Resumen Clase 6:

Cual es el problema de los índices?

La búsqueda binaria es costosa. Log base 2

Mantener los índices ordenados es costoso

Solucion 🡺 RAM nanosegundos

Objetivo 🡺 Persistencia de datos. milisegundos

Arboles Multicamino: De cada padre pueden salir mas de dos hijos.

La idea es que los arboles multicaminos lleguen a un punto donde sean balanceados.

Busqueda Binaria: Log base2: A priori es lo mas eficiente, pero debemos tener un archivo, y índices ordenados, el tema es que ordenar los índices es bastante costoso.

En la clase pasada vimos que se podira usar índices a través de un vector en RAM. Pero , para persistir los datos lo mejor seria almacenar los datos en disco. Esto lo vamos a ver, usando la estructura árbol, y persistiéndola en disco.

Una alternativa para los arboles binarios, que permita trabajar como arboles balanceados, son los Arboles AVL. Los arboles AVL son arboles binarios balanceados en altura.

En los arboles AVL, la diferencia entre el camino mas corto y el camino mas largo entre un nodo terminal (nodo hoja) y la raíz, no puede diferir en mas que 1

El árbol AVL, tiene la particularidad de que, mientras se va creando el árbol, se tiene que ir reacomodando para que siemore, en cada inserción cumpla con el criterio mencionado anteriormente.

Paginacion de arboles binarios:

Las operaciones de lectura y escritura en un archivo utilizando buffers presentan una mejora del performance. Este concepto se puede llevar a cabo cuando se genera el archivo que contiene el árbol binario.

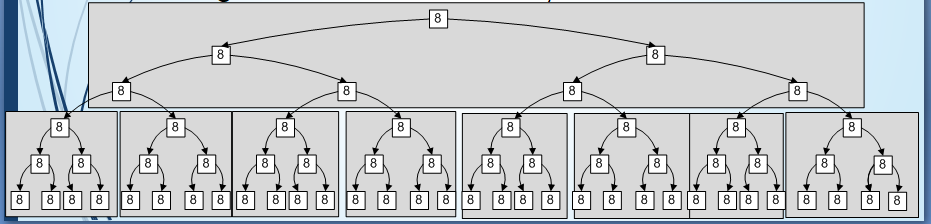
El árbol se divide en paginas, y cada pagina contiene un conjunto de nodos, los cuales están ubicados en direcciones físicas cercanas.

De esta manera, podemos lograr que al momento de transferir datos, no se acceda a disco para transferir unos pocos bytes (un solo registro), sino que se transmita una pagina completa.

Una organización de este tipo, reduce el numero de accesos a disco para poder recuperar información.

Un ejemplo en este árbol es que, al transferir la primera pagina, se están recuperando siete nodos del árbol, que representan los primeros tres niveles del árbol. (2^h) – 1 = (2^3) -1 = 7.

De esta manera, logramos encontrar un elemento en el primer recuadro con un acceso a disco, y como máximo 3 lecturas. La idea es dividir el árbol en recuadros, para minimizar los accesos a discos, pero con el costo de que el árbol debe estar balanceado para mejorar la performance.



Siguiendo dicha figura, con solo dos accesos a disco es posible recuperar un nodo especifico entre 63. (2^h)-1 = (2^6) -1 = 63.

Al dividir un árbol binario en paginas, es posible realizar búsquedas de datos almacenados en memoria secundaria, de manera mas rápida.

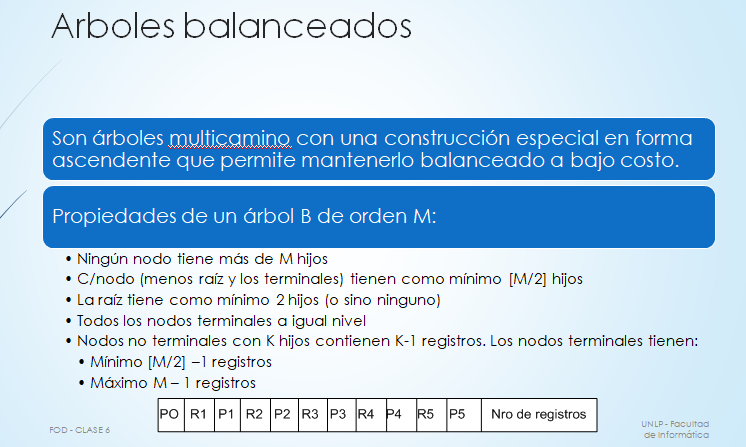
El tamaño del buffer determina el tamaño de la pagina a traer.

Es decir, si tenemos un buffer donde entran 255 NODOS, el tamaño de cada pagina seria de 255 NODOS, resultando la performance final de búsqueda de orden log\_256 (N), es decir: log\_k+1 (N)

Siendo N la cantidad de claves del archivo, y K la cantidad de nodos por pagina.

El objetivo de la paginación, es hacer que de una lectura en disco, no traigamos solo un registro, sino que traigamos por ejemplo 10.000 bytes a un buffer, de esta manera, podemos encontrar un registro en en un árbol de altura 3, con un acceso a disco y 3 lecturas

Se introduce la idea de arboles balanceados:

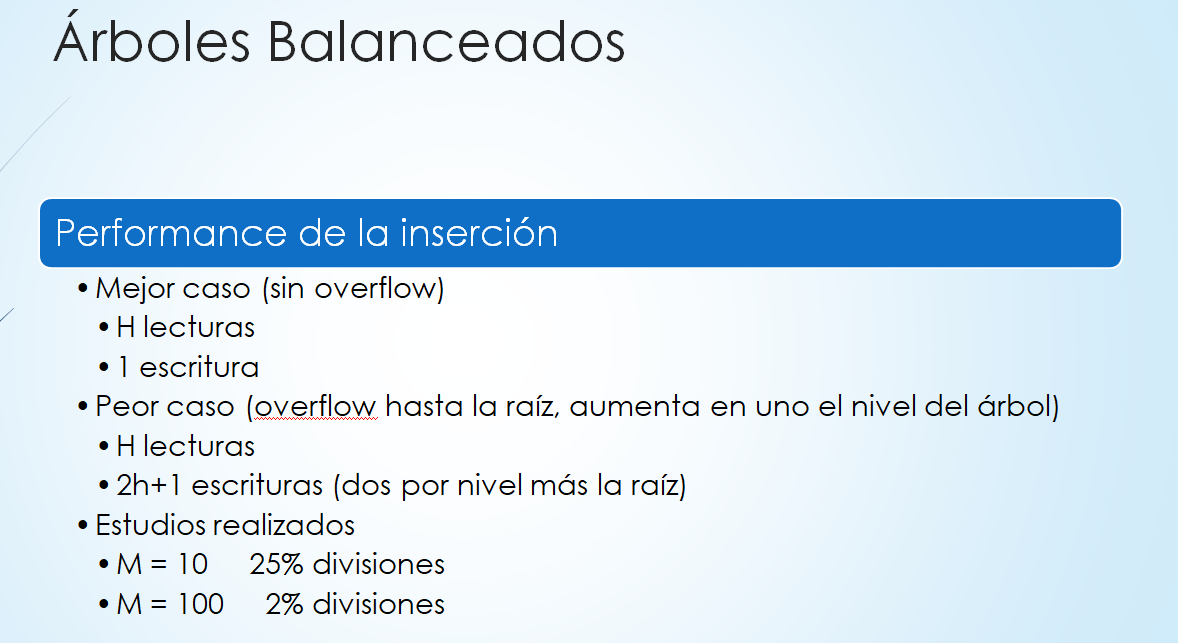
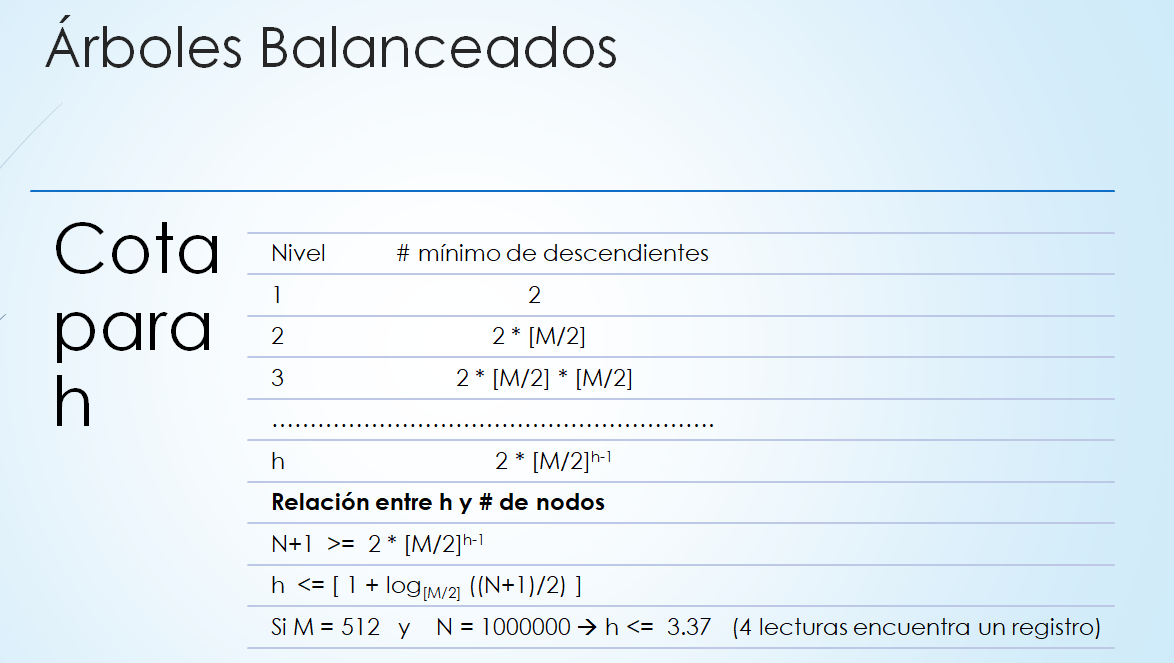


La performance de búsqueda en los arboles balanceados es:

Mejor caso: 1 lectura

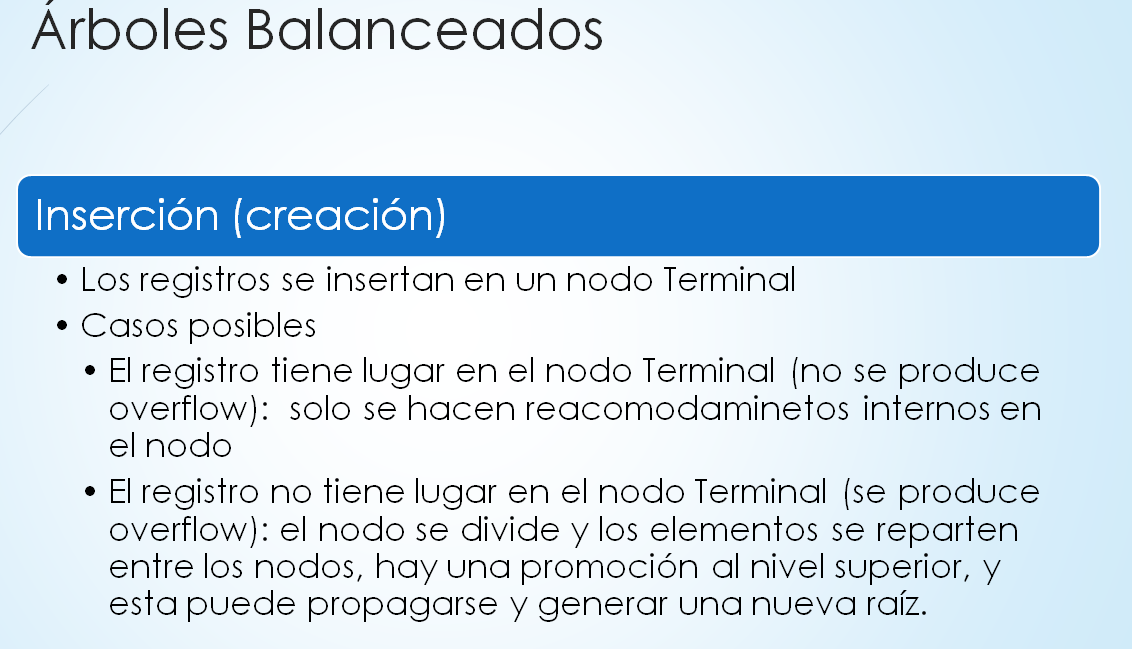
Peor caso: h lecturas (con h : altura del árbol )

* Axioma: árbol balanceado de Orden M, si el número de elementos del árbol es N 🡪 hay N+1 punteros nulos en nodos terminales.



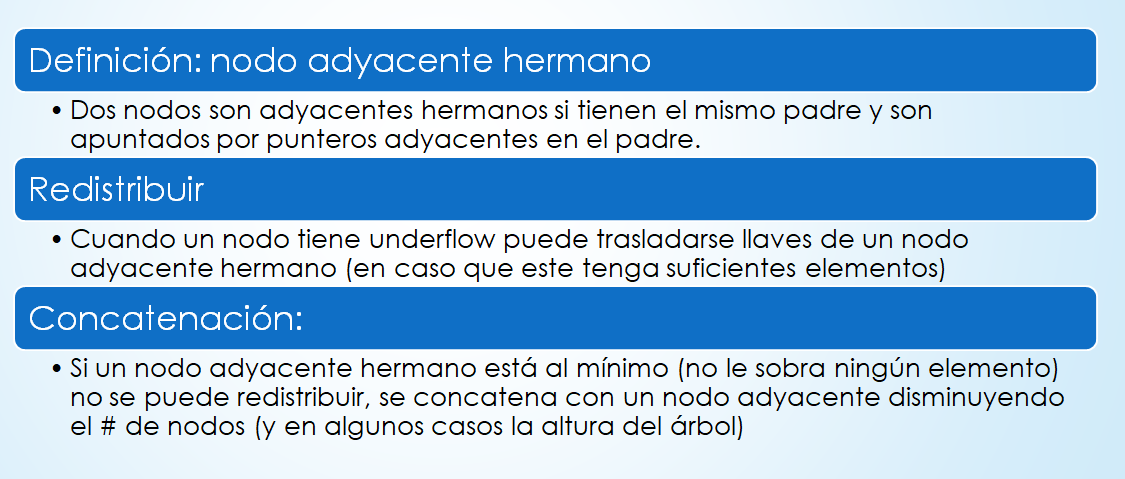
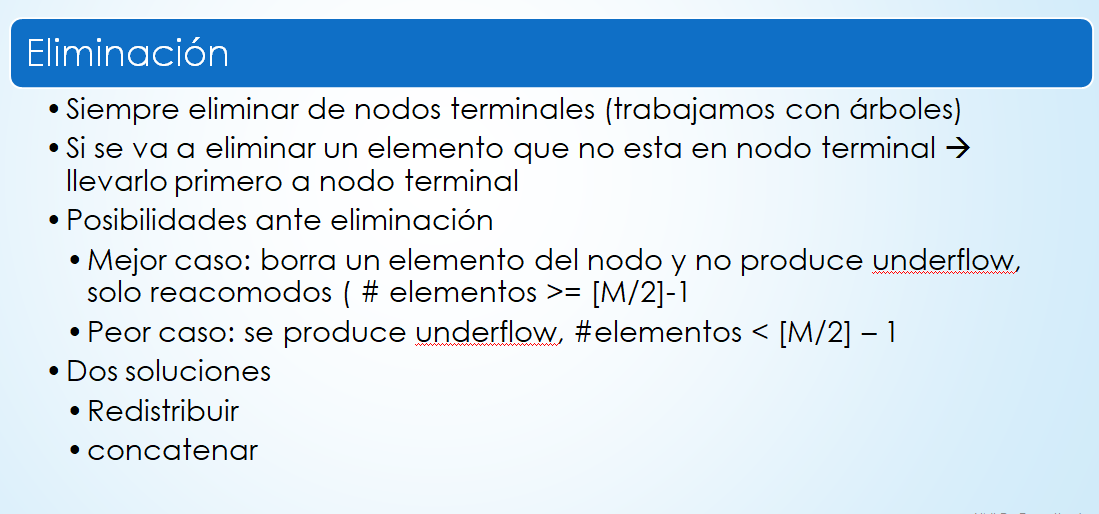
Performance de búsqueda en arboles balanceados:

* Mejor caso: 1 lectura
* Pero caso: h lecturas (con h altura del árbol)
* Cual es el valor de h?
  + Axioma: árbol balanceado de Orden M, si el número de elementos del árbol es N 🡪 hay N+1 punteros nulos en nodos terminales.



Performance de la inserción

* Mejor caso (sin overflow)
  + H lecturas
  + 1 escritura
* Peor caso (overflow hasta la raíz, aumenta en uno el nivel del árbol)
  + H lecturas
  + 2h+1 escrituras (dos por nivel más la raíz)

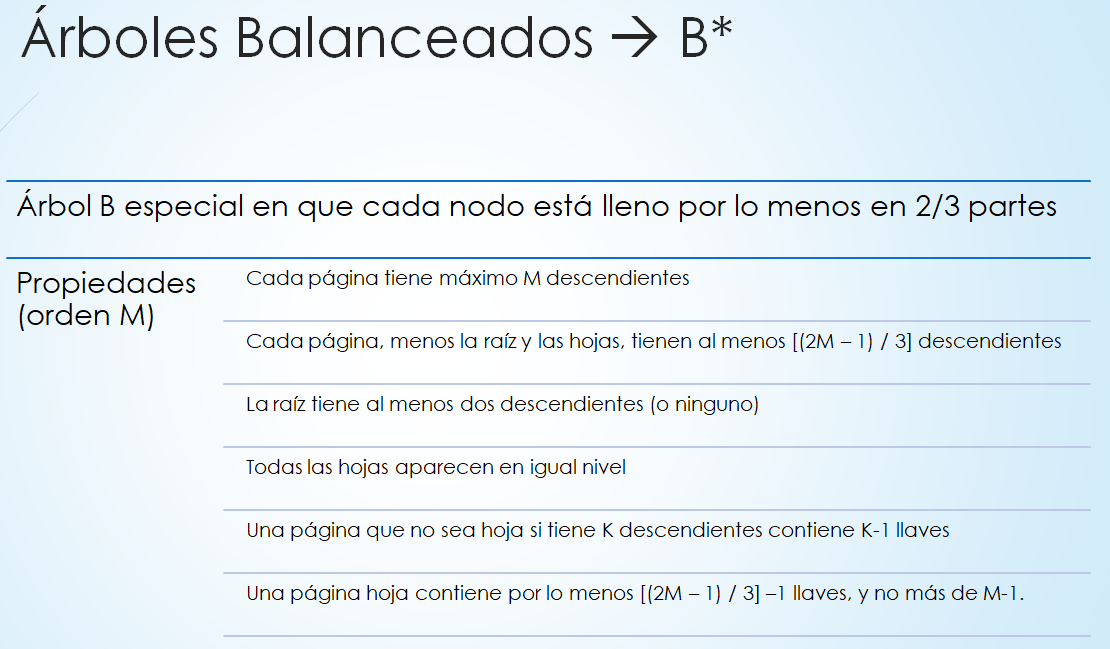


Performance de la eliminación

* Mejor caso (borra de un nodo Terminal)
  + H lecturas
  + 1 escritura
* Peor caso (concatenación lleva a decrementar el nivel del árbol en 1)
  + 2h – 1 lecturas
  + H + 1 escrituras

FIN DE ARBOLES B

EMPIEZA ARBOLES B \*



Los arboles B son arboles multicamino con una construcción especial de arboles que permite mantenerlos balanceados a bajo costo.

Propiedades de un Arbol B de orden M

LOS ELEMENTOS DENTRO DEL ARBOL SIEMPRE ESTAN ORDENADOS

* Cada nodo del árbol puede contener **como** **máximo** :

**M referencias y M-1 elementos.**

* La raíz puede tener 0 hijos, o al menos 2
* **Un nodo con X referencias directas contiene X-1 elementos.**
* Todos los nodos (**salvo la raiz**) tienen como **minimo :**

**(M/2)-1 ELEMENTOS**

Y como **máximo: M – 1 ELEMENTOS (claves)**

* Todos los nodos terminales se encuentran al mismo nivel

Uso de buffers:

Las operaciones de lectura y escritura de datos de un archivo utilizando buffers presentan una mejora de performance.

Cuando se genera el archivo que contiene el árbol binario, dicho árbol se divide en paginas, y cada pagina contiene un conjunto de nodos los cuales están ubicados en direcciones físicas cercanas.

Asi cuando se transfieren datos, no se accede a disco para transferir unos pocos bytes, sino que se transmite una pagina completa.

Una organización de este tipo, reduce el numero de acceso a disco necesarios para poder recuperar información.

Eficiencia de búsqueda en arboles b:

Consiste en contar los accesos al archivo de datos que se requieren para localizar un elemento o para determinar que el elemento no se encuentra.

El resultado, es un valor acotado entre el rango (1,H), siendo H la altura del árbol.

Si el elemento se encuentra ubicado en el nodo raíz, la cantidad de accesos requeridos es 1. En caso de localizar al elemento en un nodo terminal ,(o que el elemento no se encuentre), la cantidad de accesos serian H.

Eficiencia de inserción en un árbol B:

Una de las propiedades de los arboles B, determina que todos los elementos se insertan en los nodos terminales. Por ese motivo, es necesario realizar H lecturas para poder encontrar el nodo donde el elemento será almacenado.

(Comenzamos en la raíz, y avanzamos hasta el nodo terminal)

El peor de los casos en la insercion es que el overflow se propague hacia el nodo raíz, haciendo que, se genere un nuevo nivel en el árbol.

Performance de inserción en un árbol b:

H lecturas , 1 escritura (mejor de los casos)

H lecturas , (2 \* H )+1 (peor de los casos)

Eliminacion en arboles B:

Siempre, se elimiinan elementos en los nodos terminales.

La primera situación se da cuando se elimina la clave, y la cantidad de elementos restantes del nodo no está por debajo de la cantidad minima (M/2)-1 elementos. En este caso no se genera underflow y el proceso de la baja finaliza.

La segunda situación se da cuando se genera una situación de underflow. En este caso, el nodo al que se le borra el elemento queda por debajo del minimo: (M/2)-1 elementos.

Cuando se genera underflow, tenemos dos posibles caminos para solucionarlo:

Primer opción: Redistribuir, se podrá redistribuir siempre y cuando algún hermano pueda prestarle un elemento.

Para que un hermano adyacente pueda prestar un elemento, tiene que tener una cantidad de elementos MAYOR a la minima.

De esta manera, se promociona hacia el padre, el elemento mas chico de ese nodo, y el padre baja hacia el nodo que se produjo underflow

(Obviamente teniendo en cuenta que siempre los nodos deben quedar ordenados)

Si no es posible hacer una redistribución, se deberá hacer una fusión o concatenación.

Segunda opción: Para hacer una fusión o concatenación, se concatena en un mismo nodo: El nodo donde se elimino el elemento, su nodo adyacente, y el nodo padre, todo eso se junta en el nodo de la izquierda.

Una ventaja que presenta la redistribución es que la cantidad de nodos no se ve afectada, soli había que cambiar de posición elementos entre tres nodos (dos hijos, y el padre)

La concatenación, tiene la desventaja de producir underflow en el nodo padre, y de esa manera se propagaría el underflow hacia arriba, y se debería realizar alguno de los dos caminos explicados anteriormente.

Arbol B\*

Cada nodo del árbol puede contener como máximo M referencias,

y M-1 Elementos.

Los nodos terminales tienen como minimo: (2M-1)/3 elementos

Y como máximo M-1 Elementos.

Los nodos que no son terminales ni raíz, tienen como minimo ((2M-1)/3) referencias

Arbol B+

Cada nodo del árbol puede contener como máximo M Referencias , y

M – 1 elementos.

Los nodos terminales tienen como minimo (M / 2) -1 elementos, y como máximo M – 1 elementos.

Los nodos que no son terminales ni raíz, tienen como minimo (M/2) referencias

**Los nodos terminales, representan un conjunto de datos, y son entrelazados entre ellos.**

Esta ultima propiedad diferencia establece la principal diferencia entre arboles B y arboles B+. De esta manera, para poder realizar un acceso secuencial ordenado a todos los registros, es necesario que cada elemento aparezca almacenado en un nodo terminal.

En este tipo de arboles, se utiliza una clave como separador, pero se realiza una copia del elemento, y no el elemento en si.

Insercion en arboles \*

Los elementos se insertan siempre en nodos terminales. Si se produce una saturación, el nodo se divide y se promociona a una copia del menor de los elementos mayores hacia el nodo padre. (Si el padre no tiene espacio, se dividirá nuevamente.)

En caso de dividir un nodo no terminal, se debe promocionar hacia el padre el elemento en si, y no una copoa del mismo. Es decir**, solo ante la división de un nodo terminal se debe promocionar una copia.**

Para borrar un elemento en un árbol b+, siempre se borra de un nodo no terminal, y si hubiese una copia de ese elemento en un nodo no terminal, esa copia se mantendrá porque sigue actuando como separador.

Arbol B+ de prefijos simples: Es un árbol b + donde los separadores están representados por la minima expresión posible de la clave. Que permita decidir si la búsqueda se realiza hacia la izquierda o hacia la derecha.