



**INSTITUTO POLITÉCNICO NACIONAL  
SECRETARÍA ACADÉMICA  
DIRECCIÓN DE EDUCACIÓN SUPERIOR  
ESIME CULHUACAN**

FECHA		
DÍA	MES	AÑO

**NOMBRE ALUMNO:** \_\_\_\_\_  
Apellido paterno  
\_\_\_\_\_  
Apellido materno Nombre (s)  
**No.**  
**BOLETA** \_\_\_\_\_ **GRUPO** \_\_\_\_\_

**INGENIERÍA EN COMPUTACIÓN**  
**ASIGNATURA**  
**ESTRUCTURAS DE DATOS**  
**PROFESOR**  
**M. en C. BEATRIZ EUGENIA CORONA**  
**RAMÍREZ**

**PRÁCTICA No. 15  
ÁRBOL B**

**1. Objetivo**

El alumno usará los algoritmos de ordenación interna y externa para el ordenamiento de datos.

**2. Material y Equipo**

Computadora, Compilador C y/o Java y unidad de almacenamiento (USB)

**3. Introducción teórica**

Los diferentes tipos de árboles binarios estudiados hasta el momento fueron desarrollados para funcionar en la memoria principal de la computadora. Sin embargo, existen muchas aplicaciones en las que el volumen de información es tal, que los datos caben en la memoria principal y es necesario almacenarlos, organizados en archivos en dispositivos de almacenamiento secundario. Esta organización de archivos debe ser suficientemente adecuada como para recuperar los datos en forma eficiente.

El Árbol B es un TDA de búsqueda equilibrado, diseñado para ser usado con grandes conjuntos de datos en almacenamiento secundario. Generalmente se considera a los Árboles B como el mejor método para implementar al TDA dinámico en una unidad de disco.

A diferencia de los Árboles Binarios (que sólo podían almacenar un dato en cada nodo, induciendo así a realizar un acceso al disco cada vez que se carga un dato en el árbol antes de ser procesado), el Árbol B accede al disco mediante bloques de datos, es decir, agrupa los datos en paquetes para su lectura o escritura de así serlo.

Esta propuesta reduce bastante el número de accesos al dispositivo secundario, optimizando así el rendimiento de nuestro sistema informático.

Así por ejemplo, supongamos que tenemos un archivo con nueve mil registros de empleados (RUT, nombre, apellido, dirección, cargo) de 100 Bytes cada registro, como se muestra en la figura. Además sabemos que el bloque del disco es de 512 Bytes y que cada puntero al bloque de disco es de 2 Bytes.



### Representación de un registro

Si hubiésemos usado un ABB para administrar los datos en este archivo, tendríamos que haber accedido al disco unas 9000 veces para cargar el árbol completo en memoria principal (lo cual es bastante lento).

Ahora si usamos un Árbol B con 5 registros (500 Bytes) y 6 punteros (12 Bytes) por cada nodo, tendremos que acceder al disco unas 1800 veces. Ya que empaquetados los datos en bloques de 5 registros por nodo. Esto es mucho más rápido y eficiente que en el caso de los ABB.

Por lo tanto, ya debemos tener clara la idea que el Árbol B es el TDA óptimo para administrar una gran cantidad de datos en memoria secundaria. Ahora que sabemos el uso de los Árboles B y su forma de trabajar, les mostraremos su definición formal.

A fines de los años sesenta, R. Bayer y E. McCreight postularon un criterio muy razonable de organizar datos en un fichero externo, lo llamaron Árbol B. Se dice que un Árbol es B de orden  $n$  si:

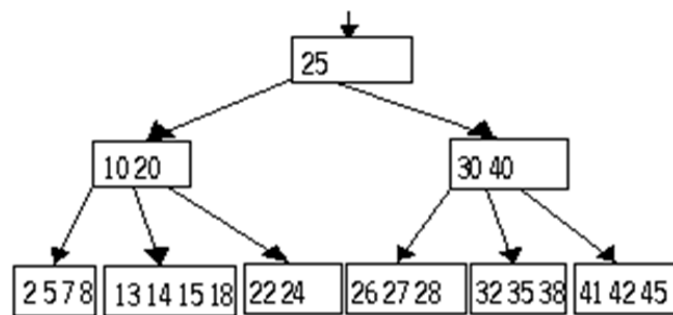
Cada página contiene a lo sumo  $2n$  elementos (llaves).

Cada página, excepto la de la raíz, contiene  $n$  elementos por lo menos.

Cada página es una página de hoja, o sea que no tiene, descendientes o tiene  $m+1$  descendientes, donde  $m$  es el número de llaves en esta página.

Todas las páginas de hoja aparecen al mismo nivel.

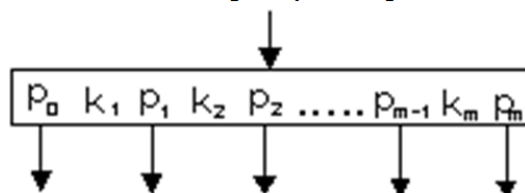
La figura muestra un Árbol B de orden 2 con 3 niveles. Todas las páginas tienen 2, 3 o 4 elementos; la excepción es la raíz que puede contener un solo elemento únicamente. Todas las páginas de hoja aparecen en el nivel 3.



Árbol B de orden 2

## Búsqueda

Examinemos la figura y un argumento de búsqueda x.



Página de Árbol B con m llaves

Suponiendo que hemos cargado en memoria primaria una página P del Árbol B, entonces podemos aplicar los métodos ordinarios de búsqueda entre las llaves  $k_1, \dots, k_m$ .

*Nota: Si m es muy grande, se puede hacer una búsqueda del tipo "Dividir para reinar". Pero si es pequeña, bastará con realizar una búsqueda Secuencial.*

Si la búsqueda fracasa, nos encontraremos en una de las siguientes situaciones:

$k_{i-1} < x < k_{i+1}$  para  $1 \leq i < m$ . Proseguimos la búsqueda en la página  $p_i$

$k_m < x$  La búsqueda prosigue en la página  $p_m$

$x < k_1$  La búsqueda prosigue en la página  $p_0$

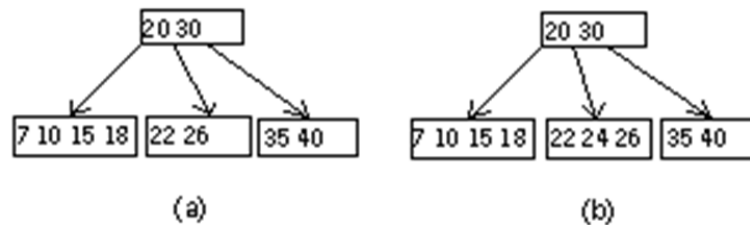
Si en algún caso el apuntador designado es NULL, esto es, si no hay página de hijo, entonces tampoco existe un elemento con la llave x en el Árbol B y la búsqueda finaliza.

## Inserción

Es interesante señalar que la inserción en un Árbol B de orden n es relativamente sencilla. Si hay que insertar un elemento en una página con  $m < 2n$  elementos, el proceso de inserción queda limitado a esa página.

Para analizar esta situación, considérese la siguiente figura que muestra un Árbol B de orden 2. Puesto que cada página en un Árbol B de orden  $n$  (excepto la raíz) contiene entre  $n$  y  $2n$  elementos, cada página del ejemplo tiene entre 2 y 4 elementos.

En cada página debe existir un indicador (que no está reflejado en la figura) para informar sobre el número de elementos que tiene la página. Primero se procede a Buscar desde la raíz hasta localizar la página apropiada para la inserción. Entonces se realiza la inserción. Refiriéndonos a la figura (a), uno puede ver que cuando se inserta el elemento 24, la Búsqueda termina sin éxito en la segunda hoja. Puesto que la hoja puede alojar otro elemento, se inserta el elemento nuevo simplemente, dando lugar al Árbol que se muestra en la figura (b).



(a) Un Árbol B de orden 2, y (b) el mismo árbol tras la inserción del elemento 24

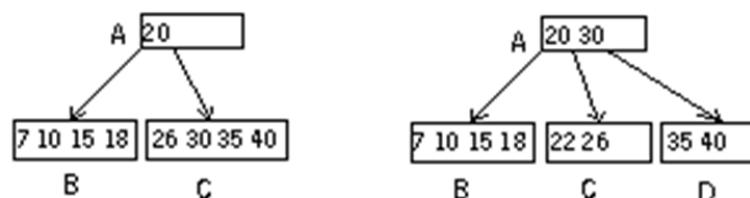
La otra situación que se presenta y la más problemática, es cuando se inserta un elemento en una página ya llena. Esto puede afectar la estructura del árbol y ocasionar asignación de páginas nuevas.

Para comprender lo que sucede en este caso, analicemos la figura que ilustra la inserción de la llave 22 en un Árbol B de orden 2. La acción se realiza en los siguientes pasos:

Se descubre que falta la llave 22; la inserción en la página C es imposible porque C ya está llena.

La página C se divide en dos páginas (esto es, se asigna una nueva página D).

Las  $2n + 1$  llaves se distribuyen uniformemente en C y D, y la llave de la mitad se sube un nivel hacia la página madre A.



Inserción de la llave 22 en un Árbol B de orden 2

Este plan tan elegante preserva todas las Propiedades típicas de los Árboles B. En particular, las páginas divididas contienen exactamente  $n$  elementos. Desde luego, la inserción de un elemento en la página madre puede hacer que ésta se desborde, con lo cual ocasiona que la división se propague. En el caso extremo, puede propagarse hasta la raíz. Es decir, la única manera en que el Árbol Búsqueda aumentar su altura. Tiene pues una manera singular de crecer: crece de las hojas hacia la raíz.

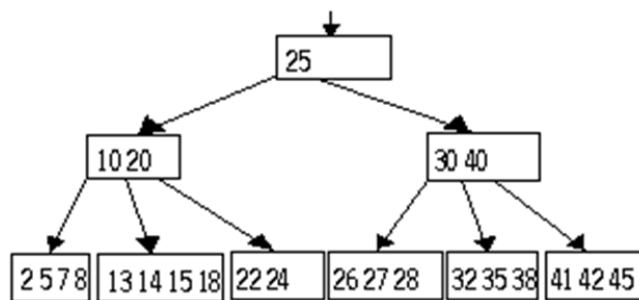
## Borrado

Se distinguen dos casos:

- a) El elemento a borrar se encuentra en una página hoja.
- b) El elemento a borrar no se encuentra en una página hoja, entonces debe ser sustituido por uno de los dos elementos adyacentes (predecesor o sucesor), que resultan estar en páginas hojas y que pueden ser borrados fácilmente.

En cualquier caso, después de la eliminación, debe seguir una comprobación del número de elementos restantes en la página, pues si  $m < n$  se tiene se estaría violando las Propiedades para el Árbol B de orden  $n$  y entonces, se requiere reorganizar el Árbol.

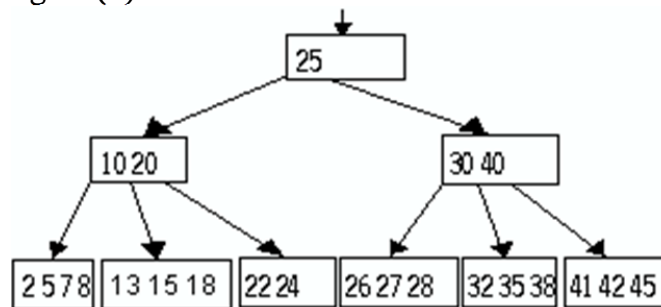
Para analizar el caso 1 (el más sencillo), considérese el Árbol B de orden 2 que se muestra en la figura (a).



(a) Un Árbol B de orden 2

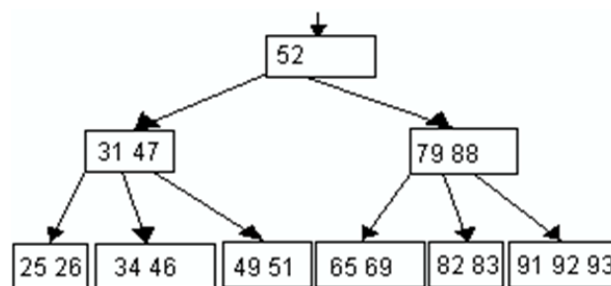
Ahora bien, supongamos que deseamos eliminar del Árbol el elemento con valor 14. El primer procedimiento es Buscar dicho elemento y determinar su posición en el Árbol (se encuentra en una página hoja). Una vez encontrado, procedemos a eliminarlo del Árbol y mover los elementos adyacentes a sus nuevas posiciones. Finalmente se debe realizar una verificación. Si la página en donde fue eliminado dicho elemento cumple con las Propiedades de Árbol B de orden 2, entonces se queda como está. En caso contrario, se debería realizar una combinación de páginas. En nuestro caso sólo debemos eliminarlo y volver a ordenar los elementos

de la página. El Árbol que se obtiene después del borrado del elemento con valor 14 se muestra en la figura (b).



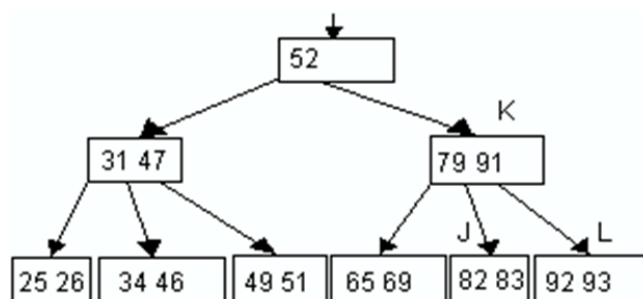
(b) El mismo Árbol B de la anterior tras el borrado del elemento 14

El caso 2 lo analizaremos a través el ejemplo de la figura (a), en el cual se muestra un Árbol B de orden 2. El elemento que deseamos borrar es el 88, y se encuentra ubicado en una página que no es hoja.



Árbol B de orden 2

Posteriormente se elimina el elemento y se reordenan los elementos restantes de la página A. Finalmente se procede a verificar el cumplimiento de las Propiedades del Árbol B de orden 2. Debido a que la página K queda con un elemento, debe hacerse una combinación entre las páginas J y L, y subir a K el elemento central, es decir el 91. Como se muestra en la figura (b).



(b) Árbol B de la Figura (a) tras el borrado del elemento 88

La reorganización se da tomando un elemento del padre, quien, a su vez debe tomar un elemento del hermano. Pero puede ocurrir que ya el hermano haya alcanzado su tamaño mínimo  $n$ , y no tenga, por tanto, posibilidad de prestar un elemento; entonces las dos páginas hermanas se unen en una sola, conteniendo las  $2n - 1$  elementos de las dos hermanas más un elemento central proveniente del padre. La combinación de páginas, causadas por cantidades de elementos en las páginas, menores a las permitidas, puede prolongarse a los niveles superiores, y en el caso extremo hasta la raíz, que cuando queda reducida a un tamaño nulo se borra, produciéndose reducción de la altura del árbol.

### **Costos**

Ya que visitar un nodo en el Árbol B conlleva a acceder a memoria secundaria, el número de páginas que se visiten en cada operación sobre el Árbol es lo que nos da la medida de su costo. Analicemos el caso de un archivo externo (ubicado en memoria secundaria) de  $N$  elementos.

#### **Costo de Buscar**

Como vimos en la sección funcionamiento de un Árbol B y su definición podemos organizar lógicamente los elementos del archivo mediante un Árbol B de orden  $n$ , en el cual cada página contiene entre  $n$  y  $2n$  elementos. De aquí que, la búsqueda de un elemento requiere a lo más  $\log_n N$  accesos al dispositivo secundario. De este modo, el costo de procesar una operación de búsqueda crece de forma logarítmica en relación con el tamaño del archivo.

El peor caso de búsqueda corresponde cuando se está buscando un elemento que está en la hoja del Árbol y además está situado al final de la página hoja.

#### **Costo de Inserción y Borrado**

La inserción o el borrado de un elemento del Árbol B podría requerir, además de la operación de búsqueda, un costo adicional que es el acceso a memoria secundaria. Es decir el costo de insertar o borrar un elemento es de:  $\log_n N + \text{Tiempo de acceso a memoria secundaria}$

Pero como el "Tiempo de acceso a memoria secundaria" es una constante (muchas veces bastante significativa), podemos obviarla si usamos el criterio asintótico. Y decir que el costo de inserción y borrado en un Árbol B de orden  $n$  que contiene las claves de un archivo externo de  $N$  elementos es a lo más de:  $\log_n N$

La tabla muestra cómo puede ser de razonable el costo logarítmico, incluso para archivos de gran tamaño. Por ejemplo, en un Árbol B de orden 50 que contenga las

claves que indexan a un fichero de un millón de registros, se puede realizar una operación de búsqueda, inserción o borrado con 4 accesos como mucho.

Tamaño del archivo (N)						
Tamaño de la página (n)	$10^3$	$10^4$	$10^5$	$10^6$	$10^7$	
10	3	4	5	6	7	
50	2	3	3	4	4	
100	2	2	3	3	4	
150	2	2	3	3	4	

Tipos de los Árboles B	
Nombre	Árbol B*
Uso	Administración de Bases de Datos (grandes)
Propiedades	Las mismas que un Árbol B básico pero además debe cumplirse que cada página está al menos 2/3 llenas.
Almacenamiento mínimo x pág.	66%, ya que las páginas están a lo menos 2/3 llenas.
Nombre	Árbol B+
Uso	Administración de Bases de Datos (grandes)
Propiedades	Las mismas que un Árbol B básico pero además debe cumplirse que los datos deben almacenarse en solamente en las hojas.
Almacenamiento mínimo x pág.	50%
Nombre	Árbol B+ Prefijo
Uso	Administración de Bases de Datos con secuencias alfabéticas.
Propiedades	Las mismas de un Árbol B+. Además, muchos datos pueden estar almacenados en un nodo, aumentando así los elementos y disminuyendo su altura.
Almacenamiento mínimo x pág.	Puede ser menor al 50% según los casos.
Nombre	Árbol B Virtual
Uso	Administración de memoria.
Propiedades	Las mismas de un Árbol B básico, pero ahora se administran los valores de los registros de los procesos en la CPU y su intercambio de RAM a ROM y viceversa
Almacenamiento mínimo x pág.	50%



---

Nombre	Árbol B Binario
Uso	Administración de Bases de Datos (pequeñas)
Propiedades	Es un Árbol B de orden 1
Almacenamiento mínimo x pág.	50%
Nombre	Árbol 2-3-4
Uso	Administración de Bases de Datos (no muy grandes)
Propiedades	Es un Arbol de orden 2
Almacenamiento mínimo x pág.	50%

---

Un árbol B+ es una variante especializada de árbol B que tiene dos tipos de nodos: internos, que sólo apuntan a otros nodos, y externos, que contienen los datos en si.

### **Ventajas de los árboles B+**

La ventaja de los árboles B+ frente a los árboles B es que los nodos internos del árbol B+ pueden tener muchos más valores clave de decisión que los nodos intermedios de un árbol B, de tal forma que el acceso global es más rápido y la altura media del árbol es menor. Hay que tener en cuenta el hecho de que en un árbol B+ hay que recorrer toda una rama hasta las hojas para encontrar un dato, mientras que en los árboles B+ se puede encontrar el dato en los nodos intermedios, incluso en el raíz.

### **Árboles 2-3**

Son los árboles B de primer orden ( $n = 1$ ).

Están formados por nodos (páginas) de uno o dos ítems (2 o 3 descendientes – de ahí su nombre de árboles 2-3) y, como árbol B, todas las hojas están a un mismo nivel.

Dado que con ellos se trabaja en memoria principal y que la representación con arreglos determina hasta un 50% de desperdicio, es usual representar la página como una lista enlazada, con 1 o 2 elementos. De esta manera, los nodos del árbol B lucen como nodos de un árbol binario normal y para distinguir entre referencias a descendientes (verticales) y referencias a "hermanos" (horizontales) se introduce un campo booleano  $h$  – hermano.

## **Inserción**

Se distinguen 4 casos, según crezca el subárbol derecho o el izquierdo: .....

El árbol crece de abajo hacia arriba y las hojas permanecen al mismo nivel.

Estos árboles presentan asimetría, el subárbol derecho suele ser más pesado.

Para corregirla se introdujeron los árboles binarios simétricos – árboles BBS (árboles seto), cuya representación requiere de una variable booleana adicional en cada nodo, para distinguir entre referencia a descendiente o a hermano. Estos árboles tienen las siguientes propiedades:

- todo nodo tiene una clave y un máximo de dos (referencias) subárboles,
- toda referencia es horizontal o vertical, no hay dos punteros horizontales consecutivos en ningún camino de búsqueda,
- todos los nodos terminales están en el mismo nivel

Esta definición garantiza que la longitud del camino, para cualquier nodo, sea a lo más de  $2 \cdot \log_2 N$  (N – número de claves).

#### 4. Desarrollo

```
class Arbol
{
    Nodo raiz;
    String[] cont;
    Nodo[] nod;
    Nodo papa;
    public Arbol(Nodo raiz)
    {
        this.raiz = raiz;
        nod=new Nodo[1000];
        cont=new String[1000];
    }
    public void insertar(Nodo puntero, String elemento)
    {
        if (puntero != null)
        {
            if (elemento.compareTo(puntero.contenido)<0)
            if(puntero.izquierdo != null)
                insertar(puntero.izquierdo, elemento);
            else
                puntero.izquierdo = new
                Nodo(elemento, puntero.profundidad + 1);
            else
                if(puntero.derecho != null)
                    insertar(puntero.derecho, elemento);
                else
                    puntero.derecho = new Nodo(elemento, puntero.profundidad + 1);
        }
    }

    public Nodo buscar(Nodo puntero, String elemento)
    {
        if (puntero != null)
        {
            if (elemento.compareTo(puntero.contenido)>0)
            if(puntero.izquierdo != null)
                return buscar(puntero.izquierdo, elemento);
            else
            {
                System.out.println("no existe elemento");
                return puntero;
            }
            else
            if(elemento.compareTo(puntero.contenido)==0)
            {
                return puntero;
            }
            else
            if(puntero.derecho != null) return
            buscar(puntero.derecho, elemento);
            else{ System.out.println("no existe elemento");
```

```

public void eliminar(String elemento)
{
    Nodo ubicacion=buscar(raiz,elemento);
    if(elemento.compareTo(ubicacion.contenido)!=0)
    {
        System.out.println("no se encontro elemento");
    }
    else
        podar(ubicacion);
}
public void podar(Nodo puntero)
{
    int i=0;
    papa=puntero;
    while(i>=0)
    {
        while(puntero.izquierdo!=null)
        {
            cont[i]=(puntero.izquierdo).contenido;
            nod[i]=puntero;
            puntero=puntero.izquierdo;
            i++;
        }
        while(((nod[i]).derecho==null) || (i<0))
        {
            i--;
        }
        if(i<0) break;
        puntero=nod[i].derecho;
        cont[i]=(puntero.contenido);
    }
    papa=null;
    String contenido;
    for(int j=0;j<100;j++){ contenido=cont[j];
    insertar(raiz,contenido);
    }
}
public void imprimir(Nodo puntero)
{
    if(puntero != null)
    {
        imprimir(puntero.izquierdo);
        System.out.println(puntero);
        imprimir(puntero.derecho);
    }
}

public static void main(String[] args)
{
    Arbol a = new Arbol(new Nodo("b",0)); //      a.imprimir(a.raiz);
    a.insertar(a.raiz,"c"); //      a.imprimir(a.raiz);
}

```

## **5. Cuestionario [Trabajo Complementario]**

1. Realice una tabla comparativa de los diferentes tipos de árboles analizados.
2. Muestre las ventajas y desventajas de cada uno de ellos.

## **6. Conclusiones**