Estructuras de datos
Clase teórica 7

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN

Contenido

- Árboles binarios de búsqueda

Material elaborado por: Julián Moreno

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

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

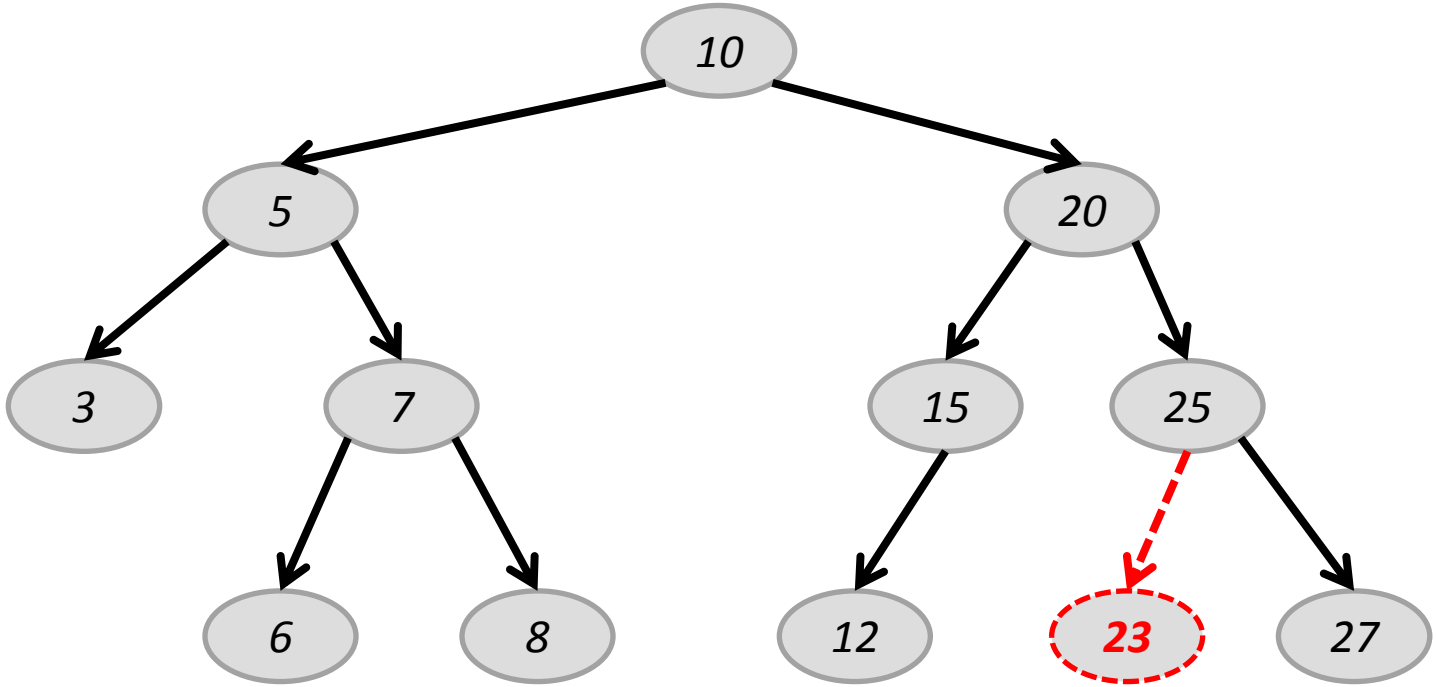
Para cada nodo *Y*, se deben cumplir las siguientes propiedades:

• Si *X* es un nodo en el subárbol izquierdo de *Y*, entonces el campo de clasificación de *X* es menor que el campo de clasificación de *Y*.

• Si *Z* es un nodo en el subárbol derecho de *Y*, entonces el campo de clasificación de *Z* es mayor que el campo de clasificación de *Y*.

Árbol binario de búsqueda

Ejemplo: ¿Dónde se ingresaría el elemento 23?



Por cierto, ¿Pre-orden? 10, 5, 3, 7, 6, 8, 20, 15, 12, 25, 23, 27

¿En-orden? 3, 5, 6, 7, 8, 10, 12, 15, 20, 23, 25, 27

¿Post-orden? 3, 6, 8, 7, 5, 12, 15, 23, 27, 25, 20, 10

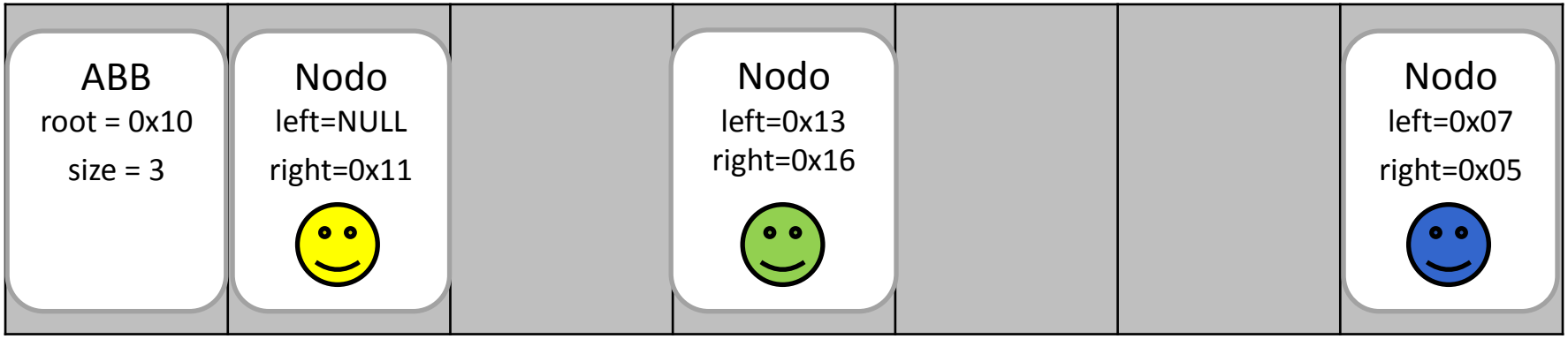
Altura de un ABB

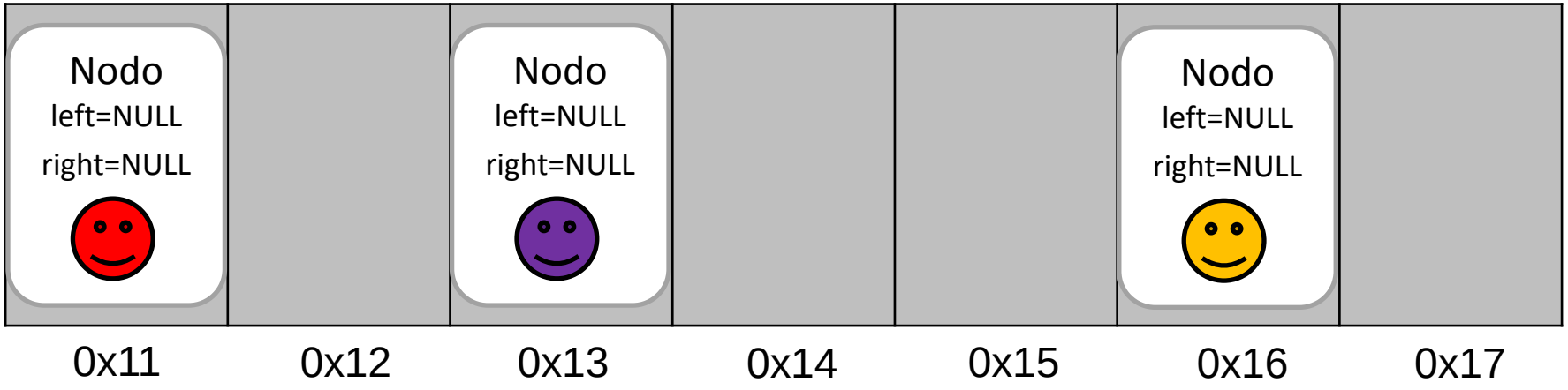
Ejemplo 1: Ingresar los siguientes elementos (en ese orden) dentro de un ABB {50, 40, 35, 30, 25, 20, 10}

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

Estructura de un ABB





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
```

¿De que va a depender la eficiencia de este algoritmo?

De la altura del árbol

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){
         aux.left = p
         return true
      }
      else if(e < aux.elm && p.left != NULL)
         aux = aux.left
      else if(e > aux.elm && p.right == NULL){
         aux.right = p
         return true
      }
      else if(e > aux.elm && p.right != NULL)
         aux = aux.right
   }
}
```

¿De que va a depender la eficiencia de este algoritmo?

De la altura del árbol

Borrado en un ABB

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

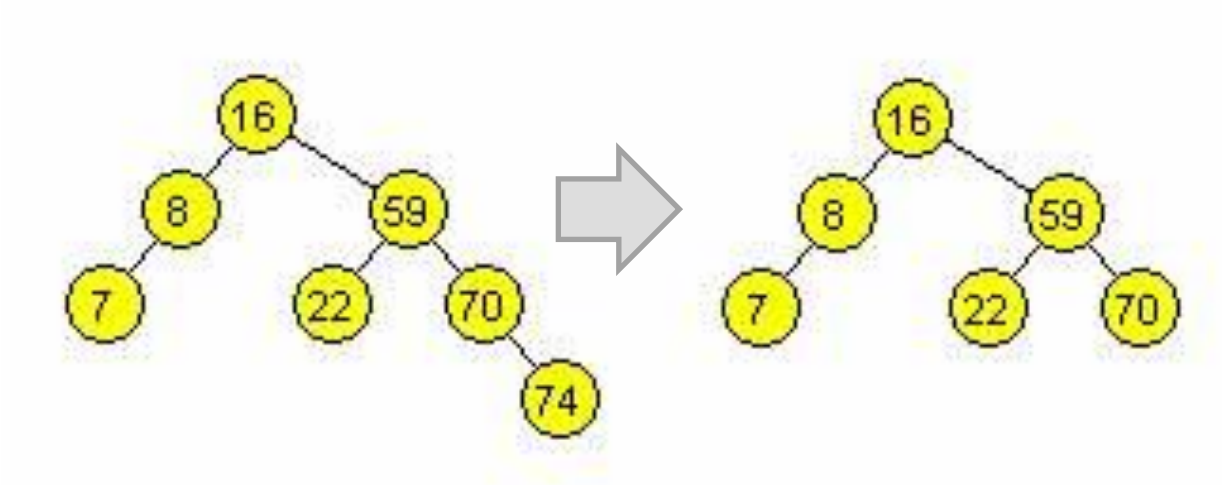
1. El elemento no está en el árbol
2. El elemento se encuentra en un nodo hoja
3. El elemento se encuentra en un nodo con un solo hijo
4. El elemento se encuentra en un nodo con los dos hijos

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



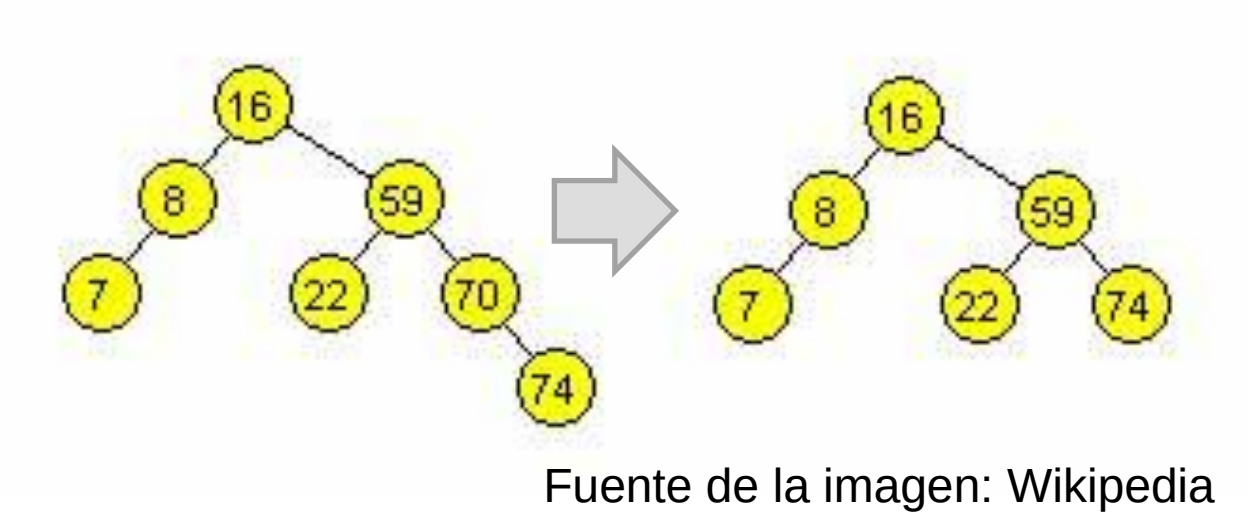
Fuente de la imagen: Wikipedia

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



Fuente de la imagen: Wikipedia

Borrado en un ABB

Caso 4: Borrado de nodo con dos hijos

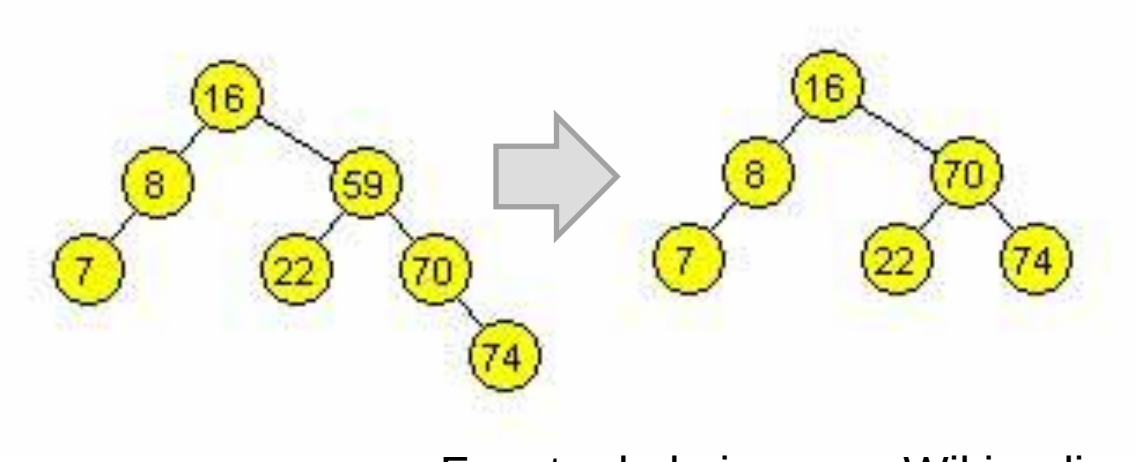
Este es el caso más complejo pues no se puede perder ninguno de los hijos y además se debe mantener las propiedades del árbol binario de búsqueda. Para lograr esto se puede utilizar una de dos alternativas:

- Reemplazar el nodo que se quiere borrar por su sucesor más a la izquierda de su subárbol derecho (“menor de los mayores”), ó

- Reemplazar el nodo que se quiere borrar por su sucesor más a la derecha de su subárbol izquierdo (“mayor de los menores”)

Borrado en un ABB

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



Fuente de la imagen: Wikipedia

```
//e es el elemento a borrar
aux1 = root
aux2 = NULL
//se busca el nodo que contiene e (aux1) y su padre (aux2)
while (aux != NULL && aux.elm != e){
   aux2 = aux1
   if (e < aux1.elm)
      aux1 = aux1.left
   else
      aux1 = aux1.right
}
if (aux == 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
   else if (aux2.left == aux1) //si era hijo izquierdo
      aux2.left = aux1.left
   else //si era hijo derecho
      aux2.right = aux1.left
   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
   else if (aux2.left == aux1) //si era hijo izquierdo
      aux2.left = aux1.right
   else //si era hijo derecho
      aux2.right = aux1.right
   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
    }
}
```

¿De que va a depender la eficiencia de este algoritmo?

De la altura del árbol

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 estuctura.

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(log(n))* en el mejor de los casos *O(n)* en el peor	No aplica	*O(log(n))* en el mejor de los casos *O(n)* en el peor	*O(log(n))* en el mejor de los casos *O(n)* en el peor

¿Cómo hacer para que siempre se garantice el mejor de los casos? … Eso le veremos la próxima clase