Árboles de Búsqueda Binaria

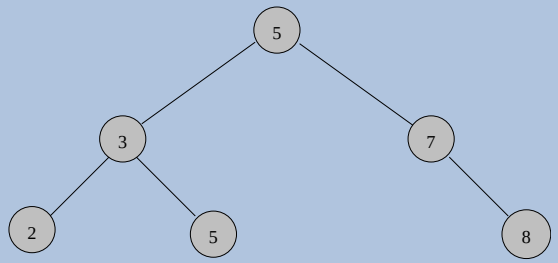
Profesor: Julio César López
jlopez@eisc.univalle.edu.co

17 de octubre de 2003

Contenido

Árbol de Búsqueda Binaria

Un árbol de búsqueda binaria está organizado en un árbol binario como se muestra en la siguiente figura:



Dicho árbol puede ser representado por una estructura de datos enlazada en la cual cada nodo es un objeto.

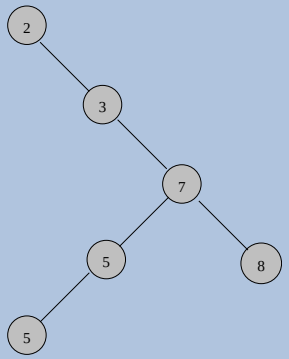
El árbol de búsqueda binaria de la figura tiene 6 nodos con altura 2.

Árbol de Búsqueda Binaria (cont.)

Las llaves en un árbol de búsqueda binaria siempre están almacenadas de modo que satisfagan la ***propiedad de árbol de búsqueda binaria***:

> Sea x un nodo en un árbol de búsqueda binaria. Si y es un nodo en el subárbol izquierdo de x, entonces $key[y] \leq key[x]$. Si y es un nodo en el subárbol derecho de x, entonces $key[x] \leq key[y]$.

El mismo conjunto de valores de la figura anterior puede ser representado en un árbol de búsqueda binaria menos eficiente:



Recorrido de un Árbol de Búsqueda Binaria

La propiedad descrita anteriormente, permite conocer todas las llaves de un árbol de búsqueda binaria, de forma ordenada, con un algoritmo recursivo llamado ***inorder tree walk***.

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2     then INORDER-TREE-WALK(left[x])
3          print key[x]
4          INORDER-TREE-WALK(right[x])
```

Recorrido (cont.)

- En el recorrido *inorder* la llave de la raíz del árbol se imprime entre los valores de su subárbol izquierdo y su subárbol derecho.
- Similarmente existe el algoritmo ***preorder tree walk***, el cual imprime la raíz antes de los valores de los subárboles, y el algoritmo ***postorder tree walk***, que imprime la raíz después de los valores en cada subárbol.
- Toma un tiempo $\Theta(n)$ recorrer un árbol de búsqueda binaria de n nodos, ya que después del llamado inicial, el procedimiento es llamado recursivamente exáctamente dos veces para cada nodo en el árbol.

Consulta de un Árbol de Búsqueda Binaria

- La operación más común realizada en un árbol de búsqueda binaria es la búsqueda de una llave almacenada en el árbol (SEARCH).
- Además, existen otras operaciones como MINIMUM, MAXIMUM, SUCCESSOR y PREDECESSOR.
- Cada una de estas operaciones pueden realizarse en un tiempo $O(h)$, donde h es la altura del árbol.

Búsqueda

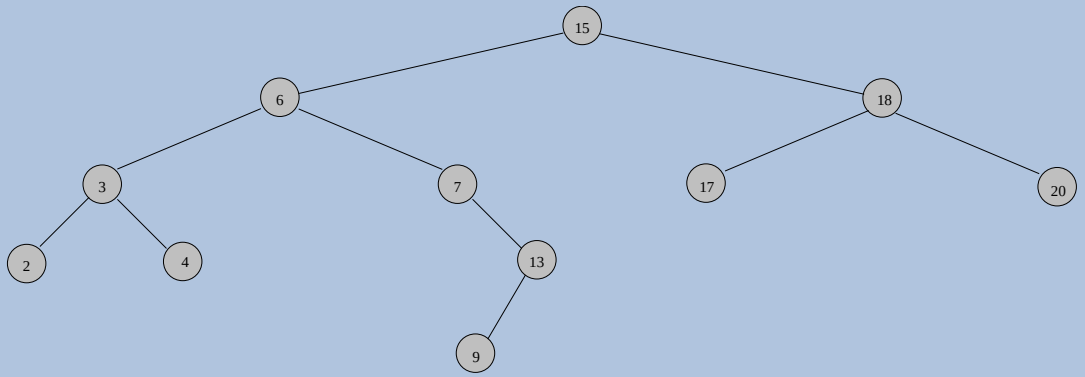
Se usa el siguiente procedimiento para buscar un nodo con una llave k dada en un árbol de búsqueda binaria con un apuntador a la raíz x:

TREE-SEARCH(x,k)
1 **if** $x =$ NIL or $k = key[x]$
2 **then return** x
3 **if** $k < key[x]$
4 **then return** TREE-SEARCH($left[x]$,k)
4 **else return** TREE-SEARCH($right[x]$,k)

Los nodos encontrados durante la recursión forman un camino desde la raíz, y por lo tanto el tiempo de ejecución de TREE-SEARCH es $O(h)$, donde h es la altura del árbol.

Búsqueda (cont.)

Dado el siguiente árbol:



La búsqueda de la llave 13 en el árbol, construiría el camino $15 \rightarrow 6 \rightarrow 7 \rightarrow 13$.

Búsqueda (cont.)

El mismo procedimiento de búsqueda puede ser escrito iterativamente cambiando la recursión por un ciclo **while**. En la mayoría de computadores, esta versión es más eficiente.

ITERATIVE-TREE-SEARCH(x,k)
1 **while** $x \neq$ NIL and $k \neq key[x]$
2 **do if** $k < key[x]$
3 **then** $x \leftarrow left[x]$
4 **else** $x \leftarrow right[x]$
5 **return** x

Mínimo y Máximo

Un elemento, en un árbol de búsqueda binaria, cuya llave es un mínimo, siempre se puede encontrar siguiendo los apuntadores al hijo *izquierdo* desde la raíz hasta que NIL sea encontrado.

El siguiente procedimiento retorna un apuntador al mínimo elemento en un subárbol con nodo raíz x.

TREE-MINIMUM(x)
1 **while** $left[x] \neq$ NIL
2 **do** $x \leftarrow left[x]$
3 **return** x

Mínimo y Máximo (cont.)

El procedimiento para hallar el máximo es simétrico.

TREE-MAXIMUM(x)
1 **while** $right[x] \neq$ NIL
2 **do** $x \leftarrow right[x]$
3 **return** x

La propiedad de árbol de búsqueda binaria, garantiza que TREE-MINIMUM y TREE-MAXIMUM son correctos.

Ambos procedimientos corren en un tiempo $O(h)$ en un árbol de altura h, ya que ellos trazan un camino descendente en el árbol.

Sucesor y Predecesor

- Dado un nodo en un árbol de búsqueda binaria, es importante saber cual es su sucesor en el orden dado por un recorrido *inorder*.
- Si todas las llaves son distintas, el sucesor de x es el nodo con la llave más pequeña, mayor a $key[x]$.
- La estructura de un árbol de búsqueda binaria permite determinar el sucesor de un nodo sin la necesidad de comparar llaves.

Sucesor y Predecesor (cont.)

El siguiente procedimiento retorna el sucesor de un nodo x en un árbol de búsqueda binaria si éste existe, de lo contrario retorna NIL (ya sea si el árbol es vacío o la llave de x es el máximo del árbol).

```
TREE-SUCCESSOR(x)
1  if right[x] ≠ NIL
2     then return TREE-MINIMUM(right[x])
3  y ← p[x]
4  while y ≠ NIL and x = right[y]
5     do x ← y
6        y ← p[y]
7  return y
```

El procedimiento TREE-PREDECESSOR es simétrico, y ambos corren en un tiempo $O(h)$ en un árbol de altura h.

Inserción

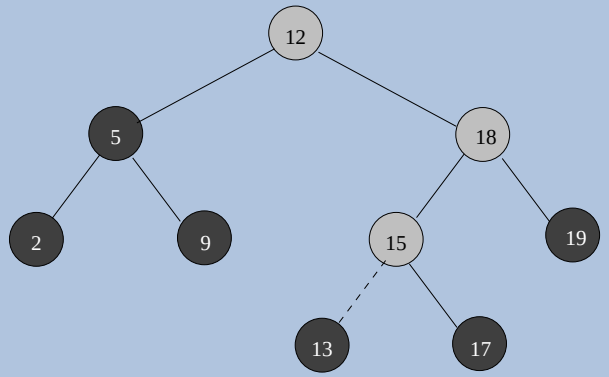
Para insertar un nuevo valor v en un árbol de búsqueda binaria T, se usa el siguiente procedimiento:

```
TREE-INSERT(T,v)
1   y ← NIL
2   x ← root[T]
3   while x ≠ NIL
4      do y ← x
5         if key[z] < key[x]
6           then x ← left[x]
7           else x ← right[x]
8   p[z] ← y
9   if y = NIL
10     then root[T] ← z
11     else if key[z] < key[y]
12             then left[y] ← z
13             else right[y] ← z
```

Este procedimiento, al igual que los anteriores, corre en un tiempo $O(h)$, siendo h la altura del árbol.

Inserción (cont.)

La siguiente figura muestra la inserción de un elemento con llave 13 en un árbol de búsqueda binaria.



Los nodos claros indican el camino desde la raíz a la posición donde el elemento es insertado. La línea punteada indica el enlace en el árbol que se agrega para insertar el elemento.

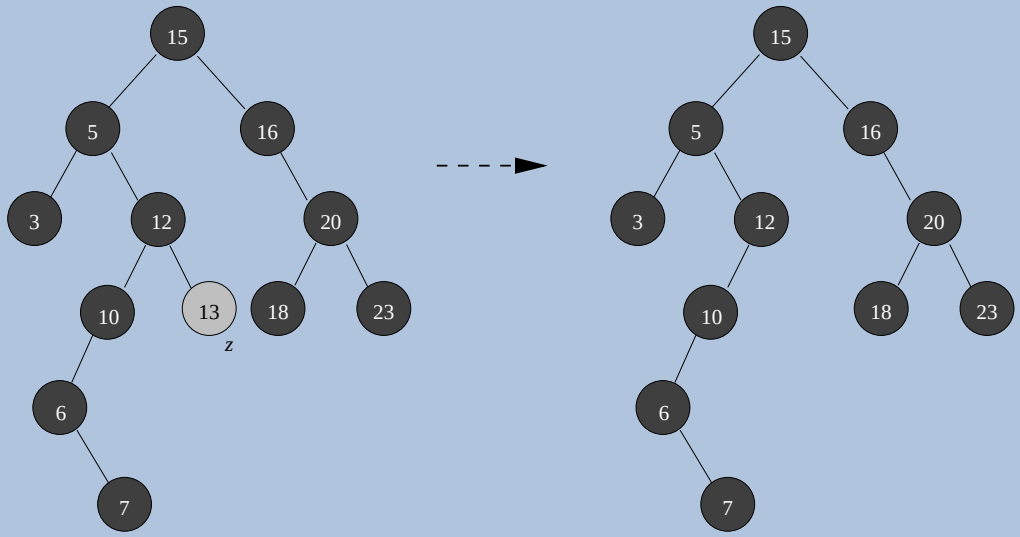
Eliminación

El procedimiento para eliminar un nodo z de un árbol de búsqueda binaria tiene tres casos:

Caso 1:

Si z no tiene hijos, se modifica su padre $p[z]$ para reemplazar z con NIL como su hijo.

Ejemplo:

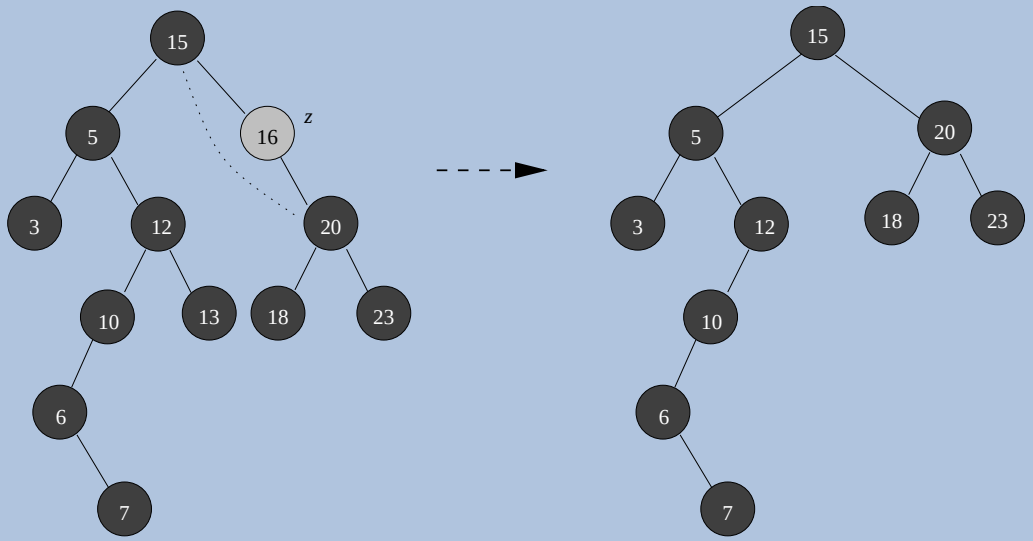


Eliminación (cont.)

Caso 2:

Si z tiene un solo hijo, simplemente se separa z del árbol.

Ejemplo:

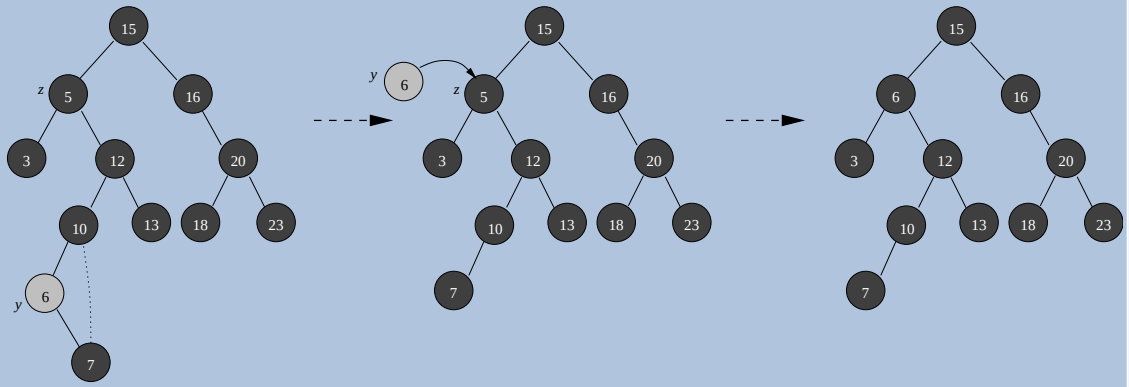


Eliminación (cont.)

Caso 3:

Si z tiene dos hijos, se separa su sucesor y, el cual tiene como máximo un hijo, y luego se reemplaza el contenido de z con el contenido de y.

Ejemplo:



Eliminación (cont.)

El código para el procedimiento TREE-DELETE es el siguiente:

```
TREE-DELETE(T,z)
1   if left[z] = NIL or right[z] = NIL
2      then y ← z
3      else y ← TREE-SUCCESSOR(z)
4   if left[y] ≠ NIL
5      then x ← left[y]
6      else x ← right[y]
7   if x ≠ NIL
8      then p[x] ← p[y]
9   if p[y] = NIL
10     then root[T] ← x
11     else if y = left[p[y]]
12             then left[p[y]] ← x
13             else right[p[y]] ← x
14  if y ≠ z
15     then key[z] ← key[y]
16          ▷ If y has other fields, copy them, too.
17  return y
```