ARBOLES

ESTRUCTURAS DE DATOS

Arboles

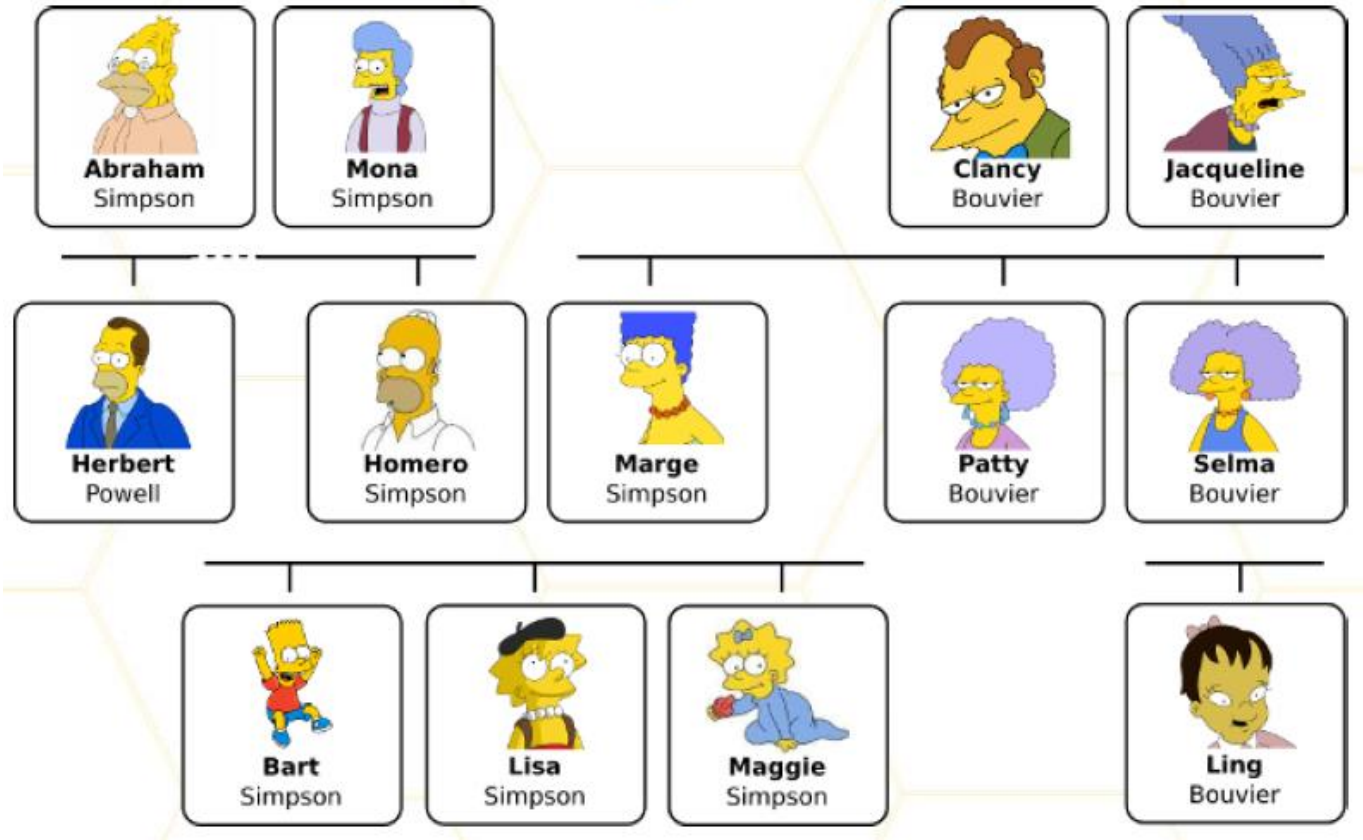
Definición

- Del latín ***arborem;*** la real academia española, define el árbol como: planta perenne, de tronco leñoso y elevado que se ramifica a cierta altura del suelo, produce ramas, que parten de un único tronco, dando lugar a una nueva copa separada del suelo.

Arboles

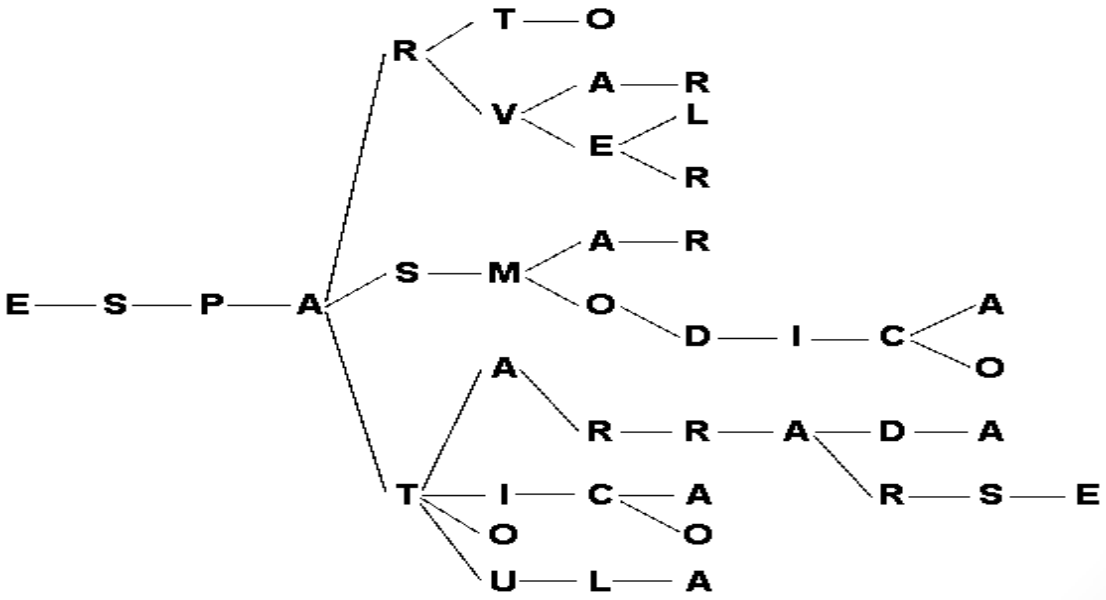
Definición

Árbol genealógico

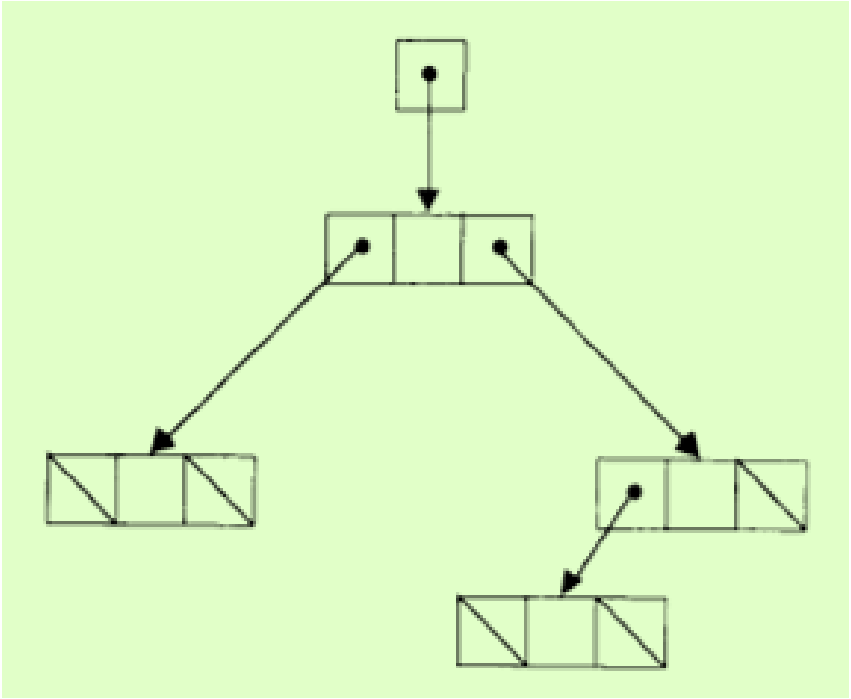


Árboles Generales

Intuitivamente el concepto de árbol implica una estructura donde los datos se organizan de modo que los elementos de información estén relacionados entre sí a través de ramas.



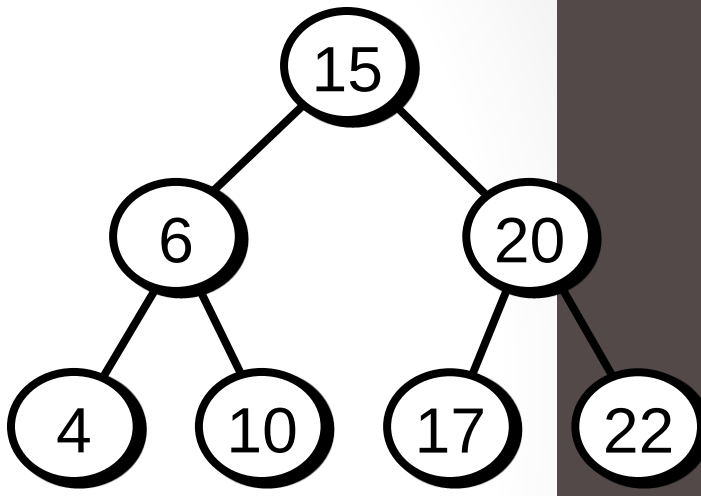
Definición



- Es un conjunto de elementos llamados nodos donde a partir de cada nodo se ramifican uno o mas sub árboles.
- Los nodos de los arboles contienen dos o más enlaces.
- Normalmente se dibujan en forma opuesta a los árboles en la naturaleza.
- Permiten jerarquizar información, es decir, agrupamiento de la información.

Árboles Generales

- **Nodos:** conjunto finito de elementos.
- **Ramas:** conjunto finito de líneas dirigidas, que conectan nodos.
- **Grado del Nodo:** número de ramas descendentes con un nodo.
- **Raíz:** primer nodo de un árbol no vacío.
- **Camino:** secuencia de nodos en los que c/nodo es adyacente al siguiente.
  - Solo existe 1 camino entre la Raíz y un Nodo cualquier.
  - La distancia de un Nodo a la Raíz determina la rapidez de búsqueda.



Raiz=15

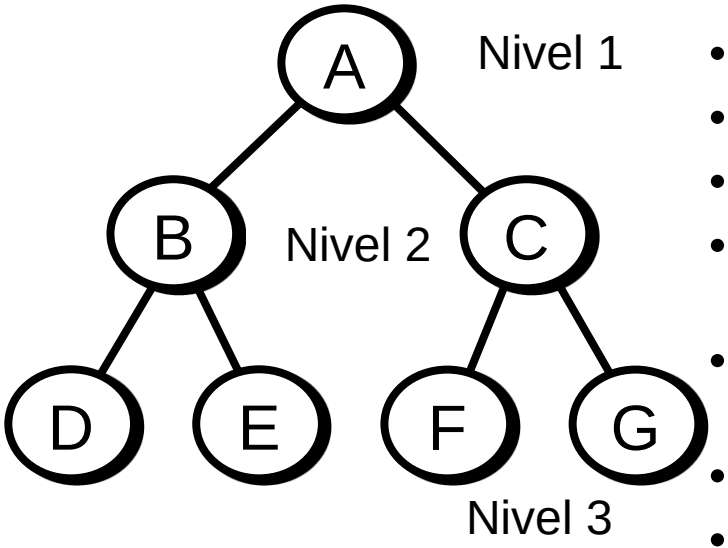
Nodos: 15, 6, 20….

Ramas(15, 6); (20, 17)

Grado del 15= G(15)=2

G(10)=0; G(20)=2

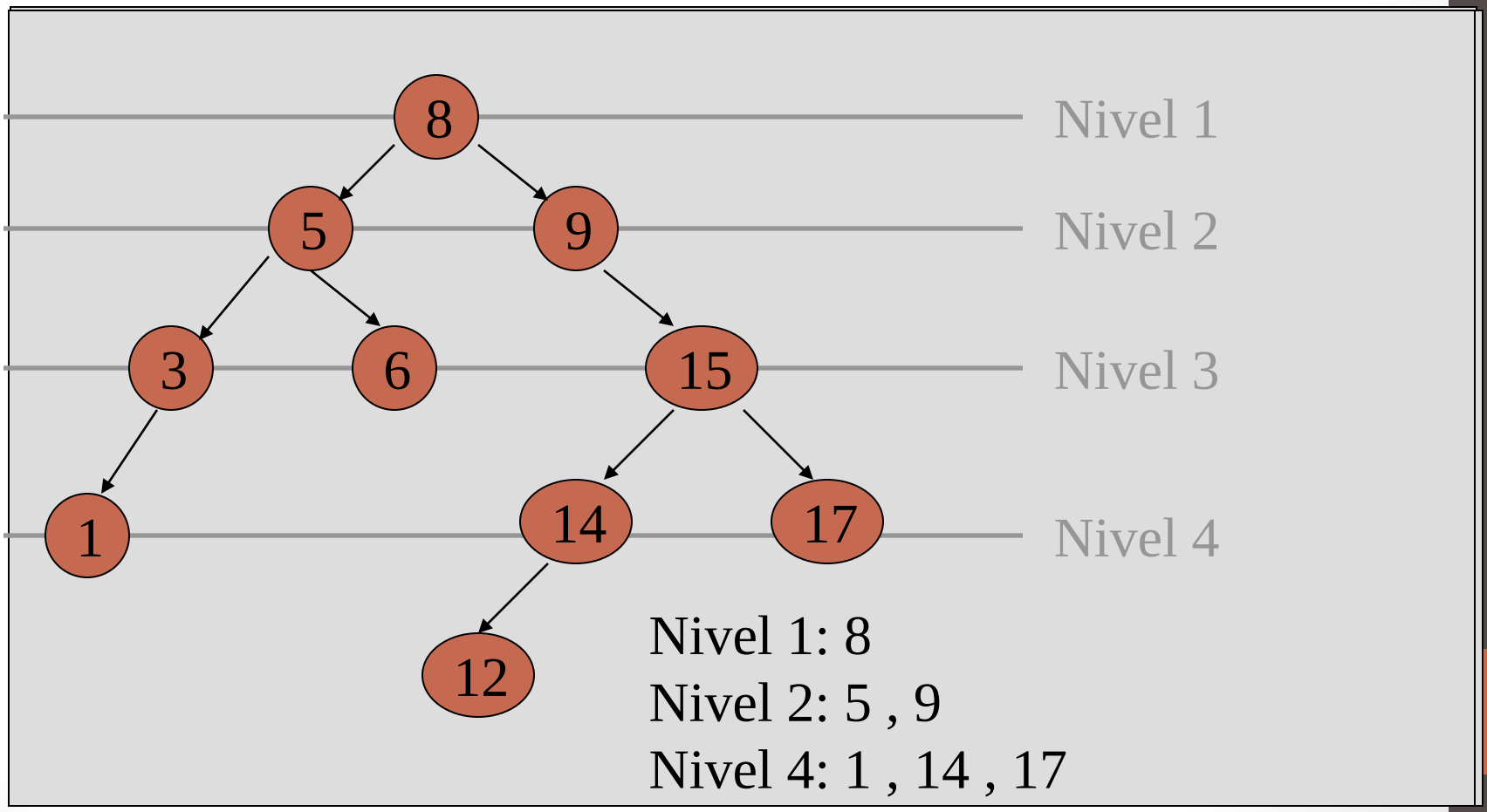
Terminología



- **Padre:** tiene Nodos sucesores.
- **Hijos:** Nodos sucesores.
- **Descendientes:** Hijos de los hijos
- **Ascendientes:** los padre y abuelos de un nodo hijo.
- **Hermanos:** 2 o mas nodos del mismo padre.
- **Hojas:** nodo sin hijos .
- **Nivel de un nodo:** distancia a la raíz.
- **Altura o profundidad de un árbol:** nivel de la hoja del camino más largo desde la raíz más uno.
  - La altura de un árbol vacío es 0.
- **Subárbol:** cualquier estructura conectada por debajo del raíz. C/nodo de un árbol es la raíz de un subárbol que se define por el nodo y todos los descendientes del nodo.

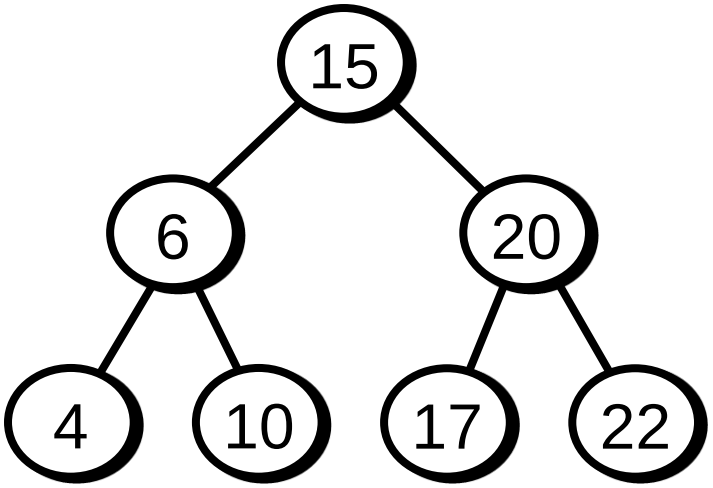
- Padres: A; B y C.
- Hijos: De A(B y C), de B (D y E)
- Descendentes de B: D y E
- Ascendientes de E: B y A.
- Hermano: {B, C}, {F, G}
- Hojas: D, E, F, G
- Altura del Árbol: 3
- Altura del Subárbol de B: 2

Ejemplos de la terminología

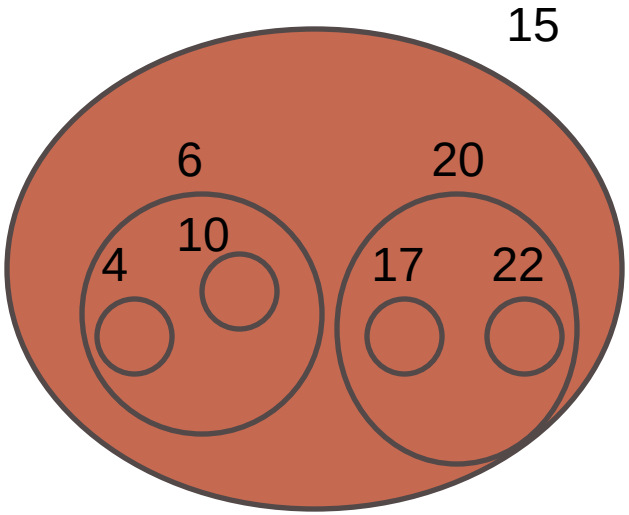


REPRESENTACION DE UN ARBOL.

a) Grafos: conjunto de nodos y aristas



b) Por diagramas de conjuntos



c) Lineal 15(6(4(),10()),20(17(),22()))

d) Tabla de jerarquias

nivel	1	2	3	4
	15			
		6		
			4	
			10	
		20		
			17	
			22	

Árbol Binario

- Árbol donde ningún nodo puede tener mas de 2 subárboles.
- En cualquier nivel n, solo puede contener de 1 a 2^{n-1} nodos

Características

- ✓ Sus nodos contienen dos enlaces.
- ✓ El valor de sus nodos pudiese se NULL.
- ✓ El nodo raíz es el primer nodo de un árbol.
- ✓ Cada enlace en el nodo raíz se refiere a un hijo.
- ✓ El hijo izquierdo es el elemento menor a su raíz.
- ✓ El hijo derecho es el elemento mayor a la raíz.
- ✓ Los hijos de un nodo se conocen como descendientes.
- ✓ Un nodo sin hijos se conoce como nodo de hoja.

Árbol Binario

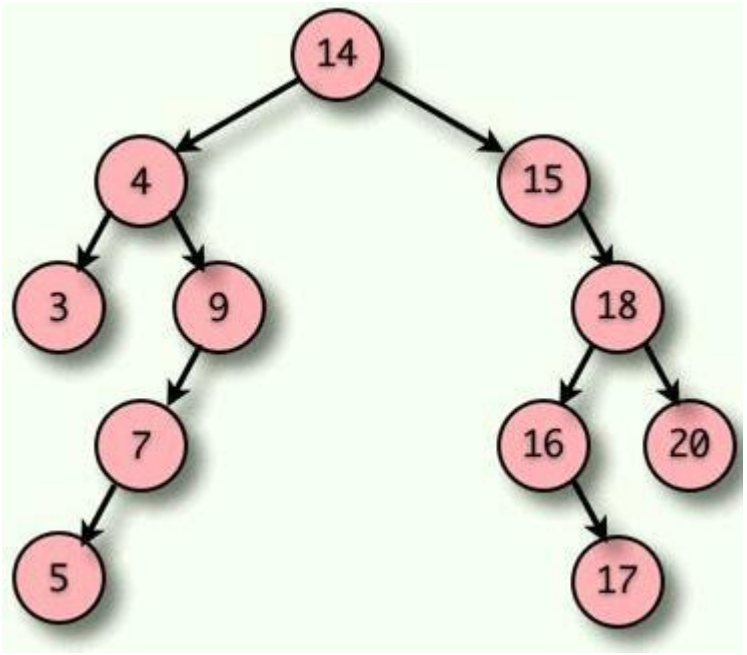
- Construir árbol con la siguiente información

```
14,15,4,9,7,18,3,5,16,4,20,17,9,14,5
```

Árbol Binario

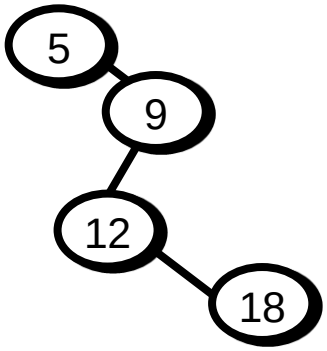
- Construir árbol con la siguiente información

```
14,15,4,9,7,18,3,5,16,4,20,17,9,14,5
```

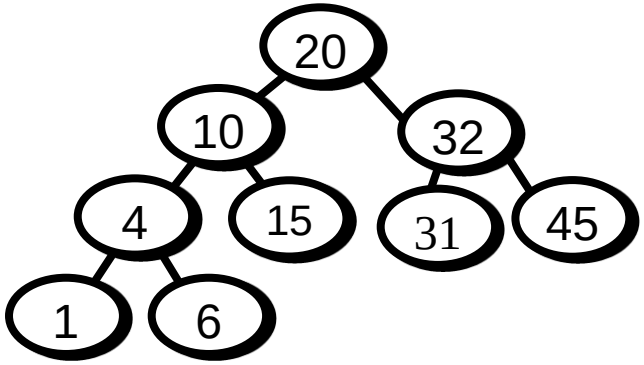


Árbol Binario

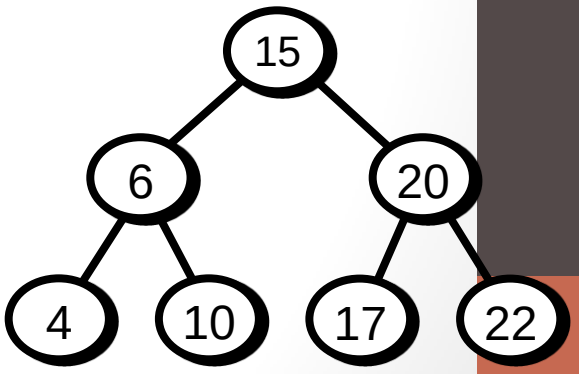
- Árbol **Completo:** de **Altura *n*** es un árbol en el que para c/nivel, del **1 al n-1, está lleno de nodos**. Todos los nodos hoja a nivel *n* ocupan posiciones a la izquierda.
- Árbol **Lleno**: tiene el máximo número de entradas para su altura: $2^{n.}$ **A Nivel *k*, tendrá 2^k–1 nodos**. (Nivel = Profundidad)
  - Lleno de Profundidad 3 = Nivel 2 => 2^{2+1}–1 nodos= 2^3–1= 7 nodos
- Árbol **Degenerado**: hay un solo nodo hoja (el 18) y cada nodo no hoja tiene solo un hijo. **Equivalente a una lista enlazada**.



Degenerado de profundidad 4



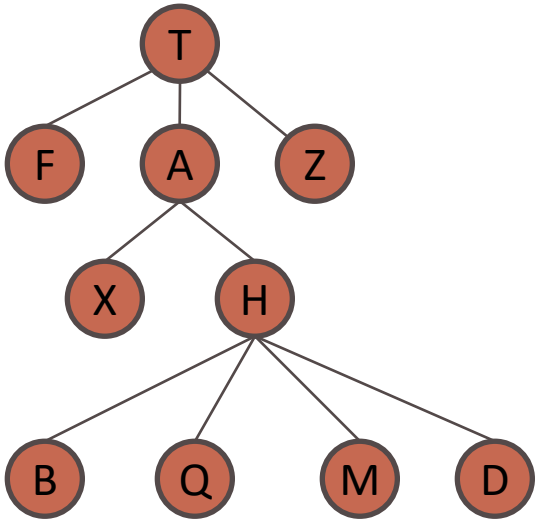
Completo de profundidad 4

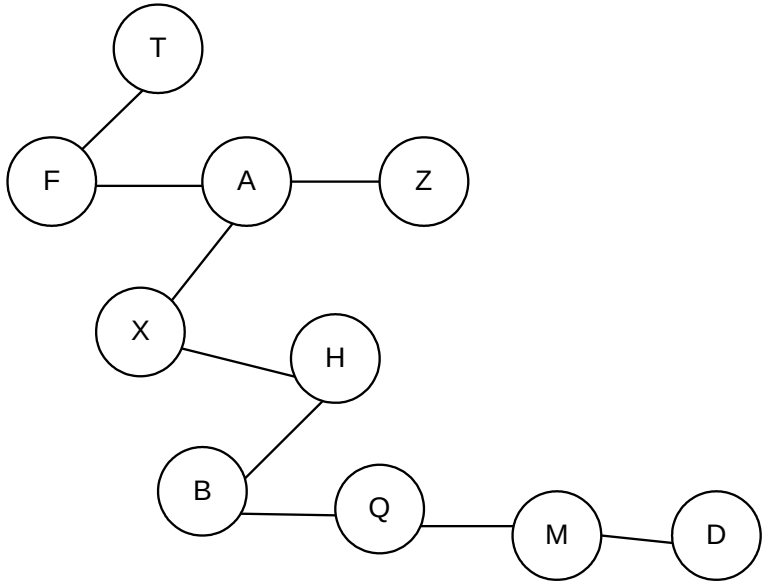


Lleno de profundidad 3

CONVERSION DE UN ARBOL GENERAL A ARBOL BINARIO

1.-La raíz del árbol general es la raíz del árbol binario.
2.-Para cada nodo se forma una lista encadenada simple con todas las ramificaciones del nodo, constituyendo los sub arboles derecho y los subárbol izquierdos lo constituyen el primer nodo izquierdo que se ramifica.

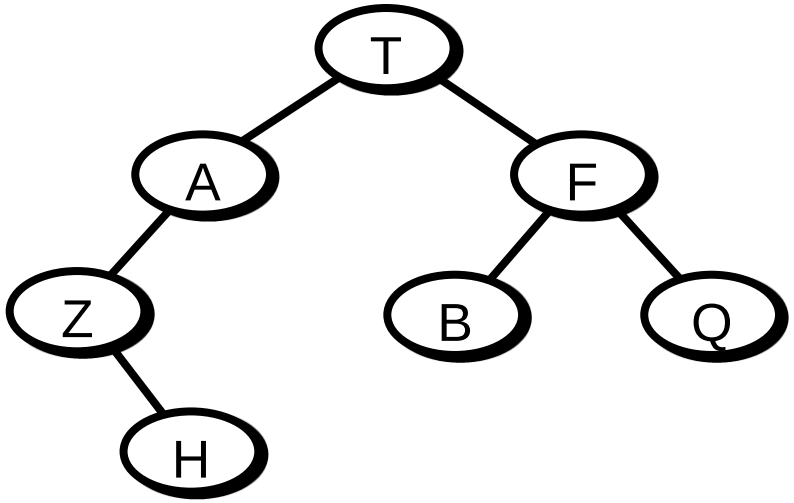




IMPLEMENTACION COMPUTACIONAL

a)**estatica**.-Se emplea un arreglo unidimensional, las celdas que almacenan nodos son a partir de la numero uno.
-Para un nodo que se encuentra en la celda i, el subárbol izquierdo se localiza en la celda 2*i y el subárbol derecho en la celda 2*i+1.

ejemplo.-

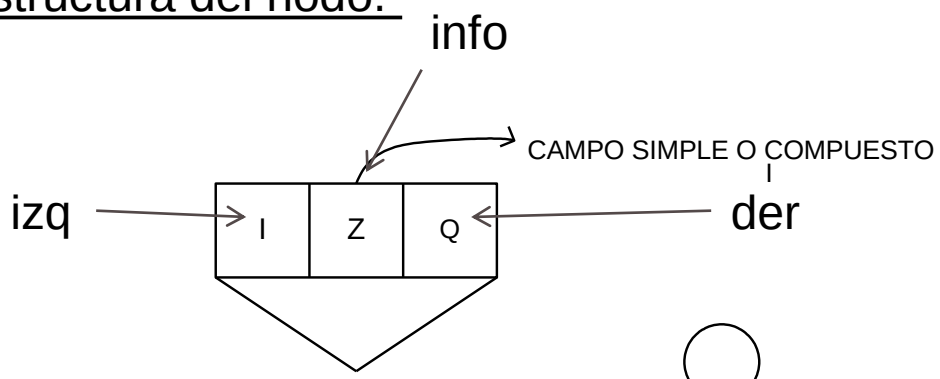


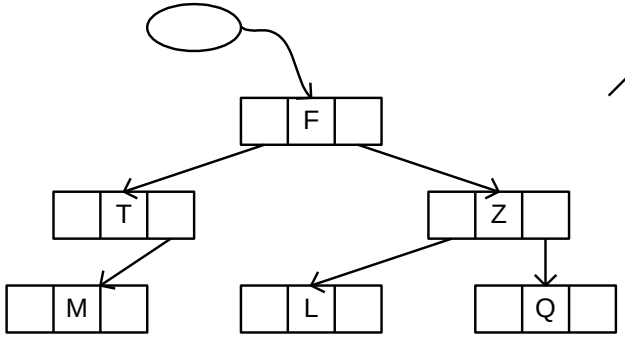
T	A	F	Z		B	Q		H												
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

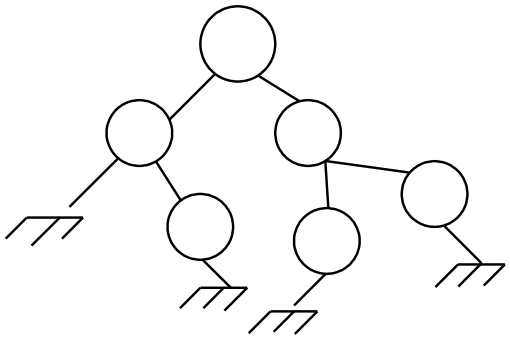
b) <u>Dinamica</u>.- empleo de estructuras enlazadas

Existe una Variable especial que almacena la direccion del primer nodo llamada “raiz”

<u>Estructura del nodo.-</u>







Estructura del Nodo del Árbol Binario

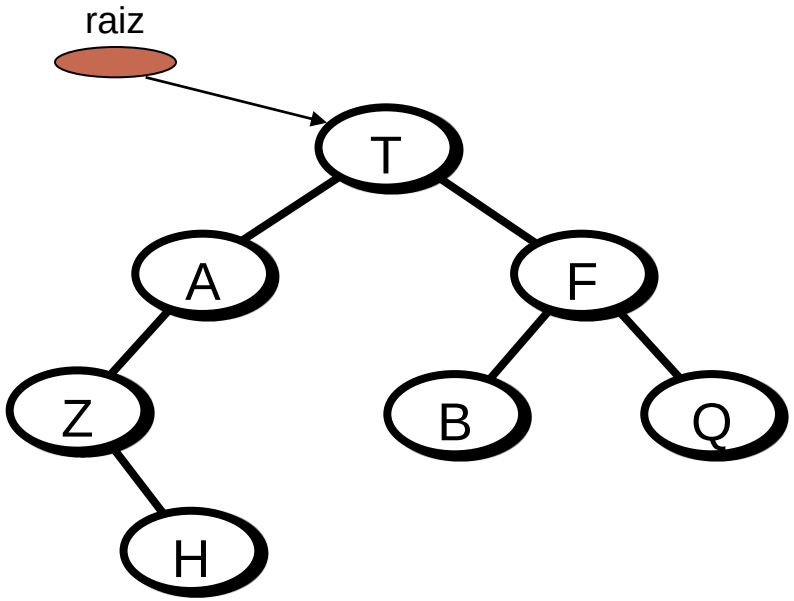
```
struct nodo
{
    int    infoo;
    struct nodo   *izq;
    struct nodo   *der;

}
typedef struct nodo  Nodo;
Nodo *ArbolBinario;
```

OPERACIONES BASICAS EN UN ARBOL BINARIO

- CREACION
- RECORRIDO.

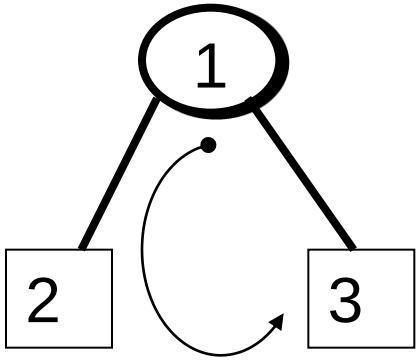
CREACION DEL ARBOL BINARIO.

El arbol se crea con una estructura y una cantidad de nodos definido.

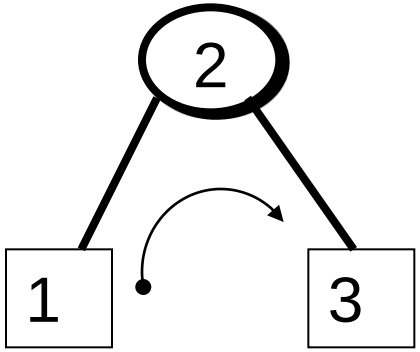


Recorrido del Árbol

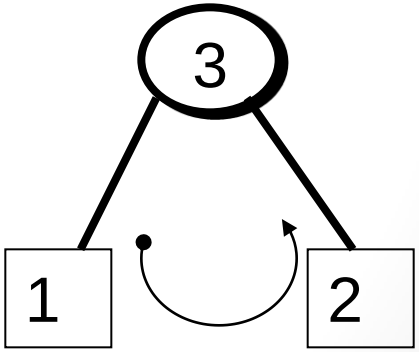
- Recorrer el Árbol significa que cada nodo sea procesado una vez y solo una en un secuencia determinada. Existen 2 enfoques generales
  - **Recorrido en Profundidad**: el proceso exige alcanzar las profundidades de un camino desde la raíz hacia el descendiente mas lejano del primer hijo, antes de proseguir con el segundo.
  - **Recorrido en Amplitud:** el proceso se realiza horizontalmente desde la raíz a todos su hijos antes de pasar con la descendencia de alguno de ellos.



Preorden RID
Raiz–IZQ–DER



EnOrden IRD
IZQ–Raiz–DER



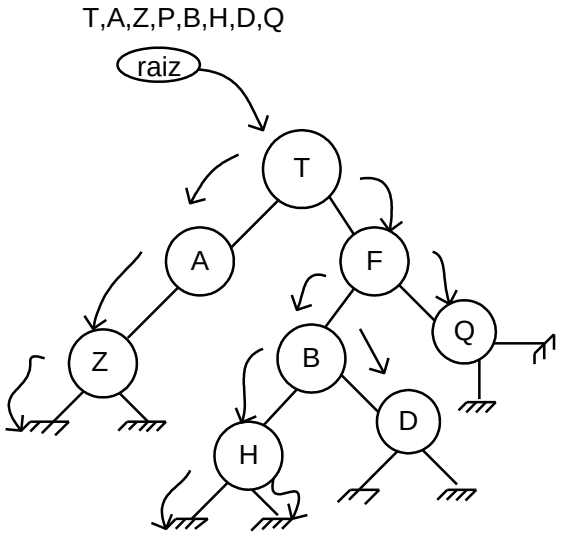
PostOrden IDR
IZQ–DER–Raiz

a) **Preorden**.-

1.-Procesar la información del nodo raiz.

2.-Recorrer el sub-arbol izquierdo en recursividad pre-orden.

3.- Recorrer el sub-arbol derecho en pre-orden.




```
inicio preorden(raiz)
  si raiz<> nulo
    procesar raiz->info
    preorden(raiz->izq)
    preorden(raiz->der)
  fin si
fin
```

T,A,Z,F,B,H,D,Q

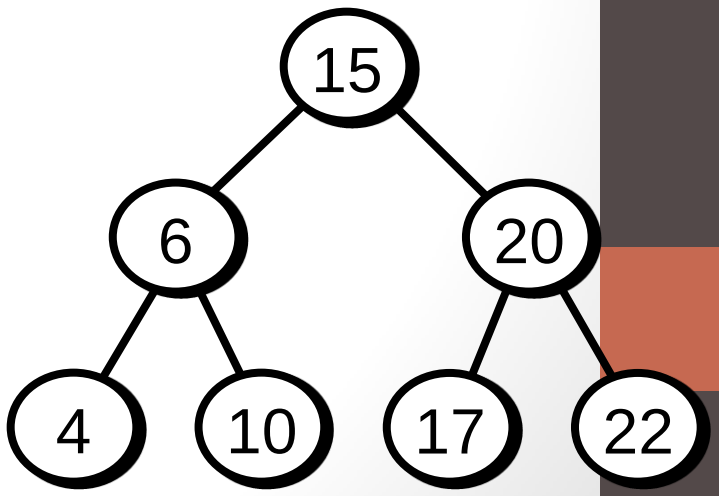
Recorrido PreOrden (RID)

```
void preOrden(ArbolBinario raíz)
{  if(raiz)
   {    visitar(raiz–>dato);
        preOrden(raiz–>izq);
        preOrden(raiz–>der);
   }
}
```

```
void visitar(TipoElemento x)
    {printf(“ %i ”, x);}
```

El recorrido en PreOrden del árbol es el siguiente:

15, 6, 4, 10, 20, 17, 22

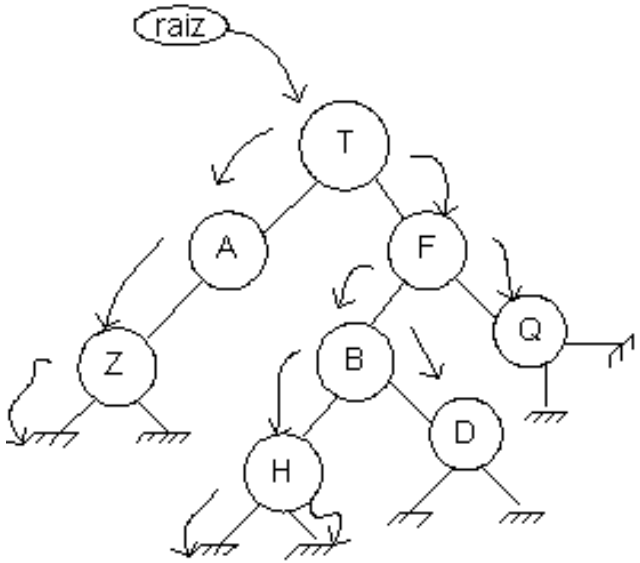


b) **<u>EnOrden</u>**.-

1.-Recorrer el sub-arbol izquierdo en recursividad en-orden.

2.-Procesar la información del nodo raíz.

3.- Recorrer el sub-arbol derecho en en-orden.




```
inicio enorden(raiz)
  si raiz<> nulo
          enorden(raiz->izq)
          procesar raiz->info
          enorden(raiz->der)
  fin si
fin
```

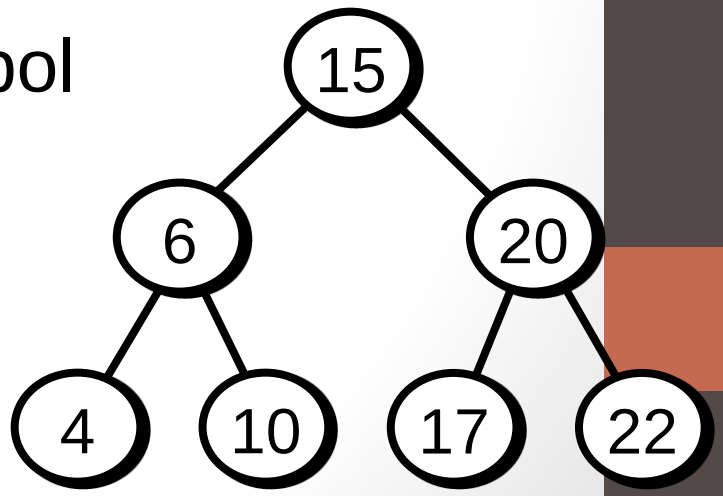
Z, A,T,H,B,D,F,Q

Recorrido EnOrden (IRD)

```
void enOrden(ArbolBinario raíz)
{  if(raiz)
   {    enOrden(raiz–>izq);
        visitar(raiz–>dato);
        enOrden(raiz–>der);
   }
}
```

El recorrido en EnOrden del árbol es el siguiente:

4, 6, 10, 15, 17, 20, 22

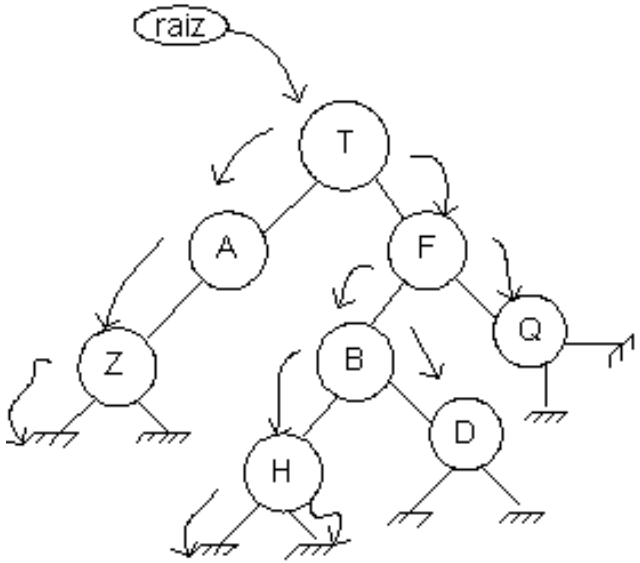


a) **PostOrden**.-

1.-Recorrer el sub-arbol izquierdo en recursividad post-orden.

2.- Recorrer el sub-arbol derecho en post-orden.

3.-Procesar la información del nodo raíz.




```
inicio postorden(raiz)
  si raiz<> nulo
          postorden(raiz->izq)
          procesar raiz->info
          postorden(raiz->der)
  fin si
fin
```

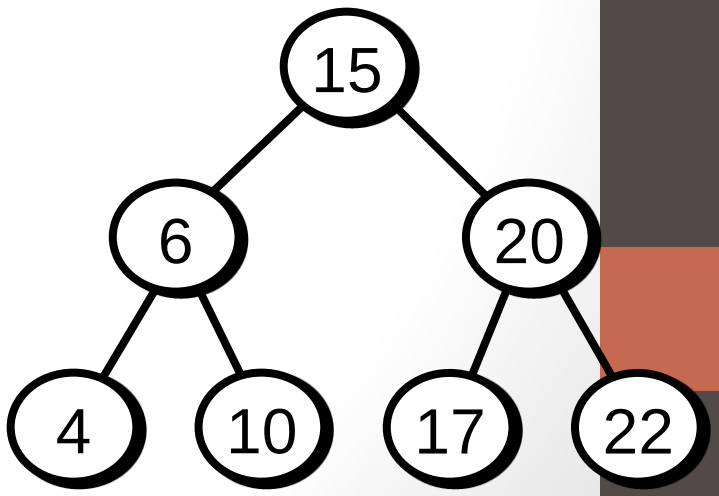
Z,A,H,D,B,Q,F,T

Recorrido PostOrden (IDR)

```
void PostOrden(ArbolBinario raíz)
{  if(raiz)
   {    PostOrden(raiz–>izq);
        PostOrden(raiz–>der);
        visitar(raiz–>dato);
   }
}
```

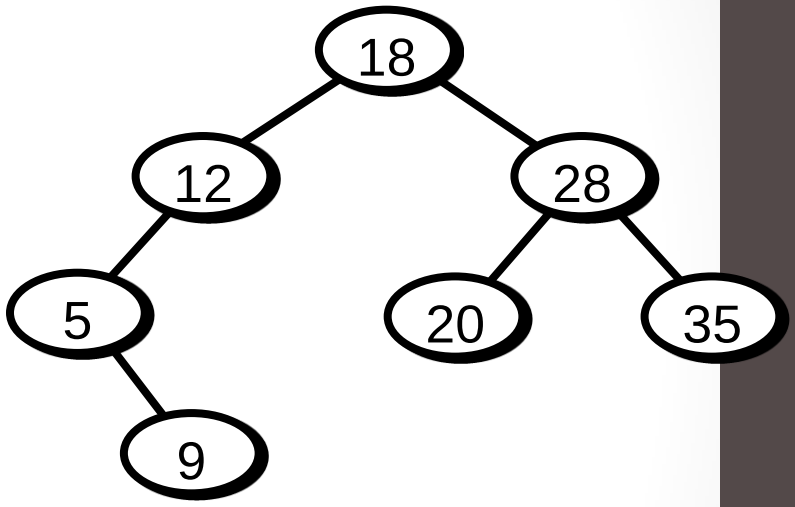
El recorrido en PostOrden del árbol es el siguiente:

4, 10, 6, 17, 22, 20, 15



Comparacion de Recorridos

- PreOrden RID
  - 18, 12, 5, 9, 28, 20, 35
- EnOrden IRD
  - 5, 9, 12, 18, 20, 28, 35
- PostOrden IDR
  - 9, 5, 12, 20, 35, 28, 18



Recorridos no recursivos

Emplear una pila como estructura de datos auxiliar.

PREORDEN:

- Se muestra la info del nodo raiz
- Entra a la pila el subárbol derecho diferente a nulo.
- Se recorre por el subárbol izquierdo procesando la información.
- Si al recorrer por la izq. Se encuentra nulo, entonces se saca de la pila y se repite el proceso.
- El algoritmo termina cuando se saca nulo de la pila o esta queda vacía.

ENORDEN:

- Se recorre por el sub árbol izquierdo introduciendo a la pila los nodos hasta encontrar un sub árbol nulo..
- Si al recorrer por la izq. Se encuentra nulo, entonces se saca de la pila, se procesa la información y se repite el proceso continuando con el sub árbol derecho.
- El algoritmo termina cuando se saca nulo de la pila o esta queda vacía.

```
Inicio preorden(raiz)
      Cima = 1
      Pila[cima] = nulo
      ptr = raiz
      Mientras ptr <> nulo hacer
            Mostrar ptr.info
            Si ptr.der <> nulo entonces
                                    Cima=cima+1
                                    Pila[cima] = ptr.der
            Fin_si
            Si ptr.izq <> nulo entonces
                  ptr = ptr^.izq
             Sino
                  ptr = pila[cima]
                  Cima=cima-1
            Fin_si
       Fin_mientras
Fin.
```

```
Inicio enorden(raiz)
        Cima = 1
        Pila[cima]=nulo
        Ptr = raiz
        Mientras ptr <> nulo hacer
          Mientras ptr <> nulo hacer
                Cima=cima+1
                Pila[cima] = ptr
                ptr = ptr.izq
          Fin_mientras
          ptr = pila[cima]
          Cima=cima-1
          Mientras ptr <> nulo hacer
                Mostrar ptr .info
                Si ptr.der <> nulo entonces
                     ptr = ptr.der
                     romper_ciclo
                Sino
                    ptr = pila[cima]
                    Cima=cima-1
                Fin_si
           Fin_mientras
         Fin_mientras
 Fin.
```

POSTORDEN:

- Se recorre por el sub árbol izquierdo introduciendo a la pila los nodos hasta encontrar un sub árbol nulo, a medida que se mete a la pila se debe marcar los nodos que tienen sub árbol derecho.

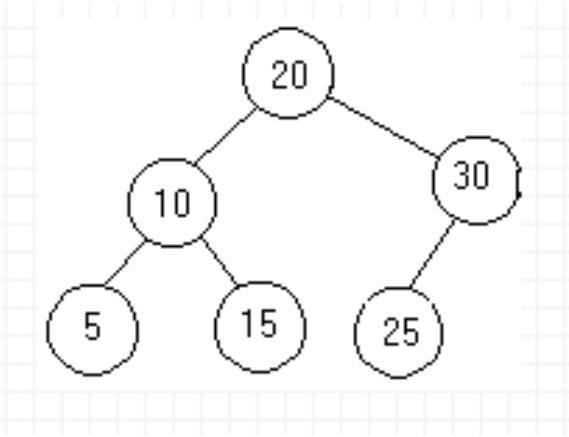
- Si al recorrer por la izq. Se encuentra nulo, entonces se saca de la pila un nodo, si el nodo no tiene sub árbol derecho, entonces se procesa la información y se continua sacando de la pila hasta encontrar un nodo con sub árbol derecho, en este caso se elimina la marca y se lo deja en la pila continuando el recorrido por la derecha del nodo.

- El algoritmo termina cuando se saca nulo de la pila o esta queda vacía.

```
//post orden no recursivo
void postden(nodo x){
    nodo pt=x;
    cima=0;
    pila[cima]=nulo;
    int control[20];
    control[cima]=1;
    while (pt!=null) {
            cima++;
            pila[cima]=pt;
            if (pt.der!=null)
                    control[cima]=1;
            else
                    control[cima]=0;
            pt=pt.izq;
            if (pt==null){
                    pt=pila[cima];
                    while (control[cima]==0){
                        mostrar(pt.info);
                        cima--;
                         pt=pila[cima];
                    }
                    control[cima]=0;
                    if (cima==0)
                              pt=null;
                    else
                             pt=pt.der;
            }
  }
}
```

RECORRIDO EN AMPLITUD.

En el recorrido por amplitud se visitan los nodos por niveles. Para ello se utiliza una estructura auxiliar tipo "cola" en la que después de mostrar el contenido del nodo, empezando por el nodo raíz, se almacenan los punteros correspondientes a sus hijos izquierdo y derecho.

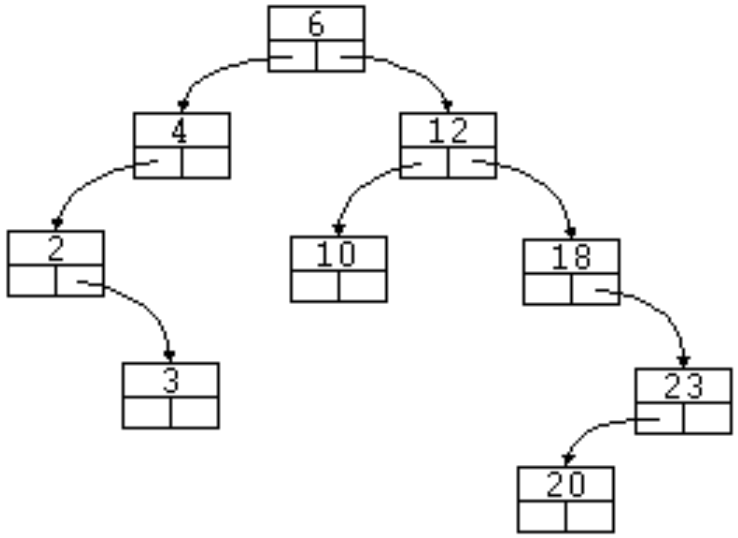


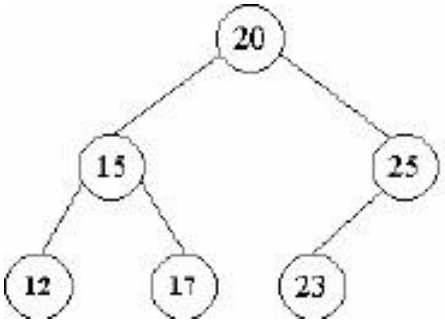
Resultado:
20, 10, 30, 5, 15, 25

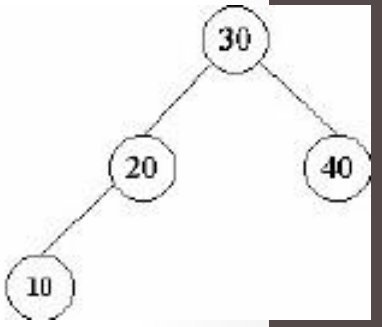
```
Amplitud
Inicio (raiz)
      Ptr = raiz
     final = 1
     Inicio=1
     cola[final] = Ptr
     Mientras (no colavacia) hacer
          ptr=cola[Inicio]
          inicio++
          Mostrar ptr.inf
          Si ptr.izq <> nulo entonces
                                    final=final+1
                                    cola[final] = ptr.izq
         Fin_si

          Si ptr.der <> nul0 entonces
                              final=final+1
                              cola[final] = ptr.der
           Fin_si
     Fin_mientras
Fin.
```

Árboles binarios de búsqueda (ABB)

Se trata de árboles de Grado 2 en los que se cumple que para cada nodo, el valor de la clave de la raíz del subárbol izquierdo es menor que el valor de la clave del nodo y que el valor de la clave raíz del subárbol derecho es mayor que el valor de la clave del nodo.







Definición: un árbol binario de búsqueda es un árbol binario, que puede estar vacío, y que si es no vacío cumple las siguientes propiedades:

- (1) Todos los nodos están identificados por una clave y no existen dos elementos con la misma clave.
- (2) Las claves de los nodos del subárbol izquierdo son menores que la clave del nodo raíz.
- (3) Las claves de los nodos del subárbol derecho son mayores que la clave del nodo raíz.
- (4) Los subárboles izquierdo y derecho son también árboles binarios de búsqueda.

Operaciones en ABB

- Buscar un elemento.
- Insertar un elemento.
- Borrar un elemento.
- Movimientos a través del árbol:
  - Izquierda.
  - Derecha.
  - Raiz.
- Información:
  - Comprobar si un árbol está vacío.
  - Calcular el número de nodos.
  - Comprobar si el nodo es hoja.
  - Calcular la altura de un nodo.
- Calcular la altura de un árbol.

Buscar un elemento

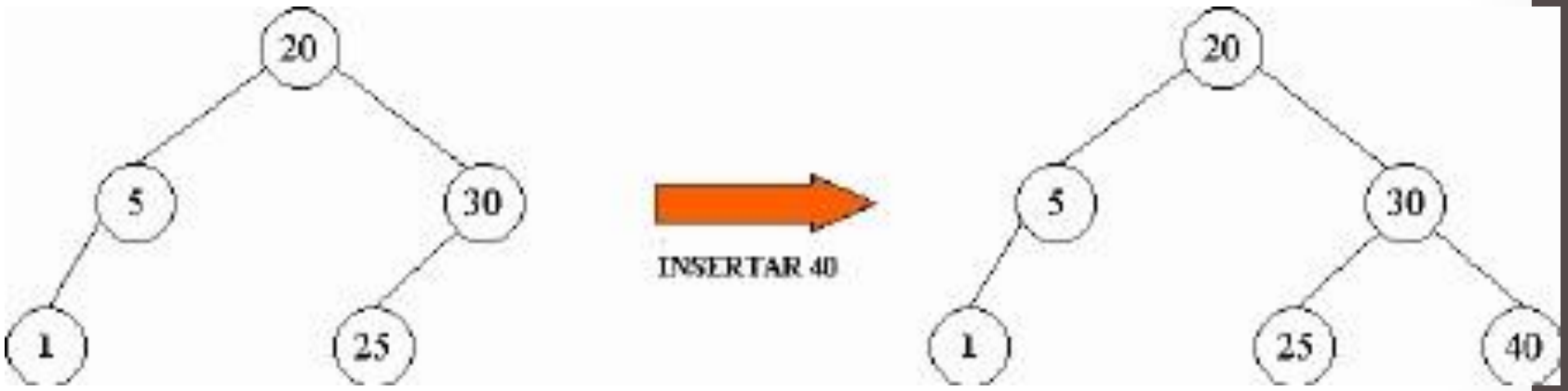
Partiendo siempre del nodo raíz, el modo de buscar un elemento se define de forma recursiva.

- Si el árbol está vacío, terminamos la búsqueda: el elemento no está en el árbol.
- Si el valor del nodo raíz es igual que el del elemento que buscamos, terminamos la búsqueda con éxito.
- Si el valor del nodo raíz es mayor que el elemento que buscamos, continuaremos la búsqueda en el árbol izquierdo.
- Si el valor del nodo raíz es menor que el elemento que buscamos, continuaremos la búsqueda en el árbol derecho

```
Inicio  Buscar (raiz, clave)
  si (raiz= NULO) entonces
       devolver(NULO)
   sino
       si (clave= raiz.info) entonces
            devolver(raiz)
       sino
           si (clave < raiz.info) entonces
                devolver(Buscar(raiz.Izq,clave))
           sino
                devolver(Buscar(raiz.der,clave))
           fin_si
       fin_si
  fin_si
Fin
```

Insertar un elemento

La inserción de un nuevo nodo en un árbol binario de búsqueda debe realizarse de tal forma que se mantengan las propiedades del árbol.



La inserción de un nodo se realiza en un arbol vacio, por lo tanto se debe buscar el sub-arbol vacio donde se inserta el nuevo nodo.

Algoritmo recursivo:

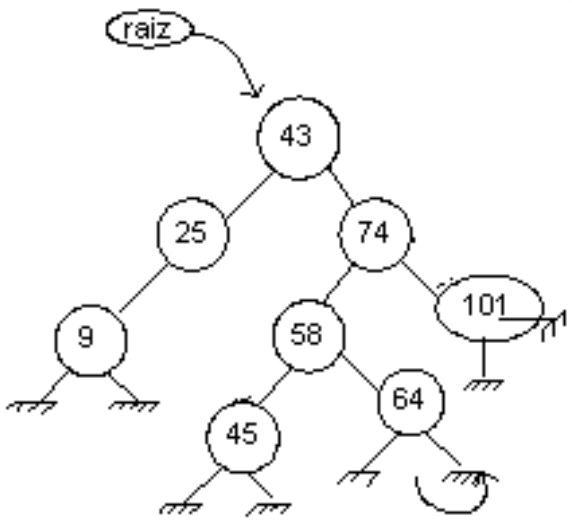
```
Inicio_adicion(raiz, clave)
        Si (raiz=nulo) entonces
              Pedir memoria en raiz
              Raiz.info=clave
              raiz.izq=raiz.der=nulo
        Sino
              Si (clave >raiz.info)  entonces
                        Adicion (raiz.der, clave)
              Sino
                   si (clave<raiz.info) entonces
                        Adicion(raiz.izq, clave)
                   fin si
              Fin si
      Fin si
 Fin
```

Eliminar un elemento

Al borrar un nodo de un ABB, se debe tener cuidado de no distorcionar la estructura del arbol.

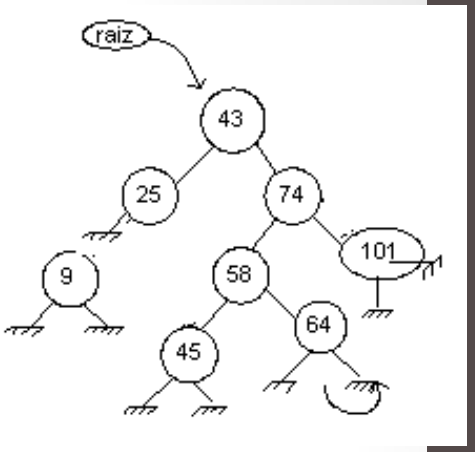
3 posibilidades:

- Borrar un nodo Hoja.
- Borrar un Nodo con un hijo.
- Borrar un Nodo con dos Hijos



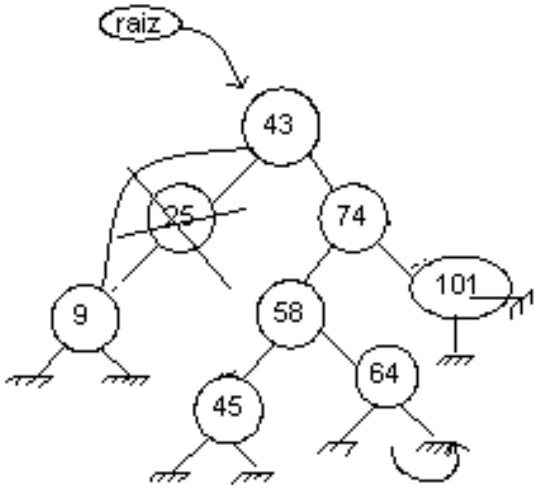
Borrar un nodo Hoja:

Se retorna al nodo padre el valor NULO



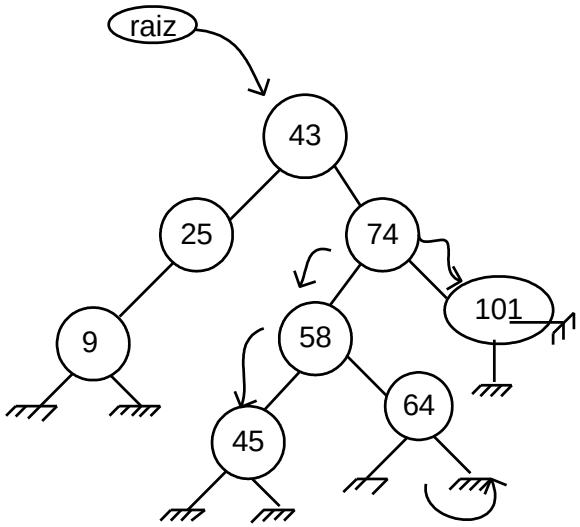
Borrar un nodo con 1 hijo:

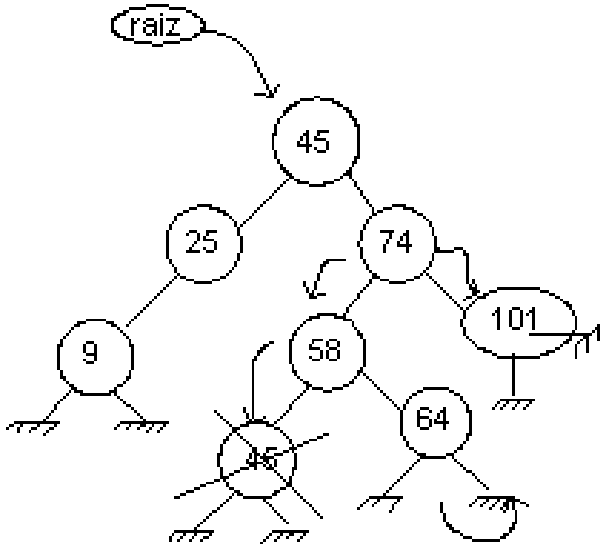
Se retorna al nodo padre la direccion del unico hijo



Borrar un nodo con dos hijos

- Buscar el menor de los mayores del nodo a eliminar. (Tiene como maximo un descendiente en el s.a derecho).
- Reemplazar el menor de los mayores al nodo que se va a eliminar.
- Eliminar el nodo que contiene el menor de los mayores.



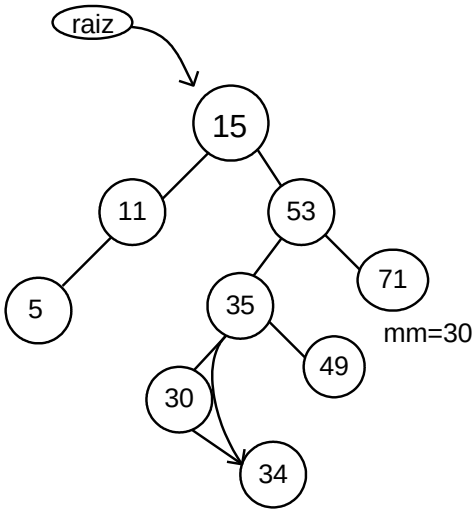


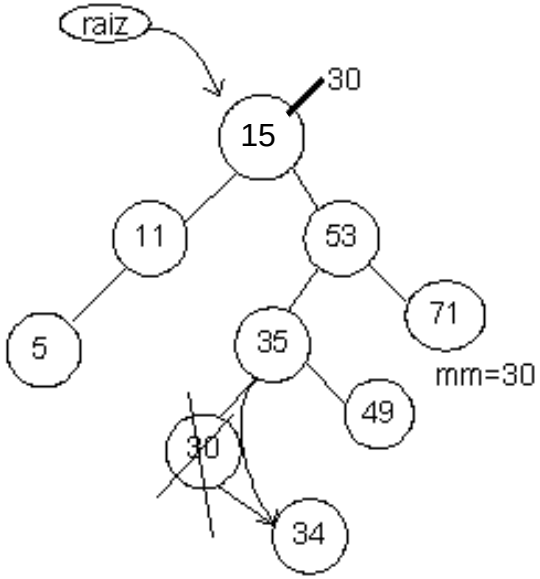
Clave = 43

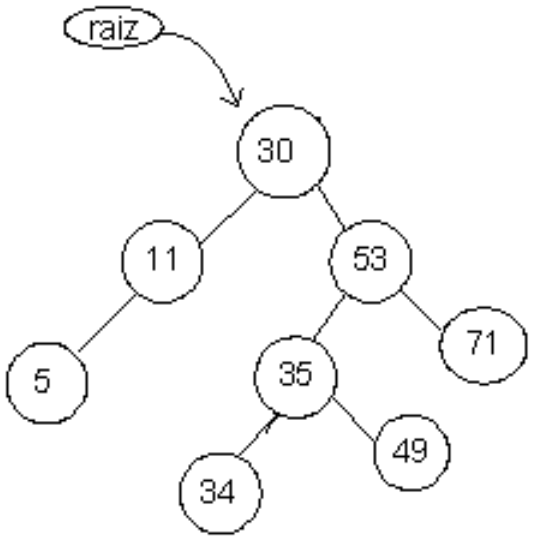
```
inicio eliminacion(raiz,clave)
    si raiz <>nulo entonces
            si (clave=raiz.info) entonces
                    eli(raiz)
            si no
                    si clave> raiz.info entonces
                                eliminacion (raiz.der,clave)
                    si no
                                eliminacion (raiz.izq,clave)
                    fin si
            fin si
     fin si
fin
```

```
inicio  eli(raiz)
    si(raiz.izq=nulo y raiz.der=nulo) entonces
            liberarnodo(raiz)
             retornar nulo
    si no
          si(raiz.izq=nulo y raiz.der<>nulo) entonces
              retornar(raiz.der)
          si no
              si(raiz.izq<>nulo y raiz.der=nulo) entonces
                    retornar(raiz.izq)
              si no
                    mm=menmay(raiz.der)
                    raiz.info=mm
                    eliminacion(raiz.der,mm)
              fin si
            fin si
     fin si
 final
```

```
inicio menmay(raiz)
    si raiz.izq=nulo entonces
            retornar  raiz.info
    si no
            menmay(raiz.izq)
     fin si
final
```



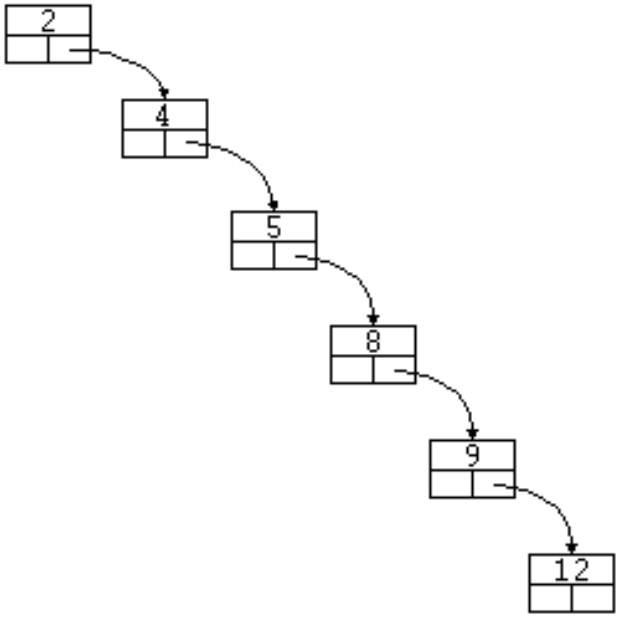


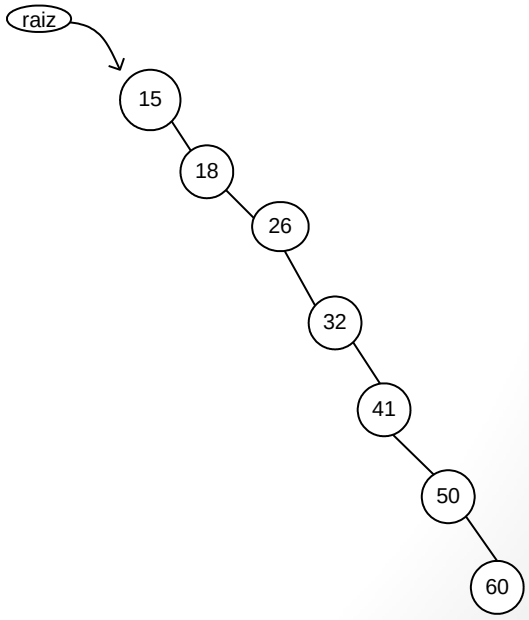


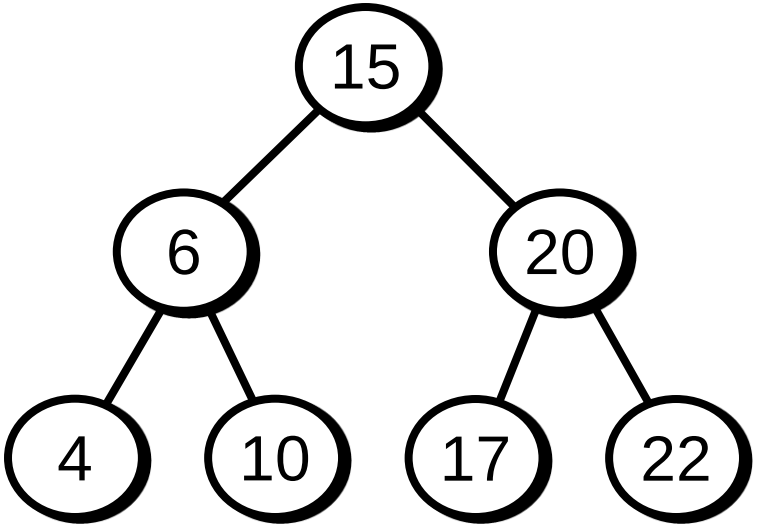
ARBOLES BALANCEADOS (AVL)

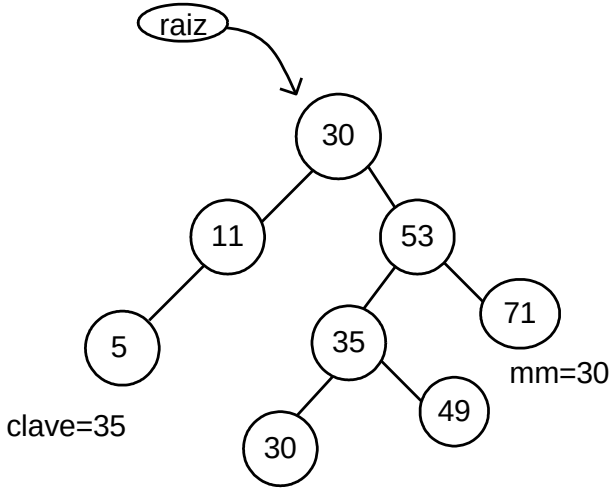
Un árbol AVL (llamado así por las iniciales de sus inventores: Adelson-Velskii y Landis) es un árbol binario de búsqueda en el que para cada nodo, las alturas de sus sub árboles izquierdo y derecho no difieren en más de 1.

ARBOL ABB DEGENERADO






15
6
20
4
10
17
22


raiz
30
11
53
5
35
71
49
30
clave=35
mm=30

Cada vez que insertamos o eliminamos un nodo en un árbol AVL pueden suceder dos cosas:

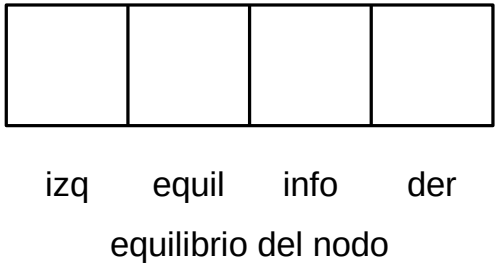
- que el árbol se mantenga como AVL
- o que pierda esta propiedad.

En el segundo caso, recuperaremos el estado AVL aplicando la rotación adecuada, es decir, se debe reestructurar el abol.

Se necesita añadir un nuevo miembro a cada nodo del árbol, para averiguar si el árbol sigue siendo AVL, **el Factor de Equilibrio.**

Cada vez que se inserta o elimina un nodo se debe recorrer el camino desde ese nodo hacia el nodo raíz actualizando los valores del factor de equilibrio de cada nodo. Cuando uno de esos valores sea 2 ó -2 se aplica la rotación correspondiente.

ESTRUCTURA DEL NODO:



Cada nodo, además de la información que se pretende almacenar, debe tener los dos punteros a los árboles derecho e izquierdo, igual que los ABB, y además un miembro nuevo: el factor de equilibrio (equi).

El factor de equilibrio es la diferencia entre las alturas del árbol derecho y el izquierdo:

equi = altura subárbol derecho - altura subárbol izquierdo

Por definición, para un árbol AVL, este valor debe ser -1, 0 ó 1.

Equi = -1

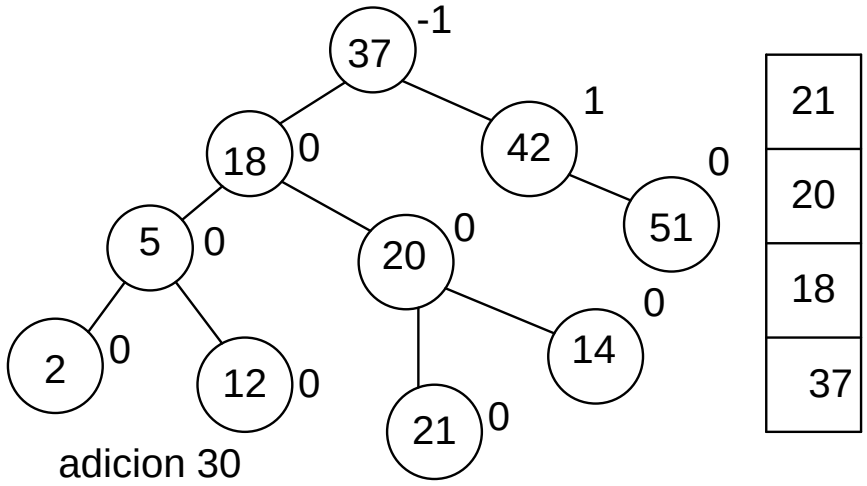
Cuando la altura del sub arbol izquierdo es mayor en 1 que la altura del sub arbol derecho

Equi = 0

Cuando la altura del sub arbol izquierdo es igual que la altura del sub arbol derecho

Equi = 1

Cuando la altura del sub arbol derecho es mayor en 1 que la altura del sub arbol izquierdo

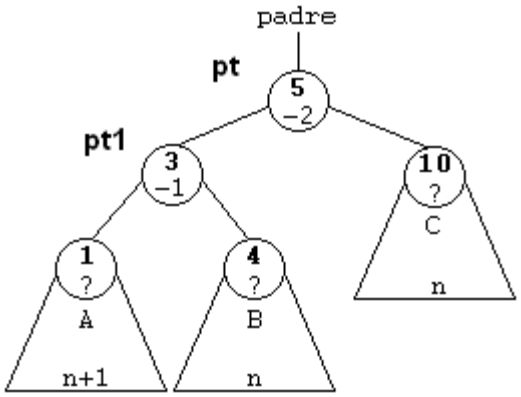


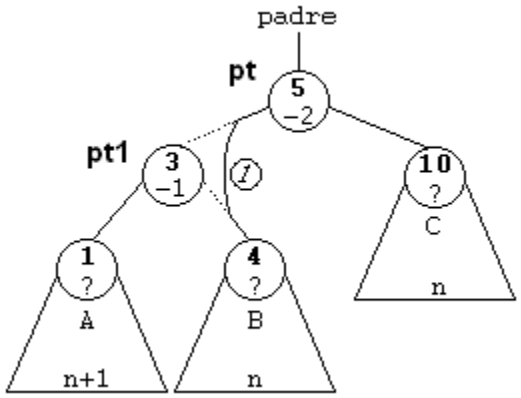
ALGORITMOS DE ROTACION

Los reequilibrados se realizan mediante rotaciones, existe cuatro posibles rotaciones que se puede aplicar.

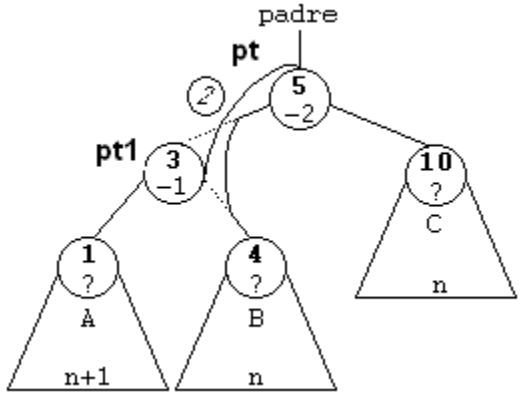
Rotación simple a la derecha (SD):

Esta rotación se usa cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su **equi** sea de -2. Y además, la raíz del subárbol izquierdo tenga un equi de -1, es decir, que esté cargado a la izquierda.

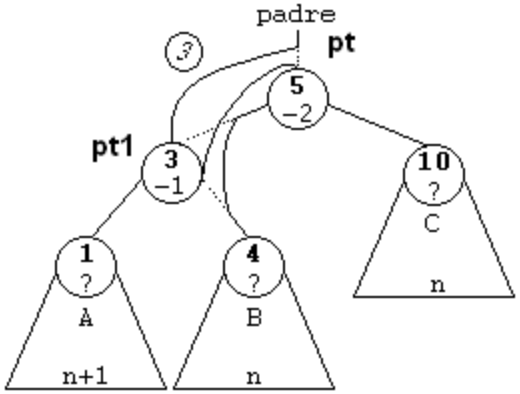




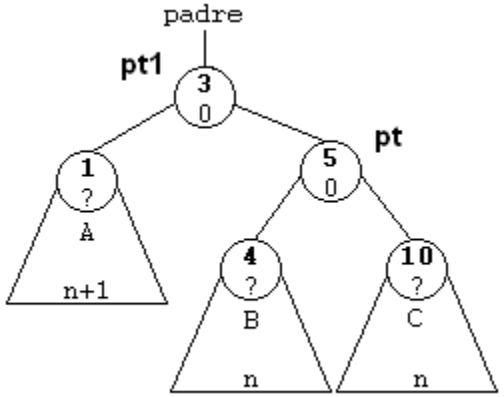
Pt.izq =pt1.der



Pt1.der =pt

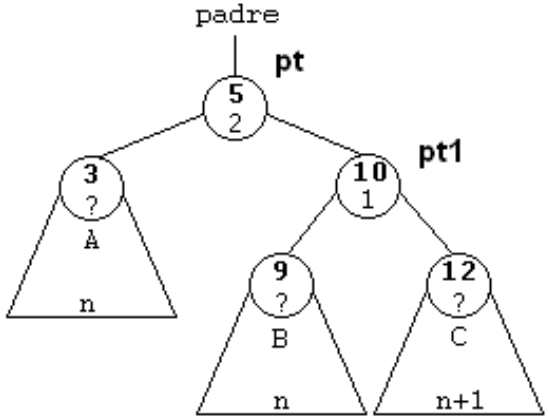


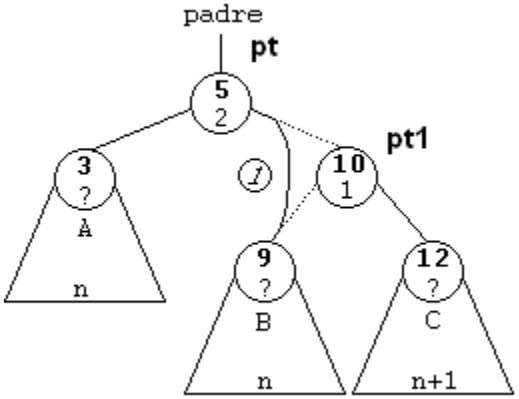
Pt =pt1



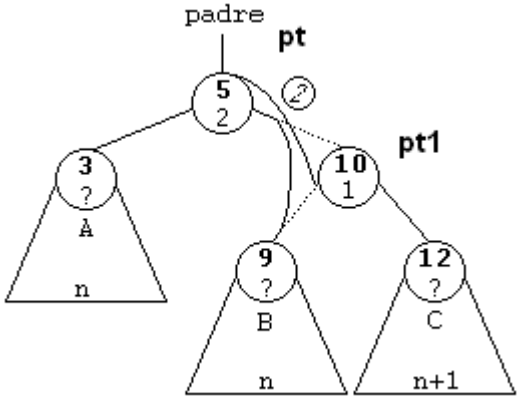
Rotación simple a la izquierda (SI):

Se trata del caso simétrico del anterior. Esta rotación se usa cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su equi sea de 2. Y además, la raíz del subárbol derecho tenga un equi de 1, es decir, que esté cargado a la derecha.

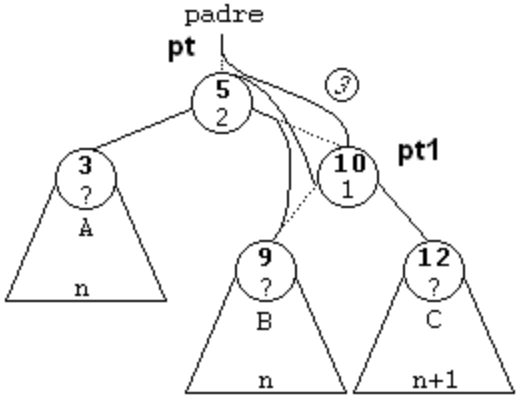




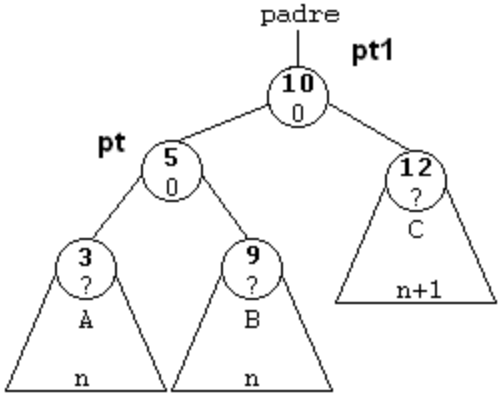
Pt.der =pt1.izq



Pt1.izq =pt

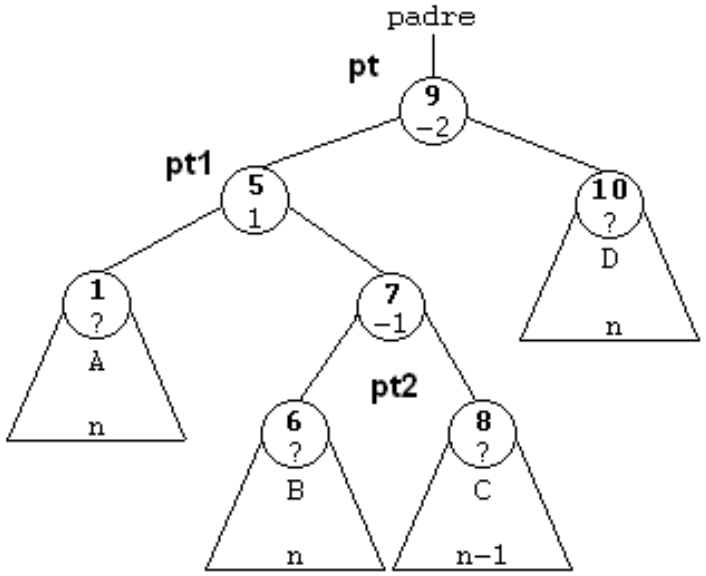


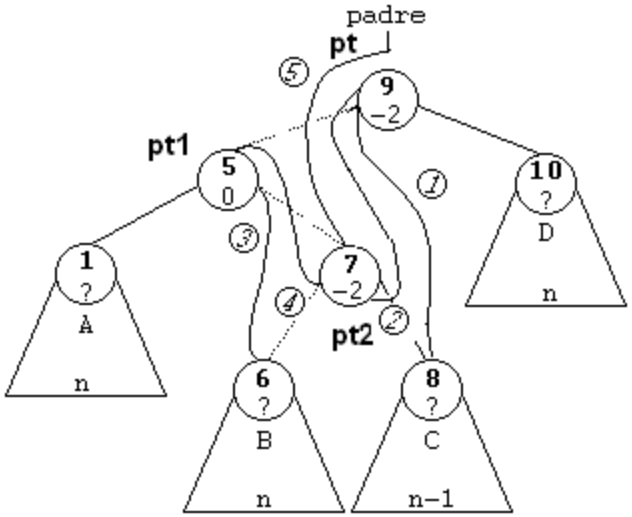
Pt =pt1



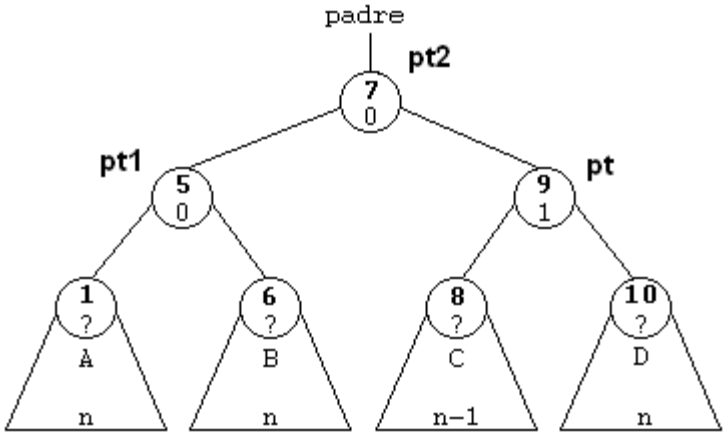
Rotación doble a la derecha (DD):

Esta rotación se usa cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su equi sea de -2. Y además, la raíz del subárbol izquierdo tenga un equi de 1, es decir, que esté cargado a la derecha.



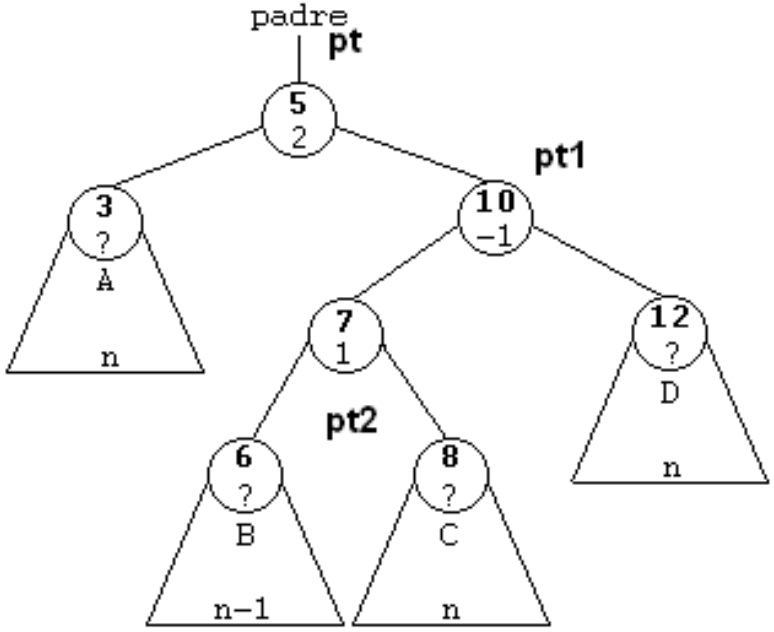


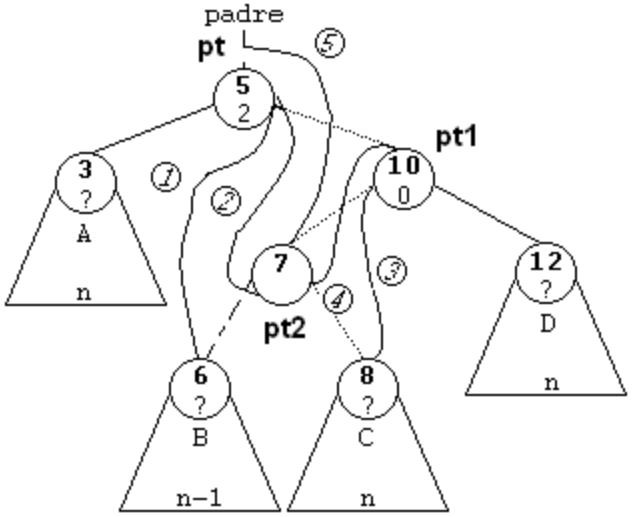
Pt.izq = pt2.der
Pt2.der = pt
Pt1.der = pt2.izq
Pt2.izq = pt1
Pt = pt2



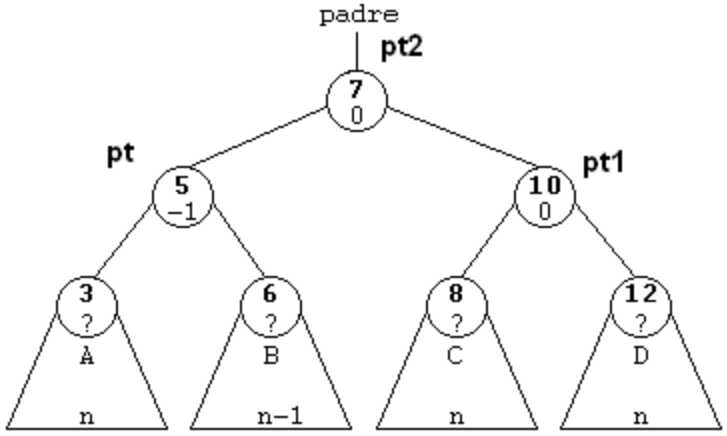
Rotación doble a la izquierda (DI):

Esta rotación se usa cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su equi sea de 2. Y además, la raíz del subárbol derecho tenga un equi de -1, es decir, que esté cargado a la izquierda. Se trata del caso simétrico del anterior.





Pt.der = pt2.izq
Pt2.izq = pt
Pt1.izq = pt2.der
Pt2.der = pt1
Pt = pt2



Reequilibrados en árboles AVL por inserción de un nodo

Al recorrer por la DERECHA para la adicion: equi se incrementa en 1

- Si equi = 0 entonces equi = 1
- Si equi = -1 entonces equi = 0
- Si equi = 1 entonces equi = 2 ROTACION A LA IZQUIERDA

Al recorrer por la IZQUIERDA para la adicion: equi se decrementa en 1

- Si equi = 0 entonces equi = -1
- Si equi = 1 entonces equi = 0
- Si equi = -1 entonces equi = -2 ROTACION A LA DERECHA

Equi nodo actual	equi del nodo derecho	equi del nodo izquierdo	Rotación
-2	No importa	-1	RSD
-2	No importa	1	RDD
2	-1	No importa	RDI
2	1	No importa	RSI

Reequilibrados en árboles AVL por borrado de un nodo

equi nodo actual	equi del nodo derecho	equi del nodo izquierdo	Rotación
-2	No importa	-1	RSD
-2	No importa	0	RSD
-2	No importa	1	RDD
2	-1	No importa	RDI
2	0	No importa	RSI
2	1	No importa	RSI