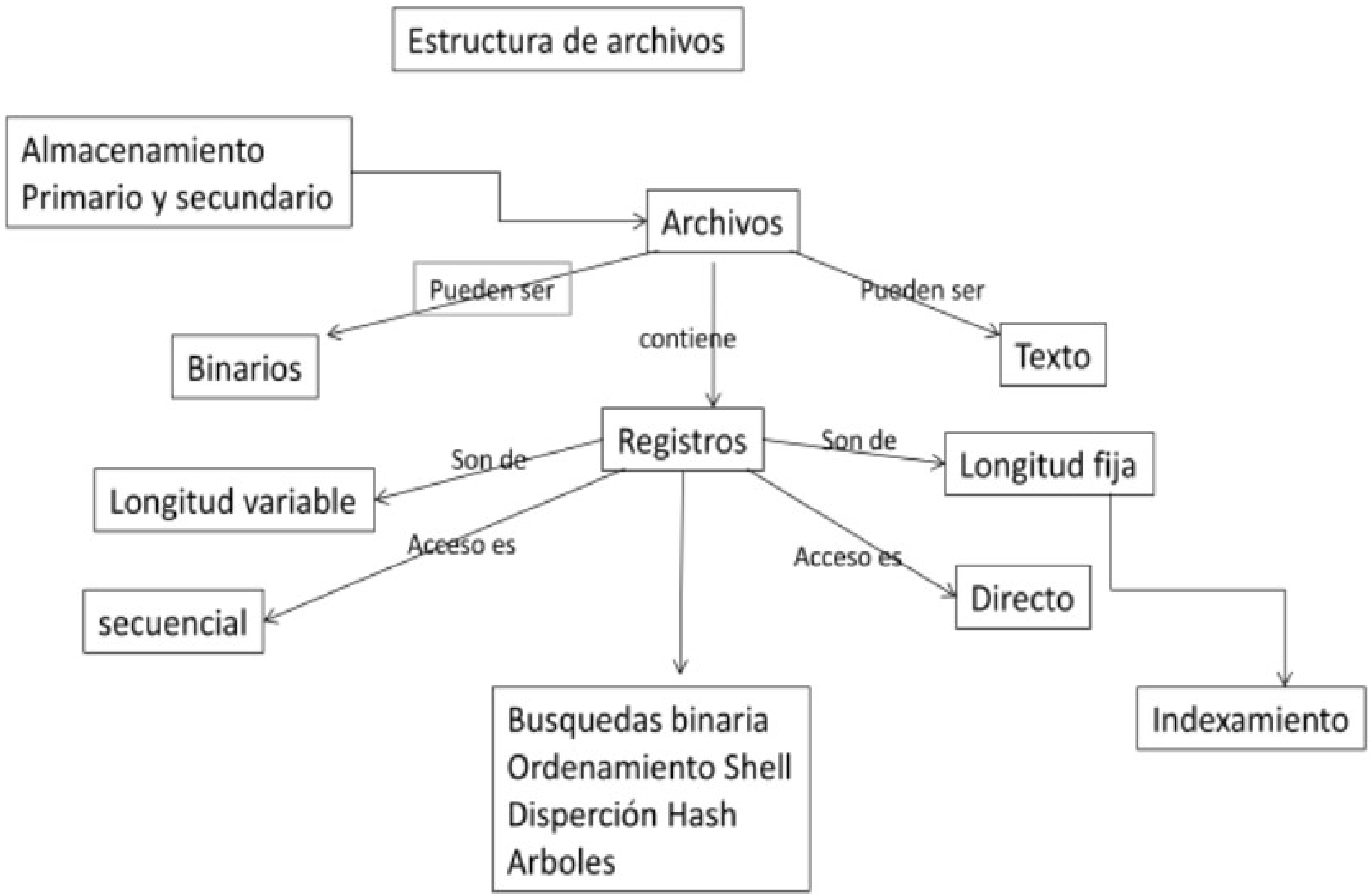
Conceptos

ESTRUCTURA CONCEPTUAL



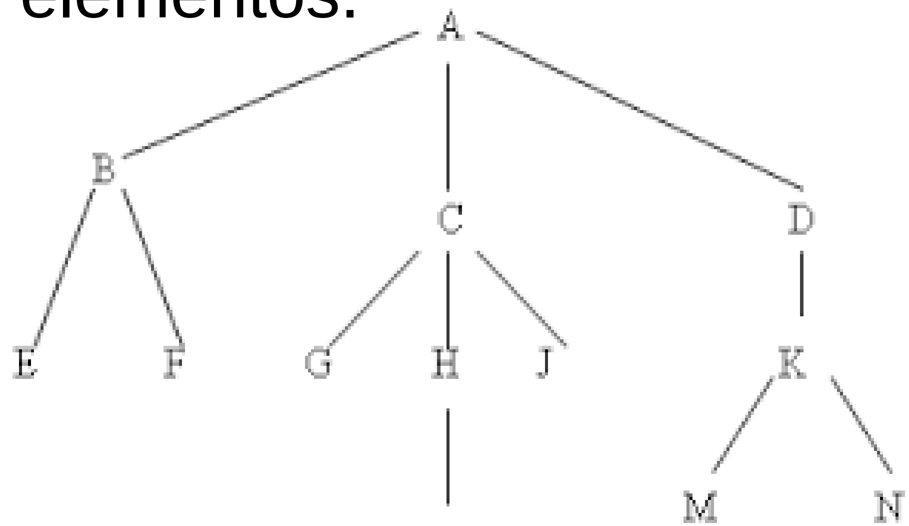
Arboles

Repaso

Los árboles representan estructuras no lineales y dinámicas.

Dinámica: puesto que la estructura del árbol puede variar durante la ejecución del programa.

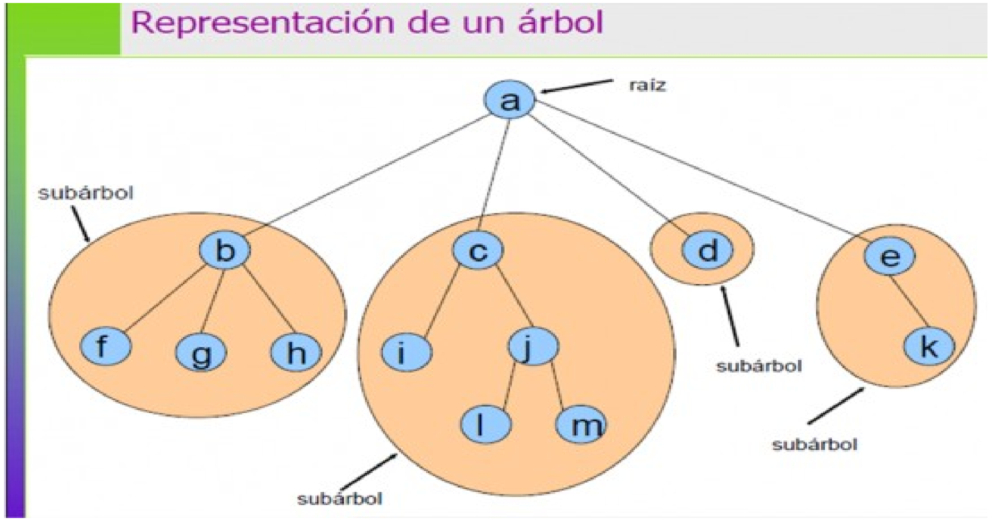
No lineales: Porque a cada elemento del árbol puede seguirle varios elementos.



Arboles

Definición

Un árbol es una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos; uno de los cuales se denomina raíz.



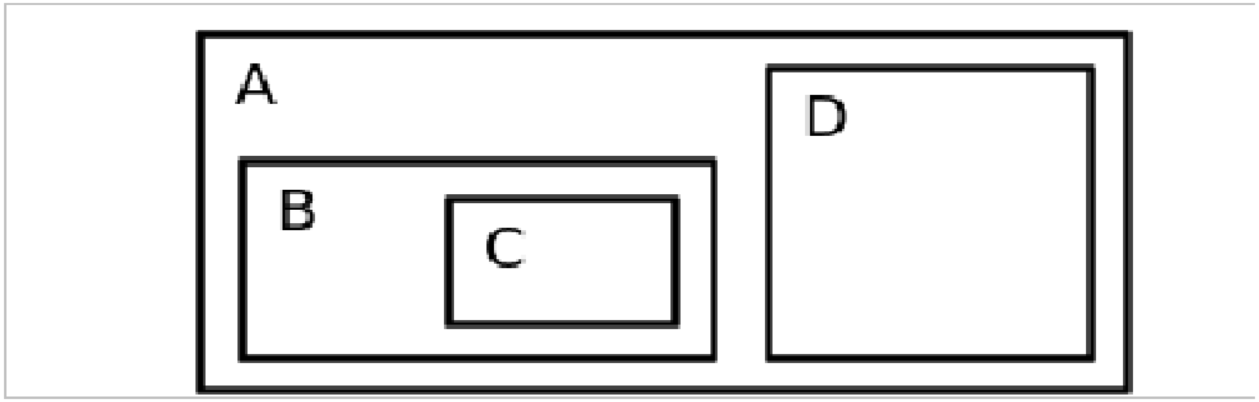
Formalmente se define un árbol de tipo T, como una estructura homogénea que es la concatenación de un elemento de tipo T junto con un número finito de árboles disjuntos llamados subárboles. Una forma particular de árbol puede ser la estructura vacía.

Arboles

Representación de Arboles

Se pueden representar como:

a) Diagramas de Venn



b) Anidación de paréntesis
(A((B(C))D))

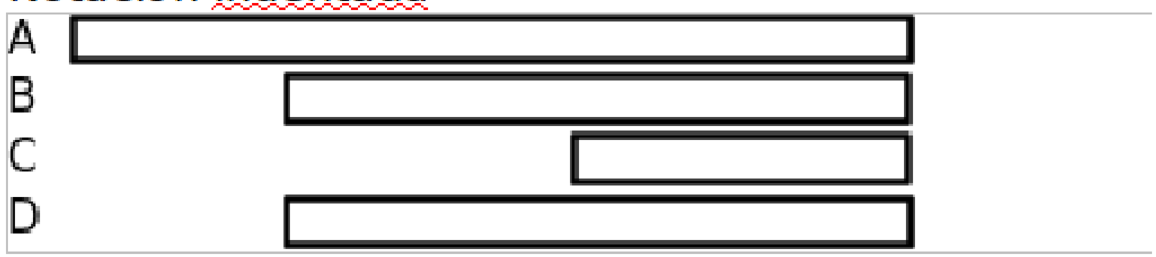
c) Por notación decimal
1.A, 1.1.B, 1.1.1.C, 1.2D

Arboles

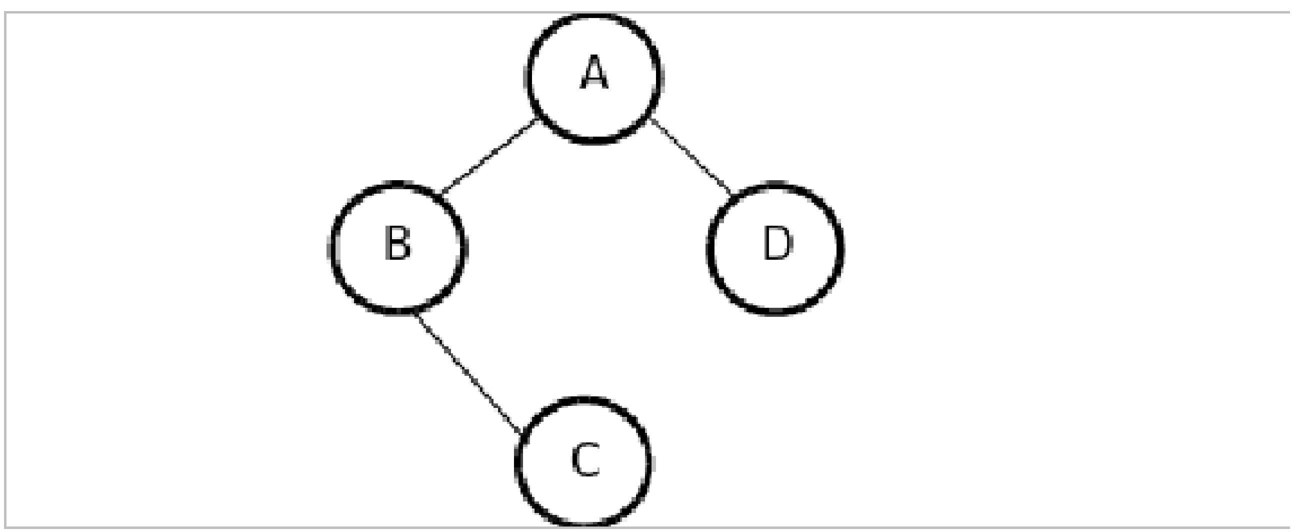
Representación de Arboles

Se pueden representar como: cont.

d) Notación indentada



e) Grafos



Arboles

Características y Propiedades de los Árboles

a. Todo árbol que no es vacío, tiene un único nodo raíz

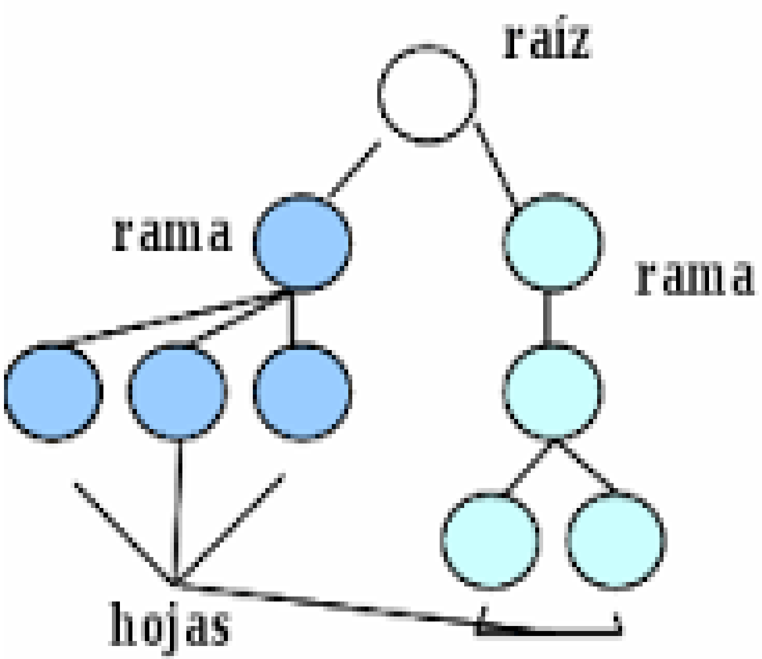
b. Sí un nodo X es descendiente de un nodo Y,
decimos que X es hijo de Y

c. Sí un nodo X es antecesor directo de un nodo Y,
decimos que X es padre de Y

d. Todos los nodos de un mismo padre,
son hermanos

e. Todo nodo que no tiene hijos
es un nodo terminal

f. Todo nodo que no es raíz, ni terminal,
es un nodo interior



Arboles

Características y Propiedades de los Árboles

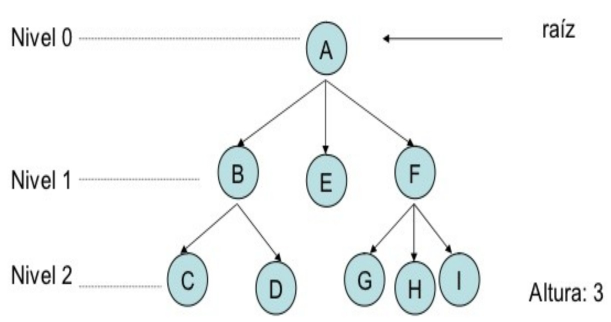
g. Grado es el número de descendientes directos de un determinado nodo.

h. Grado de un árbol es el máximo grado de todos los nodos de un árbol.

h. Nivel es el número de nodos que deben ser recorridos para llegar a un determinado nodo (desde el raíz)

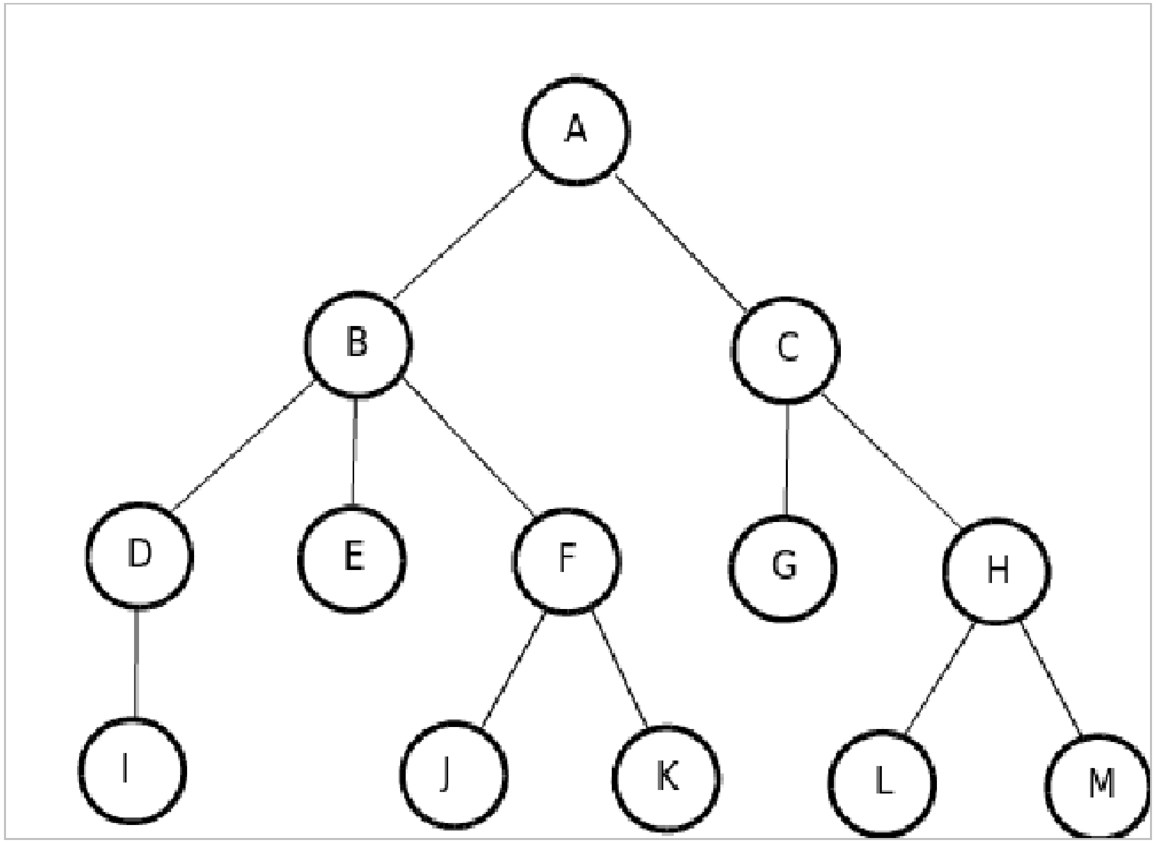
i. Altura del árbol es el máximo número de niveles de entre todos las ramas del árbol más 1.

- Si el árbol no esta vacío, entonces el primer nodo se llama **raíz.**



Ejemplo de Arboles

Ejemplo:



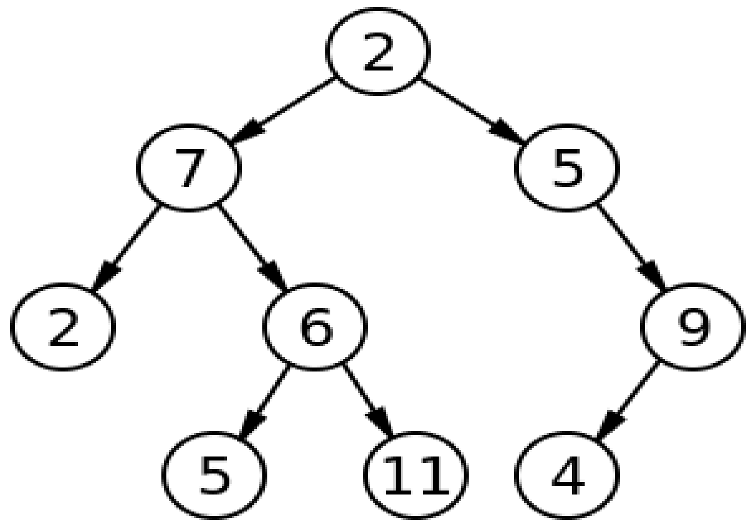
Completar:

Raíz	
Hermanos	
Terminales	
Interiores	
Grado	
Grado del árbol	
Nivel	
Altura	

Arboles Binarios

Definición

Un árbol de grado 2, es un árbol donde cada nodo puede tener como máximo 2 subárboles,
siendo necesario distinguir entre el subárbol derecho y el subárbol izquierdo.

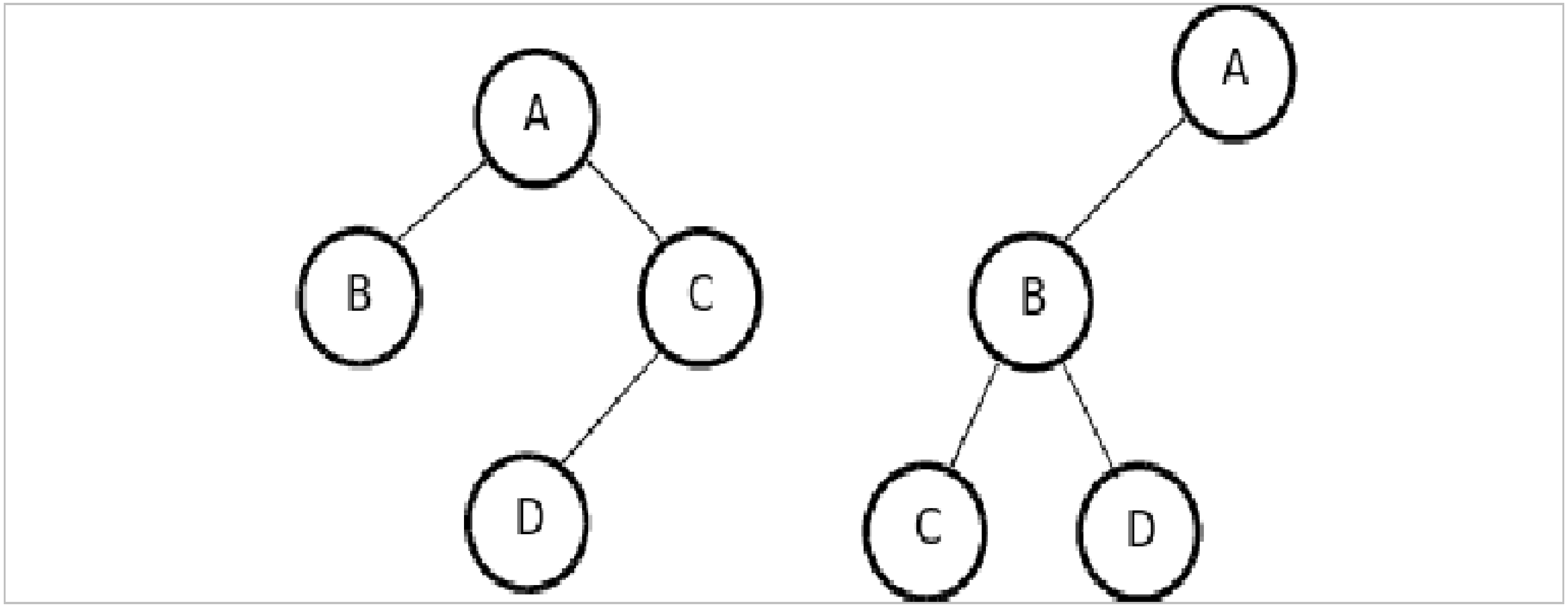


- Los arboles binarios son arboles de orden o Grado 2.
- Cada nodo puede tener como máximo dos hijos.

Arboles Binarios

Arboles Binarios Distintos

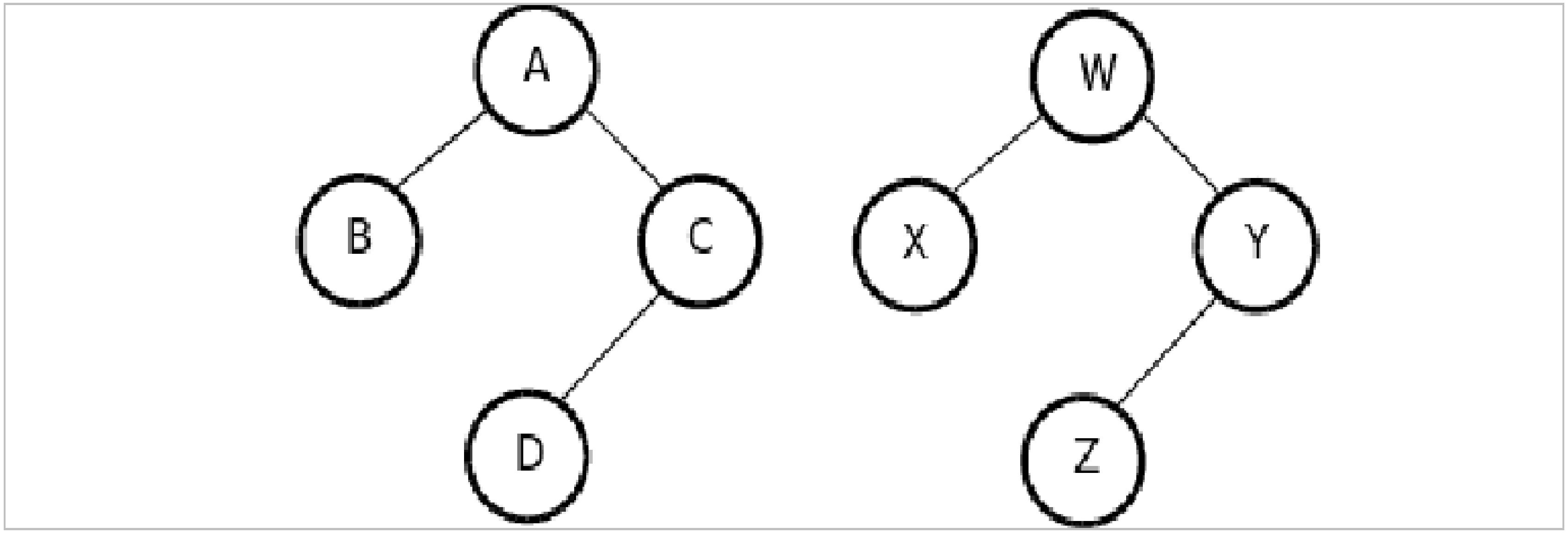
Son distintos cuando sus estructuras son diferentes



Arboles Binarios

Arboles Binarios Similares

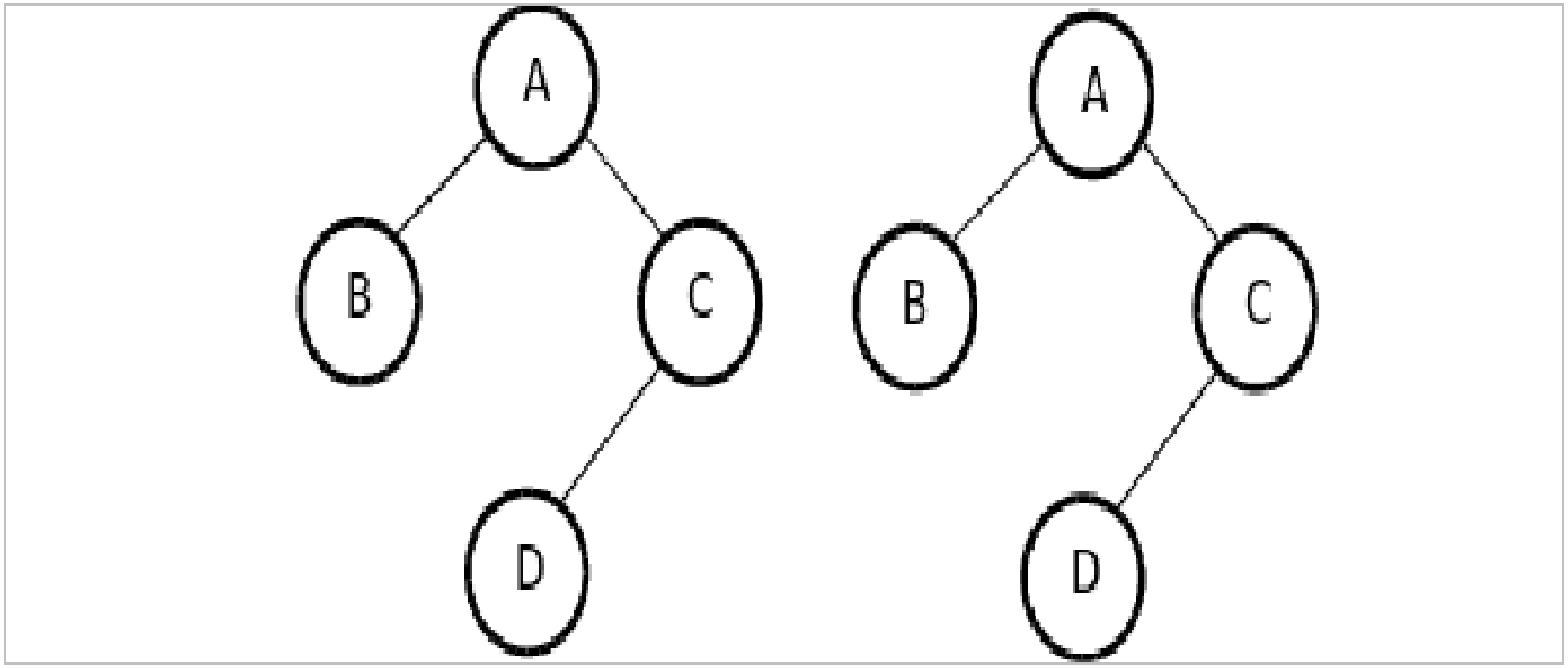
Son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos difieren entre sí.



Arboles Binarios

Arboles Binarios Equivalentes

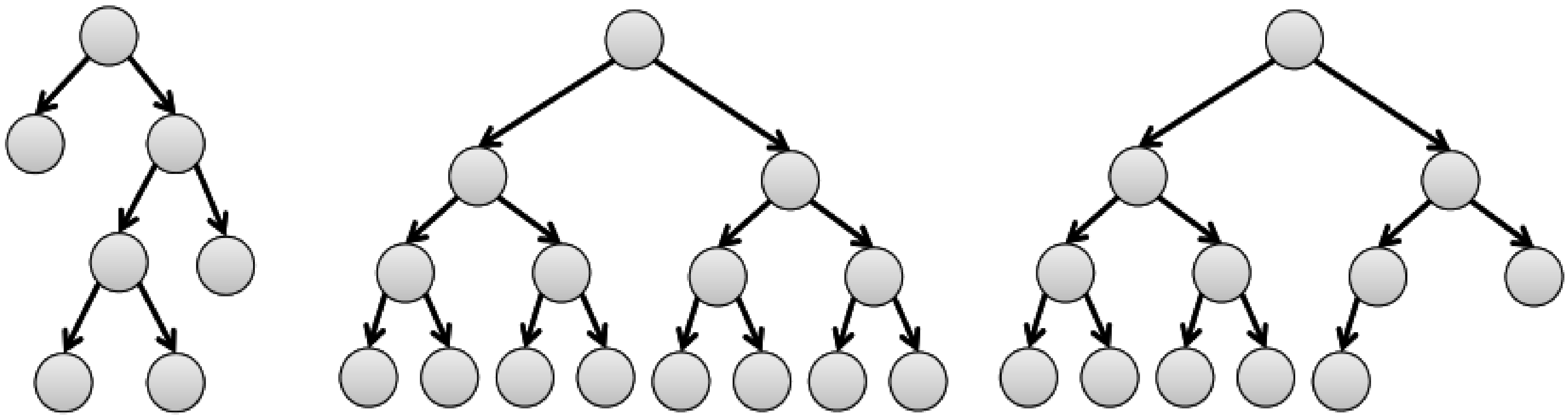
Son equivalentes cuando son similares y además los nodos contienen la misma información.



Arboles Binarios

Arboles Binarios Variantes

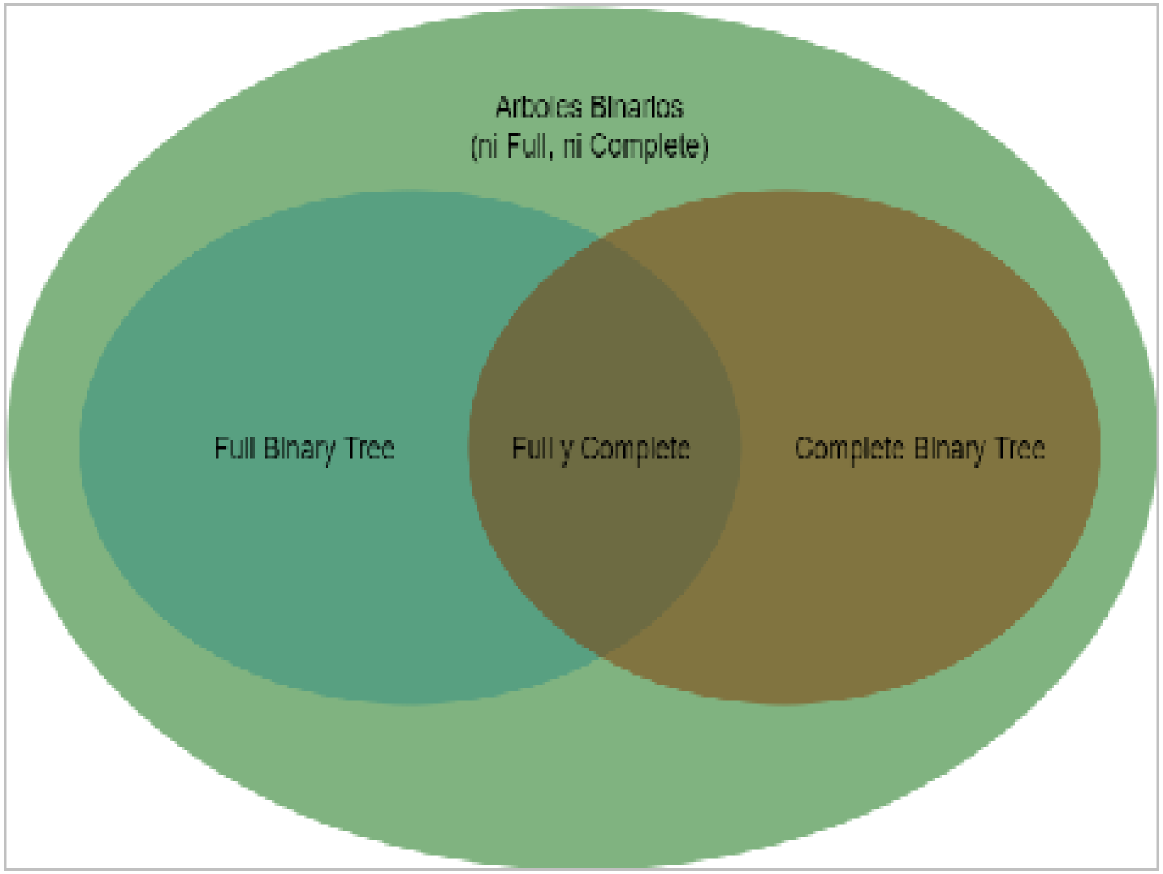
- **Árbol estricto**: Si un subárbol está vacío, el otro también. Cada nodo puede tener 0 ó 2 hijos.
- **Árbol lleno**: Árbol estricto donde en cada nodo la altura del subárbol izquierdo es igual a la del derecho, y ambos subárboles son árboles llenos.
- **Árbol completo**: Arbol lleno hasta el penúltimo nivel. En el último nivel los nodos están agrupados a la izquierda.



Arboles Binarios

Universo de los Arboles Binarios

El universo de árboles binarios se puede solapar, obteniendo el siguiente diagrama:

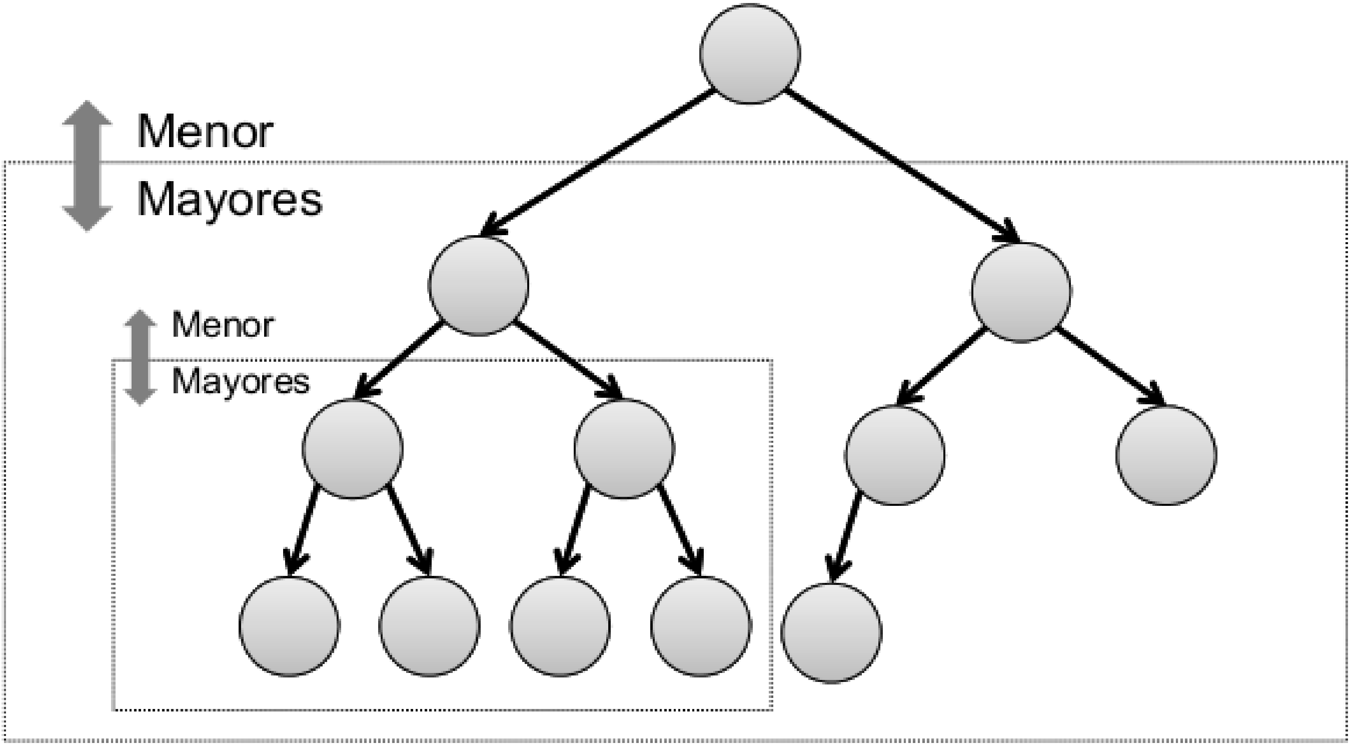


Arboles Binarios

Montículo Binarios

Un **montículo** (binario) es un **arbol completo** cuyos nodos almacenan elementos comparables mediante $\leq$ y donde todo nodo cumple la **propiedad de montículo**:

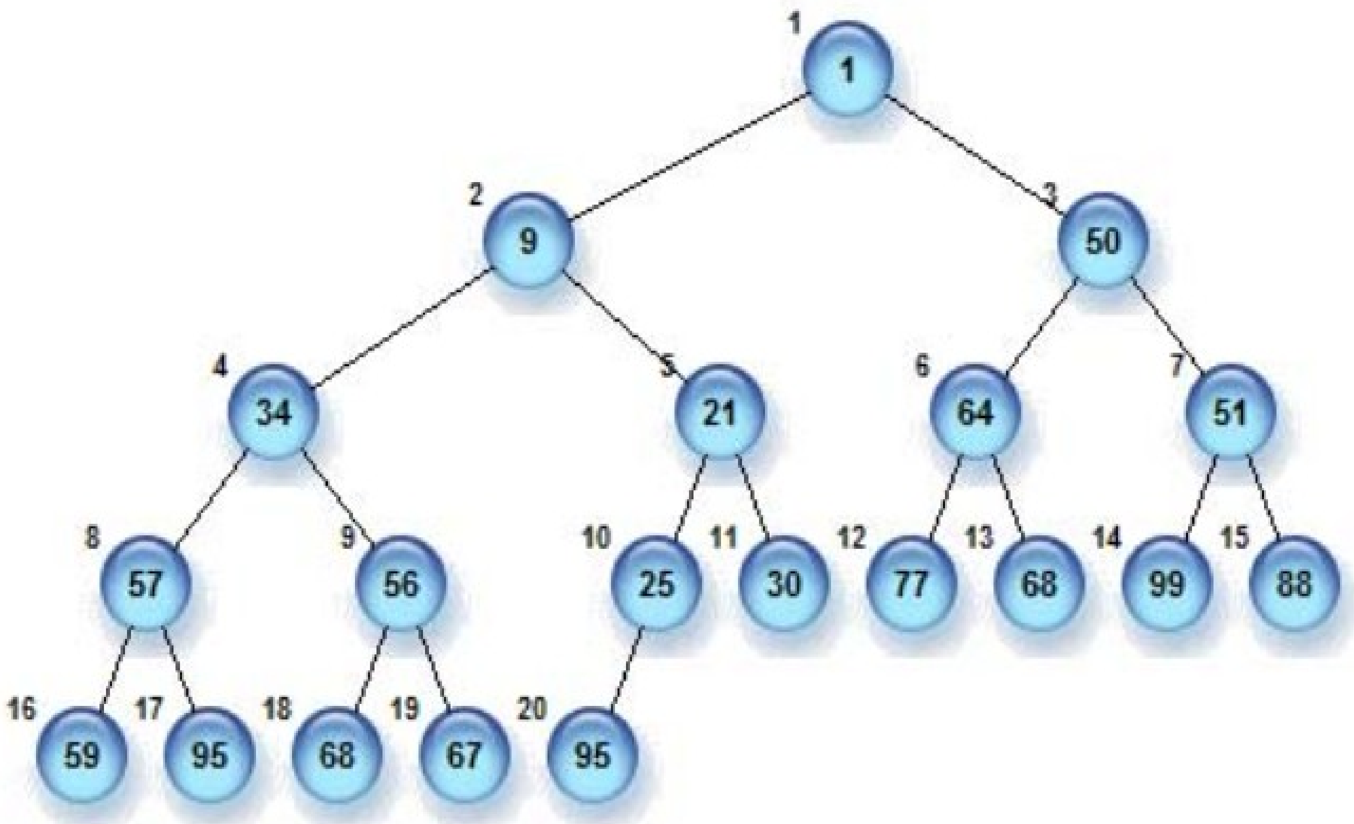
Propiedad de montículo: Todo nodo es menor que sus descendientes. (montículo **de mínimos**).



Arboles Binarios

Montículo Binarios

Ejemplo:



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	9	50	34	21	64	51	57	56	25	30	77	68	99	88	59	95	68	67	95

Arboles Binarios

Montículo Binarios

Propiedades:

- El **nodo raíz** (en primera posición del vector) es el **mínimo**.
- La **altura** de un montículo es **logarítmica** respecto al número de elementos almacenados (por ser arbol completo).
- Si un sólo elemento no cumple la propiedad de montículo, es posible restablecer la propiedad mediante **ascensos** sucesivos en el árbol (intercambiándole con su padre) o mediante **descensos** en el árbol (intercambiándole con el mayor de sus hijos). El número de operaciones es proporcional a la **altura**.
- Para **insertar** un nuevo elemento se situa al final del vector (última hoja del árbol) y se **asciende** hasta que cumpla la propiedad.
- Para **eliminar la raiz** se intercambia con el último elemento (que se elimina en O(1)) y se **desciende** la nueva raiz hasta que cumpla la propiedad.

Montículo Binarios

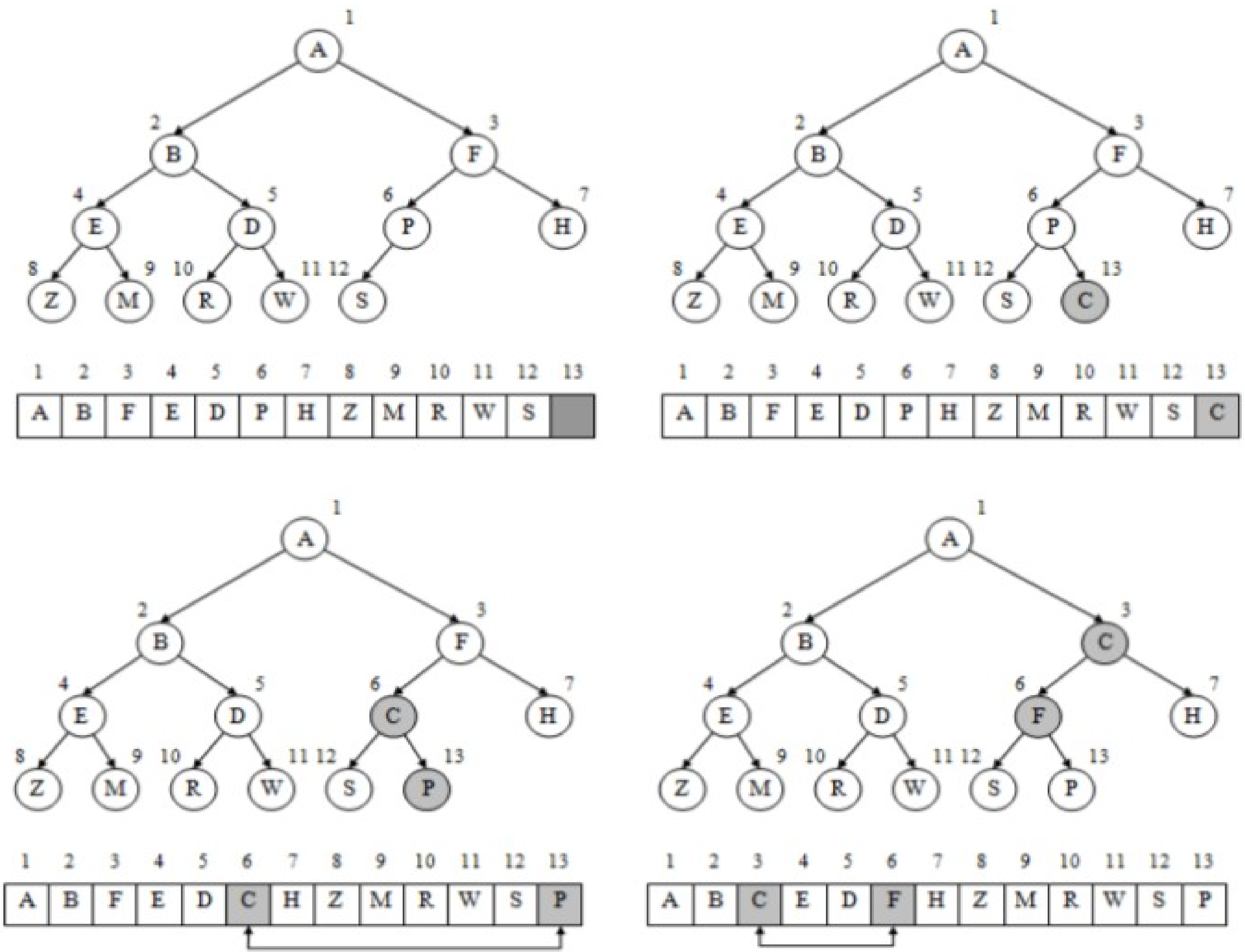
Utilidad:

- Un montículo es una representación extremadamente útil para el **TAD Cola de Prioridad**:
  - El **acceso al mínimo** es **O(1).**
  - La **inserción por valor** es **O(log n)** (tiempo amortizado).
  - El **borrado del mínimo** es **O(log n)**.
  - No usa una representación enlazada, sino un **vector**.
  - La **creación a partir de un vector** es **O(n)** y no requiere espacio adicional.
  - El borrado o modificación de un elemento, conocida su posición en el montículo, es **O(log n)**.
- Existen otras operaciones para las que no se comporta bien:
  - Para la **búsqueda** y **acceso al i-ésimo menor** se comporta igual que un vector desordenado.
  - La **fusión** de montículos (binarios) es O(n)

Arboles Binarios

Montículo Binarios

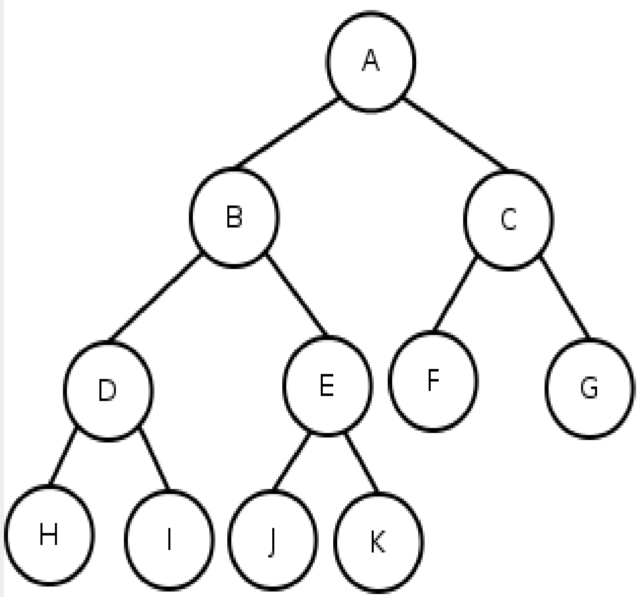
Inserción:



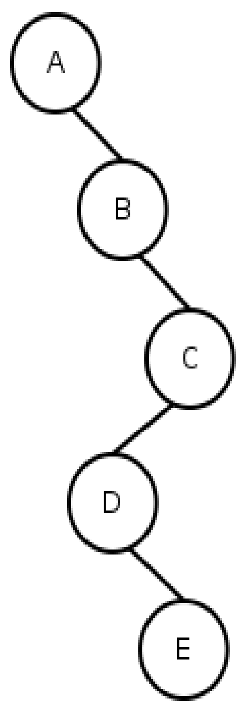
Arboles Binarios

Arboles degenerados

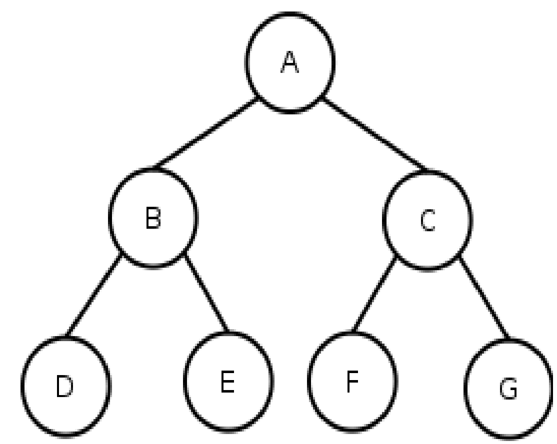
Son árboles en donde cada nodo solo tiene un hijo, motivo por lo cual se comporta como una lista, motivo por lo cual se pierde la eficiencia que tienen los arboles respecto de las listas.



a) Árbol completo de profundidad 4



b) Árbol degenerado de profundidad 5

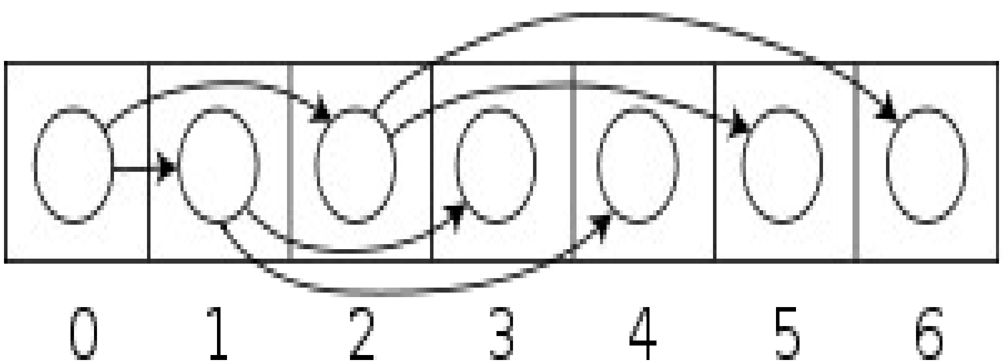


c) Árbol lleno de profunidad 3

Arboles Binarios

Representación de Arboles Binarios

Hay 2 formas de representar un árbol binario en memoria:
1. Con arreglos o listas enlazadas
2. Por medio de datos tipo puntero (variables dinámicas)



Donde el 2i+1 Hijo Izquierdo y 2i+2 hijo Derecho.
Util para Complete Binary o Full Binary.

Arboles Binarios

Representación de Arboles Binarios

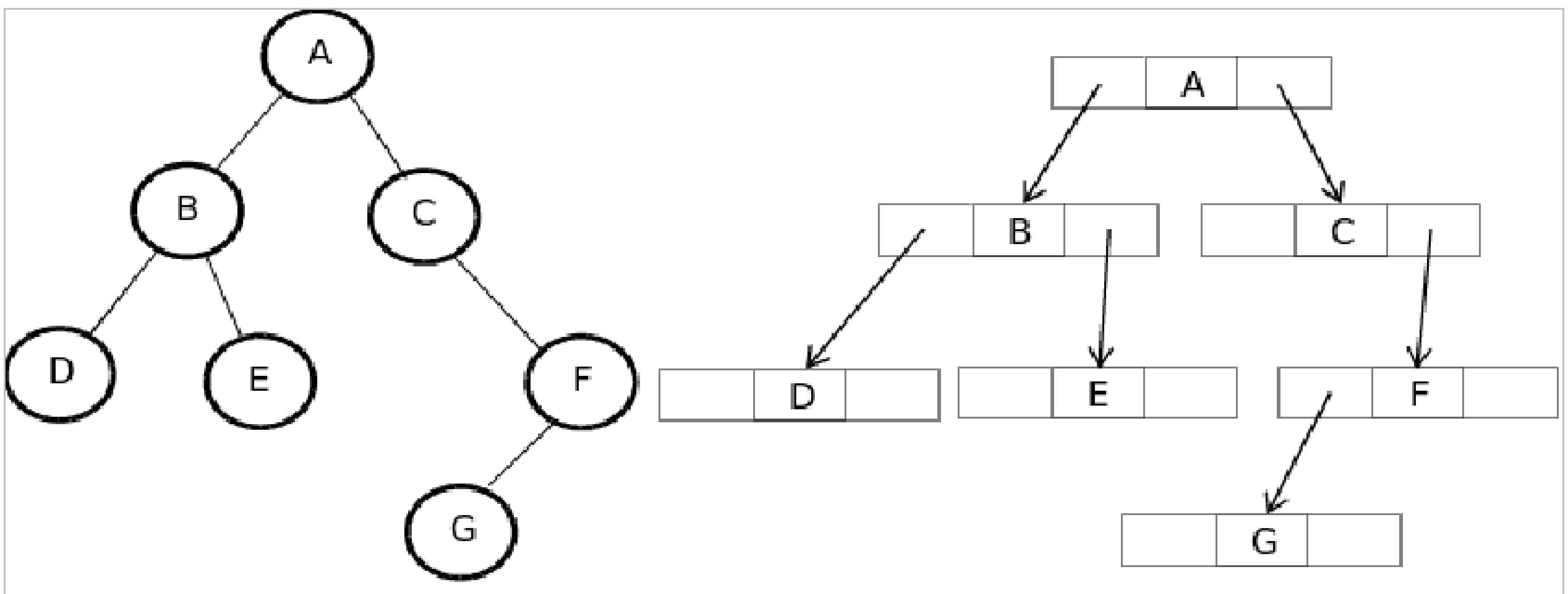
Hay 2 formas de representar un árbol binario en memoria:

1. Por medio de datos tipo puntero (variables dinámicas)
2. Con arreglos o listas enlazadas

Cada nodo tiene la siguiente estructura:

IZQ	INFO	DER

Representación Enlazada de Árboles Binarios



Arboles Binarios

Representación de Arboles Binarios

Ejemplo de Definición de Estructuras de Arboles:

Estructura en PASCAL

```
Type
  tArbol = ^tNodo;
  tNodo = RECORD
    Clave : Integer;
    Izq, Der : tArbol;
  end;
```

Estructura en C

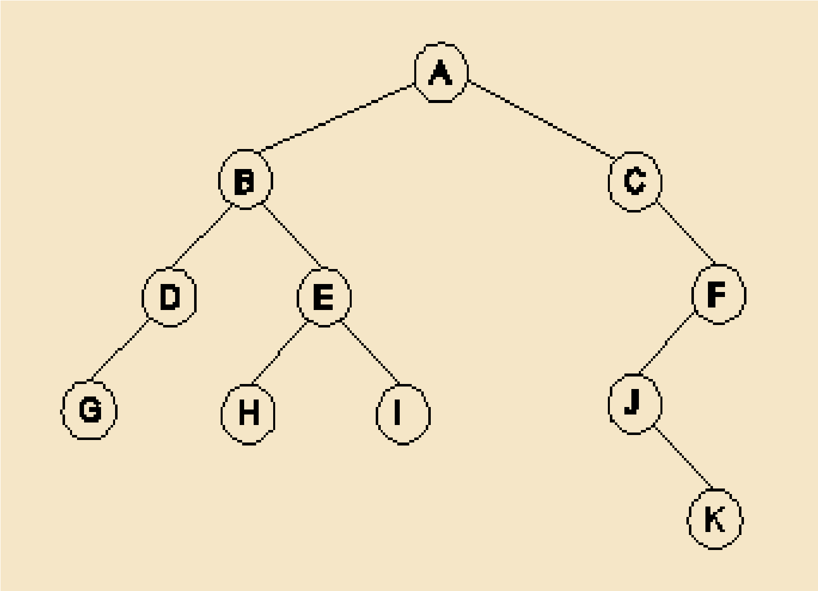
```
typedef struct nodo {
    int clave;
    struct nodo *izdo, *dcho;
}Nodo;
```

Arboles Binarios

Recorridos de Arboles Binarios

Recorrer significa visitar los nodos del árbol en forma sistemática, de tal manera que todos los nodos del mismo sean visitados una sola vez.
Existen distintas fomas de recorrer un árbol:

1. En anchura: consiste en todos los elementos del mismo nivel, y posteriormente pasar a todos los elementos del siguiente nivel.
2. En profundidad:
   a. Preorden:
      - Visitar raíz
      - Recorrer subárbol izquierdo
      - Recorrer subárbol derecho
   b. Inorden:
      - Recorrer subárbol izquierdo
      - Visitar raíz
      - Recorrer subárbol derecho
   c. Postorden:
      - Recorrer subárbol izquierdo
      - Recorrer subárbol derecho
      - Visitar raíz



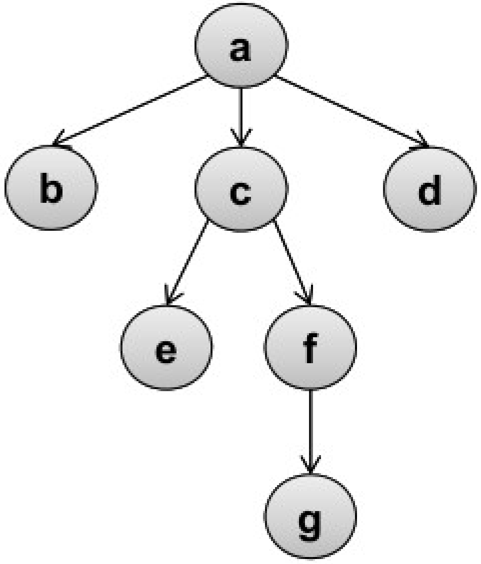

```
Inorden: GDBHEIACJKF
Preorden: ABDGEHICFJK
Postorden: GDHIEBKJFCA
```

Recorridos de Arboles

En Anchura – (a lo ancho – por niveles).

Por Niveles: Se etiquetan los nodos según su profundidad (nivel). Se recorren ordenados de menor a mayor nivel, a igualdad de nivel se recorren de izquierda a derecha.

- No recursivo: Se introduce el raiz en una cola y se entra en un bucle en el que se extrae de la cola un nodo, se recorre su elemento y se insertan sus hijos en la cola.



Por Niveles: (a) (b c d) (e f) (g)

Recorridos de Arboles Binarios

Pre-Orden - (raíz, izquierdo, derecho).

Preorden

```
void preorden (árbol*a)
        {
        if(a.raiz != null)
                {
                procesar(a.raiz);
                preorden(a.izquierdo);
                preorden(a.derecho);
                }
        }
```

Arboles Binarios

Recorridos de Arboles Binarios

Inorden - (izquierdo, raíz, derecho).

Inorden

```
void inorden (árbol*a)
        {
        if(a.raiz != null)
                {
                inorden(a.izquierdo);
                procesar(a.raiz);
                inorden(a.derecho);
                }
        }
```

Arboles Binarios

Recorridos de Arboles Binarios

Postorden - (izquierdo, derecho, raíz).

Postorden

```
void postorden (árbol*a)
        {
        if(a.raiz != null)
                {
                postorden(a.derecho);
                procesar(a.raiz);
                postorden(a.izquierdo);
                }
        }
```

Arboles Binarios

Recorridos de Arboles Binarios

Postorden - (izquierdo, derecho, raíz).

```
Postorden
void postorden (árbol*a)
        {
        if(a.raiz != null)
                {
                procesar(a.izquierdo);
                procesar(a.derecho);
                procesar(a.raiz);
                }
        }
```

Arboles Binarios

Recorridos de Arboles Binarios

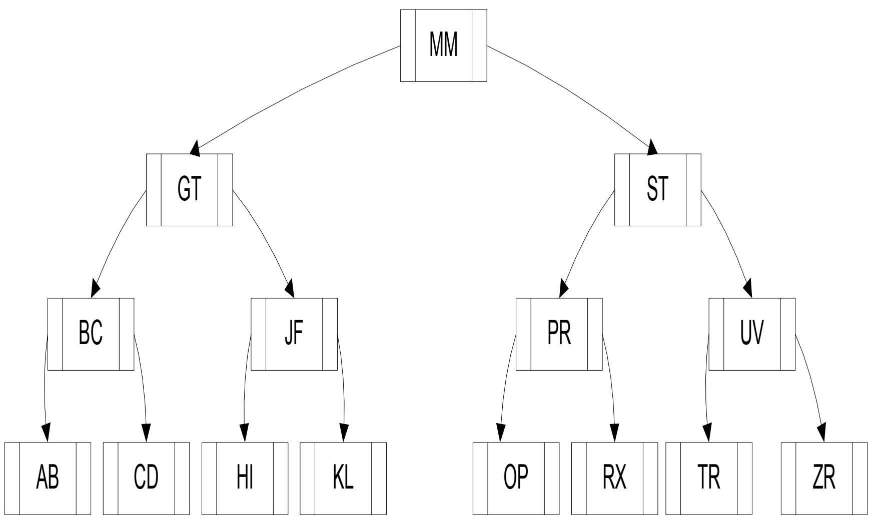
Resumen:

La diferencia entre preorden, inorden y postorden es cuándo se recorre la raíz. En los tres, se recorre primero el sub-árbol izquierdo y luego el derecho.

- En preorden, la raíz se recorre antes que los recorridos de los subárboles izquierdo y derecho
- En inorden, la raíz se recorre entre los recorridos de los árboles izquierdo y derecho.
- En postorden, la raíz se recorre después de los recorridos por el subárbol izquierdo y el derecho

Arboles Binarios

Almacenamiento en disco



Raíz à 0

	Clave	Hijo izq	Hijo Der
0	MM	1	2
1	GT	3	4
2	ST	8	11
3	BC	5	6
4	JF	7	14
5	AB	-1	-1
6	CD	-1	-1
7	HI	-1	-1

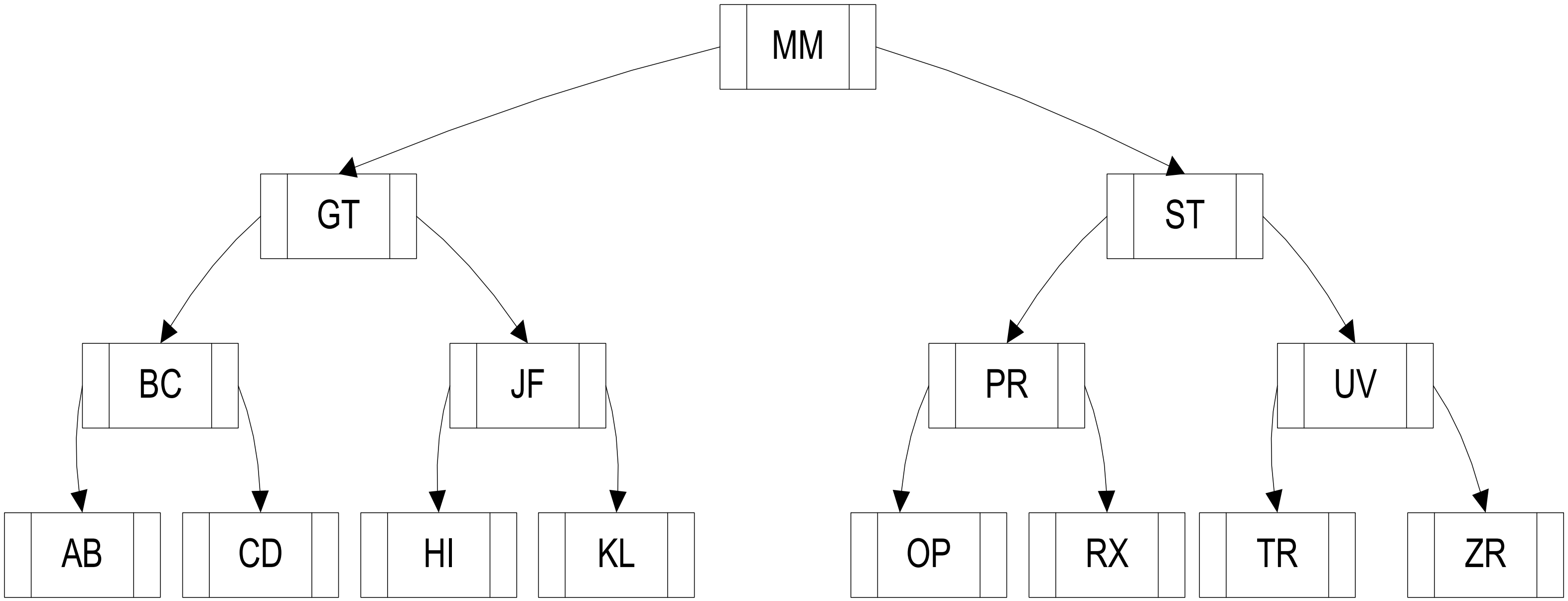
	Clave	Hijo izq	Hijo Der
8	PR	9	10
9	OP	-1	-1
10	RX	-1	-1
11	UV	12	13
12	TR	-1	-1
13	ZR	-1	-1
14	KL	-1	-1

Arboles Binarios

Almacenamiento en disco

Ejemplo ☾ supongamos estas claves

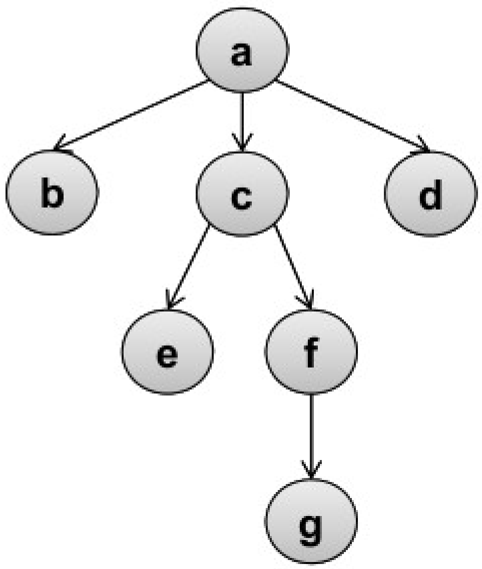
MM ST GT PR JF BC UV CD HI AB KL TR OP RX ZR



Arboles

Recorridos de Arboles en General

- **Preorden**: a,b,c,e,f,g,d
- **Postorden**: b,e,g,f,c,d,a
- **Inorden**: b,a,e,c,g,f,d
- **Por Niveles**: a,b,c,d,e,f,g



Parentizado sobre subárboles:

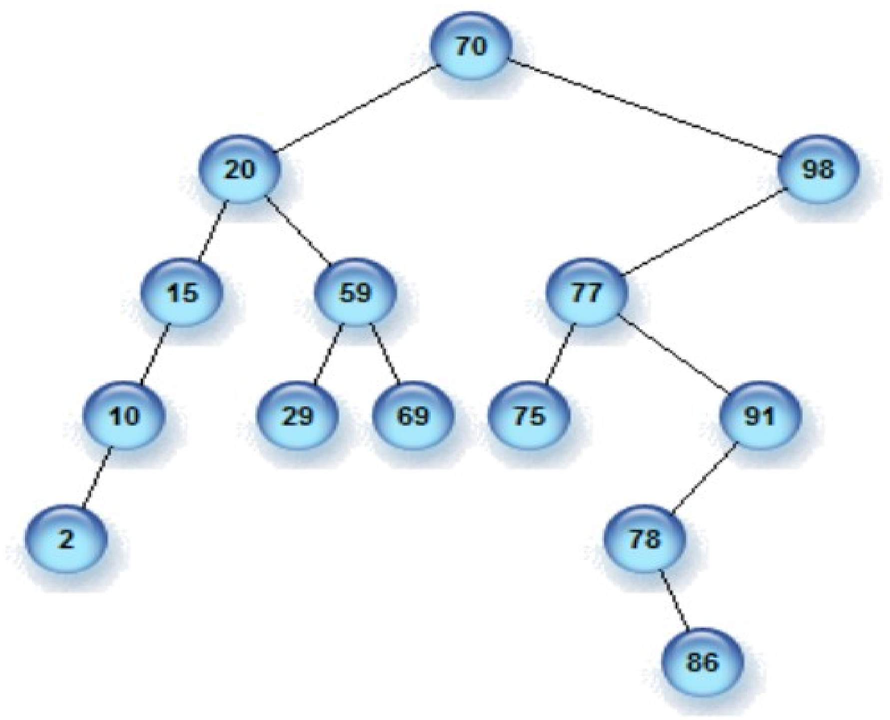
- **Preorden**: a (b) (c (e) (f (g))) (d)
- **Postorden**: (b) ((e) ((g) f) c) (d) a
- **Inorden**: (b) a ((e) c ((g) f)) (d)
- **Por Niveles**: (a) (b c d) (e f) (g)

Arboles Binarios

Árbol de Búsqueda Binario

Un árbol de búsqueda con raíz R es de búsqueda cuando no es vacio y:

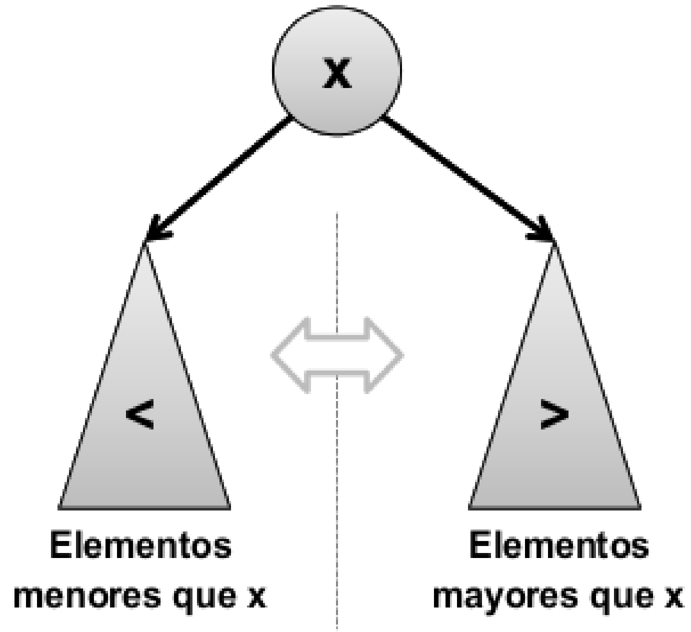
- Si tiene un subárbol izquierdo, la raíz del subárbol izquierdo es menos a R, y a la vez el subárbol izquierdo también es de búsqueda.
- Si tiene un subárbol derecho, la raíz del subárbol derecho es mayor a R, y a la vez el subárbol derecho también es de búsqueda.



Arboles Binarios

Propiedad de Árbol de Búsqueda Binario

- Un **árbol binario de búsqueda** (árbol BB) es un **árbol binario** cuyos nodos almacenan elementos comparables mediante ≤ y donde todo nodo cumple la **propiedad de ordenación**:
- **Propiedad de ordenación**: Todo nodo es **mayor** que los nodos de su subárbol **izquierdo**, y **menor** que los nodos de su subárbol **derecho**.



Arboles Binarios

Propiedad de Árbol de Búsqueda Binario

- Un recorrido **inorden** por el árbol recorre los elementos en **orden** de menor a mayor.
- El elemento **mínimo** es el primer nodo sin hijo izquierdo en un descenso por hijos izquierdos desde la raiz.
- El elemento **máximo** es el primer nodo sin hijo derecho en un descenso por hijos derechos desde la raiz.
- Para **buscar** un elemento se parte de la raiz y se desciende escogiendo el subárbol izquierdo si el valor buscado es menor que el del nodo o el subárbol derecho si es mayor.
- Para **insertar** un elemento se busca en el árbol y se inserta como nodo hoja en el punto donde debería encontrarse.
- Para **borrar** un elemento, se adaptan los enlaces si tiene 0 o 1 hijo. Si tiene dos hijos se intercambia con el máximo de su subárbol izquierdo y se borra ese máximo.

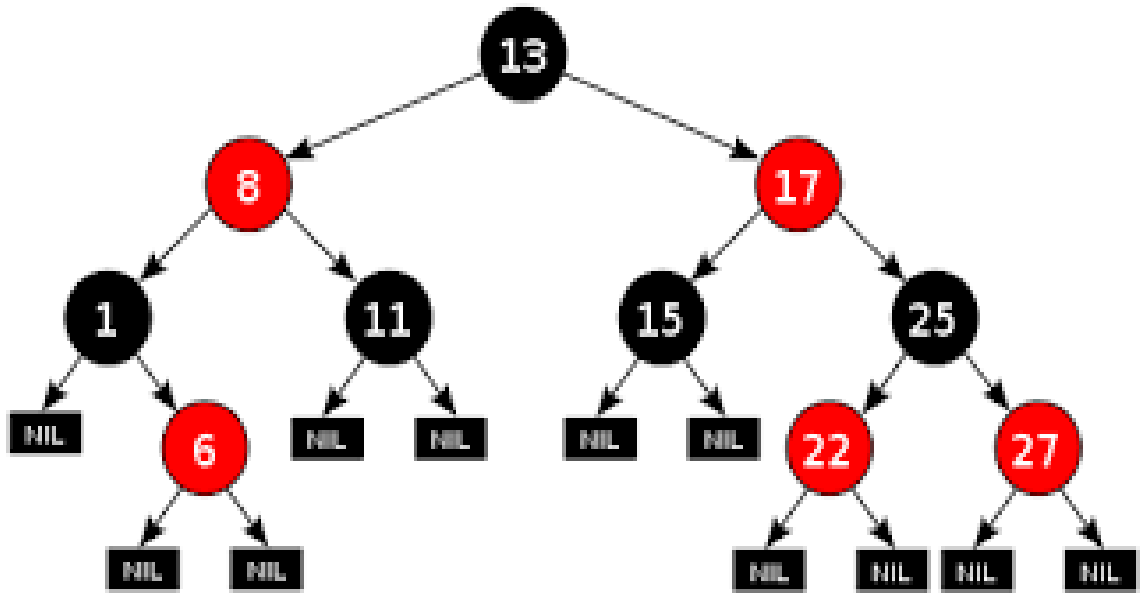
Arboles Binarios

Árbol de Búsqueda Binario

Para poder usar un árbol de binario de búsqueda es necesario que los elementos que insertemos sean comprables.

Ejemplo Almacenar:

- Enteros de menor a mayor: 5<12
- Cadenas de caracteres por orden lexicográfico: 'auto' < 'pato lucas'
- Objetos siempre que se identifiquen con una clave que sea ordenable (libros como clave el ISBN, empleados como clave el legajo, etc.)



Arboles Binarios

Árbol de Búsqueda Binario

1. Búsqueda

Básicamente podemos realizar dos tipo de operaciones:

- Operación ¿existe?: comprobar si una clave está presente en un árbol o no.
- Operación obtener: obtener un objeto cuyo identificador sea el que pedimos.

Pseudocodigo existe?

```
Existe(a: Arbol, elem: K)
	{
	if (a == NULL) return FALSE				// árbol vacío
	if (elem == a.valor) return a.valor return TRUE		// el elemento está en la raíz
	if (elem < a.valor)
		Return Existe(a.izquierdo, elem)		// recursivamente busco en árbol izq
	if (elem > a.valor)
		Return Existe(a.derecho, elem)  // recursivamente busco en árbol
```

Árbol de Búsqueda Binario

1. Búsqueda

Básicamente podemos realizar dos tipo de operaciones:

- Operación ¿existe?: comprobar si una clave está presente en un árbol o no.
- Operación obtener: obtener un objeto cuyo identificador sea el que pedimos.

Pseudocodigo obtener

```
Localizar(a: Arbol, elem: K)
        {
        if (a == NULL) return NULL                    // árbol vacío
        if (elem == a.valor) return a.valor           // devuelvo la raíz
        if (elem < a.valor)
                Return Localizar(a.izquierdo, elem) // recursivamente busco en árbol izq
        if (elem > a.valor)
                Return Localizar(a.derecho, elem)  // recursivamente busco en árbol der
        }
```

Arboles Binarios

Árbol de Búsqueda Binario

2. Insertar elementos.

La inserción se realiza a partir de la búsqueda, ya que es necesario introducir el elemento en forma ordenada.

Reglas:
Si el árbol es un árbol vacío, insertamos el elemento en la raíz.

Si la raíz del árbol es igual al elemento a insertar: si no admitimos duplicados, no insertamos.

Si la raíz del árbol es mayor al elemento a insertar, insertamos en el subárbol izquierdo.

Si la raíz del árbol es menor al elemento a insertar, insertamos en el subárbol derecha.

Arboles Binarios

Árbol de Búsqueda Binario

2. Inserción:Cont.

La inserción es similar a la búsqueda.

Si El Arbol esta vacio se crea un nuevo nodo como único contenido

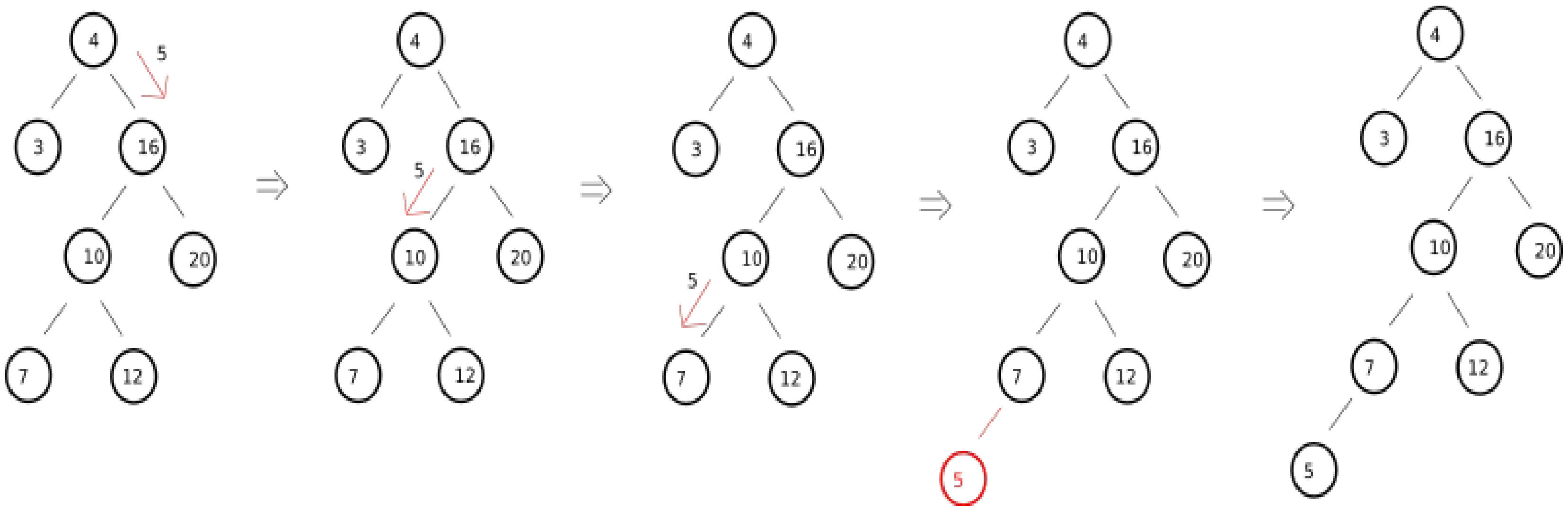
Si no lo está, y elemento < Raiz

Inserta en subárbol izquierdo.

Si elemento > Raiz

Inserta en el subárbol derecho.

Si elemento se encuentra en el árbol Actualiza?



Árbol de Búsqueda Binario

3. Eliminación de un nodo

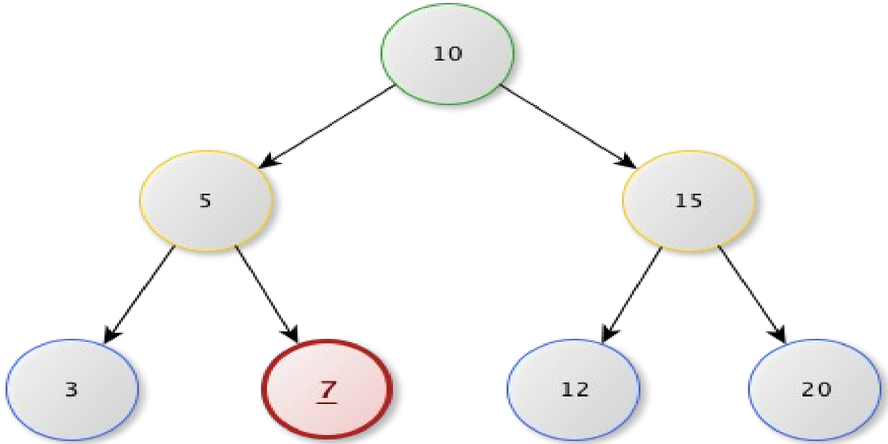
Para eliminar un elemento lo primero que hay que hacer es buscar el nodo. La eliminación depende de cuantos hijos tenga el nodo:

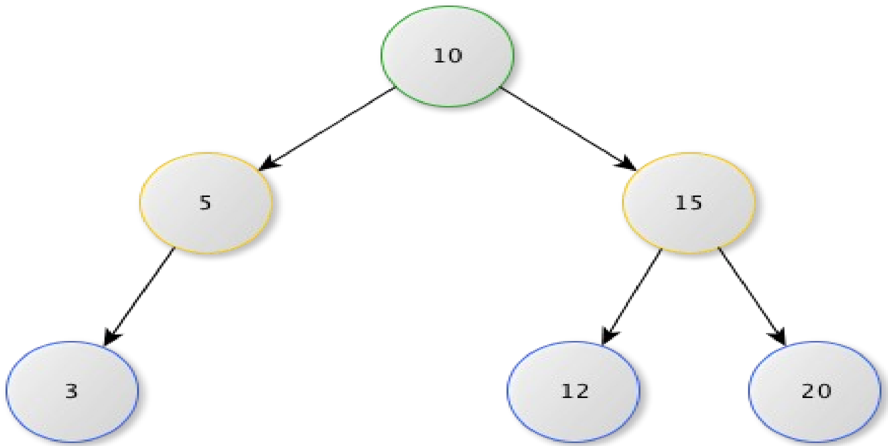
- El nodo a eliminar es hoja:
  Es la forma más sencilla. Simplemente se eliminar el nodo y se coloca null en la referencia que tenía el padre.
- Tiene un hijo:
  Si eliminamos el nodo, vamos a estar eliminando toda la rama, por lo tanto debemos:
  - Que el padre del nodo a eliminar apunte al hijo del nodo a eliminar.
  - Sustituimos el valor del nodo raíz por el valor del nodo hijo y sustituimos el derecho de la raíz por el derecho del nodo hijo.
- Tiene dos hijos: Al menos un descendiente por rama
  Hay que promover alguno de los dos nodos como nuevo Nodo
  ¿Puede ser cualquier Nodo? No, debe ser la que contenga una de estas dos claves:
  · la **mayor** de las claves **menores** al nodo que se borra.
  · la **menor** de las claves **mayores** al nodo que se borra.

Arboles Binarios

Árbol de Búsqueda Binario

El Nodo a eliminar es una hoja:



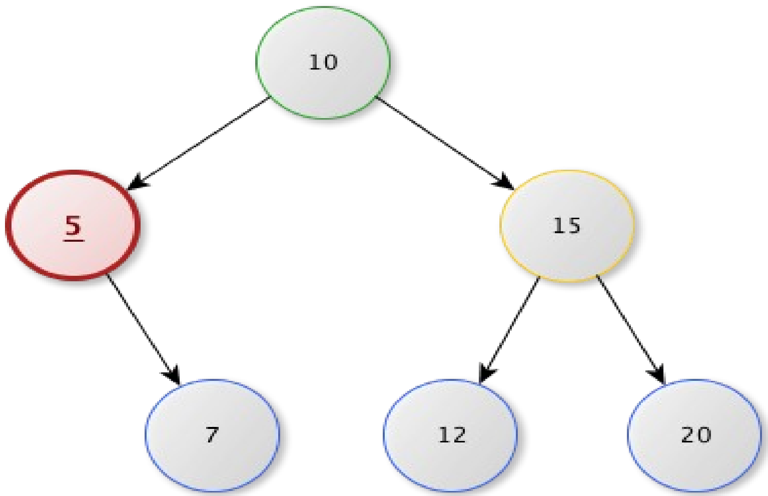


Basta con apuntar el puntero del Padre a NULL y liberar la memoria.

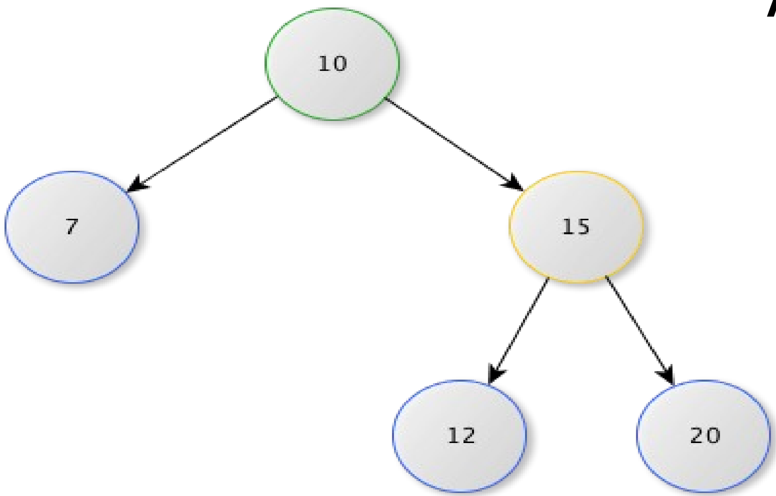
Arboles Binarios

Árbol de Búsqueda Binario

El Nodo a eliminar tiene un hijo:



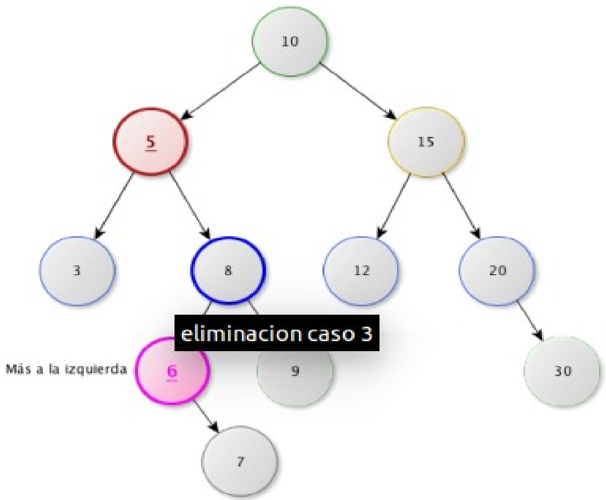
Apuntar el puntero del Padre al hijo y liberar la memoria.

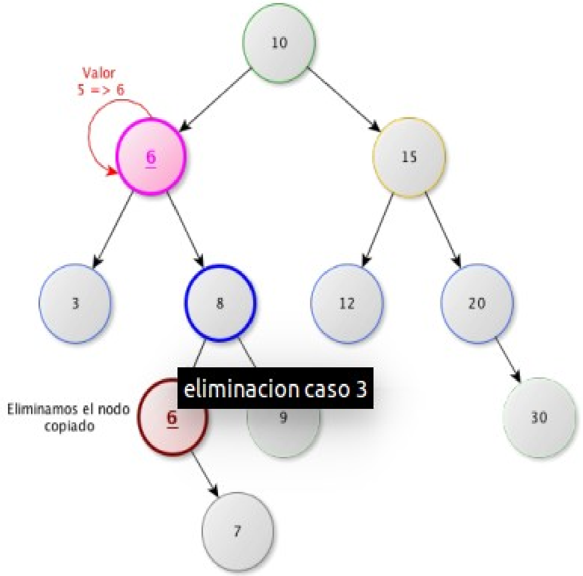


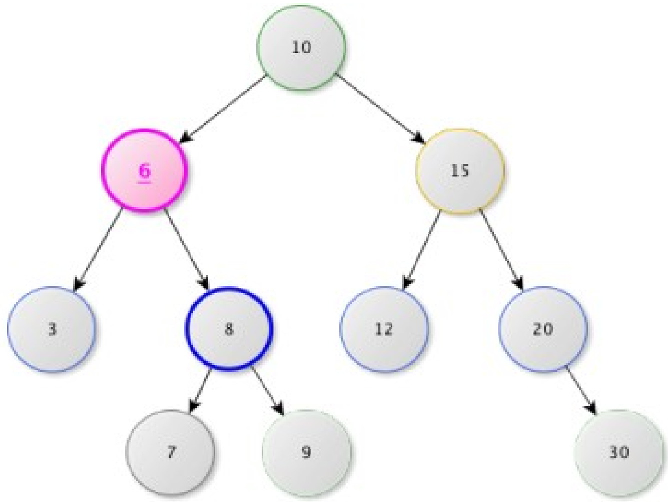
Arboles Binarios

Árbol de Búsqueda Binario

El Nodo a eliminar tiene dos hijos:





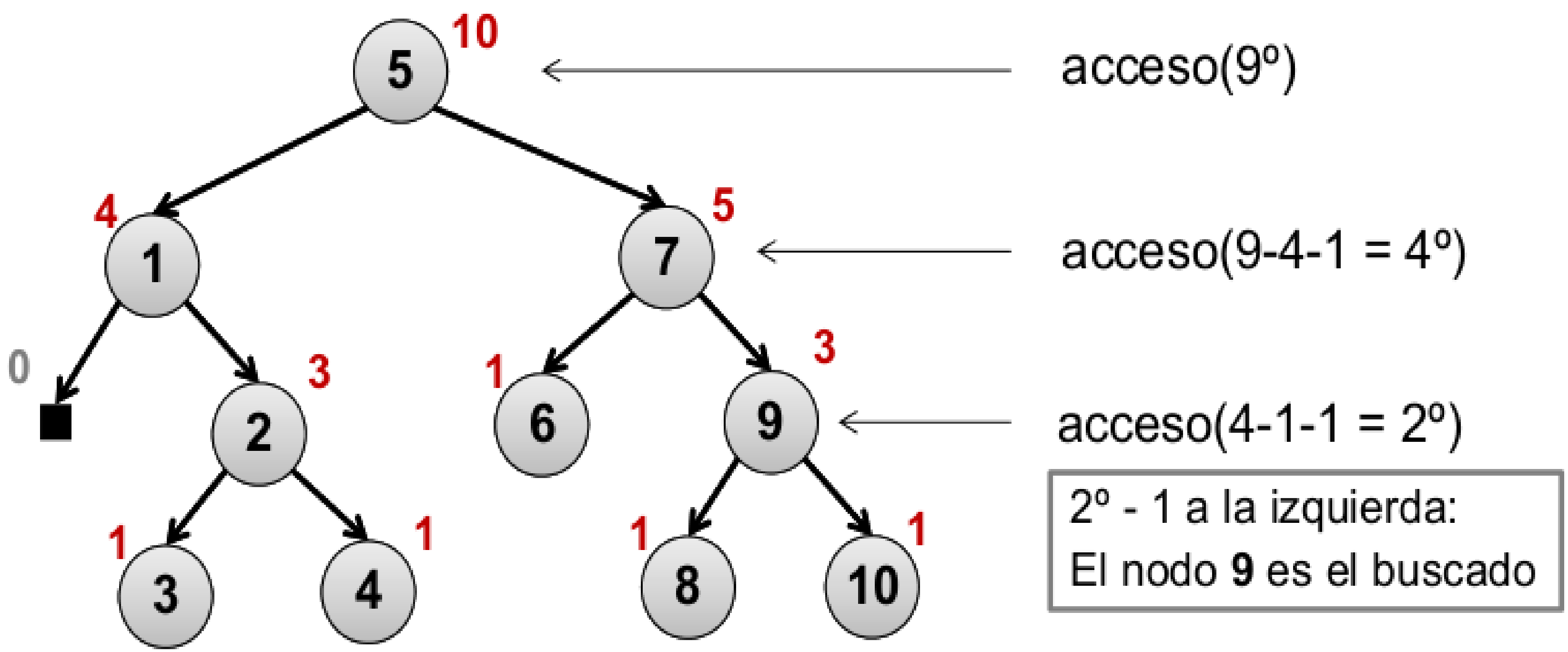


Arboles Binarios

Árbol de Búsqueda Binario

Acceso por índice:

- Es posible **extender** un ABB para que la operación de **acceso al i-ésimo menor** sea eficiente añadiendo un campo a cada nodo que indique el **número de elementos del subárbol:**



Árbol de Búsqueda Binario

Utilidad:

- Un árbol BB podría ser adecuado para representar los **TADs Conjunto, Mapa, Diccionario y Lista ordenada**:
  - El **acceso por valor** (búsqueda) es **O(h)**
  - La **inserción por valor** es **O(h)**
  - El **borrado por valor** es **O(h)**.
  - El **acceso al í-ésimo menor** (con la extensión anterior) es **O(h)**.
  - El **borrado del i-ésimo menor** es **O(h)**.
  - La **fusión** es **O(n)**.
- En las medidas de eficiencia **h** es la altura del árbol.
  - Se define **arbol equilibrado** como aquél que garantiza que su altura es logarítmica **h ∊ O(log n)**
  - Desafortunadamente, los árboles BB **no son equilibrados** (no tiene porqué cumplirse que la altura sea logarítmica).

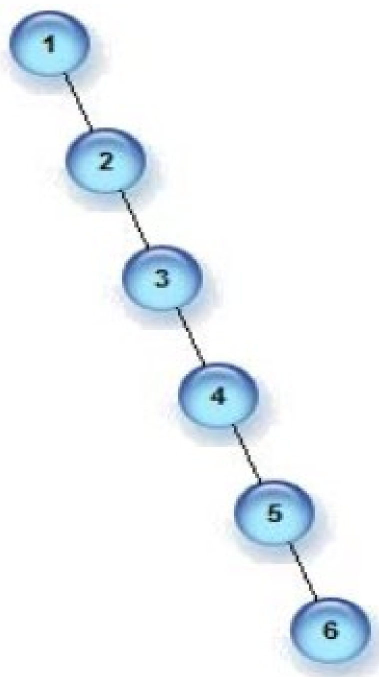
Árbol de Búsqueda

Equilibrado:

- El que un árbol BB esté equilibrado o no depende de la **secuencia de inserciones**. Desafortunadamente, el insertar elementos **en orden** provoca caer en el peor caso: Un **árbol lineal** (altura O(n), proporcional al número de elementos)

- En un árbol lineal todas las operaciones relevantes serían **O(n)**, arruinando la eficiencia.

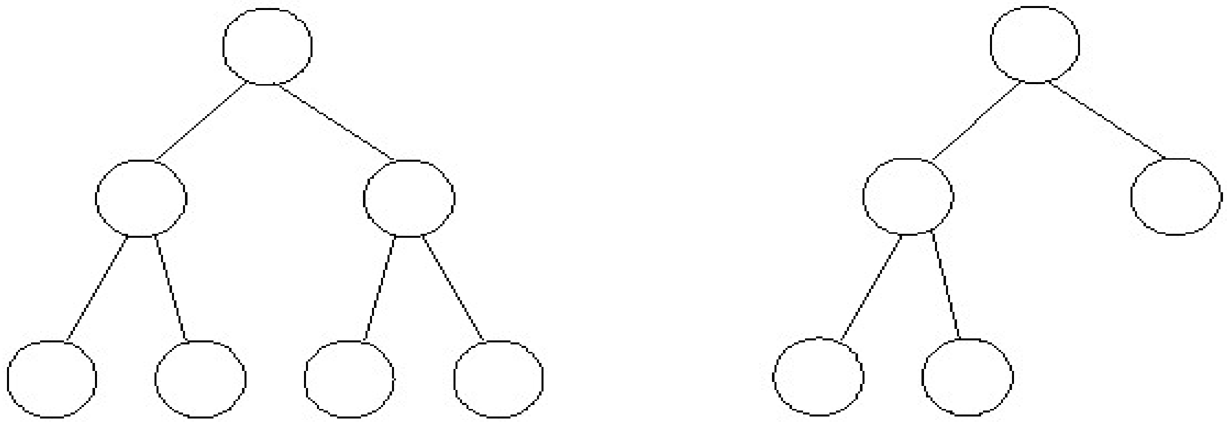
- Si los elementos se insertan al azar, se puede demostrar que la altura del árbol BB es, en promedio, logarítmica.



Arboles Binarios

Arboles Balanceados

Un árbol balanceado intenta mantener la menor profundidad en sus dos subárboles.
El balanceo o equilibrio de un árbol, hace que algunas operaciones sean mas eficientes, sobre todo las búsquedas.



Árbol Binario
Estrictamente Balanceado

Árbol Binario
NO Estrictamente Balanceado

Un árbol está balanceado cuando la altura de la trayectoria más corta hacia una hoja no difiere de la altura de la trayectoria más grande.

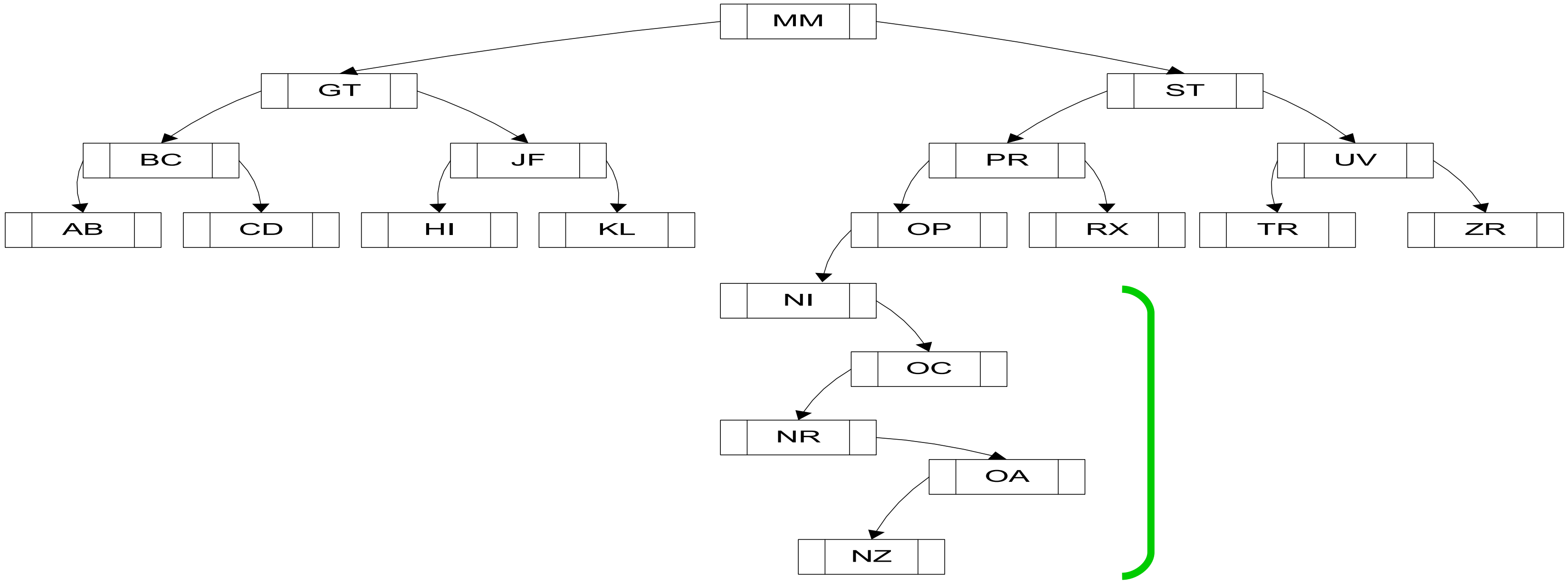
Arboles Binarios

Arboles Balanceados

Un árbol está balanceado cuando la altura de la trayectoria˙más corta
hacia una hoja no difiere de la altura de la trayectoria más grande.

Inconveniente de los binarios: se des balancean fácilmente.

Supongamos que llegan las claves : NI OC NR OA NZ



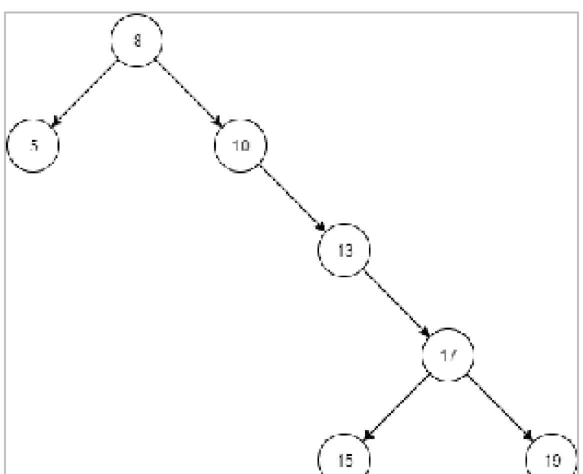
Arboles AVL

Arboles Balanceados

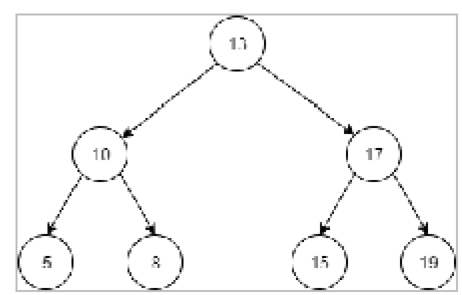
ARBOLES AVL

Un árbol AVL es un árbol binario de búsqueda (ABB) que tiene como característica que siempre está balanceado.
Su nombre deriva de las iniciales de sus creadores **Adelson-Velski y Landis.**
Un árbol balanceado intenta mantener la menor profundidad en sus dos subárboles.
Cuando se realiza una inserción o una eliminación, se comprueba si el árbol está desequilibrado y se realiza el balanceo.

Árbol degenerado:



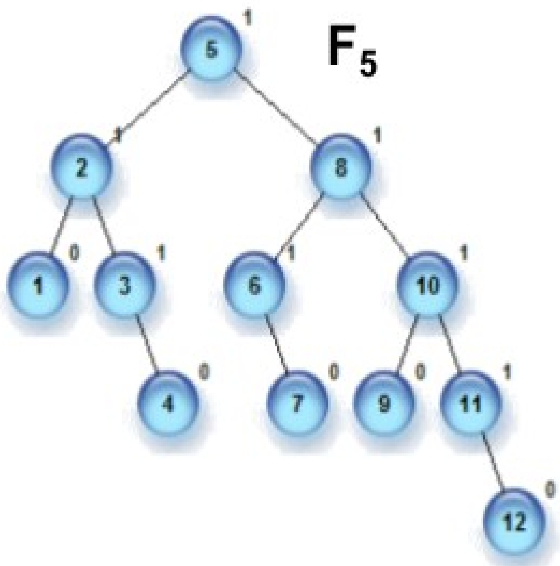
Árbol balanceado



Arboles AVL

Altura Logarítmica:

- Todo árbol binario con **equilibrado AVL** tiene altura logarítmica
- Se define **árbol de Fibonacci** (F_h) como:
  - F_{-1} es el árbol vacío.
  - F_0 es el árbol con un único nodo.
  - F_h es el árbol con subárbol izquierdo F_{h-2} y derecho F_{h-1}
- El árbol F_h, tiene altura h y número de elementos:



$$N(h) = N(h-1) + N(h-2) + 1$$

$$N(h) \in O(\phi^h) \Rightarrow h \in O(\log n)$$

Un árbol de fibonacci es el árbol AVL con mayor desequilibrio

Arboles AVL

Equilibrio

- Equilibrio (n) = altura-der (n) – altura-izq (n) describe relatividad entre subárbol der y subárbol izq.
    - + (positivo) → der mas alto (profundo)
    - - (negativo) → izq mas alto (profundo)
- **Un árbol binario es un AVL si y sólo si cada uno de sus nodos tiene un equilibrio de –1, 0, + 1**
- Si alguno de los pesos de los nodos se modifica en un valor no válido (2 o -2) debe seguirse un esquema de rotación.

Arboles AVL

OPERACIONES de BALANCEO

Para balancear un árbol, es necesario realizar rotaciones.

Existen 4 tipos de rotaciones:

- Rotación simple a la derecha (RSD)
- Rotación simple a la izquierda (RSI)
- Rotación doble a la derecha (RDD)
- Rotación doble a la izquierda (RDI)

Factor de equilibrio: es la diferencia entre las alturas del árbol izquierdo y el árbol derecho.

FE: altura subárbol derecho – altura subárbol izquierdo **FE=Hd-HI**

Para que un árbol sea AVL el valor del FE debe ser **-1, 0 ó 1:**

- -1 = cargado a la izquierda
- 0 = equilibrado
- 1 = cargado a la derecha

FE >0 DER mas Alto
FE < 0 IZQ mas Alto

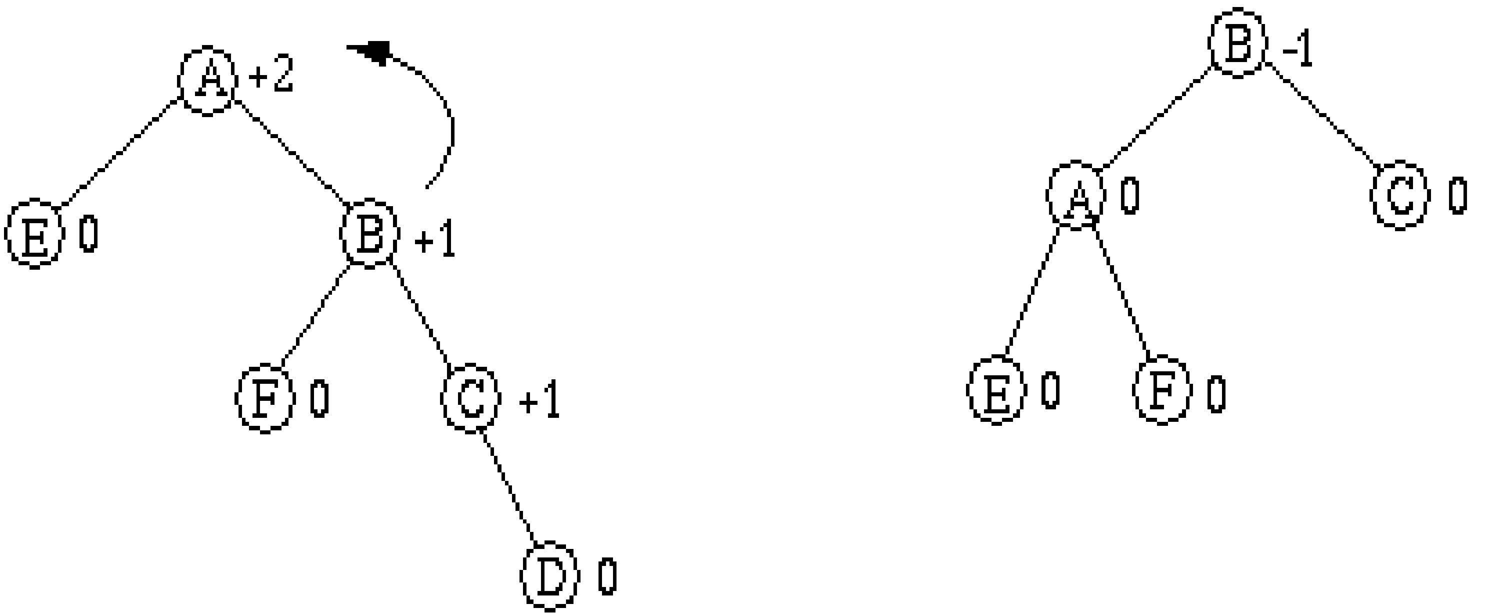
Arboles AVL

Estructura Posible para el AVL

```
struct Nodo {
    int bal;  //para almacenar el valor del equilibrio del nodo
    int dato;
    Nodo *izq;
    Nodo *der;
}
```

Arboles AVL

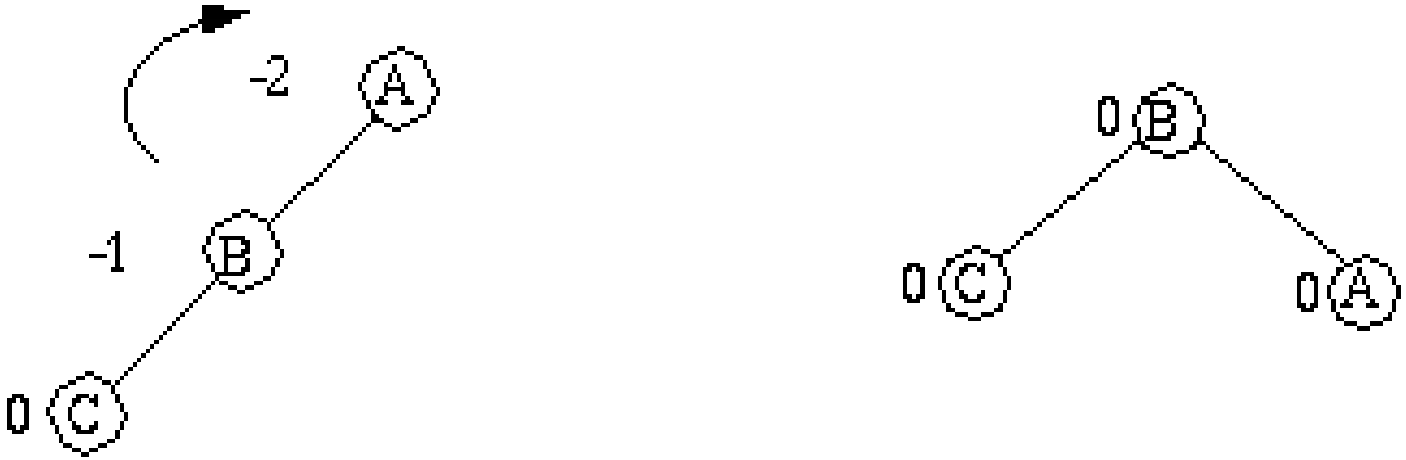
- Caso 1: Rotación simple izquierda RSI
  - Si esta desequilibrado a la izquierda y su hijo derecho tiene el mismo signo (+) hacemos rotación sencilla izquierda.



Arboles AVL

- Caso 2: Rotación simple derecha RSD
  - Si esta desequilibrado a la derecha y su hijo izquierdo tiene el mismo signo (-) hacemos rotación sencilla derecha.

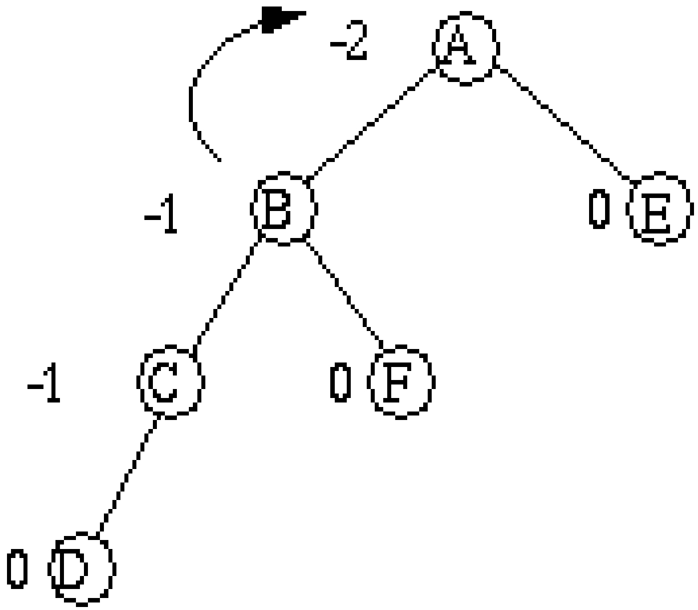
Ejemplo 1

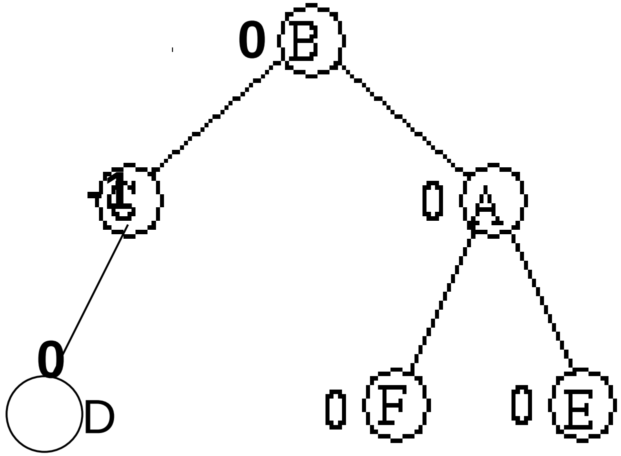


Arboles AVL

- Caso 2: Rotación simple derecha RSD

Ejemplo 2

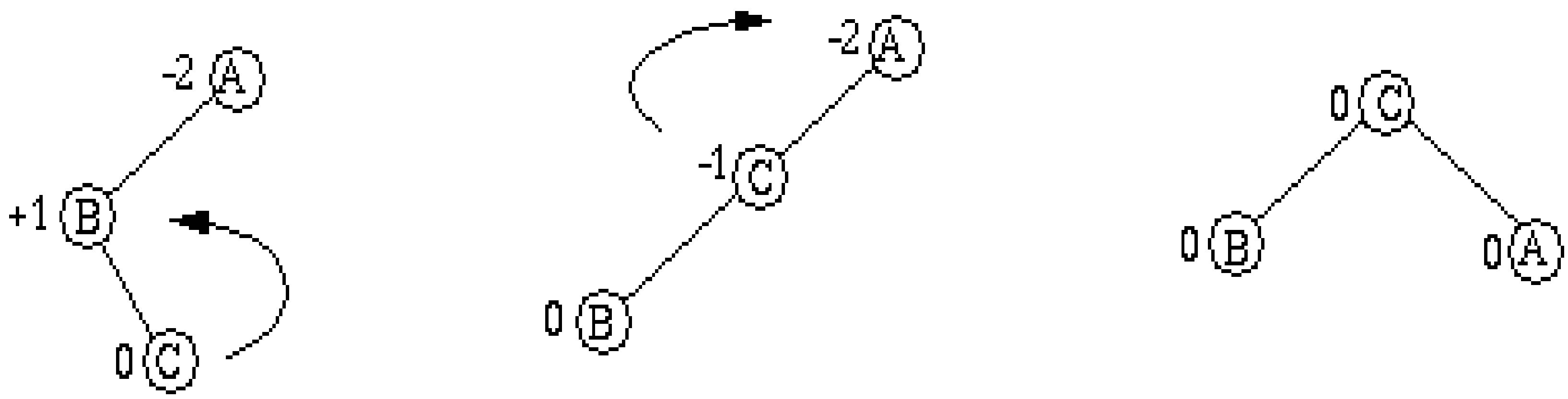




Arboles AVL

- Caso 3: Rotación doble izquierda RDI
  - Si está desequilibrado a la izquierda (FE < –1), y su hijo derecho tiene distinto signo (+) hacemos rotación doble izquierda-derecha.

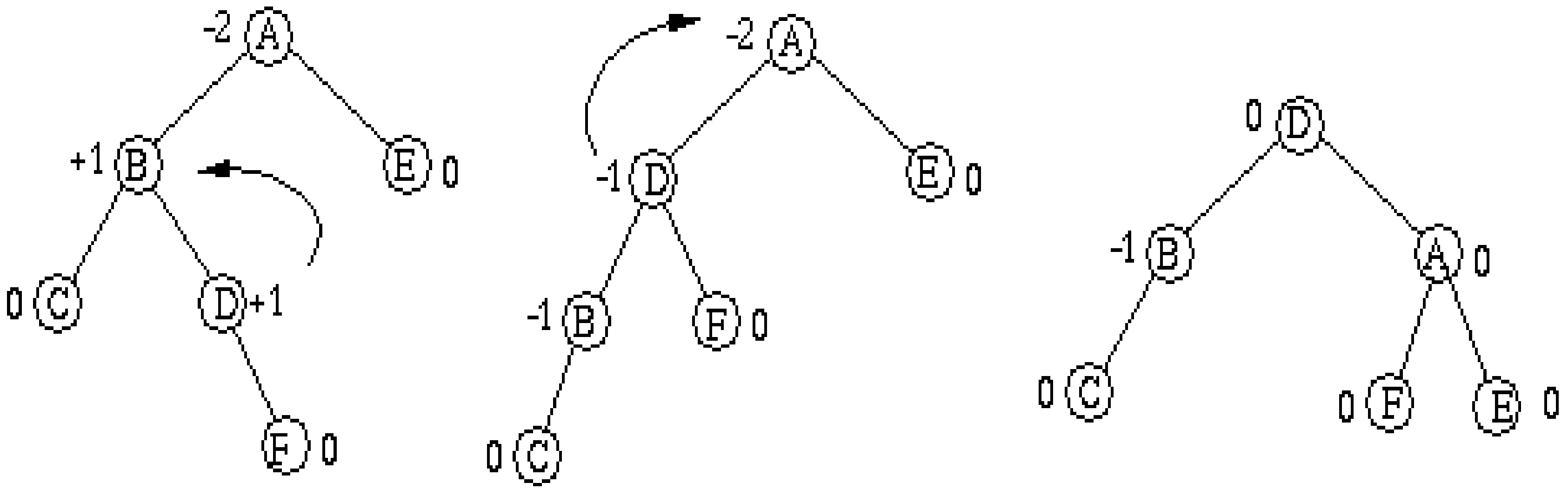
Ejemplo 1



Arboles AVL

- Caso 3: Rotación doble izquierda RDI

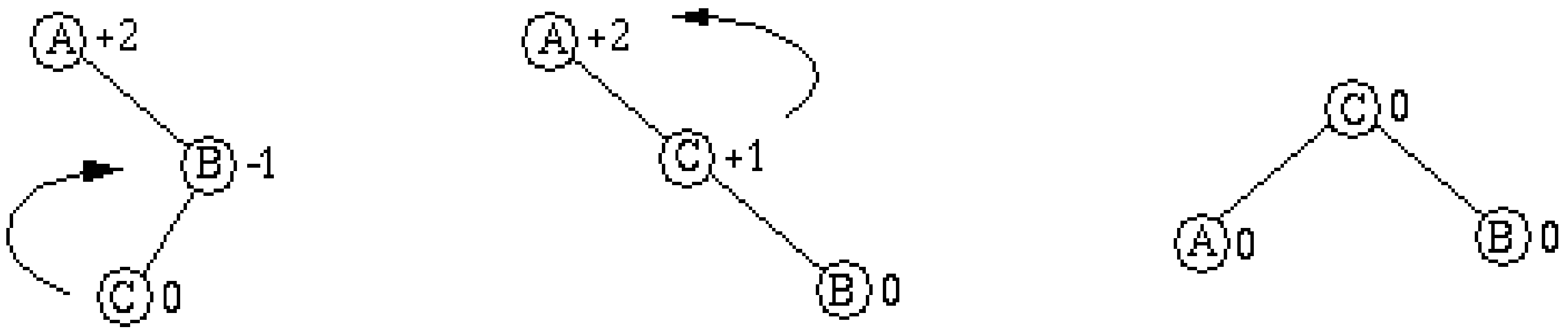
Ejemplo 2



Arboles AVL

- Caso 4: Rotación doble derecha RDD
  - Si esta desequilibrado a la derecha y su hijo izquierdo tiene distinto signo (–) hacemos rotación doble derecha-izquierda.

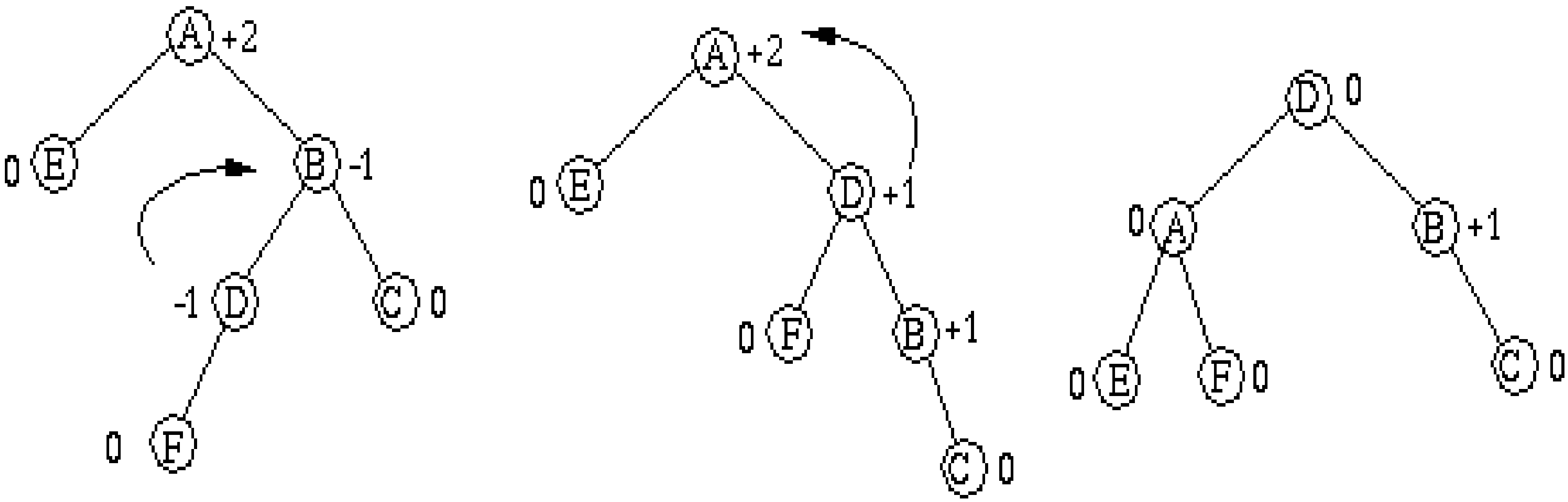
Ejemplo 1



Arboles AVL

- Caso 4: Rotación doble derecha RDD

Ejemplo 2



Arboles AVL

```
typedef struct AVL{
        int dato, FB; // FB es la Hi-Hd
        AVL *izq, *der;
        bool borrado;
} AVL;
```

```
void rotarLL(AVL* &A){ //precond: el árbol necesita una rotacion LL
        AVL* aux = A->izq->der;
        A->izq->der = A;
        A->izq->FB = 0;
        AVL* aux2 = A->izq;
        A->izq = aux;
        A->FB = 0;
        A = aux2;
}
```

```
void rotarRR(AVL* &A){ //precond: el árbol necesita una rotacion RR
        AVL* aux = A->der->izq;
        A->der->izq = A;
        A->der->FB = 0;
        AVL* aux2 = A->der;
        A->der = aux;
        A->FB = 0;
        A = aux2;
}
```

Arboles AVL

```
typedef struct AVL{
        int dato, FB; // FB es la Hi-Hd
        AVL *izq, *der;
        bool borrado;
} AVL;
```

```
void rotarLRalter(AVL* &A){ //precond: el árbol necesita una rotacion LR
        rotarRR(A->izq);
        rotarLL(A);
}
```

```
void rotarRLalter(AVL* &A){ //precond: el árbol necesita una rotacion RL
        rotarLL(A->der);
        rotarRR(A);
}
```

Arboles AVL

Inserta Por Izquierda

```
void Insert(int n, bool &aumento, AVL* &A){
	if (A == NULL){
		A = new AVL;
		A->dato = n;
		A->FB = 0;
		A->izq = NULL;
		A->der = NULL;
		aumento = true;
		A->borrado = false;
	}else{
		if (n < A->dato){
			Insert(n, aumento, A->izq);
			if (aumento){
				switch (A->FB){
					case -1:{
						A->FB = 0;
						aumento = false;
						break;
					}
					case 0:{

					}
					case 1:{
						if (A->izq->FB == 1){ // Si es 1 -> LL si es -1 -> LR
							rotarLL(A);
						}else{
							rotarLRalter(A);
						}
						aumento = false;
						break;
					}
				}
			}
		}else{
```

Arboles AVL

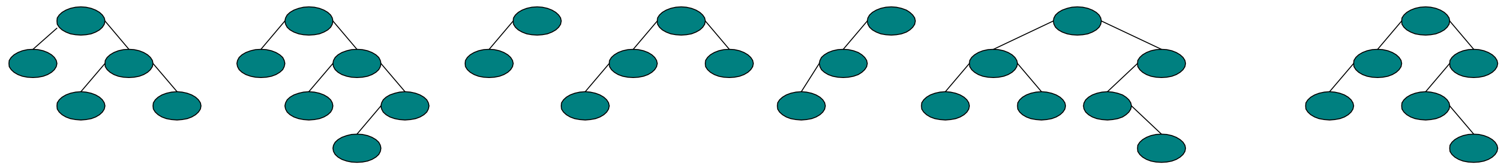
Inserta por Derecha

```
        }else{
                Insert(n, aumento, A->der);
                if (aumento){
                        switch (A->FB){
                                case -1:{
                                        if (A->der->FB == 1){ // Si es 1 n--> RL si es -1 --> RR
                                                rotarRLalter(A);
                                        }else{
                                                rotarRR(A);
                                        }
                                        aumento = false;
                                        break;
                                }
                                case 0:{
                                        A->FB = -1;
                                        aumento = true;
                                        break;
                                }
                                case 1:{
                                        A->FB = 0;
                                        aumento = false;
                                        break;
                                }
                        }
                }
        }
    }
}
```

Árboles AVL

- Árbol binario balanceado en altura (BA(1)) en el que las inserciones y eliminaciones se efectúan con un mínimo de accesos.

- Árbol balanceado en altura:
  - Para cada nodo existe un límite en la diferencia que se permite entre las alturas de cualquiera de los subárboles del nodo (BA(k)), donde k es el nivel de balance)
  - Ejemplos:



Características y Conclusiones

- Estructura que debe ser respetada

- Mantener árbol, rotaciones restringidas a un área local del árbol

- Binario: ☾ Búsqueda: $\log_2(N+1)$
- AVL: ☾ Búsqueda: $1.44 \log_2(N+2)$

- Ambas performance por el peor caso posible

Arboles Binarios Paginados

- Problemas de almacenamiento secundario, buffering, páginas de memoria, varios registros individuales, minimiza el número de accesos

- Problema: construcción descendente, como se elige la raíz?, cómo va construyendo balanceado?