Apuntes

Tema ***Arboles Binarios de Búsqueda Autobalanceados***

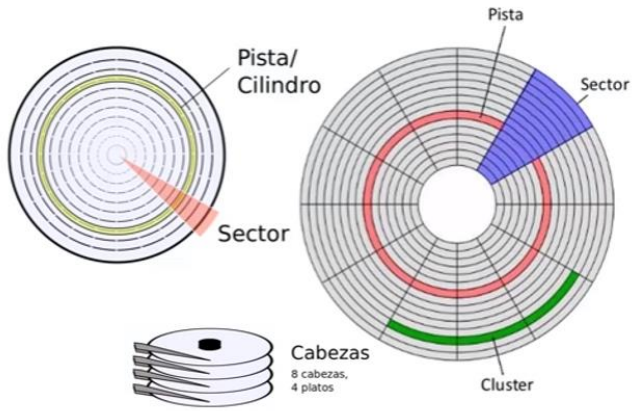
"Sería posible describir todo científicamente, pero no tendría ningún sentido; carecería de significado el que usted describiera a la sinfonía de Beethoven como una variación de la presión de la onda auditiva"
Albert Einstein

Arboles auto-balanceados B

Arboles B

Los creadores del árbol B, **Rudolf Bayer** y **Ed McCreight**, no han explicado el significado de la letra B de su nombre. Se cree que la B es de balanceado, dado que todos los nodos hoja se mantienen al mismo nivel en el árbol. La B también puede referirse a Bayer, o a Boeing, porque sus creadores trabajaban en los Boeing Scientific Research Labs por ese entonces.

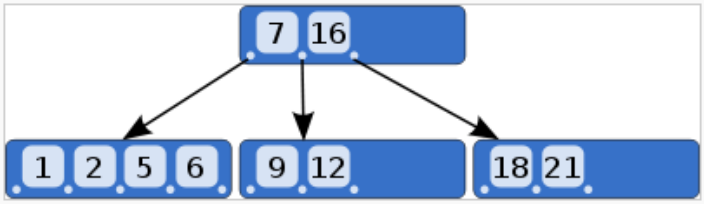
Rudolf Bayer Edward M. McCreight



Optimizada para aquellos arboles que no caben en memoria y deben cargarse parcialmente desde una memoria secundaria.

Los **árboles-B** son estructuras de arboles en la que los nodos internos deben tener un número variable de nodos hijo dentro de un rango predefinido. Cuando se inserta o se elimina un dato de la estructura, la cantidad de nodos hijo varía dentro de un nodo. Para que siga manteniéndose el número de nodos dentro del rango predefinido, los nodos internos se juntan o se parten. Dado que se permite un rango variable de nodos hijo, los árboles-B ***no necesitan rebalancearse tan frecuentemente como los árboles binarios de búsqueda auto-balanceables***. Pero, por otro lado, pueden desperdiciar memoria, porque los nodos no permanecen totalmente ocupados. Los límites *(uno superior y otro inferior)* en el número de nodos hijo son definidos para cada implementación en particular. Por ejemplo, en un **árbol-B 2-3** *(A menudo simplemente llamado **árbol 2-3**)*, el nodo sólo puede tener 2 ó 3 nodos hijo.

Un árbol-B se mantiene balanceado porque requiere que todos los nodos hoja se encuentren a la misma altura.
En el mejor de los casos, la altura de un árbol-B es: $\log_M n$
En el peor de los casos, la altura de un árbol-B es: n
Donde M es el número máximo de hijos que puede tener un nodo.

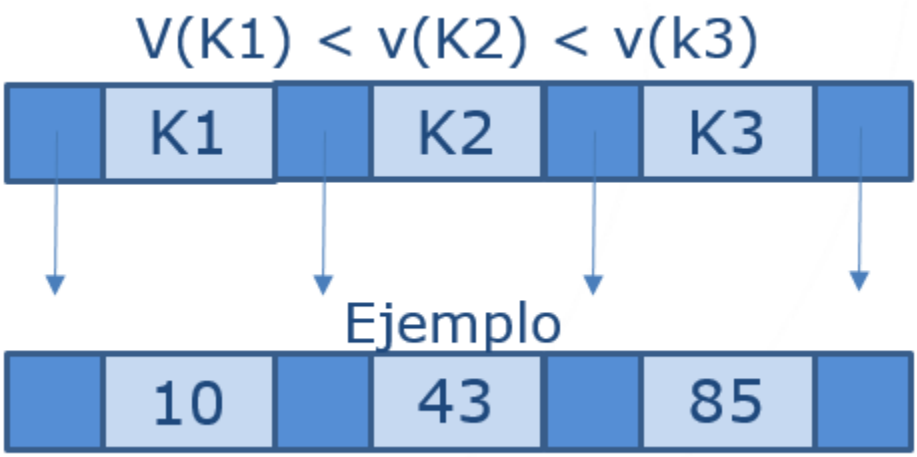


Los árboles B tienen ventajas sustanciales sobre otras implementaciones cuando el tiempo de acceso a los nodos excede al tiempo de acceso entre nodos. Este caso se da usualmente cuando los nodos se encuentran en dispositivos de almacenamiento secundario como los discos rígidos. Al maximizar el número de nodos hijo de cada nodo interno, la altura del árbol decrece, las operaciones para balancearlo se reducen, y aumenta la eficiencia. Usualmente este valor se coloca de forma tal que cada nodo ocupe un bloque de disco, o un tamaño análogo en el dispositivo. Mientras que los árboles B 2-3 pueden ser útiles en la memoria principal, y además más fáciles de explicar, si el tamaño de los nodos se ajustan para caber en un bloque de disco, el resultado puede ser un **árbol B 129-513**.

Un **árbol-B** de orden **M** *(el máximo número de hijos que puede tener cada nodo)* es un árbol que satisface las siguientes propiedades:

- Cada nodo tiene como máximo M hijos.
- Cada nodo (excepto raíz) tiene como mínimo (M)/2 claves.
- La raíz tiene al menos 1 hijos si no es un nodo hoja. (según M)
- Todos los nodos hoja aparecen al mismo nivel.
- Un nodo no hoja con k hijos contiene k-1 elementos almacenados.
- Los hijos que cuelgan de la raíz (r1, ⋯, rm) tienen que cumplir ciertas condiciones:
    - El primero tiene valor menor que r1.
    - El segundo tiene valor mayor que r1 y menor que r2, etc.
    - El último hijo tiene valor mayor que rm.

En un árbol B hay un número mayor de claves (K) y de hijos (C).



Ventaja: **Aumenta el factor de ramificación.**

- En un Árbol binario, a partir de un nodo podemos pasar como mucho a 2 hijos.
- En un Árbol B, el número de hijos a los que podemos viajar a partir de un nodo es mucho mayor.

Propiedades:

- **Grado**, es el número máximo de hijos que puede tener el nodo de un Arbol B.
- Puesto que un nodo con **K** claves tiene **C+1** hijos:
  - Un árbol B de grado **M** puede tener **M** hijos.
  - Un árbol B de grado **M** puede tener **M-1** claves.
- Dado un árbol de grado M con M hijos y M-1 claves:
  - Los nodos deben tener al menos M/2 hijos (o M/2-1 claves). El redondeo se hace siempre para arriba (función ceil).
  - La excepción es la raíz, que puede tener menos hijos de lo obligado.

Operaciones habituales

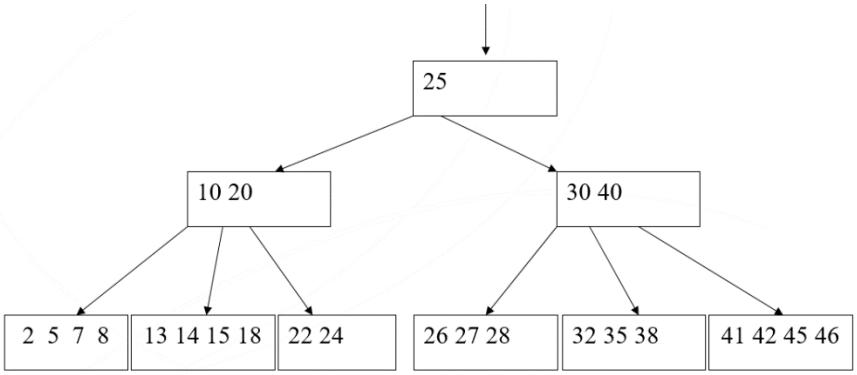
- **Búsqueda**
- **Inserción**
- **Borrado**
- **Recorrido**

Operaciones de transformación

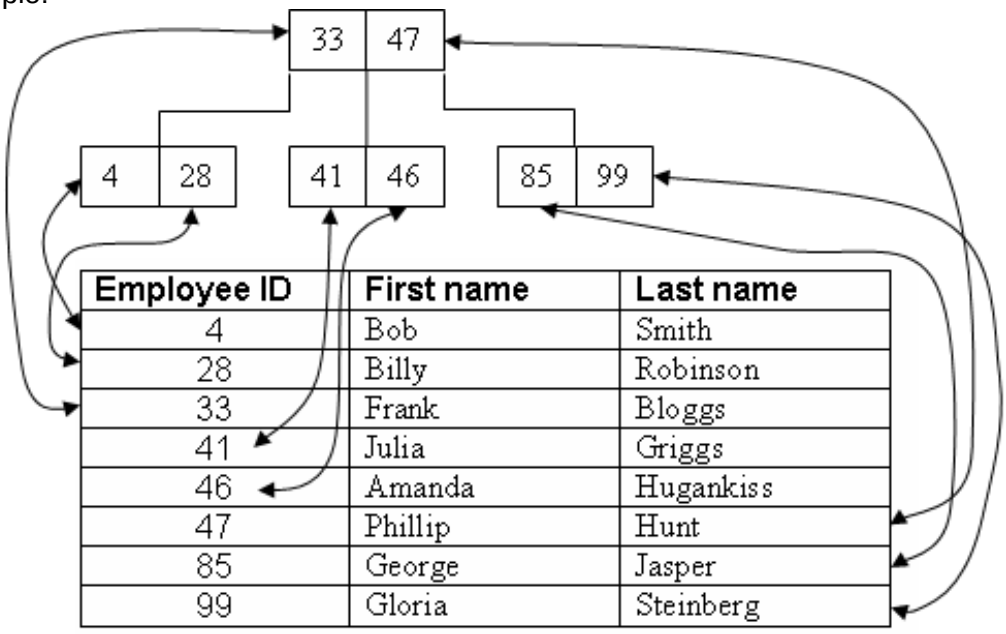
- **Dividir al insertar**
  - Ocurre cuando intentamos agregar elementos a un nodo que ya esta lleno. Hace falta reestructurar el árbol o subárbol para que se puedan insertar elementos en el
- **Reestructurar al borrar**
  - Al borrar un nodo también tenemos que tener en cuenta las propiedades de un árbol. Es decir que debemos asegurarnos que ningún nodo se encuentra debajo del límite.
- **Borrado**

- **Recorrido**

Árbol de orden 2 con 3 niveles.



Otro ejemplo:



Employee ID	First name	Last name
4	Bob	Smith
28	Billy	Robinson
33	Frank	Bloggs
41	Julia	Griggs
46	Amanda	Hugankiss
47	Phillip	Hunt
85	George	Jasper
99	Gloria	Steinberg

Operaciones:
Inserción:

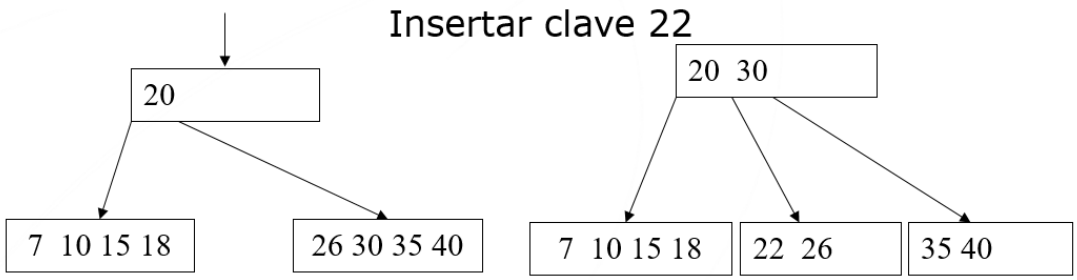
Añadir el dato a su hoja. Reorganizar la hoja.
Si se llena la hoja:
Dos hojas con el limite de elementos. Actualizar el padre
Si se llena el padre:
Partir en dos; actualizar nodo superior
Puede exigir una propagación hasta la raíz.

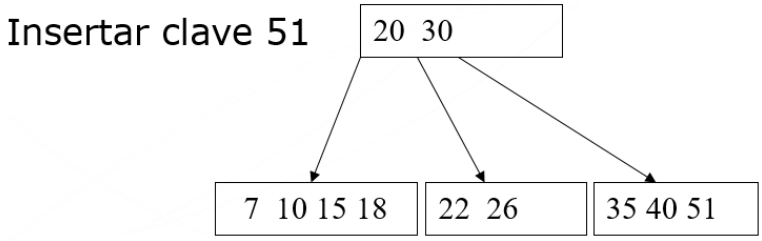
Borrado

Fusión de hojas si no alcanza el mínimo de elementos.
El padre pierde hijos. Eliminación de nodos...

La inserción en un árbol B 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.
- En una página llena, se debe realizar la asignación de páginas nuevas.
- En casos extremos, la propagación se lleva a la raíz, por lo tanto es cuando el árbol B puede crecer.





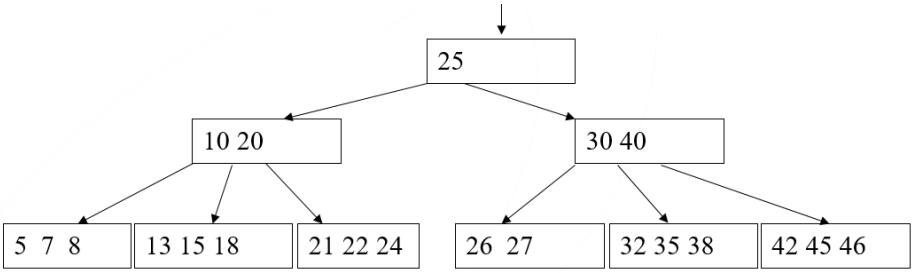
Características:

- Crecen de las hojas hacia la raíz.
- Son recursivos.
- La búsqueda de elementos se realiza como en árboles binario, solo hay que modificar la búsqueda sobre un arreglo.
- Son balanceados.
- Cada página tiene entre N y 2N elementos.

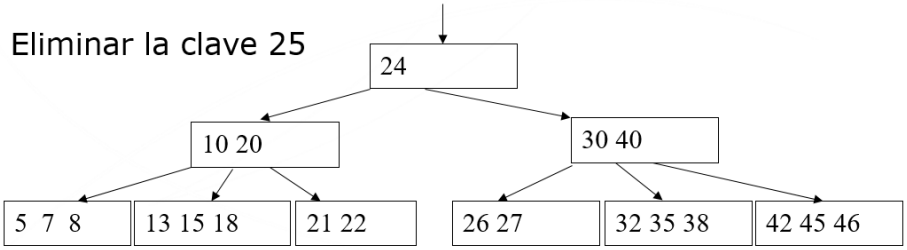
Eliminación

La eliminación de elementos en un árbol B es en teoría sencilla, pero se complica en sus detalles. Se pueden distinguir dos circunstancias:

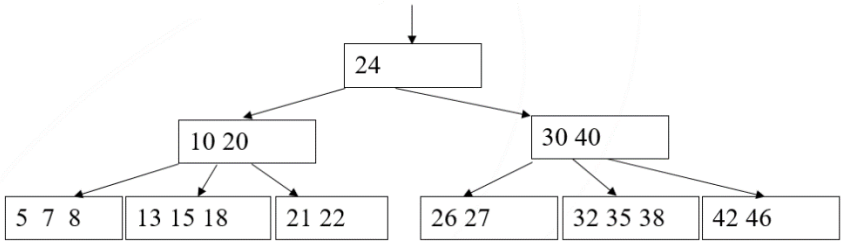
- El elemento que debe suprimirse se halla en una página de hoja, entonces el algoritmos de eliminación es sencillo.
- El elemento no se encuentra en una página de hoja; hay que sustituirlo por uno de los dos elementos lexicográficamente contiguos, que resultan estar en las páginas de hojas.



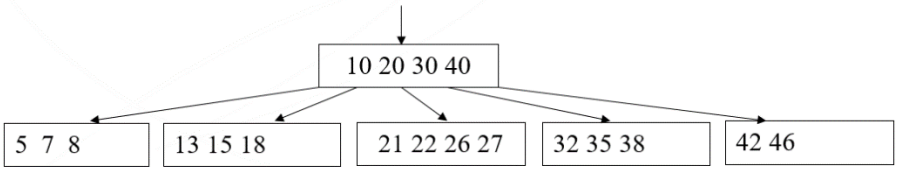
Eliminar la clave 25



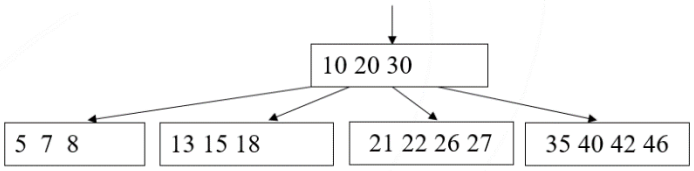
Eliminar la clave 45



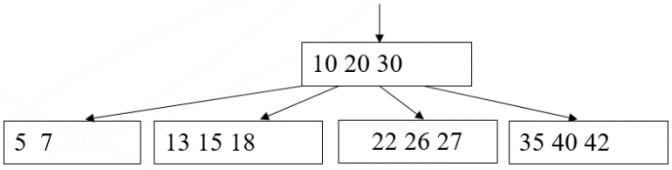
Eliminar la clave 24



Eliminar las claves 32 y 38

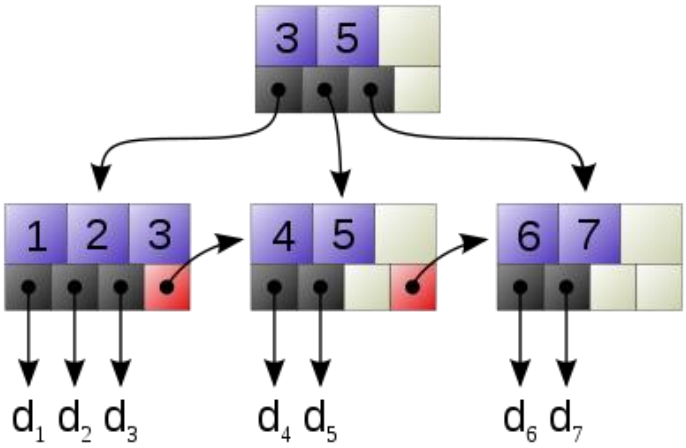


Eliminar las claves 21, 8 y 46



Arboles B +

Un árbol B+ es un tipo de estructura de datos de árbol, representa una colección de datos ordenados de manera que se permite una inserción y borrado eficientes de elementos. Es un índice, multinivel, dinámico, con un límite máximo y mínimo en el número de claves por nodo. Un **árbol B+** es una variación de un **árbol B**.
En un **árbol B+, toda la información se guarda en las hojas**. Los nodos internos sólo contienen claves y punteros. Todas las hojas se encuentran en el mismo nivel, que corresponde al más bajo. Los nodos hoja se encuentran unidos entre sí como una lista enlazada para permitir principalmente recuperación en rango mediante búsqueda secuencial.



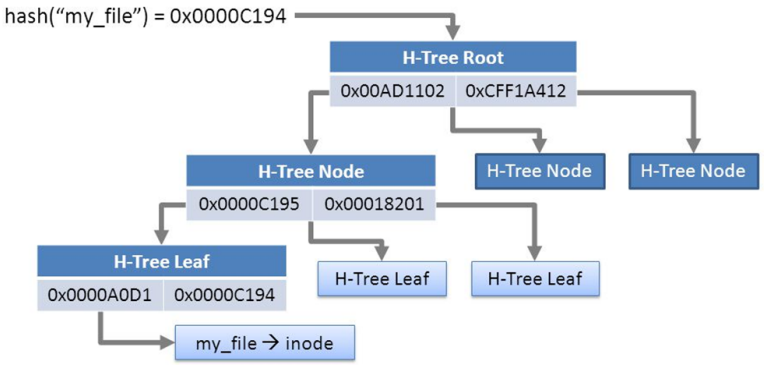
*Arboles B **

Un **árbol-B*** es una estructura de datos de árbol, una variante de Árbol-B utilizado en los sistemas de archivos HFS y Reiser4, que requiere que ***los nodos no raíz estén por lo menos a 2/3 de ocupación*** en lugar de 1/2. Para mantener esto los nodos, en lugar de generar inmediatamente un nodo cuando se llenan, comparten sus claves con el nodo adyacente. Cuando ambos están llenos, entonces los dos nodos se transforman en tres. También requiere que la clave más a la izquierda no sea usada nunca.
No se debe confundir un árbol B* con un árbol B+, en el que los nodos hoja del árbol están conectados entre sí a través de una lista enlazada, aumentando el coste de inserción para mejorar la eficiencia en la búsqueda.

Filesystem	Directory indexing algo	Hash type	Cache
ext2	list	-	+
ext3/4	htree	half_md4 + seed (earlier Legacy, TEA)	+
ufs2/NFS	dirhash	FNV (FreeBSD) DJB (OpenBSD)	+
FAT	list (btree)	-	+
NTFS	btree	-	+

- ext4 and NTFS encode directories as B-Trees to improve lookup time to O(log N)
- A B-Tree is a type of balanced tree that is optimized for storage on disk
  - Items are stored in sorted order in blocks
  - Each block stores between *m* and 2*m* items
- Suppose items *i* and *j* are in the root of the tree
  - The root must have 3 children, since it has 2 items
  - The three child groups contain items $a < i$, $i < a < j$, and $a > j$

- ext4 uses a B-Tree variant known as a H-Tree
  - The *H* stands for *hash* (sometime called B+Tree)
- Suppose you try to open("my_file", "r")



Feature	EXT4	XFS	BTRFS
Architecture	Hashed B-tree	B+ tree	Extent based
Introduced	2006	1994	2009
Max volume size	1 Ebytes	8 Ebytes	16 Ebytes
Max file size	16 Tbytes	8 Ebytes	16 Ebytes
Max number of files	4 billion	2^{64}	2^{64}
Max file name size	255 bytes	255 bytes	255 bytes
Attributes	Yes	Yes	Yes
Transparent compression	No	No	Yes
Transparent encryption	Yes	No	Planned
Copy-on-Write (COW)	No	Planned	Yes
Snapshots	No	Planned	Yes