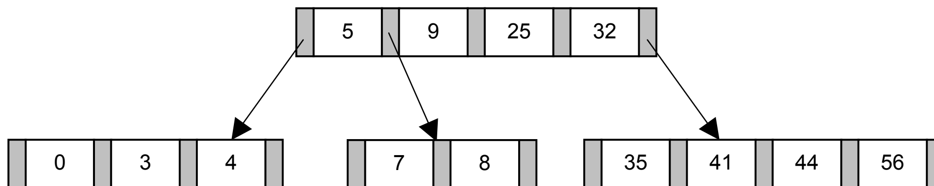
B - Trees

Definición – Usos – Operaciones

Definición

Un *árbol B* o *B-Tree*, es un tipo especializado de árbol, donde cada nodo puede contener múltiples llaves y ramas, como se puede ver en la figura a continuación.



Un árbol donde cada nodo con '*m*' cantidad de hijos – también llamado de orden *m* - tendrá a lo más m - 1 cantidad de llaves.

Un árbol B, posee las siguientes reglas de formación:

- Dentro de un nodo, las llaves deben estar ordenadas de menor a mayor.
- Cada hoja no puede tener menos de ceil(N/2) – 1 (Ver referencia [1]).
- Todos los elementos del nodo hijo enlazado más a la izquierda debe ser menor que todas las llaves del nodo padre.
- Todos los elementos de un nodo hijo, debe ser mayores que la llave izquierda del puntero y al mismo tiempo menores que la llave derecha.
- Todos los elementos del nodo hijo enlazado más a la derecha deben ser mayores que todas las llaves del nodo padre.

Las reglas de inserción, eliminación y balanceo se explican más adelante.

Una de las mayores gracias de un B-Tree, es que no hay necesidad de balancearlos, ya que se "autobalancean" al insertar o eliminar un elemento.

[1] Ceil(x) se denomina a la función que retorna el menor entero mayor igual que x.

Usos

La estructura del árbol lo hace ideal para guardar una gran cantidad de datos en memoria secundaria. Dado que el acceso a disco, cinta, etc; es un proceso muy lento en comparación con la velocidad del procesador, se debe intentar minimizar al máximo esta cantidad. Así, si se reduce la altura del árbol, se reduce también la cantidad de accesos al medio.

En una búsqueda, inserción, etc; se requiere de un acceso a disco por cada nodo que visita. El B-tree por cada visita, lee *m* datos a la vez los cuales puede procesar rápidamente y seguir con el siguiente nodo si es necesario.

Ejemplo:
Calcule la cantidad de accesos necesarios para encontrar un dato entre 1 millón de registros almacenados en disco con la utilización de un árbol binario, un B-tree de orden 100 y sin ningún tipo de estructura.

Si no utilizamos ningún tipo de estructura, entonces deberemos buscar dato por dato, hasta encontrar el que nos sirva. En el peor de los casos deberemos comparar todos los datos, teniendo 1.000.000 de accesos.

Usando el árbol binario, se sabe que la cantidad de nodos a visitar en el peor de los casos para encontrar un dato, en un árbol balanceado, es Log_2N, siendo N la cantidad de datos. Luego para este caso $Log_2 1000000 \approx 19.93 \approx 20$.

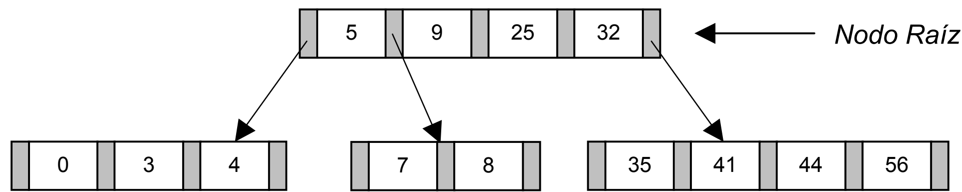
Al utilizar el B-Tree de orden 100, tendremos una situación como la siguiente.
- En el primer nivel solo estará el nodo raíz.
- En el segundo nodo tendremos a lo más 101 nodos con a lo más 100 datos cada uno. (10.100 datos)
- En el tercer nivel tendremos a lo más 101 * 101 nodos con a lo más 100 datos cada uno. (1.020.100 datos).

Así tendremos que solo se necesitarían 3 accesos a disco en el peor de los casos.

El cálculo también se puede realizar haciendo $Log_{101} 1000000 \approx 2.9935 \approx 3$.

Luego si consideramos que cada nodo, en cada posición interna, posee además de la llave de búsqueda algún otro dato asociado, como la posición en disco de un registro en especial, tenemos entonces un motor de base de datos.

Operaciones



Árbol Ejemplo

Búsqueda

La búsqueda dentro de un árbol B, es análoga a otros árboles. Sea *'a'* el dato a buscar y *'b'* el dato que comparamos dentro del nodo.
Comenzando desde el nodo raíz, de izquierda a derecha, comparamos *a* con *b*, hasta que:

a) Sean iguales, termina la búsqueda.
b) *b* sea mayor que *a*, lo que quiere decir que el dato debe estar en el nodo hijo antes que *b*. Repetimos el ciclo para el nodo hijo.
c) Se llega al último elemento de un nodo, es decir, el dato no existe en el árbol.

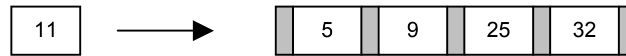
Inserción

El algoritmo de inserción es:

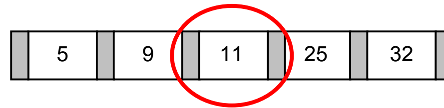
- Buscar el item a insertar.
  - Si se encuentra, no se reinserta.
  - Si no se encuentra, estaremos en la hoja donde debemos insertarlo.
- Ya que no esta el item en el árbol, lo insertamos.
  - Si hay lugar en la hoja, insértelo ahí, reordenando las llaves internas si fuese necesario.
  - Si no hay lugar, se debe proceder a partir la hoja en dos.
    - Se "inserta" el item donde correspondería como si hubiese lugar disponible.
    - Se toma el item medio (floor(N/2) + 1, ver nota [2]) del nodo y se sube al nodo padre, dejando el nodo izquierdo ya enlazado de antes, y el derecho a enlazar a la derecha del item subido. Al subir el item medio al padre, se toma como una inserción corriente a este nodo, y se aplican todas las reglas descritas.

[2] Floor(x) entrega el numero entero mayor inmediatamente menor igual que x.
Análogo a realizar una aproximación por corte.

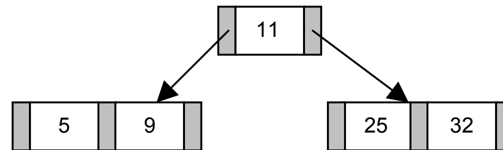
Paso 1)



Paso 2)



Paso 3)



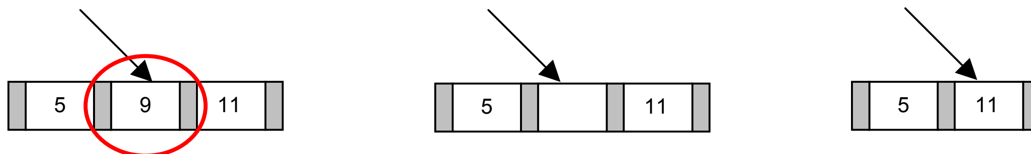
Eliminación

El proceso de remover un nodo del árbol puede complicarse según donde este se encuentre, de modo de mantener las reglas básicas de un árbol B.

Se parte por buscar el elemento a buscar. Si se encuentra, se busca cual de los siguientes casos concuerda con la situación del nodo.

Caso 1

Se desea eliminar un elemento que esta en una hoja del árbol. Si la cantidad de items luego de la eliminación es mayor igual a la cantidad mínima, simplemente se elimina. A continuación se reordenan los datos internamente en el nodo si fuese necesario.

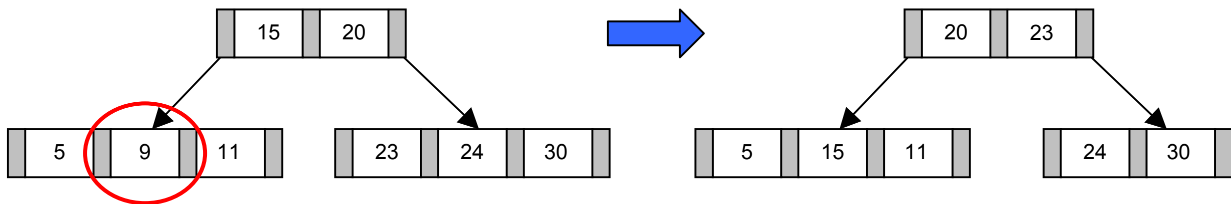


Caso 2

La eliminación también se realiza en una hoja, pero esta luego de la eliminación, queda con menos items que los mínimos permitidos.

En este caso se busca si la hoja izquierda o derecha tiene una llave (item) de sobra que podamos usar. Si es así entonces realizaremos una "rotación". Se toma la llave sobrante, se lleva al padre – reordenamos si es necesario. Luego se toma la llave en el padre más cercana al nodo donde ocurre la eliminación y se baja al ese hijo.

Veamos el caso donde la hoja derecha tiene un item que se puede usar.
Se elimina el 9. Para eso subimos el 23 y desde el padre bajamos el 15.

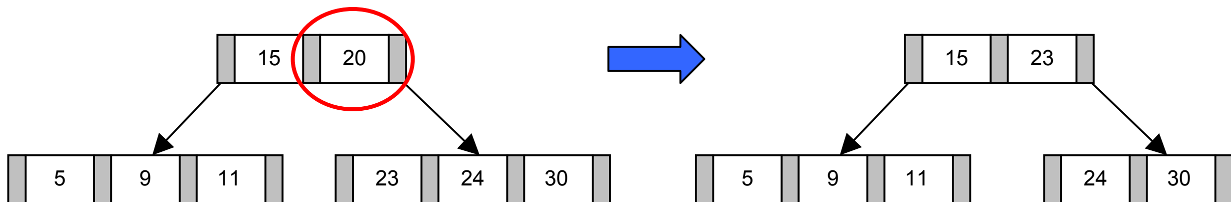


Caso 3

Se desea eliminar una llave que no esta en una rama.

Para esto, se busca el antecesor o sucesor inmediato en los hijos del nodo, de modo de reemplazar el puesto que quedará vacante. Este caso solo funciona si el hijo soporta la eliminación de una llave sin quedar con menos del mínimo requerido. Así se reemplaza la llave eliminada en el padre por la del hijo.

El caso cuando reemplazamos por una llave del nodo derecho, es el siguiente.



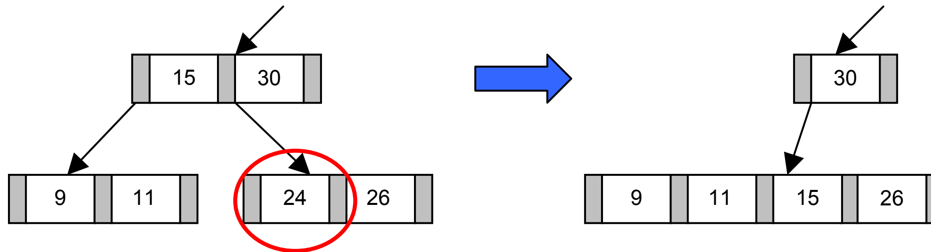
Caso 4

Este es el caso más complicado ya que genera varios cambios y mezclas de items.

Ocurre cuando se quiere eliminar una llave de un nodo hoja y este quedará con menos llaves de las permitidas, y adicionalmente, no podemos tomar ninguna de los nodos aledaños.

A este caso se le agrega cuando se borra una llave en un nodo rama (no hoja) y el hijo queda con menos nodos que los permitidos, siguiendo el procedimiento descrito en el *Caso 3*. Simplemente se lleva el item necesario hacia el padre y luego se trata al hijo como se describe a continuación.

Se debe tomar la hoja y combinarla con alguno de los nodos laterales. Para esto debemos bajar la llave que topa con ambos nodos.



El caso que el nodo combinado resultante tenga más elementos que los permitidos, se pasa el central al padre, generando dos nodos hijos.

En este caso el padre ha quedado con menos del mínimo permitido, asi que se debe combinar con el nivel superior, de la misma forma descrita anteriormente. La escepción ocurre si es que este es el nodo raíz, donde evidentemente no se tiene nodo alguno con el cual combinar.

Ideas de Implementación en JAVA

Una clase para cada nodo podría ser:

```
class Nodo {

        Integer cantidad = null;
        Integer[] llaves = null;
        Nodo[] punteros = null;

        nodo(int size) {
                this.cantidad = new Integer(size);
                this.llaves = new Integer[size];
                this.punteros = new Nodo[size + 1];
        }
}
```


Jonathan.Makuc@al.udp.cl