Estructuras de datos
Clase teórica 7

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

- Árboles
- Árboles binarios
- Árboles binarios de búsqueda

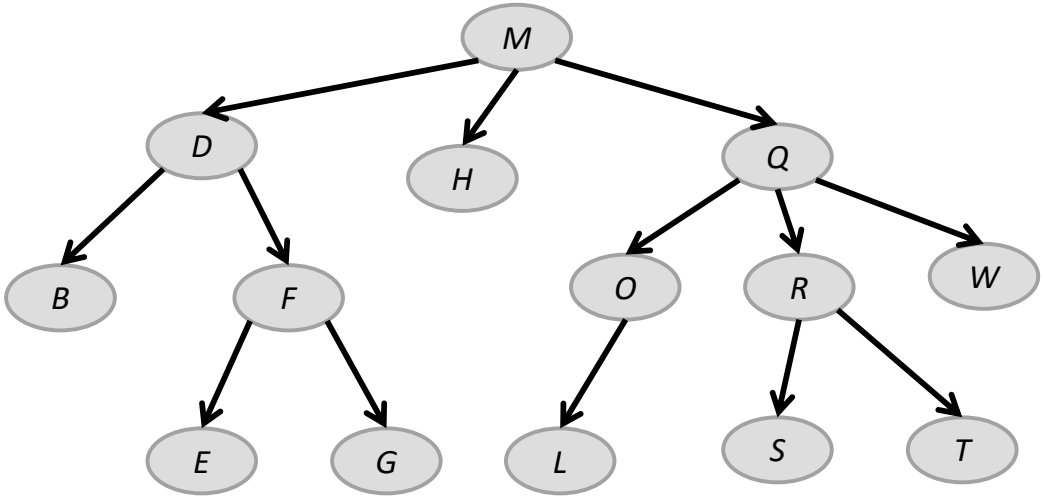
Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

¿Qué es eso de árboles?

Expectativa

Realidad



Definición y terminología

Un árbol es una estructura de datos con relaciones jerárquicas entre sus componentes. Su principal característica es que tales relaciones son “de uno a muchos”, es decir, no son necesariamente lineales (de uno a uno) como en el caso de las listas enlazadas y generalmente son unidireccionales.

Nodo: De manera similar a como se trabaja en las listas enlazadas, los nodos son contenedores para almacenar los elementos del árbol.

Arista: Son las “líneas” (en realidad las referencias) que conectan a los nodos y que representan las relaciones (jerárquicas) entre ellos. Un nodo puede tener cero, una o varias aristas que salen de él.

Terminología

Padre: *X* es padre de *Y* si hay una relación directa que va de *X* a *Y*. En la figura de ejemplo *F* es padre de *E*. Nótese que todo nodo tiene máximo un solo padre.

Hijo: Consecuentemente, *Y* es hijo de *X* si hay una relación directa que va de *X* a *Y*. En el ejemplo anterior, *E* es hijo de *F*, así como *S* es hijo de *R*. Nótese que un solo nodo puede tener varios hijos.

Hermano: *Y* y *Z* son hermanos si tienen el mismo padre. En el ejemplo anterior, *E* es hermano de *G*.

Terminología

Nodo raíz: Es el primer nodo del árbol. Se caracteriza por ser el único nodo que no tiene padre.

Nodos hojas: Son aquellos nodos que no tienen hijos.

Camino simple: El camino simple entre dos nodos es la secuencia de nodos que hay que recorrer para llegar de uno a otro. En la figura de ejemplo, el camino entre los nodos *M* y *L* es *M - Q - O - L*

Longitud de camino: Se refiere al número de nodos que este contiene (también es igual al número de aristas recorridas mas uno). En el ejemplo anterior la longitud es 4.

Terminología

Antecesores: Si *X* es el nodo raíz entonces no tiene antecesores, de otro modo el padre de *X* es su antecesor, así como todos los antecesores del padre de *X*. Otra forma de verlo es que los antecesores de *X* son aquellos nodos que se encuentran en el único camino simple que va desde *X* (sin incluirlo) hasta la raíz del árbol. En la figura de ejemplo los antecesores de *F* son *D* y *M*

Sucesores: Si *Y* es un nodo hoja entonces no tiene sucesores. De otro modo, cada hijo de *Y* es su sucesor, así como todos los sucesores los hijos de *Y*. En la figura de ejemplo los sucesores de *D* son *B, F, E* y *G*.

Terminología

Subárbol: Dado algún nodo de un árbol, dicho nodo junto con todos sus sucesores, forman un subconjunto del árbol.

Nivel o profundidad: El nivel de un nodo se define como la longitud del camino simple entre él y la raíz. La raíz tiene un nivel de 1. En la figura de ejemplo el nodo *W* está en el nivel 3.

Altura del árbol: Se define como el máximo nivel presente en el árbol. En la figura de ejemplo la altura del árbol es 4.

Árbol binario

Existen varios tipos de árboles, cada uno de ellos con propiedades especiales y usos específicos, pero en general todos se ajustan a la definición de árbol vista previamente.

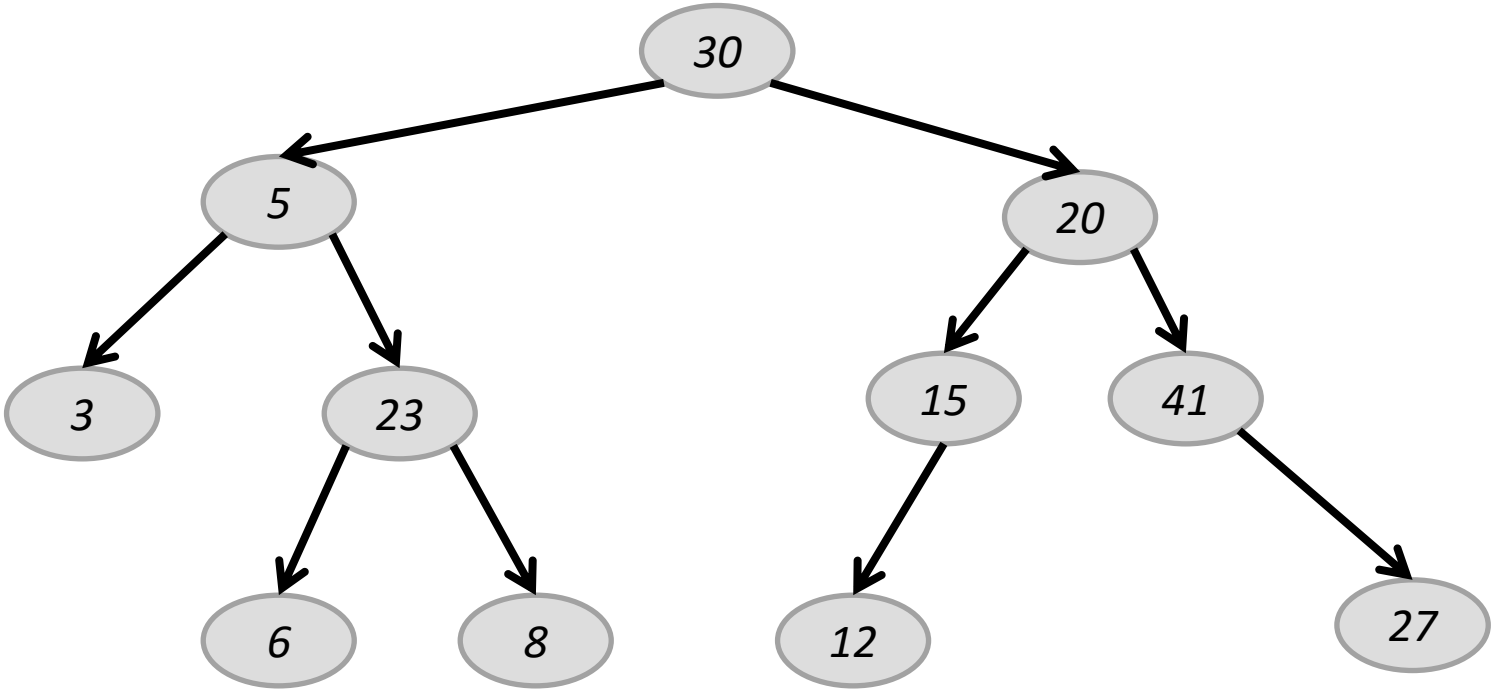
Un árbol binario por ejemplo tiene las siguientes propiedades distintivas:

- Cada nodo puede tener como máximo 2 subárboles, y por tanto no puede tener más de dos hijos.

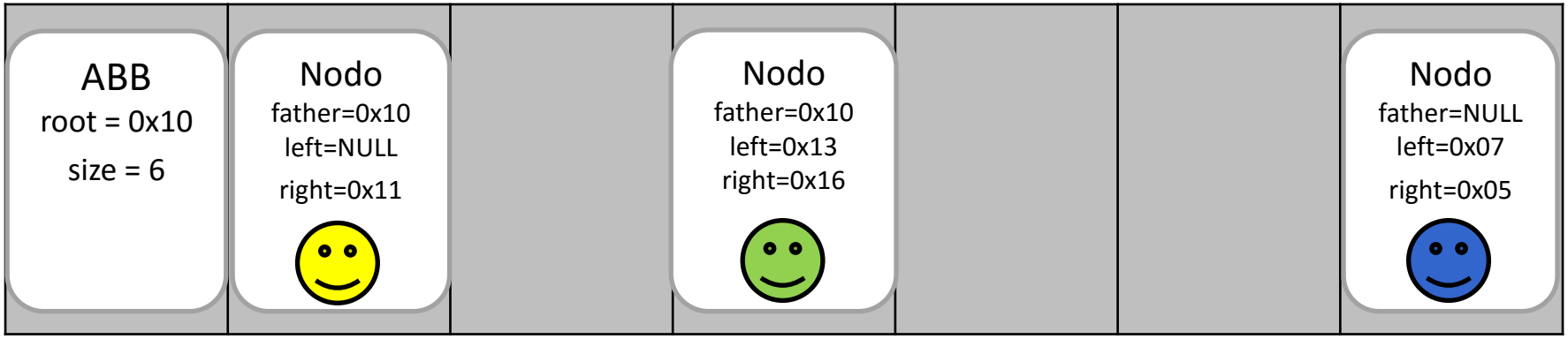
- Cada subárbol se identifica como el subárbol izquierdo o el subárbol derecho de su padre.

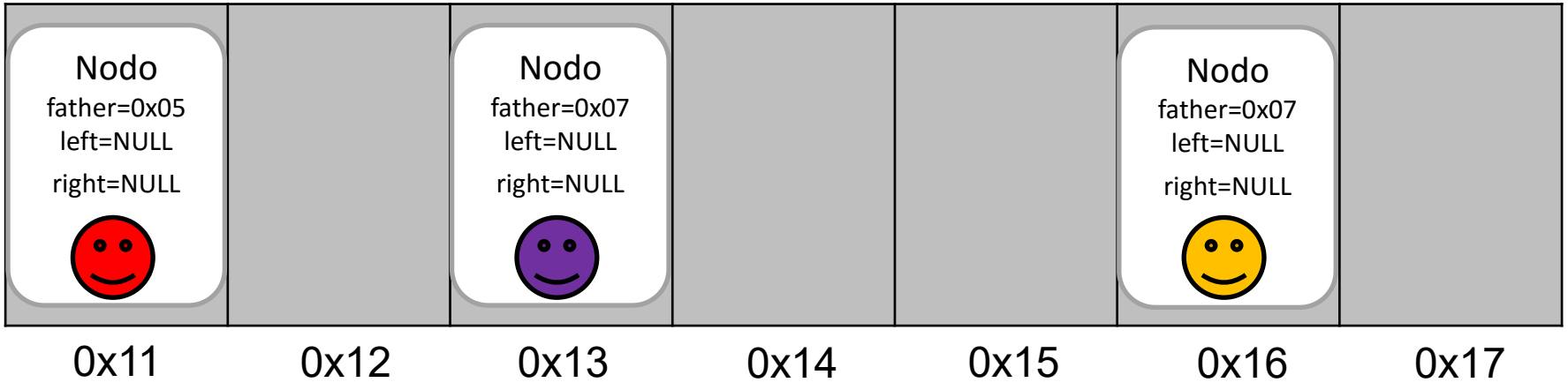
- Puede ser vacío.

Árbol binario



Estructura de un árbol binario





Recorridos típicos en árboles binarios

Pre-orden: Se procesa primero la raíz del subárbol, luego el subárbol izquierdo y por último el subárbol derecho.

```
//Este pseudo-código se limita a mostrar los elementos
void preOrder(nodo n){

   System.out.prinln(n.elm);

   if (n.hijoIzquierdo != null)
      preOrder(n.hijoIzquierdo);

   if (n.hijoDerecho != null)
      preOrder(n.hijoDerecho);
}
```

Recorridos típicos en árboles binarios

En-orden: Se procesa primero el subárbol izquierdo, luego la raíz del subárbol y por último el subárbol derecho.

```
//Este pseudo-código se limita a mostrar los elementos
void inOrder(nodo n){

  if (n.hijoIzquierdo != null)
      inOrder(n.hijoIzquierdo);

  System.out.prinln(n.elm);

  if (n.hijoDerecho != null)
      inOrder(n.hijoDerecho);
}
```

Recorridos típicos en árboles binarios

Pos-orden: Se procesa primero el subárbol izquierdo, luego el subárbol derecho y por último la raíz del subárbol.

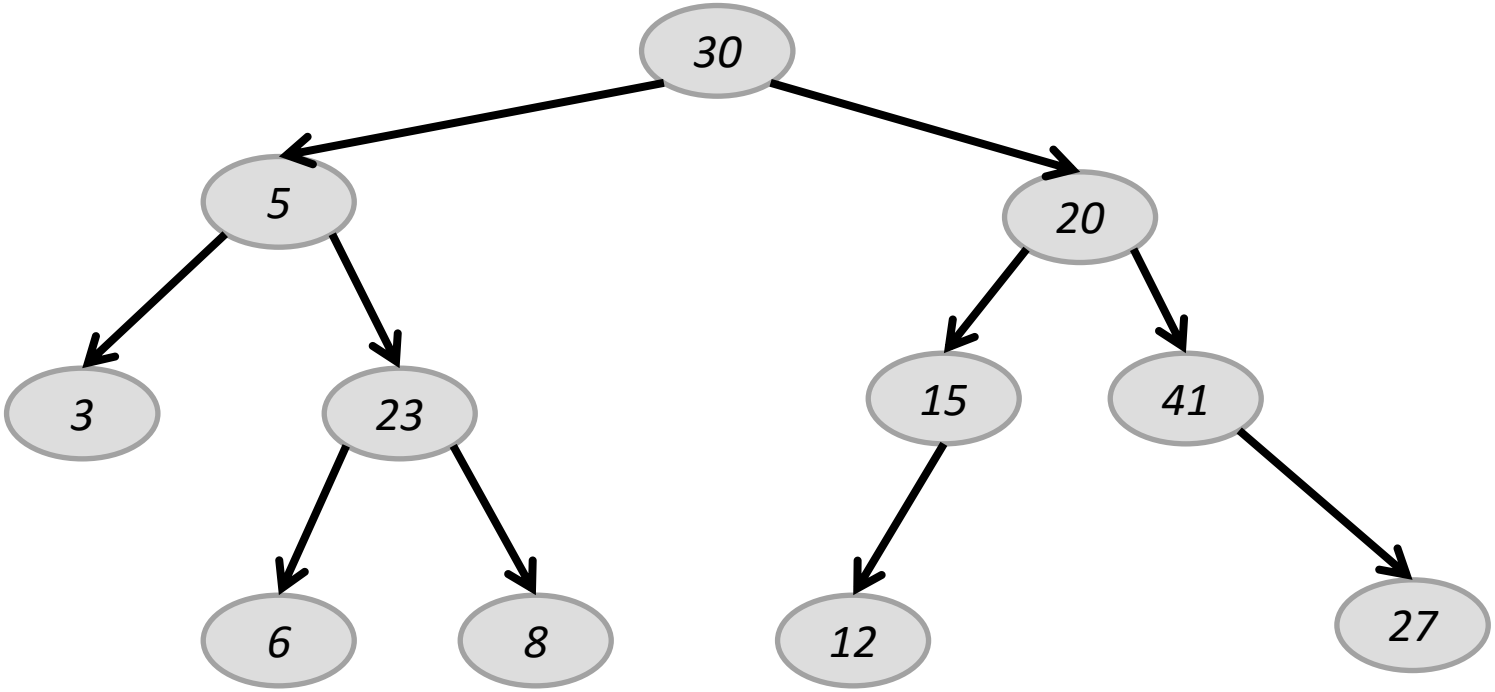
```
//Este pseudo-código se limita a mostrar los elementos
void posOrden(nodo n){

   if (n.hijoIzquierdo != null)
      posOrder(n.hijoIzquierdo);

   if (n.hijoDerecho != null)
      posOrder(n.hijoDerecho);

   System.out.prinln(n.elm);
}
```

Recorridos típicos en árboles binarios



Pre-orden: 30, 5, 3, 23, 6, 8, 20, 15, 12, 41, 27

En-orden: 3, 5, 6, 23, 8, 30, 12, 15, 20, 41, 27

Pos-orden: 3, 6, 8, 23, 5, 12, 15, 27, 41, 20, 30

Árbol binario de búsqueda

Un árbol binario de búsqueda es un tipo especial de árbol binario en el cual la posición de cada nodo en el árbol está determinada por el valor de alguno de los campos del elemento guardado en el nodo (generalmente el campo clave), el cual se conoce como campo de clasificación.

El árbol binario de búsqueda, como indica su nombre, hace que el proceso de buscar un nodo que contenga un elemento en particular sea altamente eficiente.

Árbol binario de búsqueda

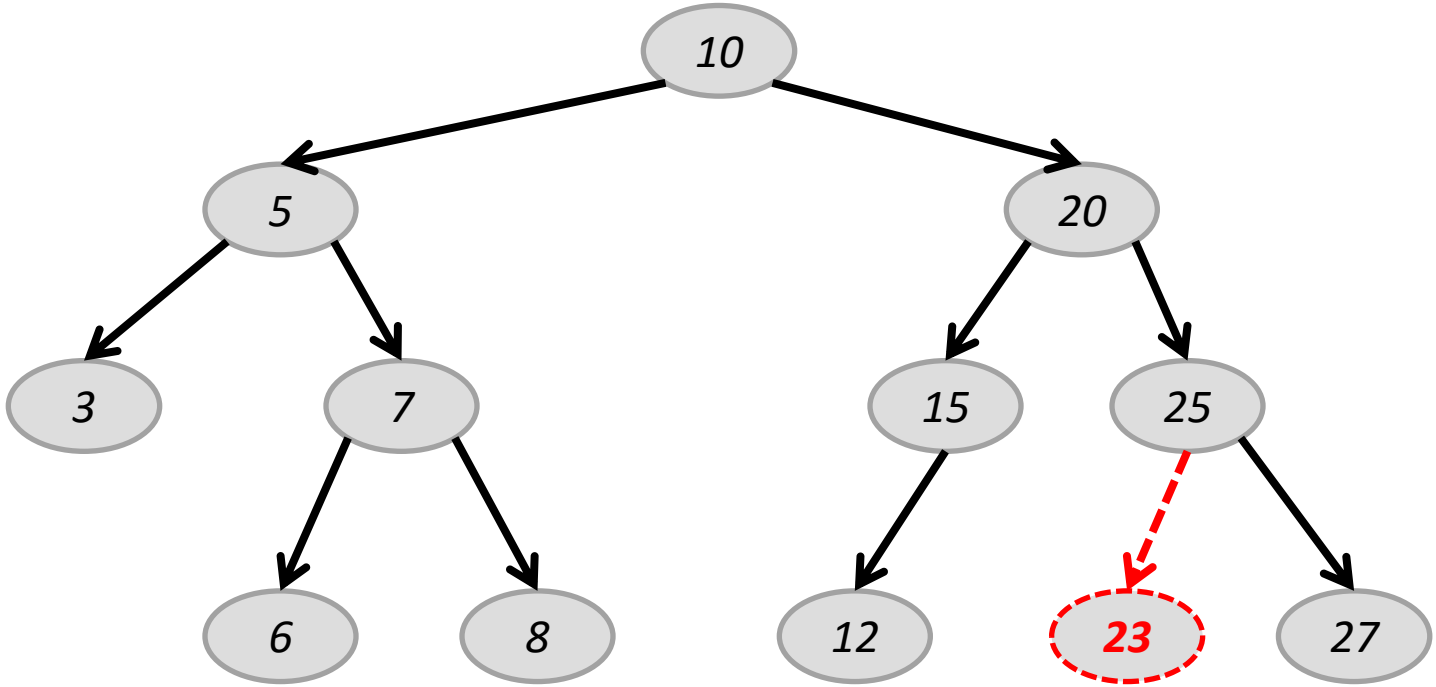
Para cada nodo *Y*, se deben cumplir las siguientes propiedades:

• Si *X* es un nodo en el subárbol izquierdo de *Y*, entonces *X* (y todos sus sucesores) son menores que *Y*.

• Si *Z* es un nodo en el subárbol derecho de *Y*, entonces *Z* (y todos sus sucesores) son mayores que *Y*.

Árbol binario de búsqueda

Ejemplo: ¿Dónde se ingresaría el elemento 23?



Altura de un ABB

Ejemplo 1: Ingresar los siguientes elementos (en ese orden) dentro de un ABB {50, 40, 35, 30, 25, 20, 10}

Ejemplo 2: Ingresar los siguientes elementos (en ese orden) dentro de un ABB {30, 20, 40, 10, 50, 25, 35}

A partir de estos ejemplos, ¿cuál es la altura en el peor de los casos de un árbol binario de búsqueda con *N* elementos? ¿cuál en el mejor? *N* y *log(N)* respectivamente

Búsqueda en un ABB

```
// e es el elemento a buscar
p = root // p es un nodo
encontrado = true
while(p.elm != e){
   if(e < p.elm && p.left != NULL){
       p = p.left
   }
   else if(e > p.elm && p.right != NULL){
       p = p.right
   }
   else{
       encontrado = false
       break
   }
}
return encontrado
```

¿Cuál es la eficiencia de este algoritmo? O(N)

Inserción en un ABB

```
// e es el elemento a insertar
p = new nodo(e)
if (root == NULL)
   root = p
else{
   aux = root
   while(true){
      if (e == aux.elm)
         return false
      else if(e < aux.elm && p.left == NULL){
         p.father = aux
         aux.left = p
         return true
      }
      else if(e < aux.elm && p.left != NULL)
         aux = aux.left
      else if(e > aux.elm && p.right == NULL){
         p.father = aux
         aux.right = p
         return true
      }
      else if(e > aux.elm && p.right != NULL)
         aux = aux.right
   }
} size ++
```

¿Cuál es la eficiencia de este algoritmo?

O(N)

Borrado en un ABB

A la hora de borrar un elemento, hay que tener en cuenta las siguientes cuatro casos:

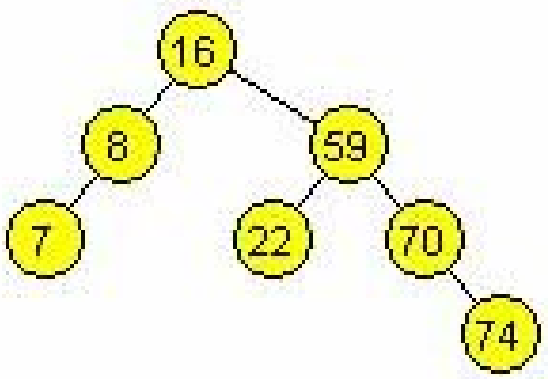
1. El elemento no está en el árbol
2. El elemento se encuentra en un nodo hoja
3. El elemento se encuentra en un nodo con un solo hijo
4. El elemento se encuentra en un nodo con los dos hijos

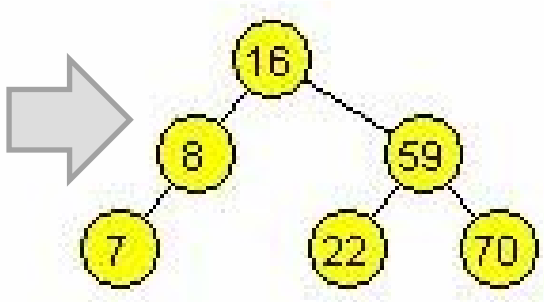
Borrado en un ABB

Caso 2: Borrado de nodo hoja

Este es el caso más sencillo, simplemente se elimina la relación de su nodo padre.

Ejemplo: borrado del nodo con código 74

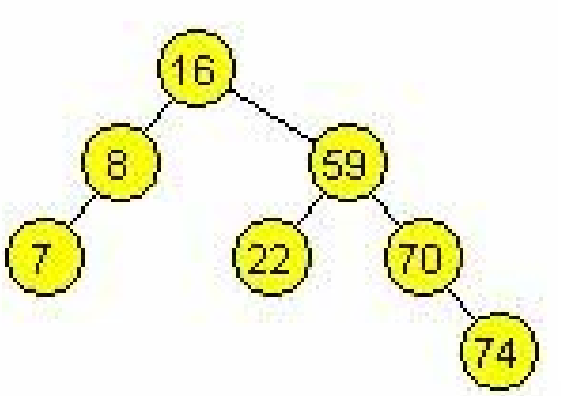


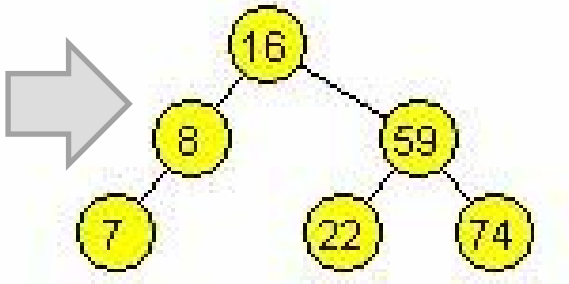


Borrado en un ABB

Caso 3: Borrado de nodo con un solo hijo
En este caso su hijo pasa a ser hijo de su padre.

Ejemplo: borrado del nodo con código 70





Borrado en un ABB

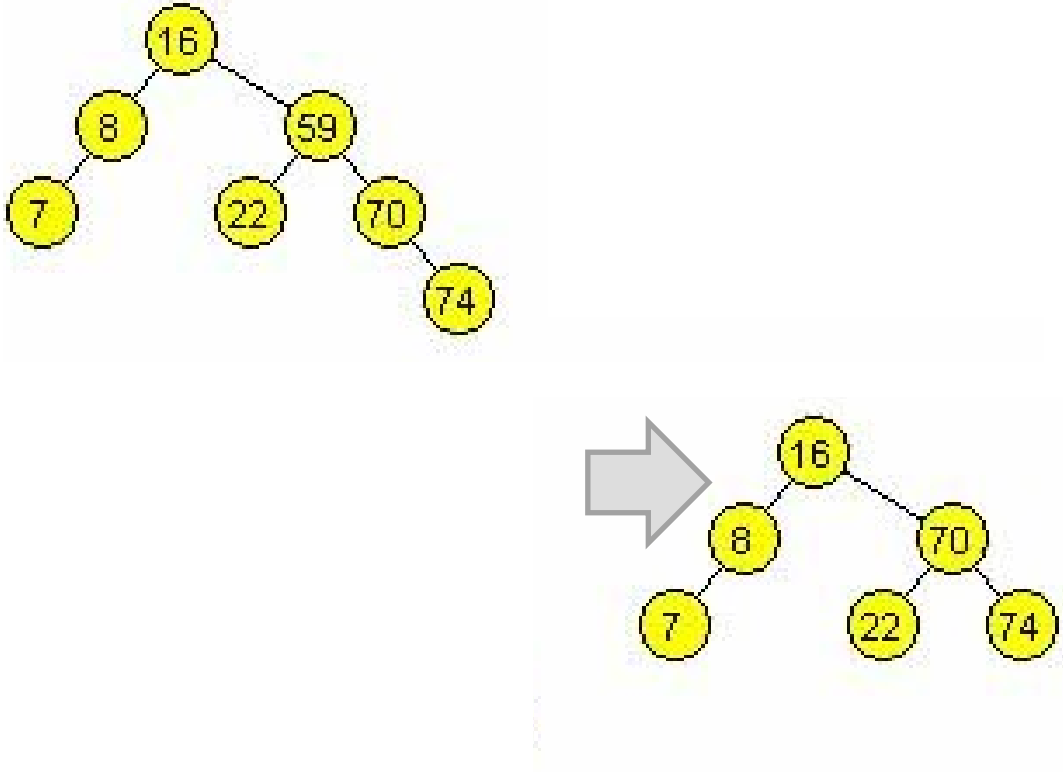
Caso 4: Borrado de nodo con dos hijos
Este es el caso más complejo pues no se puede perder ninguno de los hijos y además se debe mantener las propiedades del árbol binario de búsqueda. Para lograr esto se puede utilizar una de dos alternativas:

• Reemplazar el nodo que se quiere borrar por su sucesor más a la izquierda de su subárbol derecho (“menor de los mayores”), y si ese sucesor tiene hijo derecho, ese pasará a ser hijo izquierdo de su abuelo; ó

• Reemplazar el nodo que se quiere borrar por su sucesor más a la derecha de su subárbol izquierdo (“mayor de los menores”) , y si ese sucesor tiene hijo izquierdo, ese pasará a ser hijo derecho de su abuelo

Borrado en un ABB

Ejemplo: borrado del nodo con código 59 (buscando el menor de los mayores)



```
//e es el elemento a borrar
aux1 = root
aux2 = NULL
//se busca el nodo que contiene e (aux1) y su padre (aux2)
while (aux1 != NULL && aux1.elm != e){
   aux2 = aux1
   if (e < aux1.elm)
      aux1 = aux1.left
   else
      aux1 = aux1.right
}
if (aux1 == NULL)
   return false //caso 1
else{
   if (aux1.left == NULL && aux1.right == NULL){ //caso 2
      if (aux1 == root) //único elemento del árbol
         root = null
      else if (aux2.left == aux1) //hijo izquierdo
         aux2.left = NULL
      else //hijo derecho
         aux2.right = NULL
      return true
   }
   //...
```

```
//...
//Si no es hoja pero solo tiene subárbol izquierdo, caso 3
else if (aux1.left != NULL) && aux.right == NULL){
   if (aux1 == root)
      root = aux1.left
      aux1.father = NULL
   else if (aux2.left == aux1) //si era hijo izquierdo
      aux2.left = aux1.left
      (aux1.left).father = aux2
   else //si era hijo derecho
      aux2.right = aux1.left
      (aux1.left).father = aux2
   return true
}
//Si no es hoja pero solo tiene subárbol derecho, caso 3
else if (aux1.left == NULL && aux1.right != NULL){
   if (aux1 == root)
      root = aux1.right
      aux1.father = NULL
   else if (aux2.left == aux1) //si era hijo izquierdo
      aux2.left = aux1.right
     (aux1.right).father = aux2
   else //si era hijo derecho
      aux2.right = aux1.right
     (aux1.right).father = aux2
   return true
}   //...
```

```
    //...

    //caso 4
    else{//Buscando el menor de los mayores
        aux3 = aux1.right
        aux4 = aux1
        while (aux3.left != NULL){
            aux4 = aux3
            aux3 = aux3.left
        }
        //Se hace el reemplazo y se actualizan los enlaces
        aux1.elm = aux3.elm
        if (aux1 == aux4)
            aux1.right = aux3.right
        else
            aux4.left = aux3.right
        return true
    }
}
size--
```

¿Cuál es la eficiencia de este algoritmo? O(N)

Indexación en un ABB

Como ya sabemos, a diferencia de los arreglos o las listas enlazadas, los árboles no tienen un “esquema lineal”

Adicionalmente, como vimos en los ejemplos, las operaciones de inserción y borrado en un ABB producen que la posición de los elementos en un momento determinado no necesariamente tengan que ver con el “orden” en que entraron a la estructura.

Por tanto en los ABB, y en general en los árboles, hablar de índices usualmente no tiene sentido.

Tabla resumen

Recapitulando la clase de hoy tenemos que:

Estructura	Inserción	Indexación	Búsqueda	Borrado
Árbol binario de búsqueda	*O(n)*	No aplica	*O(n)*	*O(n)*

¿Cómo hacer para que siempre se garantice el que el peor de los casos sea *log(n)* y no *n*? … Eso le veremos la próxima clase