Árbol B - Teoría

En general, habíamos visto distintas formas de acceder a un conjunto de datos en una estructura tipo archivo: los índices y hashing.

Si bien en términos de tiempos de acceso las funciones de hash tendían a minimizar la cantidad de operaciones de comparación para obtener un registro físico a partir de una clave, este tipo de estructura no sirve cuando queremos realizar un recorrido secuencial de los datos.

La principal desventaja en archivos secuenciales indexados es que al crecer el archivo, baja la performance por lo cual es necesario reorganizar el archivo.

Se hace necesaria una estructura de acceso que me permita:
- acceder a un registro con una clave determinada en un tiempo acotado.
- recorrer secuencialmente un conjunto de datos entre dos valores de clave dados

sin necesidad de consultar todo el archivo físico.

Generalizando el concepto de árbol binario de búsqueda (que requería como valor óptimo $\boldsymbol{log_2 (m+1)}$ accesos para una búsqueda) podemos pasar a un árbol n-ario de manera de en cada paso de una búsqueda dividir en hasta 'n' veces el conjunto de claves por consultar, si es que el árbol esta lleno. De esta forma se asegura un número máximo de accesos de $\boldsymbol{log_n (m+1)}$.

Hasta el momento no consideramos las estructuras de árboles n-arios como óptimos para estructuras de búsquedas en memoria, debido a que de 'n' nodos se desperdiciaban una elevada cantidad de campos de las celdas.

Pero lo que ahora estamos considerando es el acceso a grandes cantidades de información, cientos de miles o millones de registros, por lo que las estructuras de acceso residirán primariamente en disco (no se regeneran cada vez que se van a utilizar los datos).

Entonces el tiempo adquiere más preponderancia que el espacio en el criterio para la selección de un método de acceso a grandes volúmenes de datos.

La estructura de ***árbol B*** , parte del concepto de los árboles n-arios de búsqueda, (de n-1 claves y n punteros en cada nodo).
La diferencia más notable es que en este árbol, los punteros a los datos se encuentran siempre en el último nivel (el de los maximales). Para ello sigue ciertas reglas para las altas/bajas que le permiten ser un árbol equilibrado, donde todos los nodos maximales se encuentran en el mismo nivel.
De esta forma se adquiere uniformidad en cuanto a la cantidad de accesos para obtener el registro correspondiente a una clave, ya que mantiene para todas ellas la misma profundidad.

Definición formal

La formulación inicial de la estructura fue propuesta por Bayer y McCreight en 1972, y fue adaptada por Knuth en 1973. Esta última es la definición más corriente, que damos a continuación:

Sea ***T*** un árbol-B de orden ***r*** y raíz ***t*** entonces satisface las siguientes propiedades:

- Cada nodo en el árbol tiene a lo sumo r hijos:
  $\forall$ y $\in$ T / | R(y)| <= r

- Cada nodo, a excepción de la raíz y de las hojas (los maximales) tiene al menos r/2 hijos.
  $\forall$ y $\in$ T / y <> t and y $\notin$ Max(T) => | R(y)| >= r/2

- Todas las hojas (nodos maximales) se encuentran en el mismo nivel.
  $\forall z_i, z_j \in$ T / $z_i, z_j \in$ Max(T) => $|\delta(x, z_i)| = |\delta(x, z_j)|$

- El nodo raíz tiene al menos 2 hijos, a no ser que la raíz sea también una hoja.
  Si t $\notin$ Max(T), | R(t) | >= 2

- Un nodo con r hijos tiene r-1 claves.

Dadas estas propiedades, se puede deducir una fórmula para la performance de la recuperación de datos mediante el uso de un árbol B.

Sea un árbol B de orden r y K claves, la cantidad de niveles del árbol o altura h esta acotada por la siguiente fórmula:

$$h <= 1+\log_{(r/2)+1} ((K+1)/2)$$

Esta formula viene dada por el hecho de que cada nodo no raíz tiene al menos r/2 punteros, por lo que el primer acceso (por la raíz) solo divide al conjunto en 2, y cada acceso posterior divide al conjunto en al menos r/2 partes. Es fácil ver aquí que, cuanto r es mayor, menor será la cantidad de accesos requerida.

Todos los sistemas de bases de datos relacionales han optado por esta estructura como la más conveniente para crear los índices asociados a las tablas.
En estas implementaciones, el usuario no define el grado r del árbol sino que solo define la clave sobre la que se indexan los datos.
El DBMS según el caso, ocupa para cada nodo el espacio que normalmente puede leer al realizar un I/O sobre disco, aprovechándolo para la cantidad máxima posible de claves por nodo.
Recordar que h es función de K y de r, por lo que solo se puede 'jugar' con la r.

Definición de Split

Un split puede producirse en el momento de dar de altas claves en un árbol B.

Al momento de querer dar de alta una clave en un Nodo maximal, si el mismo se encuentra lleno, o en el caso de estar en la carga inicial del árbol se llega al Porcentaje de llenado del nodo (Load Factor), se deberá partir el Nodo maximal en Dos Nodos maximales, colocando la mitad de las claves en cada uno de ellos y enviando la clave media al Nodo de Nivel Superior. (Ver Ejemplo 1)

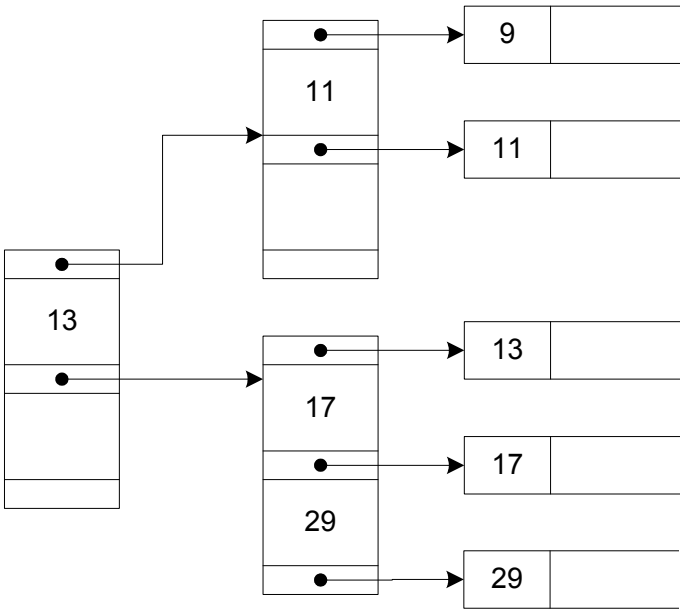
Puede suceder que dicho Nodo se encuentre lleno o que llegue a su % de llenado (sólo en la carga inicial), en dicho caso se deberá realizar un Split en dicho Nodo, repitiendo el mismo proceso.

El peor caso es que el efecto se propague por toda la rama de nodos hasta la raíz, y haya que particionar la raíz y crear una nueva raíz, aumentando la profundidad del Árbol B. (Ver Ejemplo 2)

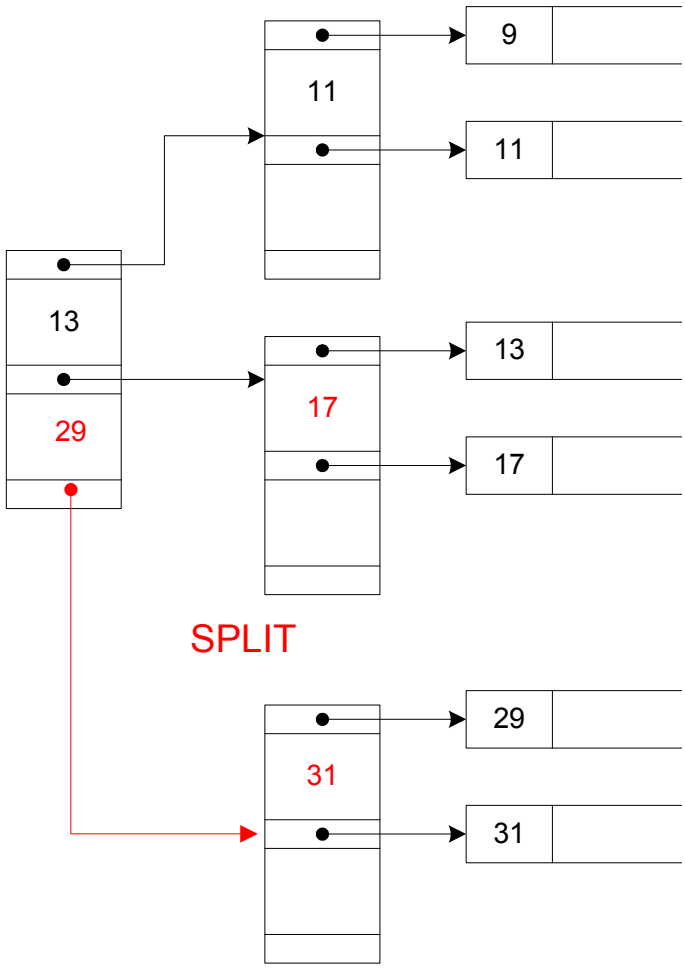
El conveniente minimizar la probabilidad de ocurrencia de splits en un Árbol B. Para ello se utiliza el Load Factor en la Carga Inicial del Árbol B.

Ejemplo1. Split de un Nodo Maximal

Dado el sig. Árbol B con r= 3 y LF=100%

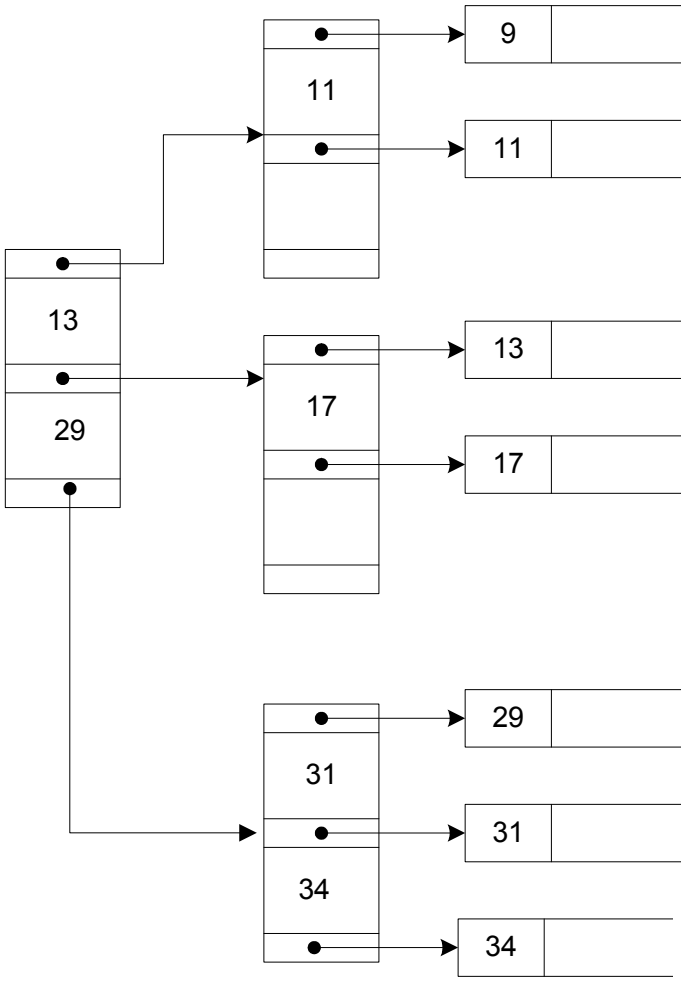


Al dar de Alta la clave 31 en el siguiente Árbol B se producirá un split del Nodo Maximal con claves 17 y 29.

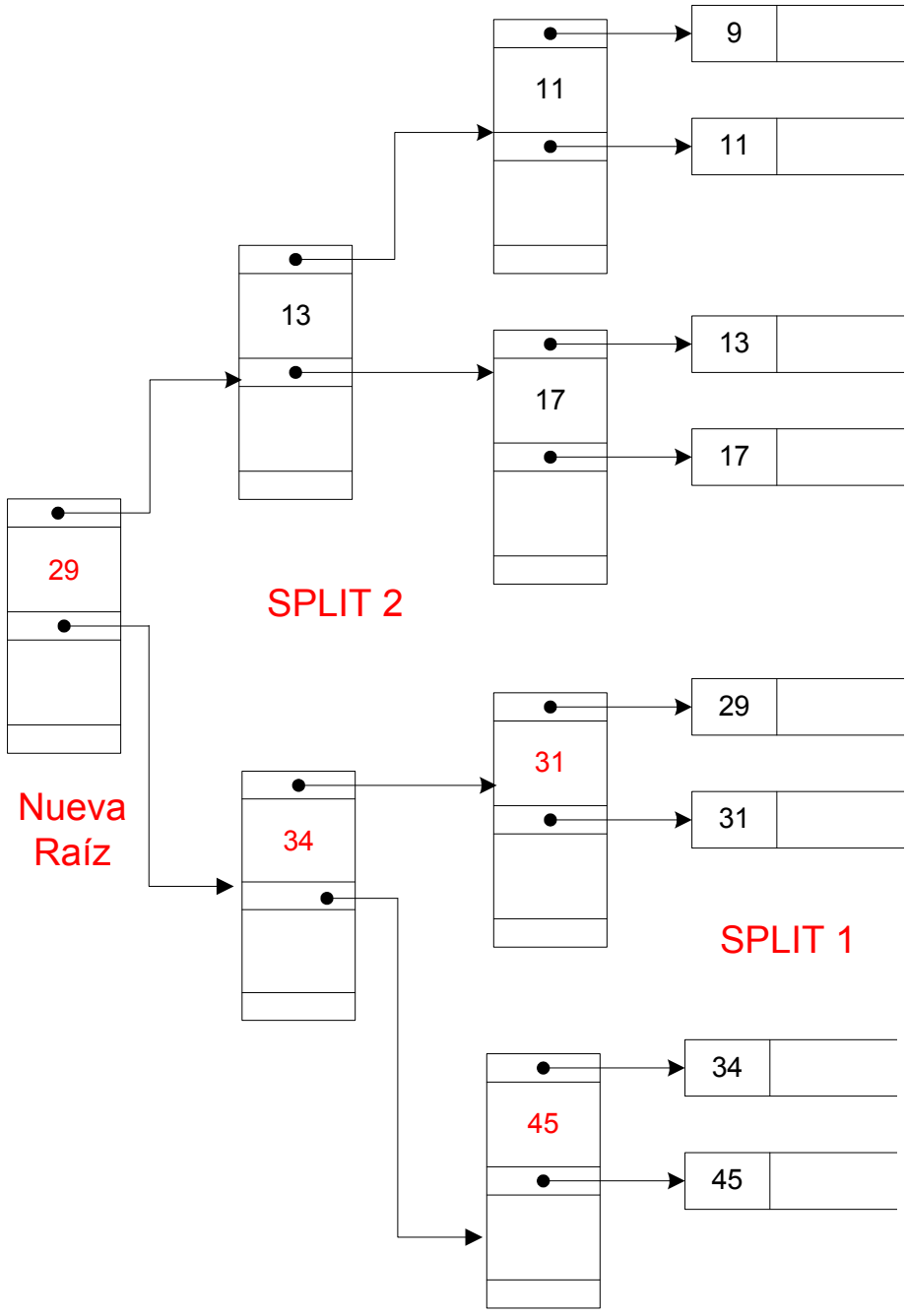


Ejemplo 2. Split de un Nodo Maximal que origina un split en el nodo raíz.

Dado el sig. Árbol B con r= 3 y LF=100%



Al dar de Alta la clave 45 en el Árbol B anterior se producirán dos splits, uno del Nodo Maximal con claves 31 y 34 y otro de la raíz del Árbol.



Definición de Factor de Carga (Load factor / Fill factor)

El Factor de carga es un porcentaje de carga de los nodos del Árbol B, el cuál se aplica sólo en el proceso de carga inicial del Árbol B.

El Load Factor puede variar dependiendo de las claves a insertar o del tipo de archivo o tabla asociado a dicho Árbol B.

Load Factor tendiendo al 100 %

1. Cuando el árbol B se representa un índice asociado a una tabla que sea de Consulta, con pocas actualizaciones, o que sea histórica. En dicho caso conviene que el índice sea lo más compacto posible.
2. Cuando las claves del índice son una secuencia incremental o decremental, eso significa que el árbol B siempre va crecer en una dirección, por lo tanto no tiene sentido dejar huecos en nodos medios.

Load Factor entre el 75 al 85 %

1. Cuando el árbol B se representa un índice asociado a una tabla que sea de mucha actualización (inserts, updates o deletes) en forma online.