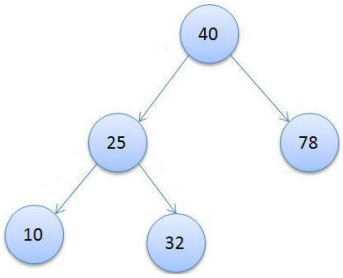
# AYED – TEORIA

## ARBOLES BINARIOS

- Definición: Un árbol binario es una colección de nodos, tal que:
    - puede estar vacía
    - puede estar formada por un nodo distinguido R, llamado raíz y dos sub-árboles T1 y T2, donde la raíz de cada subárbol Ti está conectado a R por medio de una arista.
- Descripción y terminología:
    - Cada nodo puede tener a lo sumo dos nodos hijos.
    - Cuando un nodo no tiene ningún hijo se denomina hoja.
    - Los nodos que tienen el mismo nodo padre se denominan hermanos.
    - *Camino*: desde n1 hasta nk, es una secuencia de nodos n1, n2, …,nk tal que ni es el padre de ni+1, para $1 \leq i < k$.
        - La longitud del camino es el número de aristas, es decir k-1.
        - Existe un camino de longitud cero desde cada nodo a sí mismo.
        - Existe un único camino desde la raíz a cada nodo.
    - *Profundidad*: de n sub i es la longitud del único camino desde la raíz hasta n sub i.
        - La raíz tiene profundidad cero.
    - *Grado* de ni es el número de hijos del nodo ni.
    - *Altura* de ni es la longitud del camino más largo desde ni hasta una hoja.
        - Las hojas tienen altura cero.
        - La altura de un árbol es la altura del nodo raíz.
    - *Ancestro/Descendiente*: si existe un camino desde n1 a n2, se dice que n1 es ancestro de n2 y n2 es descendiente de n1.
    - *Árbol binario lleno*: Dado un árbol binario T de altura h, diremos que T es lleno si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h).
        - Es decir, recursivamente, T es lleno si:
            - 1.- T es un nodo simple (árbol binario lleno de altura 0), o
            - 2.- T es de altura h y sus sub-árboles son llenos de altura h-1.
    - *Árbol binario completo*: Dado un árbol binario T de altura h, diremos que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.
    - *Cantidad de nodos en un árbol binario lleno*: Sea T un árbol binario lleno de altura h, la cantidad de nodos N es (2 elevado a h+1 – 1).
    - *Cantidad de nodos en un árbol binario completo*: Sea T un árbol binario completo de altura h, la cantidad de nodos N varía entre (2 h) y (2 elevado a h+1 – 1)
- Recorridos:




    - *Preorden*: Se procesa primero la raíz y luego sus hijos, izquierdo y derecho. 40, 25, 10, 32, 78

```
public class ArbolBinario<T> {
 private NodoBinario<T> raiz;
 . . .
 public void printPreorden() {
   System.out.println(this.getDatoRaiz());
   if (!this.getHijoIzquierdo().esVacio())
    this.getHijoIzquierdo().printPreorden();
   }
   if (!this.getHijoDerecho().esVacio()){
    this.getHijoDerecho().printPreorden();
   }
 }
 public boolean esVacio() {
    return (this.getDatoRaiz()==null);
 }
}
```

- *Inorden*: Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho. 10, 25, 32, 40, 78

```
Public class ArbolBinario <T> {
        Private NodoBinario <T> raiz;
        . . .
        Public void printPreorden() {
                If (!this.getHijoIzquierdo().esVacio()){
                        this.getHijoIzquierdo().printPreorden();
                }
                System.out.println(this.getDatoRaiz());
                If (!this.getHijoDerecho().esVacio()){
                        this.getHijoDerecho().printPreorden();
                }
        }
        Public boolean esVacio() {
                Return (this.getDatoRaiz()==null);
        }
}
```

- *Postorden*: Se procesan primero los hijos, izquierdo y derecho, y luego la raíz. 10, 32, 25, 78, 40

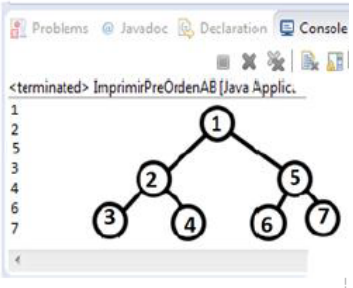
```
Public class ArbolBinario <T> {
        Private NodoBinario <T> raiz;
        . . .
        Public void printPreorden() {
                If (!this.getHijoIzquierdo().esVacio()){
                        this.getHijoIzquierdo().printPreorden();
                }
                If (!this.getHijoDerecho().esVacio()){
                        this.getHijoDerecho().printPreorden();
                }
                System.out.println(this.getDatoRaiz());

        }
        Public boolean esVacio() {
                Return (this.getDatoRaiz()==null);
        }
}
```

- *Por niveles*: Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc. 40, 25, 78, 10, 32

```
public class ArbolBinario<T> {

 private NodoBinario<T> raiz;
 . . .
 public void recorridoPorNiveles() {
  ArbolBinario<T> arbol = null;
  ColaGenerica<ArbolBinario<T>> cola = new ColaGenerica<ArbolBinario<T>>();
  cola.encolar(this);
  cola.encolar(null);
  while (!cola.esVacia()) {
    arbol = cola.desencolar();
    if (arbol != null) {
       System.out.print(arbol.getDatoRaiz());
       if (!arbol.getHijoIzquierdo().esVacio()){
           cola.encolar(arbol.getHijoIzquierdo());
       if (!arbol.getHijoDerecho().esVacio()){
           cola.encolar(arbol.getHijoDerecho());
    } else
       if (!cola.esVacia()) {
           System.out.println();
           cola.encolar(null);
       }
    }
 }}
```




**ARBOLES BINARIOS DE BUSQUEDA**

- Definición: es una colección de nodos conteniendo claves, que debe cumplir con una propiedad estructural y una de orden.
- Propiedades:
  - La propiedad estructural: es un árbol binario.
  - La propiedad de orden: para cada nodo N del árbol se cumple que todos los nodos ubicados en el subárbol izquierdo contienen claves menores que la clave del nodo N y los nodos ubicados en el subárbol derecho contienen claves mayores que la clave del nodo N.

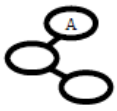
**ARBOLES AVL**

Un árbol AVL (Adelson–Velskii–Landis) es un árbol binario de búsqueda que cumple con la condición de estar balanceado.

- Propiedad de balanceo: cumple que, para cada nodo del árbol, la diferencia de altura entre el subárbol izquierdo y el subárbol derecho es a lo sumo 1.
- Características:
  - La propiedad de balanceo es fácil de mantener y garantiza que la altura del árbol sea de O(log n).
  - En cada nodo del árbol se guarda información de la altura.
  - La altura del árbol vacío es -1.
- Rebalanceo: Hay 4 casos posibles de desbalanceo a tener en cuenta, según donde se hizo la Inserción. El nodo A es el nodo desbalanceado.

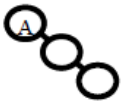
1. Inserción en el Subárbol IZQ del hijo IZQ de A


2. Inserción en el Subárbol DER del hijo IZQ de A




3. Inserción en el Subárbol IZQ del hijo DER de A


4. Inserción en el Subárbol DER del hijo DER de A

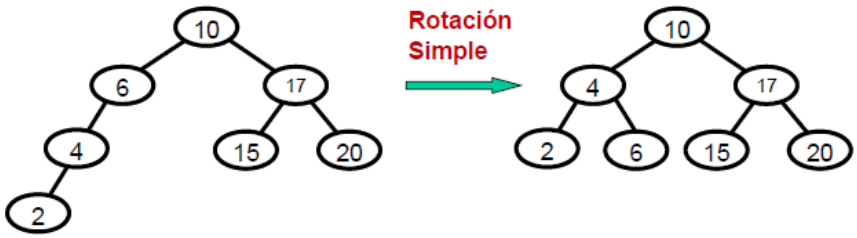



La solución para restaurar el balanceo es la ROTACION.

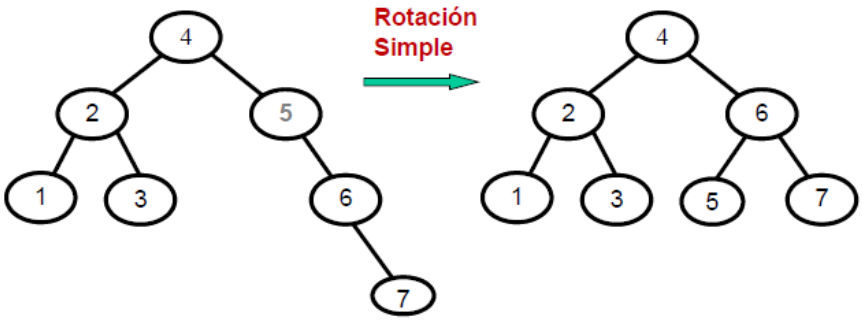
- Rotación: La rotación es una modificación simple de la estructura del árbol, que restaura la propiedad de balanceo, preservando el orden de los elementos.
  - Existen dos clases de rotaciones:

- Rotación Simple: Casos 1 y 4: inserción en el lado externo
- Rotación Doble: Casos 2 y 3: inserción en el lado interno
- Soluciones simétricas: En cada caso, los subárboles están opuestos.

- Rotación Simple Izquierda (RSI):

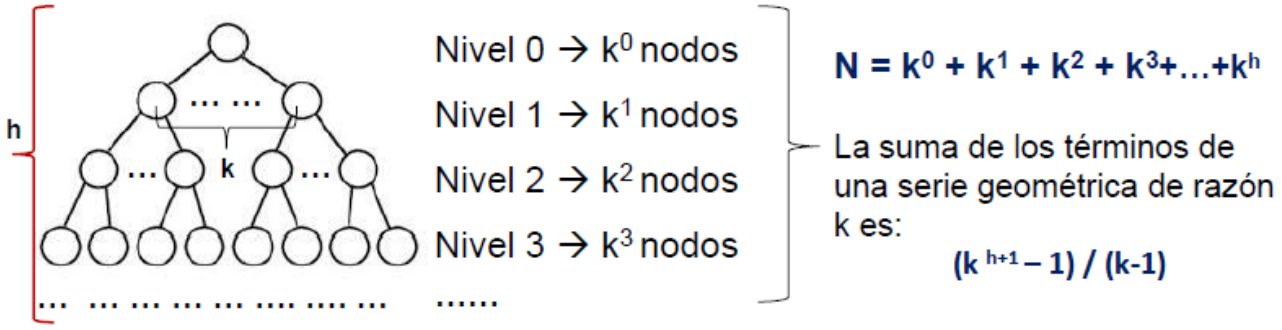



- Rotación Simple Derecha (RSD):

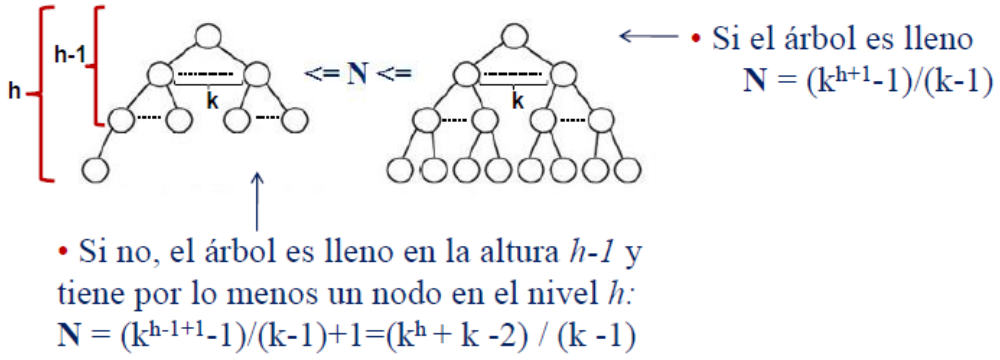



**ARBOLES GENERALES**

- Definición: Un árbol es una colección de nodos, tal que:
  - puede estar vacía. (Árbol vacío)
  - puede estar formada por un nodo distinguido R, llamado raíz y un conjunto de árboles T1, T2, ...Tk, k≥0 (sub árboles), donde la raíz de cada sub árbol Ti está conectado a R por medio de una arista.
- Descripción y terminología:
  - *Grado del árbol*: es el grado del nodo con mayor grado.
  - *Árbol lleno*: Dado un árbol T de grado k y altura h, diremos que T es lleno si cada nodo interno tiene grado k y todas las hojas están en el mismo nivel (h). Es decir, recursivamente, T es lleno si:
    - 1.-T es un nodo simple (árbol lleno de altura 0), o
    - 2.-T es de altura h y todos sus sub-árboles son llenos de altura h-1.
  - *Árbol completo:* Dado un árbol T de grado k y altura h, diremos que T es completo si es lleno de altura h-1 y el nivel h se completa de izquierda a derecha.
  - *Cantidad de nodos en un árbol lleno:* Sea T un árbol lleno de grado k y altura h, la cantidad de nodos N es (k elevado a la h+1, - 1) / (k-1) ya que:



- *Cantidad de nodos en un árbol completo:* Sea T un árbol lleno de grado k y altura h, la cantidad de nodos N varía entre (k elevado h,+ k - 2) / (k-1) y (k elevado a la h+1, - 1) / (k-1) ya que:

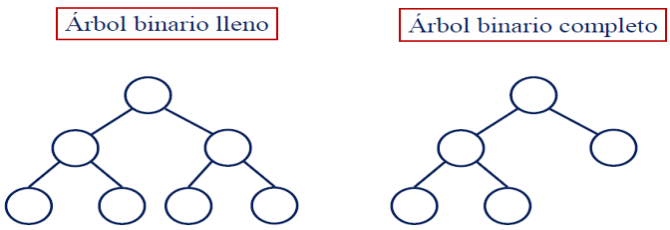



**COLAS DE PRIORIDAD**

Una cola de prioridad es una estructura de datos que permite al menos dos operaciones:

- Insert: Inserta un elemento en la estructura.
- Delete Min: Encuentra, recupera y elimina el elemento mínimo.

- Implementaciones:
  - Lista ordenada:
    - Insert tiene O(N) operaciones.
    - Delete Min tiene O(1) operaciones.
  - Lista no ordenada:
    - Insert tiene O(1) operaciones.
    - Delete Min tiene O(N) operaciones
  - Árbol Binario de Búsqueda
    - Insert y Delete Min tienen en promedio O(log N) operaciones
- **Heap binaria:** es una implementación de colas de prioridad que usa punteros y permite implementar ambas operaciones con O(log N) operaciones en el peor caso.
  - *Propiedades:*
    - *Propiedad Estructural:* Una Heap es un árbol binario completo.
      - En un árbol binario lleno de altura h, los nodos internos tienen exactamente dos hijos y las hojas tienen la misma profundidad.
      - Un árbol binario completo de altura h es un árbol binario lleno de altura h-1 y en el nivel h, los nodos se completan de izquierda a derecha.




- El número de nodos n de un árbol binario completo de altura h, satisface:

$$2^h \leq n \leq (2^{h+1}-1)$$

Demostración:

- Si el árbol es lleno, $n= 2^{h+1}-1$
- Si no, el árbol es lleno en la altura *h-1* y tiene por lo menos un nodo en el nivel *h:*

$n= 2^{h-1+1}-1+1=2^h$

La altura *h* del árbol es de **O(log n)**

- Dado que un árbol binario completo es una estructura de datos regular, puede almacenarse en un arreglo, tal que:
    - La raíz esta almacena en la posición 1.
    - Para un elemento que está en la posición i:
        - El hijo izquierdo está en la posición 2*i.
        - El hijo derecho está en la posición 2*i+1
        - El padre está en la posición | i/2 |

- *Propiedad de orden:*
    - Min Heap: el elemento mínimo esta almacenado en la raíz. El dato almacenado en cada nodo es menor o igual al de sus hijos.
    - Max Heap: el elemento máximo esta almacenado en la raíz. El dato almacenado en cada nodo es mayor o igual al de sus hijos.
- *Implementación:* un Heap H consta de:
        - Un arreglo que contiene los datos.
        - Un valor que me indica el número de elementos almacenados.
    - Ventaja:
        - No necesita usar punteros.
        - Fácil implementación de las operaciones.
    - Insert: el dato se inserta como último ítem de la Heap. Se debe hacer un filtrado hacia arriba para restaurar la propiedad de orden.
    - Filtrado hacia arriba: restaura la propiedad de orden intercambiando k a lo largo del camino hacia arriba desde el lugar de inserción. El filtrado termina cuando la clave k alcanza la raíz o un nodo cuyo padre tiene una clave menor. Ya que el algoritmo recorre la altura de la Heap, tiene O(log n) intercambios.

- *BuildHeap:*
    - Teorema: es un árbol binario lleno de altura h que contiene 2elevando h+1 – 1 nodos, la suma de las alturas de los nodos es: 2elevando h+1, -1 – (h+1)
    - Demostración: Un árbol tiene 2elevado i nodos de altura h – i

$$S = \sum_{i=0}^{h} 2^i (h-i)$$

S = h + 2 (h-1) + 4 (h-2) + 8 (h-3) + ……2h-1(1)

    - Un árbol binario completo no es un árbol binario lleno, pero el resultado obtenido es una cota superior de la suma de las alturas de los nodos en un árbol binario completo.
    - Un árbol binario completo tiene entre 2elevado a la h y 2elevando a la h+1, -1 nodos, el teorema implica que esta suma es de O(n) donde n es el número de nodos.
    - Este resultado muestra que la operación BuildHeap es lineal.
- *Ordenación de vectores:* Dado un conjunto de n elementos y se los quiere ordenar en forma creciente, existen dos alternativas:
    - *A)* Algoritmo que usa una Heap y requiere una cantidad aproximada de (n log n) operaciones.
      Construir una MinHeap, realizar n DeleteMin operaciones e ir guardando los elementos extraídos en otro arreglo.
    - *B)* Algoritmo HeapSort que requiere una cantidad aproximada de (n log n) operaciones, pero menos espacio.
      Construir una MaxHeap con los elementos que se desean ordenar, intercambiar el último elemento con el primero, decrementar el tamaño de la Heap y filtrar hacia abajo. Usa sólo el espacio de almacenamiento de la heap.