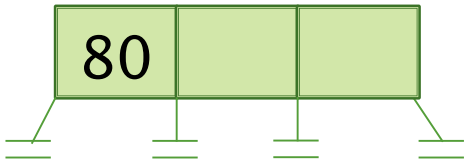
Insertar (30)

Insertar (5)

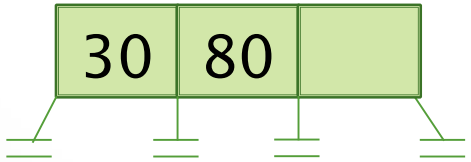


En el siguiente ejemplo se muestra la inserción de elementos en un árbol de 4 vías (árbol 3 - 4)

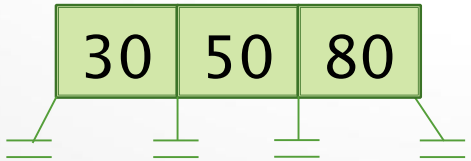
Insertar (80)



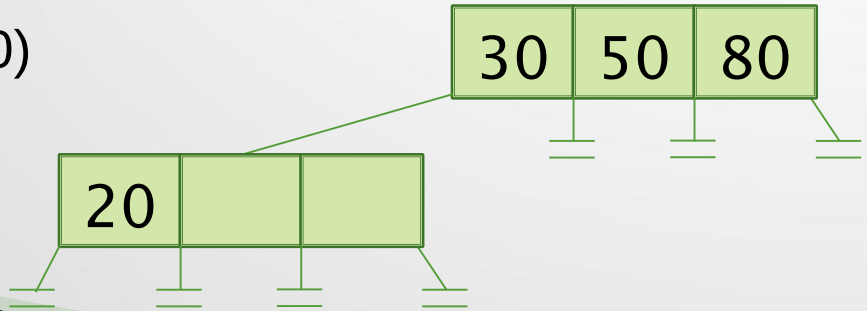
Insertar (30)



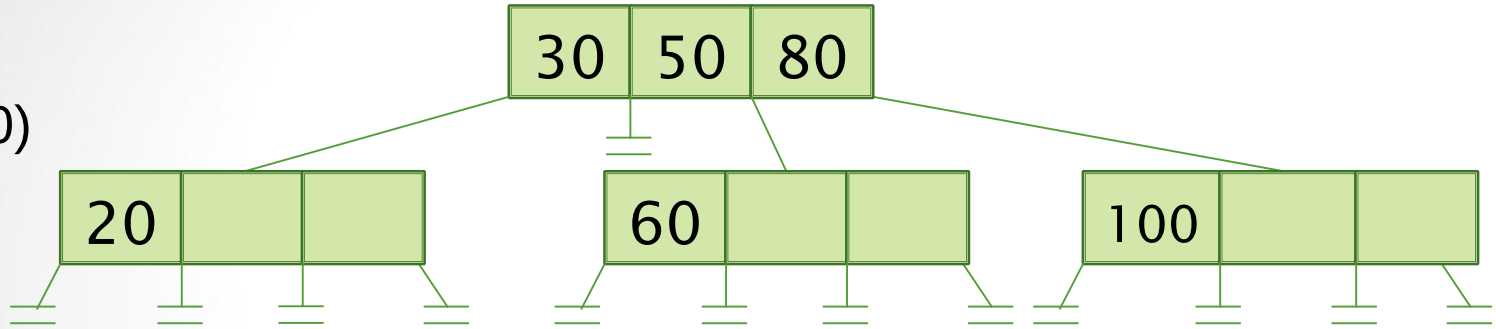
Insertar (50)



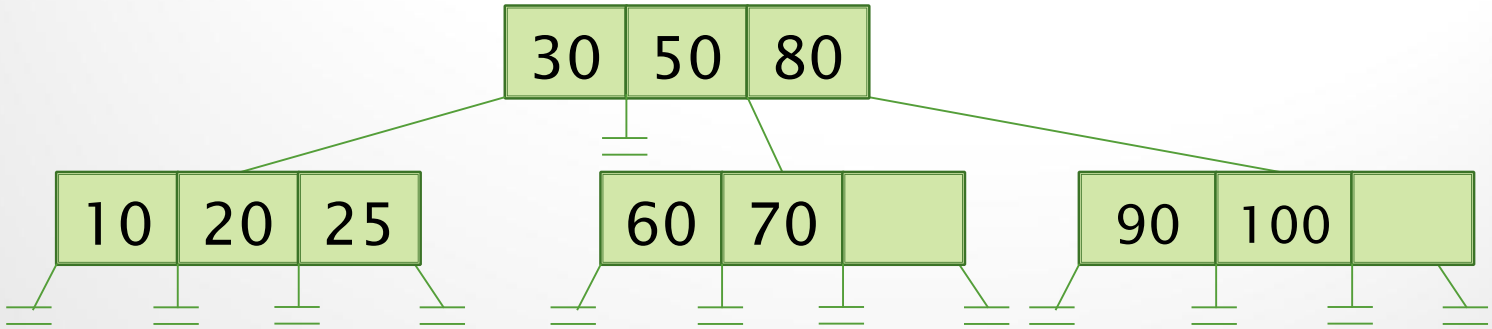
Insertar (20)



Insertar (60)
Insertar (100)



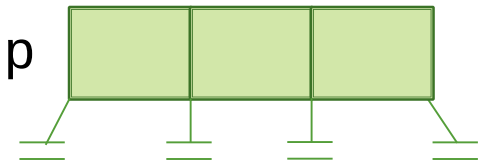
Insertar (10)
Insertar (25)
Insertar (70)
Insertar (90)



```
public class Nodo {
    public static int M = 4; //vias
    private int Elemento[];
    private Nodo Hijo[];
    private boolean Estado[];

    public Nodo(){
        Elemento = new int[M-1];
        Estado = new boolean[M-1];
        Hijo = new Nodo[M];
        for(int i = 0; i<M-1; i++){
            Estado[i] = false;
            Hijo[i] = null;
        }
        Hijo[M-1] = null;
    }
    public int getElem(int i){
        return Elemento[i-1];
    }
        public Nodo getHijo(int i){
        return Hijo[i-1];
    }
    public boolean Ocupado(int i){
        return Estado[i-1];
    }
    public boolean Vacio(int i){
        return !Estado[i-1];
    }
    public void setElem(int x, int i){
        Elemento[i-1] = x;
        Estado[i-1] = true;
    }
    public void setHijo(Nodo P, int i){
        Hijo[i-1] = P;
    }
    public void setVacio(int i){
        Estado[i-1] = false;
    }
    public int CantVacias(){
        int c = 0;
        for(int i = 0; i<Estado.length; i++){
            if(Estado[i] == false){
                c++;
            }
        }
        return c;
    }
    public int CantOcupados(){
        return (M-1) - CantVacias();
    }
    public boolean Lleno(){
        return (CantVacias() == 0);
    }
} //end class Nodo
```

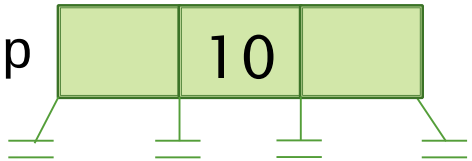
Asumamos M=3

Nodo p=new Nodo();



boolean b=p.esVacio(2); true

p.setElem(2,10)

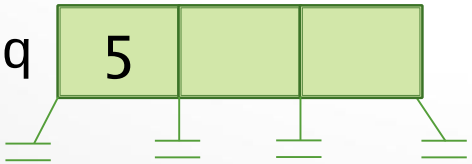


boolean b=p.esVacio(2); false

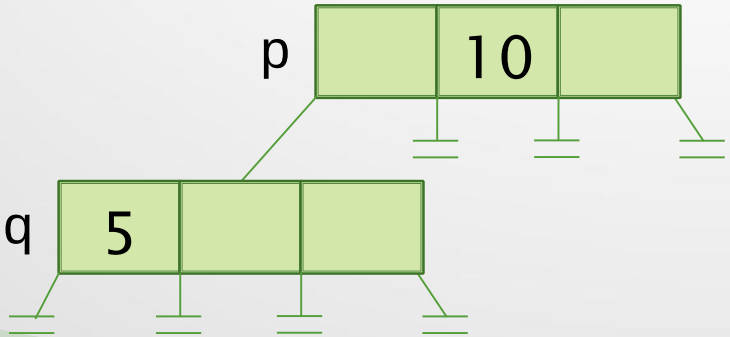
int a=p.getElem(2); a=10

Nodo q=new Nodo();

q.setElem(1,5)



p.setHijo(1,q)



IMPLEMENTACIÓN DE LA CLASS ARBOL

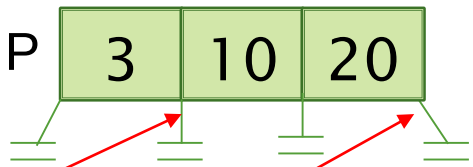
```
public class ArbolM {
    private Nodo raiz;

    public ArbolM(){
        raiz = null;
    }
    ……..
    ……..

} //end class arbol
```

```
private boolean esHoja(Nodo P){
        for(int i=1; i<=P.M; i++){
            if(P.getHijo(i) != null)
                return false;
        }
        return true;
}
```

Se utilizará 2 métodos privados:

private int getHijoDesc(Nodo P, int X)

Dado el nodo P y el dato a insertar x, nos devuelve el número de hijo donde debe insertarse x (Cuando P esté con todos sus elementos usados)



Si el elemento a insertar, existe, la función devuelve -1

getHijoDesc(P,7)=2

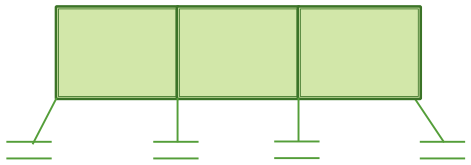
getHijoDesc(P,30)=4

```
private int getHijoDesc(Nodo P, int x){
        int i=1;
        while(i < P.M){
            if(x < P.getElem(i))
                return i;
            if(x == P.getElem(i))
                return -1;
           i++;
        }
        return P.M;
}
```

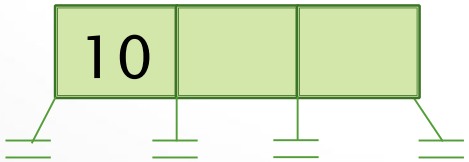
private void insertaOrd(Nodo P, int X)

Habiendo espacio en el nodo P, inserta X y los datos se ordenan (Si P no tiene espacio o X ya existe, en P no ocurre nada)

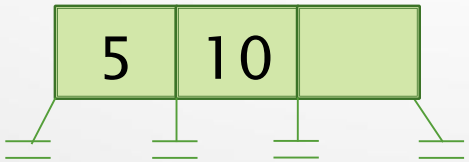
P vacío



InsertarOrd (P,10)



InsertarOrd (P,5)



```
private void InsertaOrd(Nodo P, int x)
{
      for(int i = 1; i<P.M; i++){
          if(P.Vacio(i)){
                P.setElem(x,i);
                return;
          }
          else
          {
              if(x==P.getElem(i))
                  return;
              else
              {
                   if(x < P.getElem(i)){
                       Recorrer(P,i);
                       P.setElem(x,i);
                       return;
                   }
              }
          }
      }
}
```

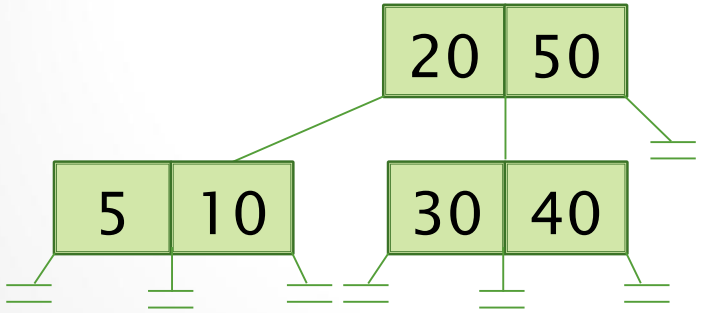
```
private void Recorrer(Nodo P, int i){
     int num1 = P.getElem(i);  int num2=0;
     int c = P.CantOcupados();
     while(i <= c){
          if(P.Ocupado(i+1))
              num2 = P.getElem(i+1);
          P.setElem(num1,i+1);
          num1 = num2;
          i++;
     }
 }
```

```
public void Insertar(int x){
        if(raiz == null){
            raiz = new Nodo();
            raiz.setElem(x,1);
        }
        else
        {
            Nodo P = raiz; Nodo AP=null; int i = 0;
            while(P != null){
                AP = P;
                if(!P.Lleno()){
                    InsertaOrd(P, x);
                    return;
                }
                i=getHijoDesc(P, x);
                if(i == -1){
                    return; // x esta en el arbol
                }
                P = P.getHijo(i);
            }
            Nodo Q = new Nodo();
            Q.setElem(x,1);
            AP.setHijo(Q,i);
        }
    }
```

RECORRIDOS

Se mantienen los mismo recorridos que se hacen sobre los arboles binarios de búsqueda.

PreOrden, InOrden, PostOrden



InOrden : 5 10 20 30 40 50

PreOrden : 20 5 10 50 30 40

```
public void InOrden(JTextArea ta) {
      InOrden(raiz,ta);
}
```

```
private void InOrden(Nodo P,JTextArea jta){
        if(P == null)
            return;
        else
        {
            if(esHoja(P))
            {   int i = 1; //muestra todos los elementos de nodo P
                while(i<=P.CantOcupados()) {
                  jta.append(String.valueOf(P.getElem(i)+ "  "));
                  i++;
                }
            }
            else
            {
                for(int i=1; i<=P.M-1; i++){
                    InOrden(P.getHijo(i),jta);
                    jta.append(String.valueOf(P.getElem(i)+ "  "));
                }
                InOrden(P.getHijo(P.M),jta);
            }
        }
    }
```