

Árboles B, B* y B+

Índice

Índice	2
¿Qué contiene este apunte?:	3
Introducción:	3
Árboles B	4
Características y propiedades de un árbol B.	4
Operaciones en Árboles B	6
Ejemplo de Árbol B:	7
Árboles B*	12
Características y propiedades de un árbol B*.	12
Operaciones en Árboles B*	13
Ejemplo de Árbol B*:	14
Árbol B+	20
Características y propiedades de un árbol B+.	20
Operaciones en Árboles B*	21
Árboles B virtuales	22

¿Qué contiene este apunte?:

Este apunte pretende ser una ayuda para el alumno para comprender la utilidad y funcionamiento de los árboles B, B* y B+. Esta pensado para ser un complemento a la clase de árboles dada por la cátedra y clarificar posibles dudas o inquietudes.

En este apunte suponemos que el lector tiene conocimientos básicos de programación, de estructuras de datos y árboles (binarios y n-arios).

Introducción:

La necesidad de administrar grandes conjuntos de información organizadas en registros o unidades lógicas y de acceder a ellas en forma rápida es un problema importante en la historia y presente de la informática.

Diversas soluciones se propusieron, con resultados igualmente diversos. La generación de índices es quizás la solución mas utilizada. Pero al momento de construir el índice se deben tener en mente ciertas consideraciones:

- La memoria RAM generalmente no es suficiente para contener todo el índice
- Los accesos a disco son muy lentos y representan la mayor parte del tiempo necesario para hacer una búsqueda por ser necesarios muchos seeks.
- Es muy costoso mantener actualizado el índice en disco
- Pueden identificarse en general dos tipos de acceso. El acceso por referencia a un registro en particular o el acceso para recorrer secuencialmente un conjunto de varios registros. Para cada tipo de acceso, la estructura óptima para el índice será diferente.

Para disminuir la cantidad de accesos a disco se utilizan estructuras de árbol, de las cuales se mantiene en memoria la parte que se está utilizando y el resto se conserva en disco.

La idea es reducir al mínimo posible la cantidad de accesos en promedio que deben hacerse, no para una lectura en particular sino para todo el tiempo de uso de la estructura. Así, si se diseña una estructura de índice que será accedida principalmente para accesos por referencia, será preferible una estructura que reduzca la cantidad de seeks necesarios para cada búsqueda puntual. En cambio, si el tipo de búsqueda predominante será para posicionarse en un registro dado y recorrer secuencialmente a partir del mismo, será mejor una estructura que aunque requiera una mayor cantidad de seeks para un acceso por referencia (el primero), requiera pocos accesos para leer secuencialmente el resto de los datos.

La utilización de árboles Binarios, binarios balanceados, AVL paginados son una opción.

Se presenta a continuación ejemplos de cantidad promedio de accesos a disco y ventajas y desventajas de su utilización.

Búsqueda binaria:

$$\log_2 [M/N] - 1$$

M: cantidad de registros lógicos

N: Factor de bloqueo

Mantener los datos en orden para hacer las búsquedas es muy costoso.

Árbol binario balanceado:

$$\text{Piso}(\log_2 N) + 1$$

En realidad nada garantiza que un árbol binario esté balanceado, podría degenerar a una lista enlazada si los datos se entran en orden. El uso de un árbol binario evita el costo de mantener los datos en orden.

Árbol AVL paginado:

$$\log_{k+1}(N+1)$$

k: cantidad de claves por página

N: cantidad total de claves

La reorganización de los nodos para mantener el balanceo es relativamente poco costosa pero por ser árboles binarios, su profundidad tiende a ser muy grande.

Árboles B

El árbol B fue desarrollado para mantener estructuras de datos cuyo contenido se va modificando con el tiempo (Alta y bajas) de forma de poder encontrar en forma rápida y eficiente un elemento en particular. Para ello se busca que la profundidad del árbol sea la menor posible. Se requería también que la modificación del contenido no sea muy costosa en tiempo y espacio. Están pensados para disminuir la cantidad de accesos a disco, y la posibilidad de mantener en memoria la parte que se está utilizando y el resto conservarlo en el disco.

La cantidad de seeks necesaria para un acceso por referencia a una clave en particular es baja:

$$1 + \log_{\text{Techo}(m/2)} [(N+1)/2]$$

Características y propiedades de un árbol B.

Los árboles B son árboles que tienden a ser anchos y poco profundos. Los nodos del árbol B contienen un conjunto de registros o claves de registros que se mantienen ordenados (a efectos de facilitar la búsqueda dentro del mismo). y referencias a nodos descendientes. Cada elemento del nodo tiene dos referencias a descendientes, una a nodos cuyas claves son menores a la clave del elemento y otra a mayores.



I) 1 elemento de un nodo de un árbol B. II) nodo de con 3 claves y 4 referencias a descendientes

Existe cierta discrepancia entre diferentes autores en el momento de clasificar a los árboles B. Generalmente se habla de árboles de orden "n". Donde, dependiendo el autor, n puede ser el número máximo o mínimo de claves o de descendientes por nodo. De aquí en adelante no hablaremos de orden de un árbol B, sino que lo caracterizaremos por medio de algún rasgo del mismo y mediante este se podrá inferir el resto de sus características.

Todo árbol B debe cumplir con los siguientes axiomas:

- Todas las ramas tienen igual profundidad
- Si un nodo tiene k claves, tiene siempre k + 1 descendientes.
- Cada clave en un nodo está asociada siempre con dos referencias. Una a otro nodo que contiene elementos con claves menores y la otra a un nodo con elementos con claves mayores.
- Existe una relación entre las características de cantidad de claves y descendientes mínimos y máximos por nodo (establecidas a continuación).

Se puede caracterizar a un árbol B por las siguientes propiedades.

- Cantidad máxima de claves por nodo (k).
- Cantidad máxima de descendientes por nodo (m)
- Cantidad mínima de claves por nodo (excepto la raíz que puede tener menos claves).
- Cantidad mínima de descendientes por nodo (excepto la raíz y las hojas).

Si conocemos, por ejemplo el número máximo de claves (k) por nodo se pueden calcular:

à Cantidad máxima de descendientes por nodo: $m = k + 1$

à Cantidad mínima de claves de: $\lfloor k/2 \rfloor^1$

à Cantidad mínima de descendientes por nodo: $\lceil m/2 \rceil^2$

De la misma forma se podría realizar la determinación de propiedades conociendo el número máximo de descendientes (m) por nodo.

Se puede ver también simple mediante operatoria matemática, si tenemos en cuenta que k y m son números naturales, que:

$$\lceil m/2 \rceil = \lceil (k+1)/2 \rceil = \lceil k/2 + 0.5 \rceil = \lfloor k/2 \rfloor + 1$$

Por lo tanto entre la cantidad mínima de claves y descendientes existe una diferencia de 1 (misma diferencia entre máximos de claves y descendientes)

Otra característica a tener en cuenta a la hora de construir un árbol B es determinar si todos los nodos mantienen los elementos almacenados o únicamente los nodos hoja (quedando para los nodos interiores simplemente la clave de búsqueda). El segundo método, si bien agrega redundancia a la información almacenada (ocupando mas espacio en soporte de información), trae beneficios que a la hora de explicar el funcionamiento de los árboles B+ se verán claramente. Los ejemplos y caracterizaciones de Árbol B en este apunte están pensados para el primer caso.

¹ La raíz puede tener como mínimo 0 claves (si el árbol esta vacío)

² Las hojas no tienen descendientes (por definición de árbol) y la raíz puede tener 0 descendientes (si el árbol esta vacío) o como mínimo 2 descendientes (cuando esta se divide en dos)

Operaciones en Árboles B

Se pueden identificar tres tipos de operaciones en un árbol B:

- Inserción de elementos
- Eliminación de elementos
- Búsqueda de elementos.

Búsqueda:

Se comienza la búsqueda en el nodo raíz. Si se encuentra el elemento buscado se devuelve. Si no, se toma el elemento del nodo con clave inmediatamente superior a la buscada y se obtiene el nodo referenciado por el puntero izquierdo de esa clave. Si no existe en el nodo un elemento mayor al de la clave, se busca el elemento con mayor clave (el ultimo) y se obtiene el nodo al que referencia por la derecha. Se prosigue buscando y descendiendo de la misma forma, a través de los niveles del árbol hasta encontrarlo o llegar al nivel hoja donde debería estar. Si en este último nivel no esta, la búsqueda retornará que no encontró el elemento buscado.

Inserción:

Los elementos siempre se agregan en un nodo hoja. Primero se utiliza el algoritmo de búsqueda para determinar la posición en donde se ingresara el elemento. A la hora de agregar el elemento en el nodo determinado, pueden ocurrir 2 cosas:

- a) El nodo tiene lugar (Alta sin overflow): En ese caso se agrega al nodo normalmente, manteniendo dentro del mismo el orden entre elementos.
- b) El nodo no tiene lugar (Alta con overflow): en ese caso se realiza un split. El split consiste en tomar el conjunto de elementos del nodo y el a agregar y generar dos nodos. Cada uno con la mitad del todo y promoviendo la clave de valor medio a un nivel superior(que de no existir se genera uno nuevo con la clave promovida). Puede ocurrir que al promoverse una clave al nivel superior esta produzca un overflow en el mismo. En este caso se procederá a realizar un split en este nivel superior. Si se debe hacer un split de la raíz, entonces el árbol crece en profundidad.

Eliminación:

Se busca la clave a eliminar (mediante la operación de búsqueda).

Puede ocurrir que la clave a borrar se encuentre en un nodo hoja o en uno no hoja. Si la clave esta en un nodo interno (no hoja), se intercambia el elemento con el de la clave inmediatamente superior o inferior (que por particularidad del árbol B siempre estará en un nodo hoja). Y luego se elimina la clave como si esta estuviese en una hoja. La decisión de intercambiarlo con el inmediatamente superior o con el inmediatamente inferior debe mantenerse en todas las bajas.

Si la clave esta en un nodo hoja se borrar la clave normalmente. En ese momento puede ocurrir que el nodo afectado contenga menos claves que las mínimas permitidas. Este caso se conoce como underflow y se puede aplicar alguno de los siguientes métodos para solucionarlo:

Redistribución: Se busca si en alguno de los siblings("siblings" se llama a los nodos vecinos, que dependen del mismo padre) tiene mas claves que el mínimo. En caso afirmativo se realiza una rotación de claves que involucra al nodo, al sibling y al padre, bajando la clave del padre, que se encuentra de separador entre el sibling y el nodo, al nodo con underflow y subiendo del sibling un elemento para cubrir el lugar del separador.

Concatenación: En el caso que la redistribución no puede llevarse a cabo, se procede a unir el nodo afectado con alguno de sus siblings y con el separador en el nodo padre, quedando un único nodo entre los afectados (operación inversa al alta con split).

Ejemplo de Árbol B:

Se desea armar un árbol cuya cantidad máxima de claves sea 5 ($k=5$).
Por lo tanto:

à Cantidad máxima de descendientes por nodo: $m = k + 1 = 6$

à Cantidad mínima de claves de: $\lfloor k/2 \rfloor = 2$

à Cantidad mínima de descendientes por nodo: $\lceil m/2 \rceil = 3$

Inicialmente el nodo raíz esta vacío:

.	
---	--	---	--	---	--	---	--	---	--	---

Raíz con 0 descendientes y 0 claves.

1) Alta de 33:

.	33
---	----	---	--	---	--	---	--	---	--	---

Raíz con 1 clave y 0 descendientes.

2) Alta de 20:

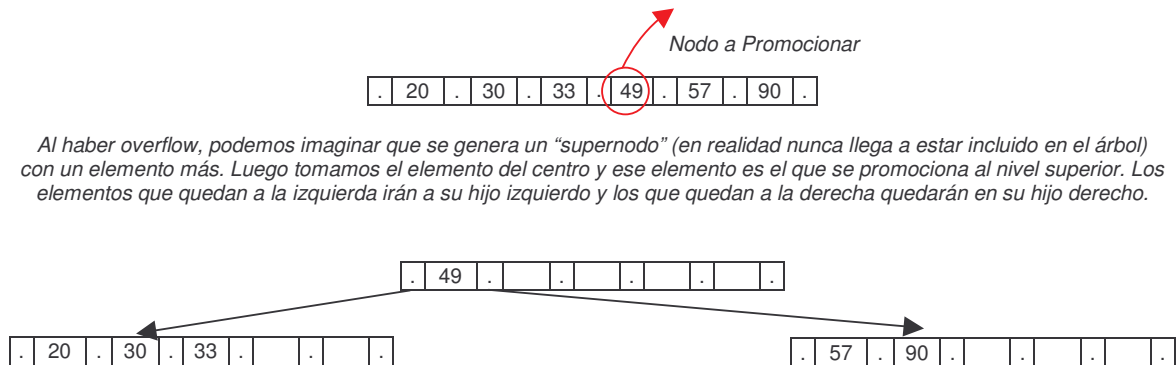
.	20	.	33
---	----	---	----	---	--	---	--	---	--	---

Raíz con 2 claves y 0 descendientes.

3) Alta de 57,90, 49:

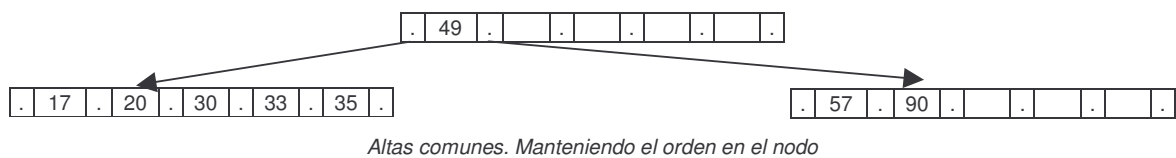
.	20	.	33	.	49	.	57	.	90	.
---	----	---	----	---	----	---	----	---	----	---

4) Alta de 30 (alta con overflow):

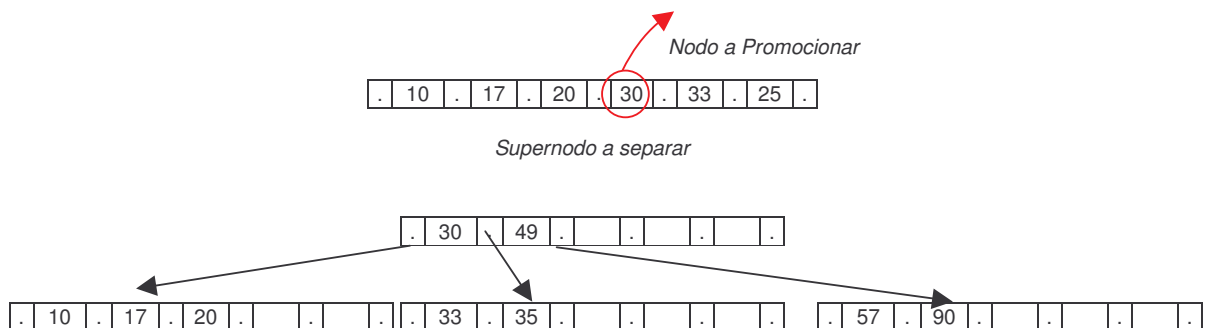


Split. Nueva raíz con 1 clave. Demás niveles con claves necesarias. La división de 3 elementos a la izquierda y dos a la derecha podría haber sido al revés (2 y 3). No obstante una vez tomado una de las dos opciones, se debe respetar siempre en el resto de los split (un algoritmo de resolución siempre mantiene la misma postura)

5) Alta de 17 y 35:



6) Alta 10(alta con overflow):



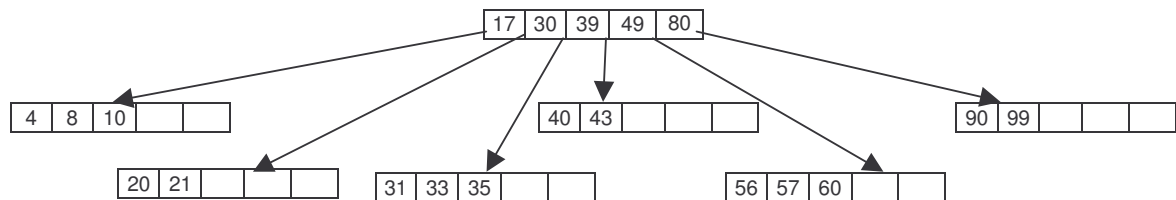
Alta con overflow. Promoción de una clave a la raíz

7) Alta de 21, 40, 80, 56, 8, 39, 43, 60:



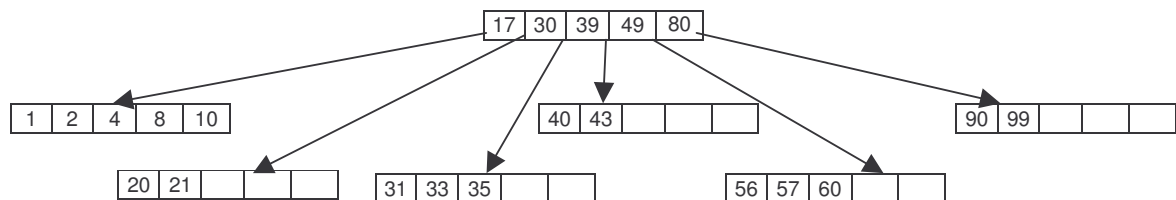
8) Alta de 4, 31, 99:

(a partir de este paso no se muestran los espacios reservados para las referencias por una cuestión de espacio en el dibujo)



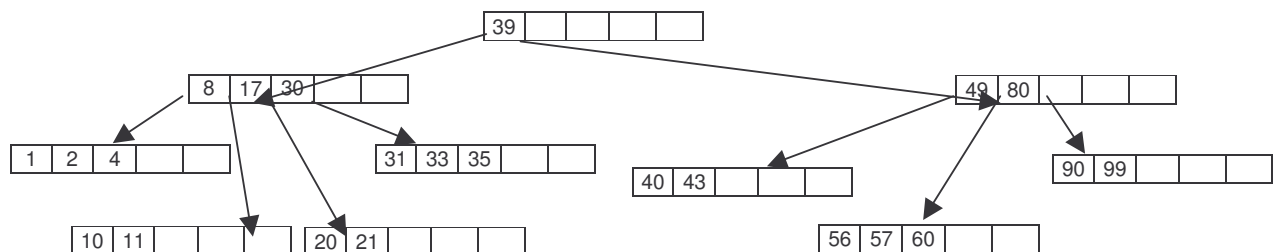
Cada alta generó un split en cada uno de los nodos hoja del árbol anterior

9) Alta de 1, 2:



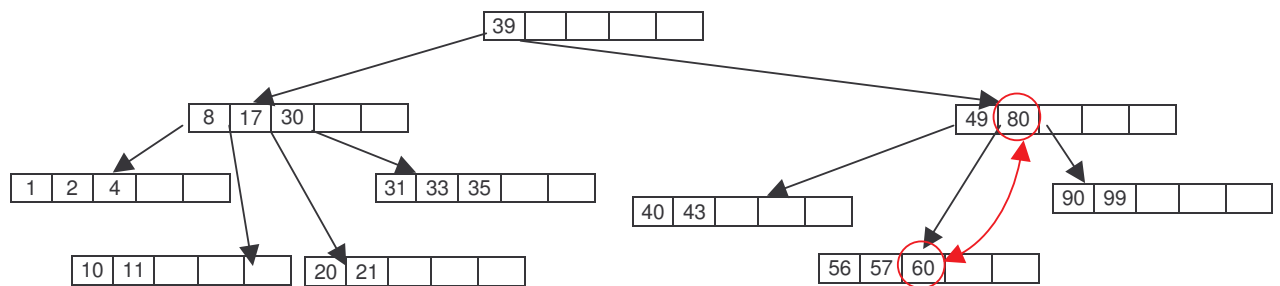
Cada alta generó un split en cada uno de los nodos hoja del árbol anterior

10) Alta 11:

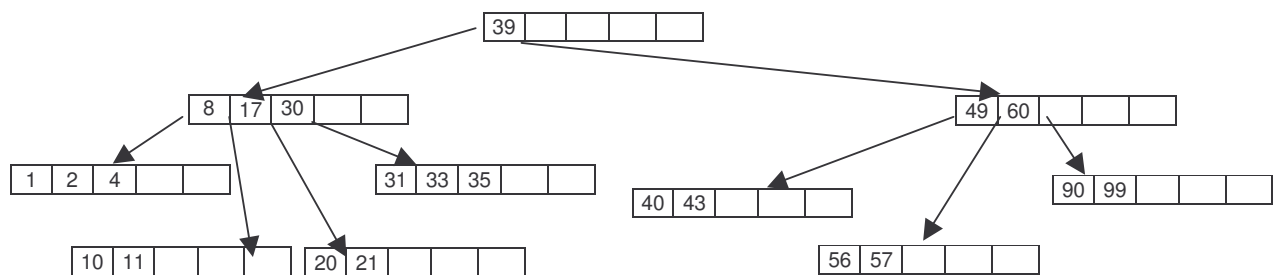


Split. Se produce overflow en hoja, se promociona un elemento a la raíz y en esta también se produce overflow. Como consecuencia se realiza un split y se genera una nueva raíz. El árbol creció en profundidad. Los nodos no hojas y no raíz tienen al menos los mínimos de descendientes por nodo

11) Baja de 80 (elemento que no esta en hoja)

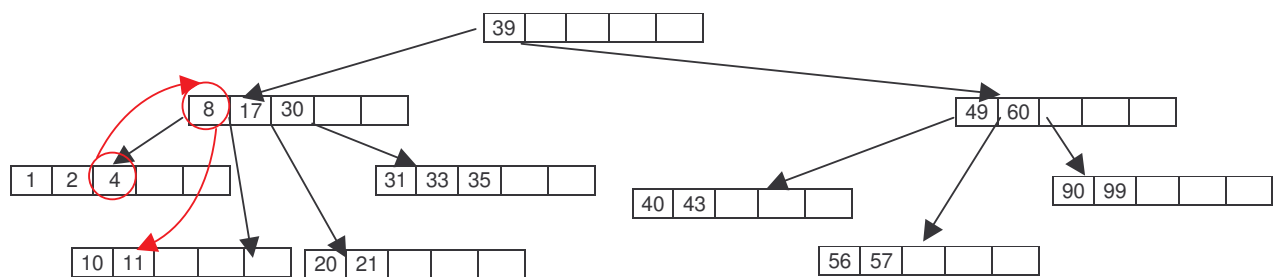


Como el elemento a dar de baja está en un nodo no hoja, debe intercambiarse con

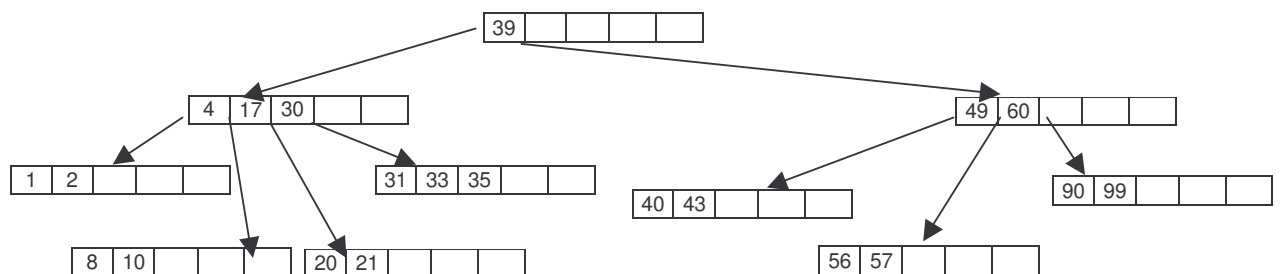


Luego eliminamos el elemento 80 "normalmente"

12) Baja de 11 (Baja con Redistribución)

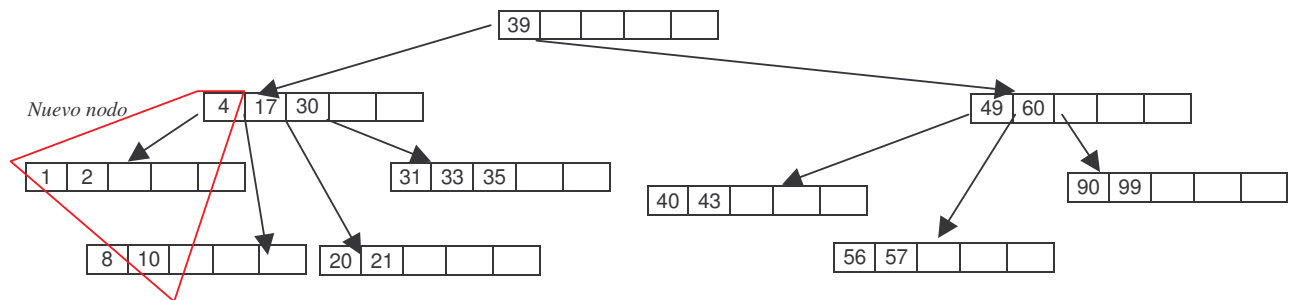


Debido a que el sibling a izquierda tiene más elementos que el mínimo, se redistribuyen las claves 4 y 8

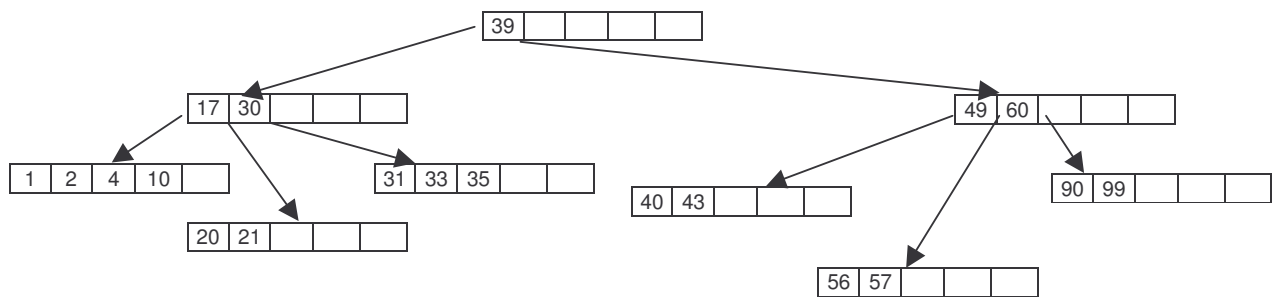


Así evitamos modificar la estructura.

13) Baja de 8 (Baja con concatenación):



Al eliminar el elemento con clave 8, el nodo que ocupaba quedará en underflow. Como los siblings no tienen elementos de más, se deben concatenar 2 nodos, bajando además el separador del nodo padre.

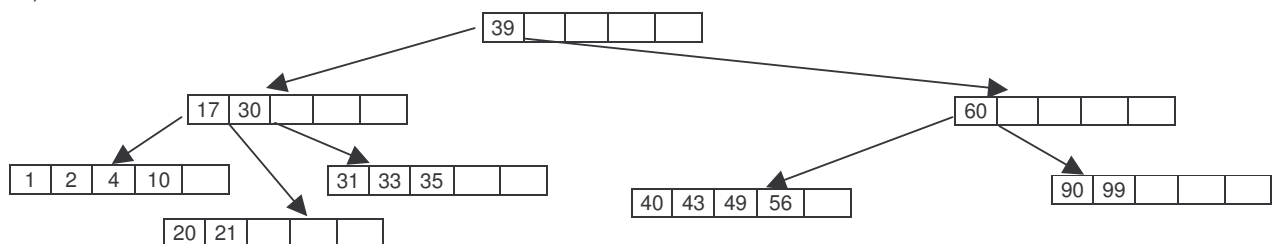


El árbol luego de la concatenación.

14) Baja de 57 (Baja con reducción de la profundidad del árbol):

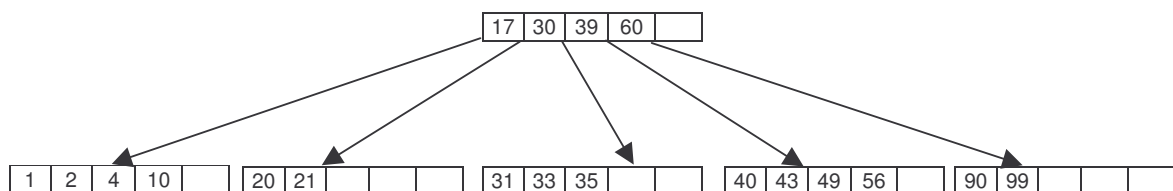
Estudiaremos el caso paso a paso:

a)



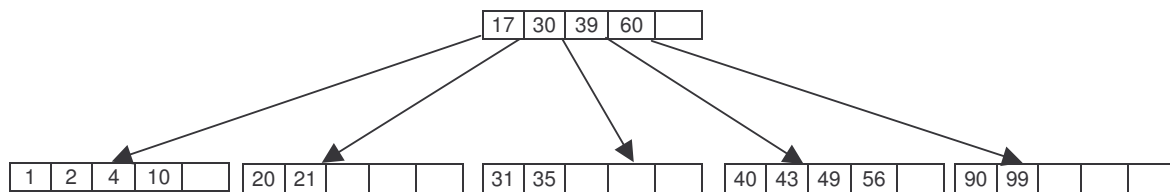
La eliminación de la clave 57 produce la concatenación de dos nodos en uno. Disminuyendo en 1 los elementos del nodo padre. Esto produce que en el nodo padre se produzca un underflow

b)



El underflow produce una nueva concatenación en el nivel superior, determinando la disminución de la altura del árbol.

15) Baja de 33 (baja normal):



Con este último ejemplo se han mostrado todos los casos posibles y el lector debería ser capaz de resolver el alta y baja de elementos en cualquier situación en un árbol B

Árboles B*

El árbol B* (B asterisco, B Estrella o B Star) es una variante del árbol B. La ventaja de su utilización reside en un menor crecimiento en la altura del árbol lograda gracias a la aplicación de redistribución en el alta. Otra ventaja del árbol B* es el aprovechamiento del espacio en el nodo (en soporte) al cambiar las políticas para decidir cuando hacer un split o una concatenación

Características y propiedades de un árbol B.*

Los árboles B* tienen un comportamiento similar a los árboles B. Pero a diferencia del anterior se aplica redistribución en el alta logrando detener hasta último momento la necesidad de los split con el consecuente mayor utilización del espacio en soporte. Además este comportamiento asegura que a la hora de realizar un split al menos dos nodos estén llenos, por lo que el split se realiza tomando dos siblings y dividiéndolos en 3 nodos 2/3 llenos.

Todo árbol B* tiene que cumplir con los siguientes axiomas:

- Todas las ramas tienen igual profundidad
- Si un nodo tiene k claves, tiene siempre $m + 1$ descendientes.
- Cada clave en un nodo está asociada siempre con dos referencias. Una hacia otro nodo que contiene elementos con claves menores y la otra a un nodo con elementos con claves mayores.
- Existe una relación entre las características de cantidad de claves y descendientes mínimos y máximos por nodo (establecidas a continuación).

Se puede caracterizar a un árbol B* por las siguientes propiedades:

- Cantidad máxima de claves por nodo (k).
- Cantidad máxima de descendientes por nodo (m)
- Cantidad mínima de claves por nodo (excepto la raíz que puede tener menos).
- Cantidad mínima de descendientes por nodo (excepto la raíz y las hojas).

Si conocemos, por ejemplo el número máximo de claves (k) por nodo se pueden calcular:

à Cantidad máxima de descendientes por nodo: $m = k + 1$

à Cantidad mínima de claves de: $\lfloor k \cdot 2/3 \rfloor^3$

à Cantidad mínima de descendientes por nodo: $\lceil (2m - 1)/3 \rceil^4$

³ La raíz puede tener como mínimo 0 claves (si el árbol está vacío)

⁴ Las hojas no tienen descendientes (por definición de árbol) y la raíz puede tener 0 descendientes

Se presenta a continuación una tabla ilustrativa de las relaciones existentes entre las características:

k	m	k(min)	m(min)
2	3	1	2
3	4	2	3
4	5	2	3
5	6	3	4
6	7	4	5
7	8	4	5
8	9	5	6
9	10	6	7

La raíz: un caso especial.

En el momento de construir y mantener un árbol B* surge rápidamente una cuestión no menor: Si el split se realiza de 2 nodos a 3, ¿cómo se realiza el split en la raíz donde no existen siblings?. Esta pregunta, nada trivial, tiene varias respuestas. En este momento propondremos varias soluciones, todas validas:

- 1) Manejar la raíz como un nodo de un árbol B
- 2) Permitir a la raíz tener un tamaño mayor y al realizar un split dividirlo en 2 hojas 2/3 llenas
- 3) Permitir a la raíz tener un tamaño mayor y al realizar un split dividirlo en 3 hojas 2/3 llenas

En el primer caso todos los nodos tendrán el mismo tamaño (pudiendo utilizar la misma estructura para todos los nodos). Pero al realizar el split los nodos no cumplirán con los mínimos de claves necesarios.

En el segundo caso el tamaño del nodo raíz será diferente a los demás nodos. Los nodos creados a partir del split cumplirán con los mínimos de claves necesarios.

En el tercer caso el tamaño del nodo raíz será mucho mayor a los demás nodos. Los nodos creados a partir del split cumplirán con los mínimos de claves necesarios y aparecerán directamente 3 nodos dependientes de la raíz.

La decisión de que método utilizar, los presentados u otros, es una decisión de implementación.

Operaciones en Árboles B*

Se pueden identificar tres tipos de operaciones en un árbol B:

- Inserción de elementos
- Eliminación de elementos
- Búsqueda de elementos.

Búsqueda:

Su funcionamiento es el mismo que en el árbol B

Inserción:

Se realiza mediante la búsqueda la ubicación del nodo donde debe agregarse el elemento. Pueden ocurrir 2 situaciones:

El nodo tiene lugar: Se procede a agregar el elemento normalmente (de la misma manera que en un árbol B)

El nodo está completo: En este caso se trata en primer lugar de realizar una redistribución con alguno de sus siblings y el elemento separador del nivel superior. En caso de no ser posible se realiza el split tomando para eso un sibling completo, juntando los dos y realizando un split a 3 nodos con $2/3$ de sus claves completas y promoviéndose una clave al nivel superior.

Eliminación:

Se procede de la misma manera que en el árbol B, teniendo en cuenta que la baja es el proceso inverso de la inserción. Por lo tanto al ocurrir un underflow de un nodo y no ser posible la redistribución, se realizara una concatenación de 3 nodos a 2.

Ejemplo de Árbol B*:

Realizaremos un ejemplo de un árbol B* con cantidad máxima de claves por nodo de 5 ($k=5$)

Por lo tanto:

à Cantidad máxima de descendientes por nodo: $m = k + 1 = 6$

à Cantidad mínima de claves de: $\lfloor k \cdot 2/3 \rfloor = 3$

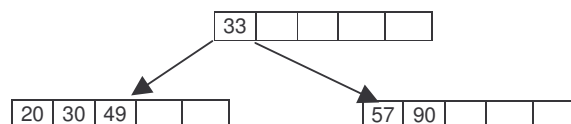
à Cantidad mínima de descendientes por nodo: $\lceil (2m - 1)/3 \rceil = 4$

1) En primer lugar tomaremos una raíz de tamaño igual al resto de los nodos.

Por ejemplo si la siguiente raíz:

20	33	49	57	90
----	----	----	----	----

Alta de 30:



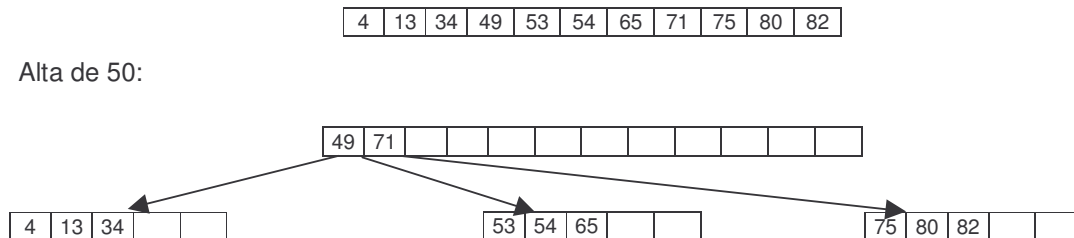
El overflow produce que se divida como en un árbol B. Aun así que el nodo de la derecha no cuenta con la cantidad mínima de claves que exige la definición de árbol B. Pero al ser un split de la raíz se acepta.*

2) Otra opción es permitir un mayor tamaño del nodo raíz para que al realizar el split queden 3 nodos con $2/3$ de sus claves completas.

Para eso se debe tomar como tamaño de la raíz de $2 \cdot m + 1$ claves (donde m es la cantidad máxima por nodo de claves). Al ingresar la clave siguiente y producirse un overflow se generan 3 nodos con $2/3$ de claves y se promueven a una nueva raíz 2 elementos.

En el ejemplo de un árbol B* con 5 claves por nodo como máximo, tendremos una raíz con hasta 11 claves.

Por ejemplo:



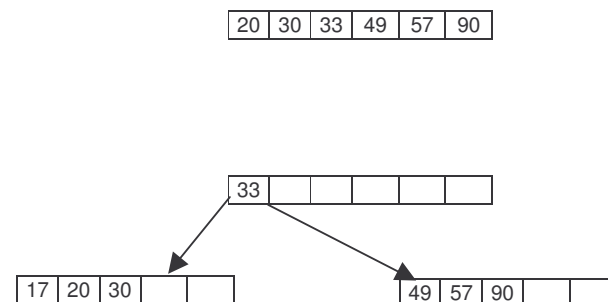
El overflow produce que se divida en 3 nodos con 2/3 de sus claves llenas. La raíz queda con 2 elementos.

2) Ahora estudiaremos el caso en el que se admite un tamaño de raíz mayor al resto de los demás nodos, permitiendo una división de la raíz en 2 nodos con 2/3 de las claves llenas.

Primero vemos el tamaño que debería tener la raíz:

La cantidad mínima de claves es 3. Se necesita que la división produzca 2 nodos con 2/3, por lo tanto la raíz debería permitir 6 claves. Quedando al agregarse un séptimo elemento dos nodos con 3 elementos y un elemento separador en la raíz.

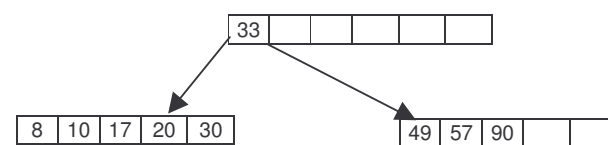
Gráficamente:



El overflow produce que se divida y se satisface la exige de claves mínimas.

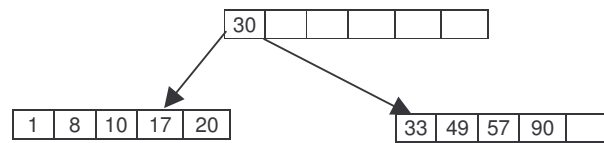
Seguiremos analizando los siguientes ejemplos a partir de este último modelo.

3) Alta de 8 y 10



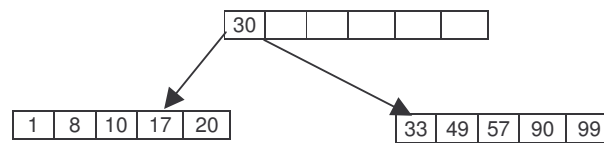
Altas comunes.

4) Alta de 1 (Alta con redistribución)

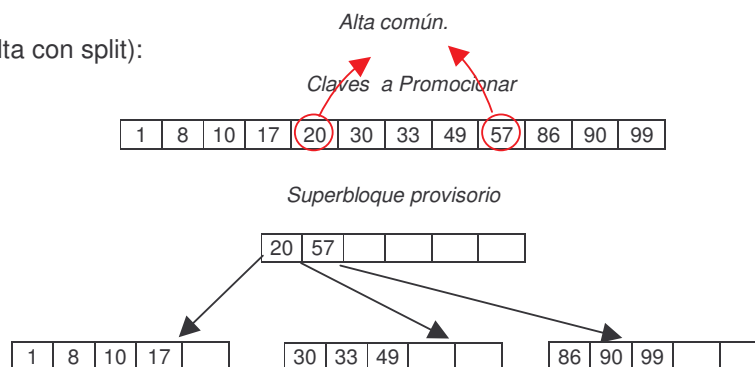


El nodo está lleno, pero su sibling no. Por lo tanto se realiza una redistribución de claves evitando un split.

5) Alta de 99:

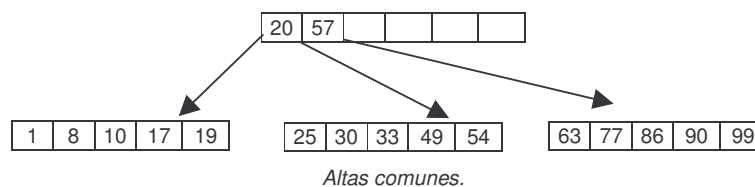


6) Alta de 86 (alta con split):

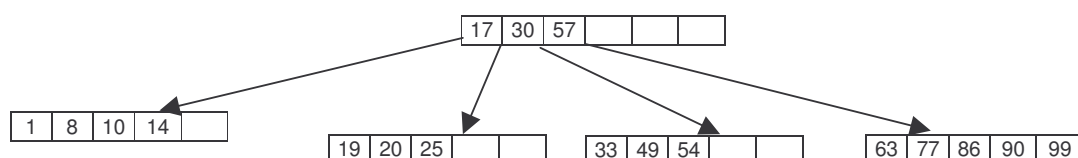


Se agrega un elemento nuevo al nodo que está completo. El split se realiza junto a su sibling y el elemento separador del nodo superior. Luego del split quedan 3 nodos con al menos 2/3 de sus claves llenas y quedan 2 claves al nodo raíz.

7) Alta de 25, 63, 77, 54, 19:

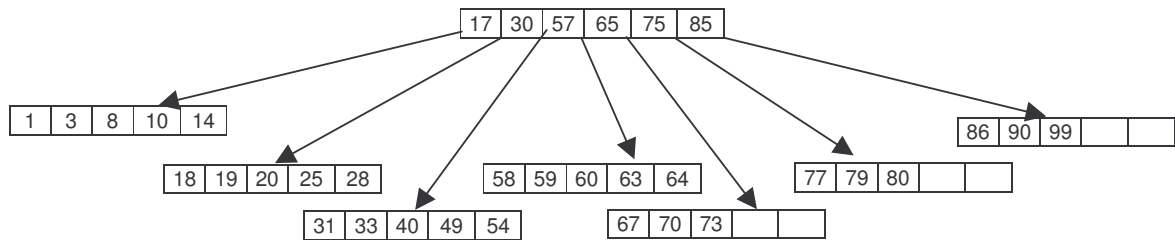


8) Alta de 14 (alta con split):

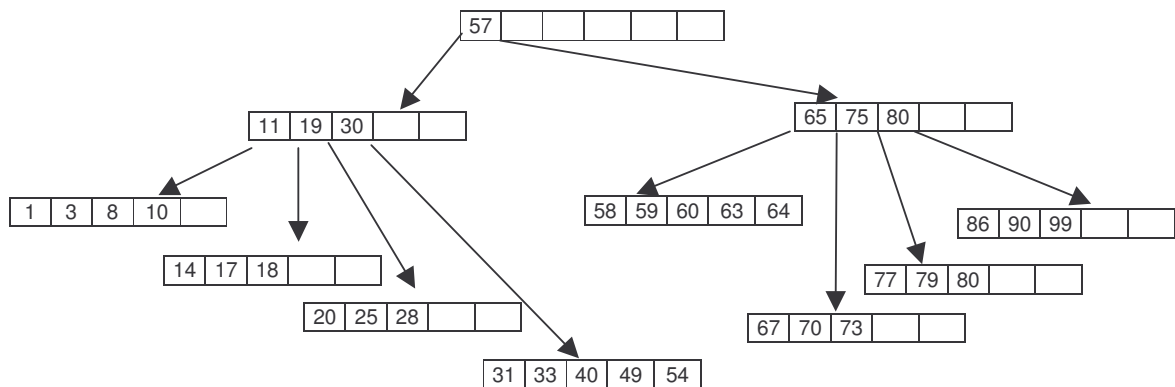


Se produce overflow en el nodo y su sibling también está completo. Por lo tanto, con ambos se realiza un split determinando la aparición de un nuevo nodo y promoviendo una nueva clave a la raíz. Tanto el nuevo nodo, el nodo que sufrió el overflow como su sibling quedan con al menos 2/3 de sus claves completas.

9) Supongamos que llegamos a la siguiente situación:

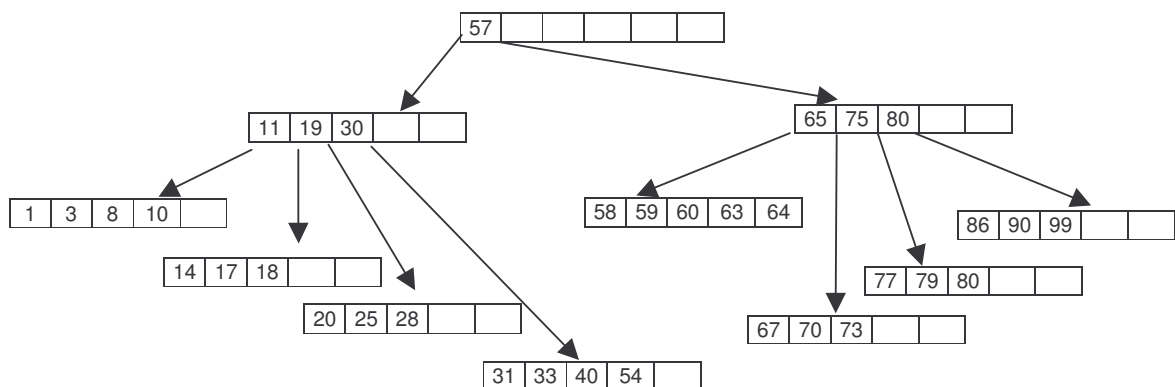


Y se realiza el alta del 11:



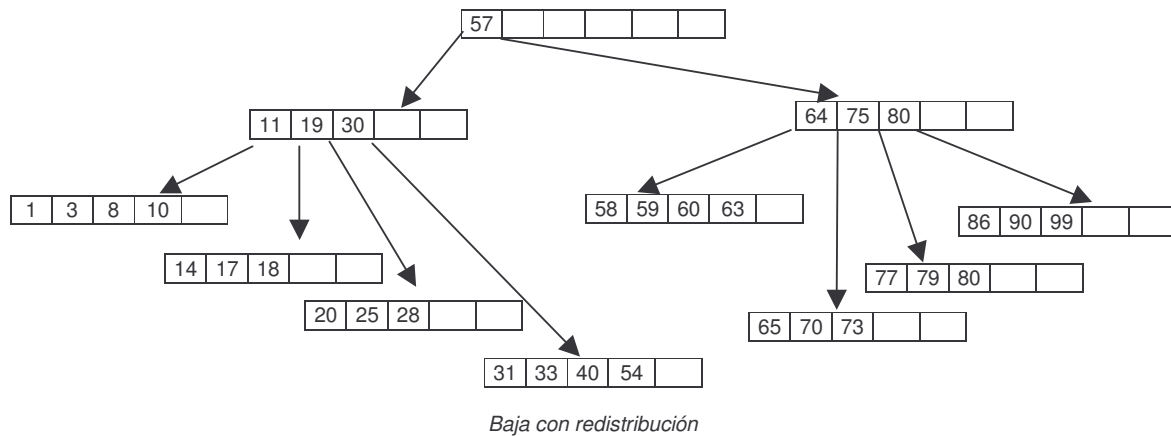
El nodo de destino del 11 está completo, así como sus siblings. Por lo tanto se realiza un split de 2 a 3 promoviendo una nueva clave a la raíz. A su vez la raíz, al estar llena y tratar de ingresar un nuevo elemento produce un split de la raíz, que al ser tratada de forma especial se divide en 2 con 2/3 de sus elementos llenos (teniendo en cuenta el método que se eligió para desarrollar este árbol) y se genera una nueva raíz, agregando un nivel al árbol

10) Baja de 49:



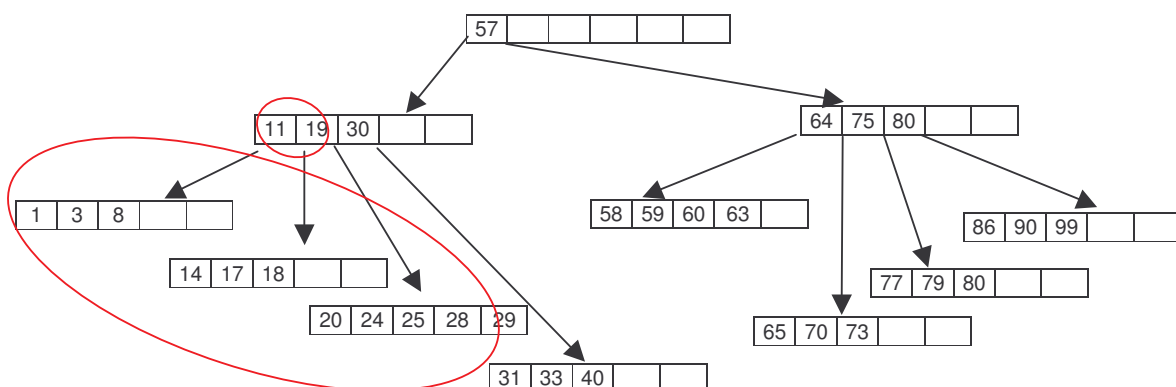
Baja común.

11) Baja de 67:



12) Baja del 8:

Supongamos que nos encontramos frente a la situación:

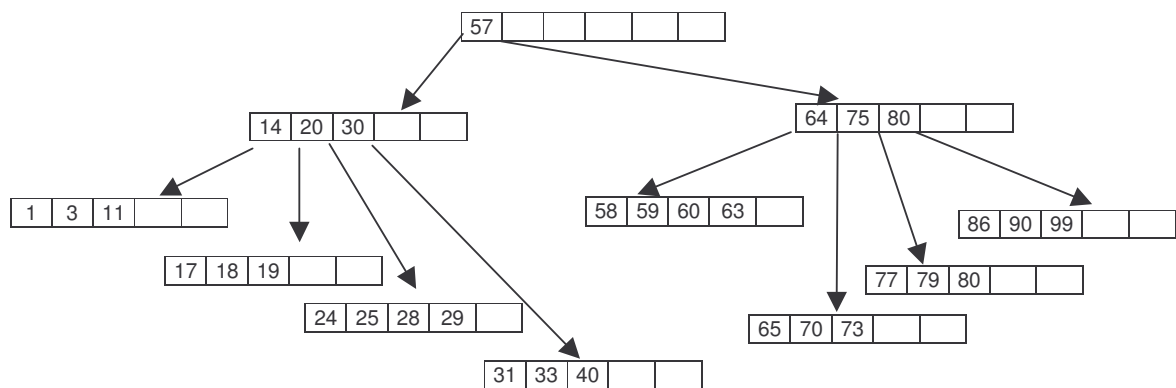


Y se desea eliminar el elemento 8.

A simple vista parecería tratarse de un caso de underflow con concatenación. Mas esto no es así. La concatenación se realizaría con un total de 12 claves (10 restantes en los 3 nodos de la izquierda y 2 del nodo superior). Estos elementos deben entrar en 1 separador y 2 nodos de 5 elementos: 11 lugares disponibles... ¿Qué es lo que está mal?

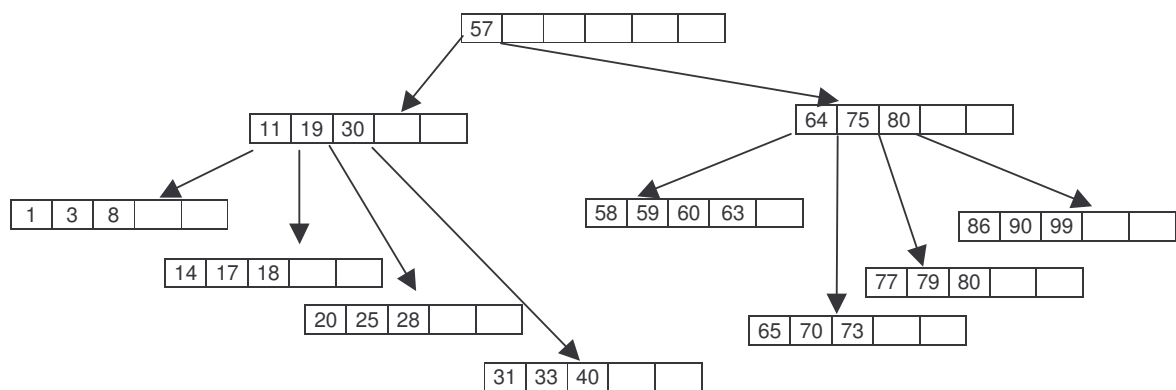
La respuesta se obtiene si observamos que en el caso de los nodos en los extremos, donde existe un único sibling, la concatenación se realiza incluyendo el nodo que le sigue al sibling disponible (respetando la reducción de 3 a 2). Por lo tanto se debe permitir la redistribución con este nodo más distante.

El resultado de la operación sería el siguiente:



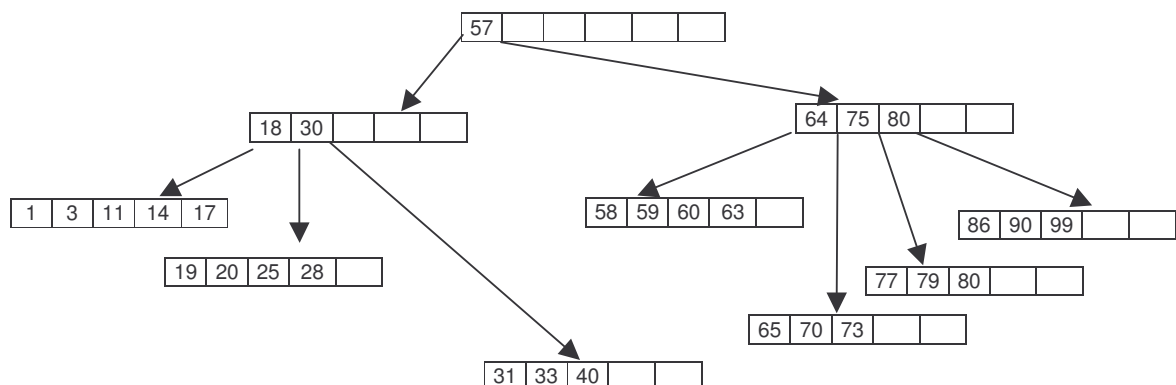
Caso especial de baja con redistribución de los nodos de los extremos.

13) Por último supongamos que hubiéramos llegado a la siguiente situación:

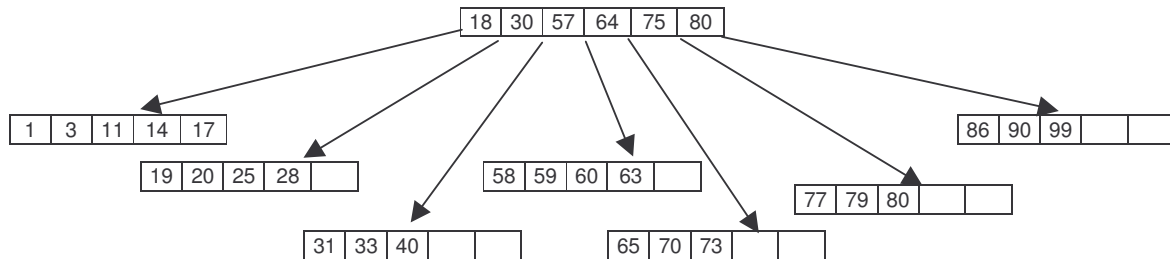


Y se realiza la baja del 8

En primer lugar, nos es posible realizar redistribución, por lo tanto de 3 nodos se concatenan quedando dos.



Al mismo tiempo se produce un underflow en el nodo padre. Por lo tanto ocurre una nueva concatenación en el nivel superior:



Se realiza una disminución de la altura en el árbol con la concatenación de dos nodos en la raíz (recordar que se eligió para la raíz la separación en alta de 1 a 2 con 2/3 de claves llenas)

Árbol B+

Los árboles B y B* son buenos para la búsqueda de un elemento del árbol mediante su clave, limitando la cantidad de accesos al soporte drásticamente con respecto a métodos propuestos antes de su aparición. No obstante, si lo que se desea es encontrar un elemento y a partir de allí leer secuencialmente los siguientes elementos, nos encontramos a un problema difícil de solucionar que nos lleva al peor de los casos de tener que recorrer gran cantidad de nodos y realizando gran cantidad de accesos a soporte.

En ese caso se desearía tener una organización donde las claves se encuentren físicamente contiguas, ordenadas y una facilidad y gasto de pocos recursos para el mantenimiento (alta y bajas de claves).

El árbol B+ surge como una adecuada solución a este problema, combinando la estructura de un árbol B con un set-secuencial, permitiendo tanto el acceso por referencia como el acceso secuencial.

Características y propiedades de un árbol B+.

Un set-secuencial es un conjunto de bloques de tamaño fijo que contienen una cantidad máxima de elementos. Estos elementos se encuentran ordenados en base a sus claves. Por otra parte cada bloque cuenta con una referencia al bloque anterior y otra al posterior.

Los bloques se manejan en forma similar a los nodos de los árboles B, al producirse un overflow o un underflow se realiza redistribución, concatenación o split según corresponda.

Esta estructura permite recorrer secuencialmente todas las claves por bloque y al llegar al final pasar al siguiente para seguir leyendo.

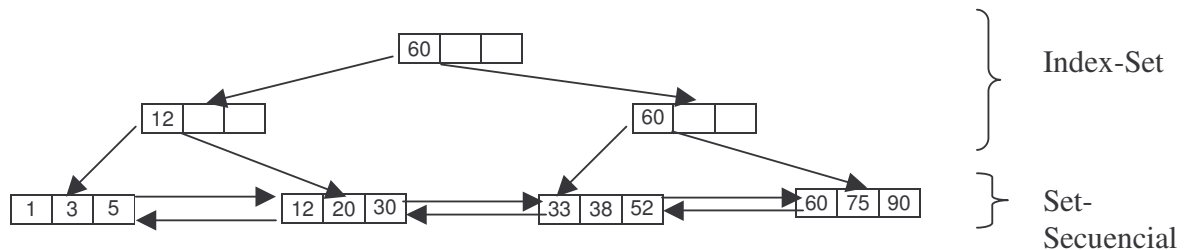


Ejemplo de set-secuencial.

Por otro lado sobre este set secuencial se construye un index-set. Este es un árbol B cuyas hojas apuntan a los bloques del set secuencial. Cada elemento contenido en un nodo de este árbol B no es necesariamente una clave completa, sino que puede ser tan sólo un separador, es decir, un valor que permite saber si la clave que se busca está en el subárbol que está a la izquierda del separador o en el de la derecha.

Estos separadores pueden ser una copia del valor de una clave o simplemente algún prefijo que permita hacer la separación. A los árboles B+ que guardan prefijos para separar en vez de claves completas se los llama árboles B+ de prefijo simple y utilizan un algoritmo para determinar el prefijo mas corto que pueda usarse como separador en cada caso.

Por lo tanto, todas las claves de un árbol B+ aparecen en el set secuencial. En el index-set se repiten las algunas claves (o partes de ellas) que permite la búsqueda indexada.



Operaciones en Árboles B*

Búsqueda:

Se comienza buscando por la raíz y si la clave buscada es menor a la leída se baja al subárbol de la izquierda. En cambio, si es mayor o igual, se baja al subárbol de la derecha. Es muy importante tener en cuenta que si se busca una clave (por ejemplo 60), no importa que se la encuentre en algún nodo del árbol puesto que en tal caso estará actuando tan sólo como separador. Siempre habrá que recorrer el árbol hasta el nivel de las hojas y recién ahí se encontrará la referencia al bloque del secuence set en el cual se aloja realmente la clave

Inserciones:

Las inserciones se manejan comenzando por el set secuencial. Se identifica por medio del algoritmo de búsqueda en que bloque del secuence set se deberá insertar la clave. Si el bloque no está lleno se inserta la clave y se debe verificar que el separador siga sirviendo, si no es así se actualiza (esto será necesario cuando se inserte una clave al principio de un bloque). Si el bloque está lleno se deberá hacer un split en el cual no se promueven claves sino que todas se distribuyen en los dos bloques resultantes. Una vez distribuidas las claves se selecciona un separador y es este separador lo que se promueve al Index Set. Si al promover este separador se produce un overflow en el nodo en el que se lo debe insertar, se procede como en un árbol B común.

Bajas :

Las bajas se hacen también sobre el set secuencial. Si al hacer una baja se produce un underflow en el bloque del set secuencial, se intenta en primer lugar redistribuir las claves con los bloques vecinos y si se puede redistribuir, se actualiza el separador de bloques en el Index Set. Si no puede aplicarse redistribución porque los bloques contiguos tienen la cantidad mínima de claves se hace una concatenación y se elimina del árbol el separador que separaba los bloques, lo cual podría producir un underflow en el nodo del Index Set. Dicho underflow se maneja de la forma habitual en un árbol B. Al darse de baja un elemento, si su clave se encuentra en el index set, no es necesario eliminarla, ya que solo actúa como separador.

Árboles B virtuales

La organización en nodos que contienen varias claves hace que el árbol B sea una estructura que resulte muy fácil de paginar. Se puede trabajar manteniendo en memoria uno o varios nodos por vez y el resto en disco. Cuando se necesita acceder a un nodo que no está en memoria se baja a disco el que hace mas tiempo que no se accede (Least Recently Used) y se sube el que se necesita en su lugar. Como cualquier búsqueda que se haga comienza siempre por la raíz, esta estará siempre en memoria. En algunas pruebas hechas con un árbol B+ se ha calculado que manteniendo cerca del 15% del total de nodos del árbol en memoria, se logra disminuir la cantidad promedio de accesos a disco por búsqueda a menos de 1. Esta reducción debería ser aún mayor en árboles B y B* puesto que estos no necesitan llegar hasta el último nivel en cada búsqueda.