Estructuras de Datos
Árboles

Tópicos

- Búsqueda binaria
- Árboles de búsqueda binaria
  - Búsqueda en un ABB
  - Inserción en un ABB
  - Eliminación en un ABB

Búsqueda binaria

Supongamos que se dispone del arreglo a, de tamaño n en el cual queremos buscar un elemento x. El mismo puede estar o no presente en el arreglo.
Para realizar esto podemos usar una Búsqueda Lineal, es decir, vamos recorriendo secuencialmente cada elemento comparándolo con x hasta que sea encontrado o hasta que todos los elementos hayan sido comparados.

Búsqueda binaria

Pensemos ahora qué sucedería si ahora nuestro arreglo a, de tamaño n tiene almacenado un conjunto de elementos ordenados de menor a mayor.
¿Podemos seguir haciendo de la Búsqueda Lineal? La respuesta es que sí pero, con la mejora que, en caso de encontrar un elemento mayor al buscado finalice. En ese caso ya sabemos que el elemento buscado x no se encuentra en el arreglo a.

Búsqueda binaria

Sin embargo existe una variante de la Búsqueda Lineal llamada Búsqueda binaria o dicotómica sobre arreglos ordenados la cual consiste en:

para buscar un elemento `x` dentro del arreglo `a`, de tamaño `n` se debe:

- buscar el índice de la posición media del arreglo (en donde se busca el elemento) que se denotará `m`. Inicialmente, `m = n/2`.
- si `a[m]=x` se encontró el elemento (fin de la búsqueda), en caso contrario se sigue buscando en el lado derecho o izquierdo del arreglo dependiendo si `a[m]<x` ó `a[m]>x` respectivamente.

Búsqueda binaria

La variable `i` representa el primer casillero del arreglo en donde es posible que se encuentre el elemento x, la variable `j` representa el último casillero del arreglo hasta donde x puede pertenecer, con `j>=i`.

Inicialmente, `i=0` y `j=n-1`.

Búsqueda binaria

En cada iteración,

- Si el conjunto es vacío (`j-i < 0`), o sea si `j < i`, entonces el elemento `x` no está en el conjunto (búsqueda infructuosa).
- En caso contrario, `m = (i+j)/2`.
  Si `x=a[m]`, el elemento fue encontrado (búsqueda exitosa).
  Si `x<a[m]` se modifica `j=m-1`, sino se modifica `i=m+1` y se sigue iterando.

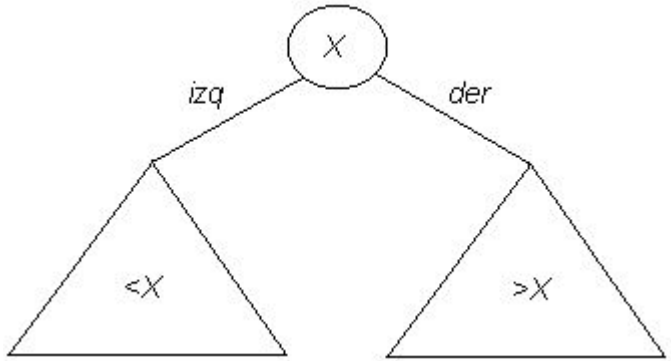
Búsqueda binaria

A continuación se muestra una versión iterativa del algoritmo. Se deja para pensar la versión recursiva del mismo.

```
#define NO_ENCONTRADO -1
int binary_search(int* a, int n, int x) {
  int i=0, j=n-1, m, posicion = NO_ENCONTRADO;
  while (i<=j && posicion == NO_ENCONTRADO) {
    m=(i+j)/2;
    if (x==a[m]) posicion = m;
    else if (x<a[m]) j=m-1;
    else i=m+1;
  }
  return posicion;
}
```

Árboles de búsqueda binaria

Un Árbol de búsqueda binaria, en adelante ABB, es un árbol binario en donde todos los nodos cumplen la siguiente propiedad (sin perder generalidad se asumirá que los elementos almacenados son números enteros): si el valor del elemento almacenado en un nodo N es X, entonces todos los valores almacenados en el subárbol izquierdo de N son menores que X, y los valores almacenados en el subárbol derecho de N son mayores que X.



Búsqueda en un ABB

Esta operación retorna un puntero al nodo en donde se encuentra el elemento buscado, x, o NULL si dicho elemento no se encuentra en el árbol. La estructura del árbol facilita la búsqueda:

- Si el árbol está vacío, entonces el elemento no está y se retorna NULL.
- Si el árbol no está vacío y el elemento almacenado en la raíz es X, se encontró el elemento y se retorna un puntero a dicho nodo.
- Si X es menor que el elemento almacenado en la raíz se sigue buscando recursivamente en el subárbol izquierdo, y si X es mayor que el elemento almacenado en la raíz se sigue buscando recursivamente en el subárbol derecho.

Búsqueda en un ABB

Como podemos ver esta función es recursiva y, podemos escribirla de la siguiente manera tomando la definición de BTree comentada en la presentación anterior:

```
BTree btree_search(BTree nodo, int dato){
    if (btree_empty(nodo))
        return NULL;
    if (nodo->dato == dato)
        return nodo;
    if (nodo->dato>dato)
        return btree_search(nodo->left, dato);
    else
        return btree_search(nodo->right, dato);
}
```

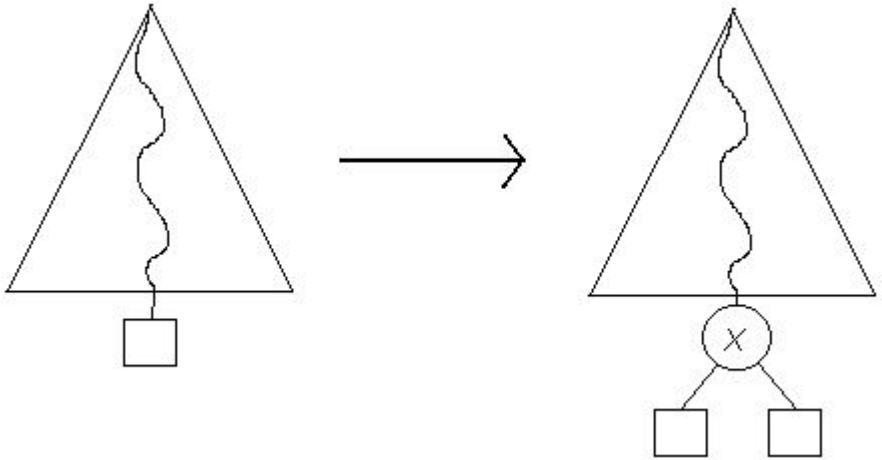
Búsqueda en un ABB

Otra versión sería, usando el if ternario:

```
BTree btree_search(BTree nodo, int dato){
    if (btree_empty(nodo) || nodo->dato == dato) return nodo;
    return nodo->dato>dato?btree_search(nodo->left, dato):btree_search(nodo->right, dato);
}
```

Inserción en un ABB

Para insertar un elemento X en un ABB, es equivalente a buscarlo, ya que nos posicionamos en el lugar donde debería haber estado usando la propiedad de ABB.



Inserción en un ABB

El código sería:

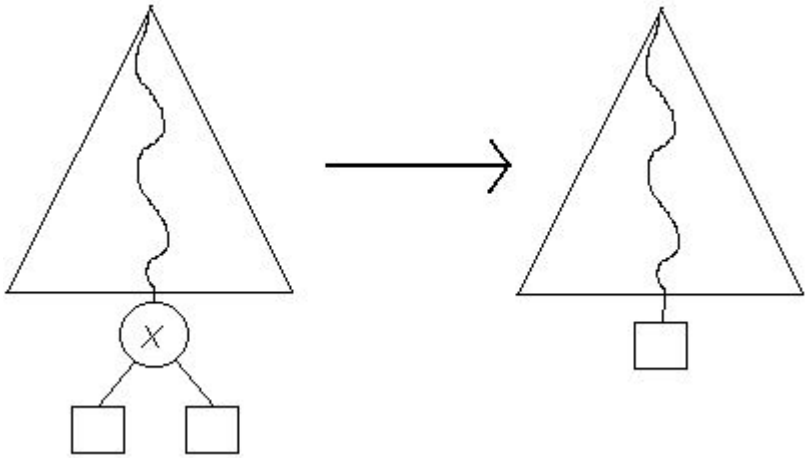
```
BTree btree_insert(BTree nodo, int dato) {
    if (btree_empty(nodo)) {
        nodo = malloc(sizeof(BTNodo));
        nodo->left = NULL;
        nodo->right = NULL;
    }
    else if (nodo->dato > dato)
        nodo->izq = btree_insert(nodo->izq, dato);
    else
        nodo->der = btree_insert(nodo->der, dato);

    return nodo;
}
```

Eliminación en un ABB

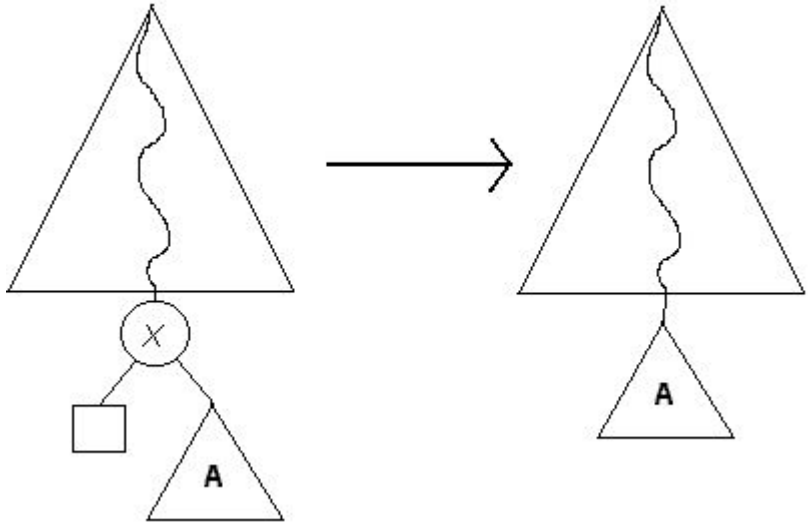
Primero se realiza una búsqueda del elemento a eliminar, digamos X. Si la búsqueda fue infructuosa no se hace nada, en caso contrario hay que considerar los siguientes casos posibles:

1. Si X es una hoja sin hijos, se puede eliminar inmediatamente.



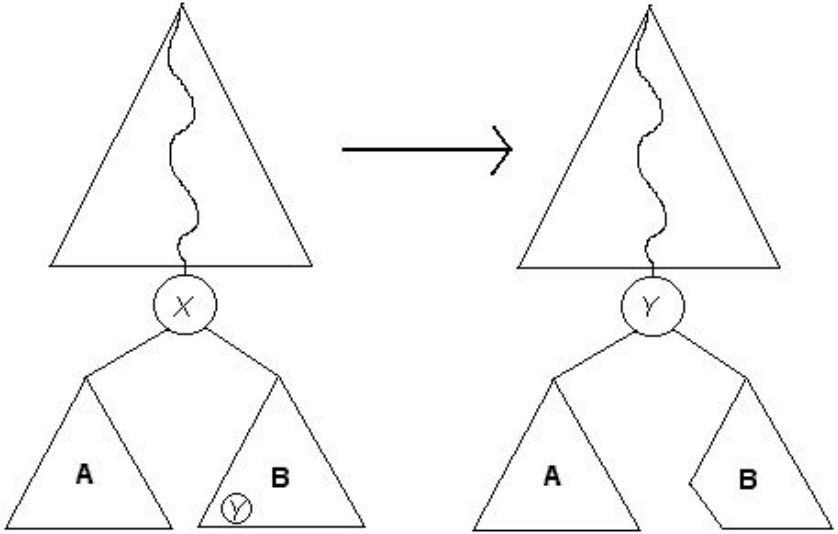
Eliminación en un ABB

2. Si X tiene un solo hijo, entonces se cambia la referencia del padre a X para que ahora referencie al hijo de X.



Eliminación en un ABB

3. Si X tiene dos hijos, es el caso complicado, tenemos que analizar qué dato se puede ubicar en el lugar de X para mantener la propiedad de ABB y que implique la menor cantidad de cambios posible.
   Si tomamos a Y = mínimo nodo del subárbol derecho de X, es decir, sería el nodo más a la izquierda del subárbol derecho de X.



Eliminación en un ABB

3. Este nodo Y tiene la característica de ser menor que todos los que están en el subárbol derecho de X y, mayor que los que están en el subárbol izquierdo de X. Es decir, lo podemos ubicar directamente en el lugar de X y seguir preservando la propiedad de ABB.
   Análogamente, se podría hacer tomando el máximo nodo del subárbol izquierdo de X.