Tipos de árboles especiales

Arboles binarios balanceados

- Un árbol balanceado (llamados AVL por sus “descubridores” Adelson, Velski y Landis) es un árbol binario donde la altura de cada subárbol no difiere de 1 para cada nodo.

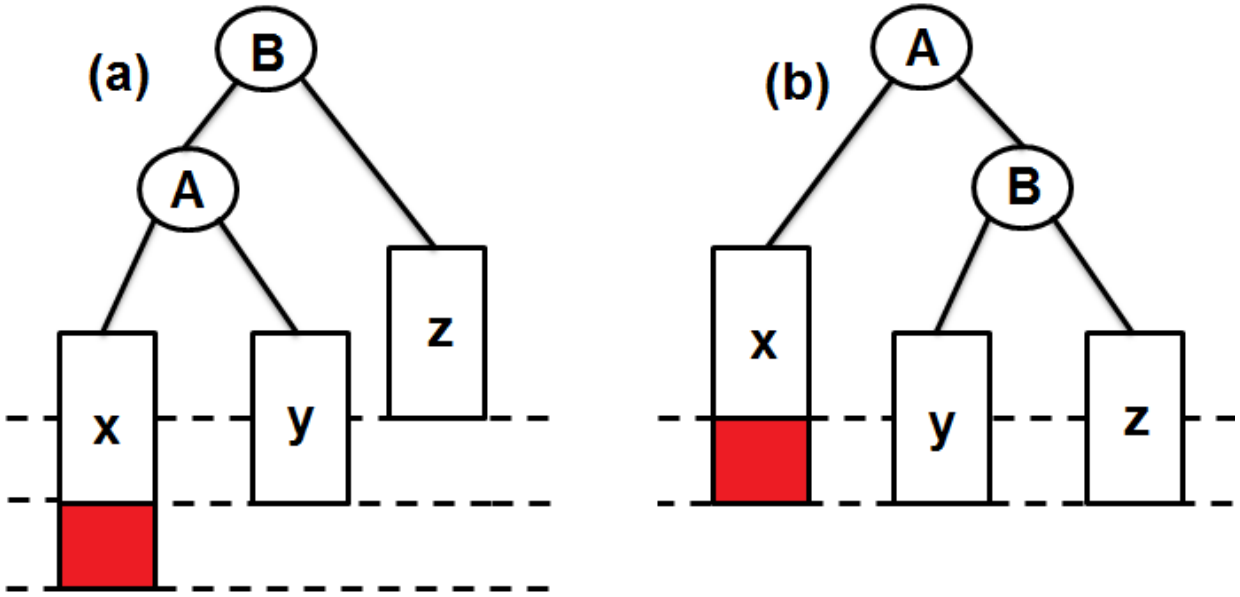
- AlturaArbolIzquierdo – AlturaArbolDerecho < 2

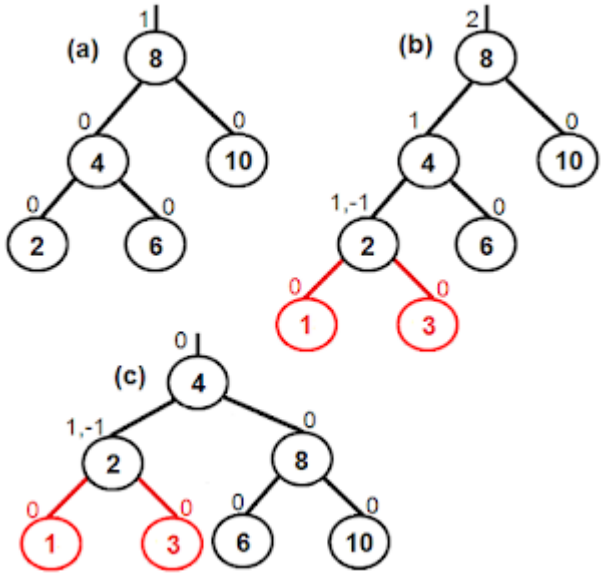
- Esto trata de impedir el desbalance derivado de inserciones en desorden, manteniendo las cualidades de búsqueda binariar del árbol
.

Arbol Binario Balanceado

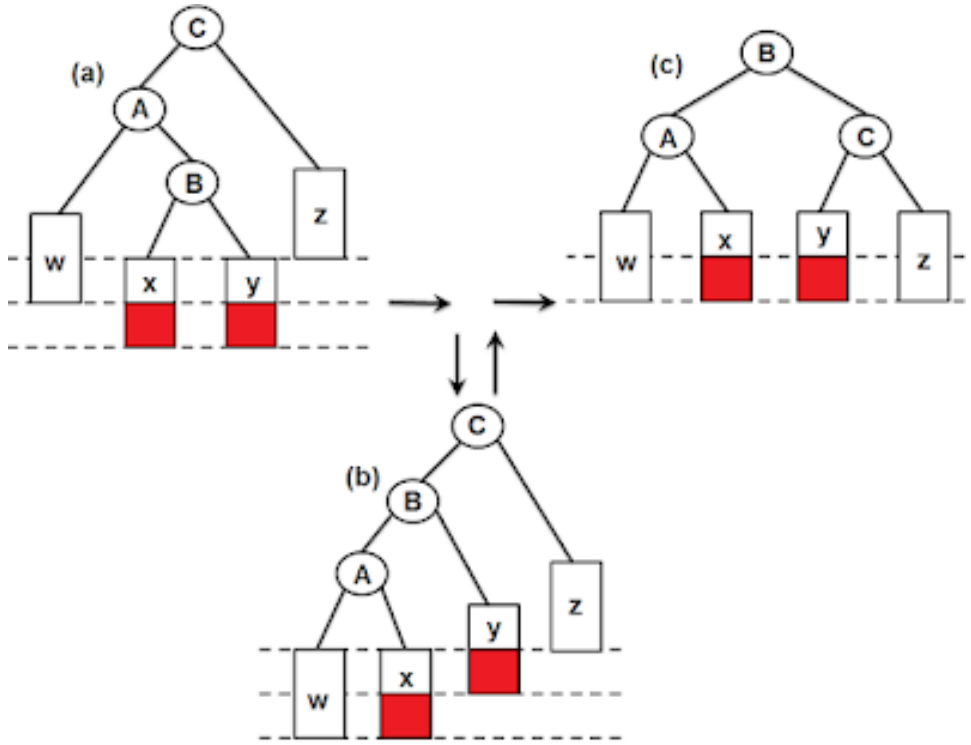
- Tiene 4 operaciones básicas que se eejecutan después de insertar un nodo
  - Rotación simple derecha
  - Rotación simple izquierda
  - Rotación doble izquierda derecha
  - Rotación Doble derecha izquierda

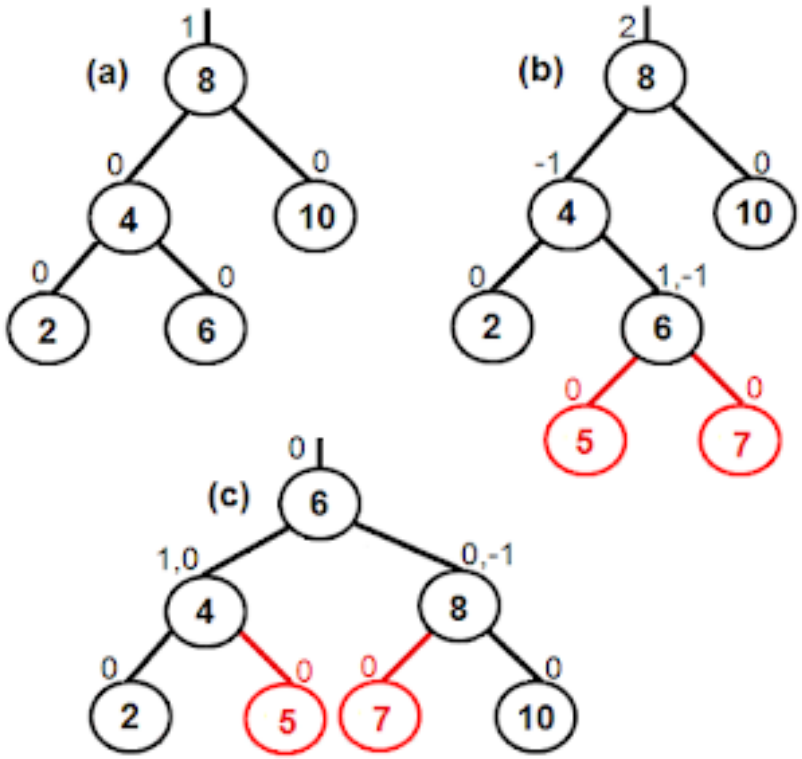
Rotaciones simples:





Rotaciones Complejas





Arbol Red-Black

- Un árbol red-black (RB) es un árbol que tiene un atributo extra en cada nodo: el color.

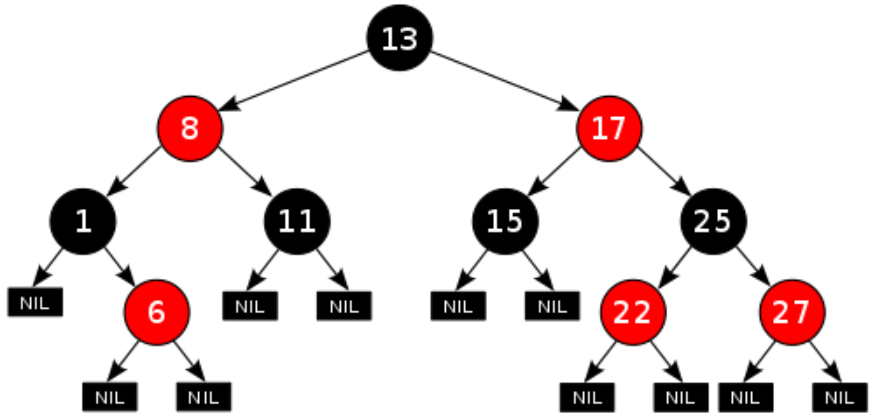
- El propósito de este atributo es asegurarse que el árbol este “aproximadamente” balanceado

- La idea es que ninguna ruta entre 2 nodos sea mas del doble que cualquier otra ruta.

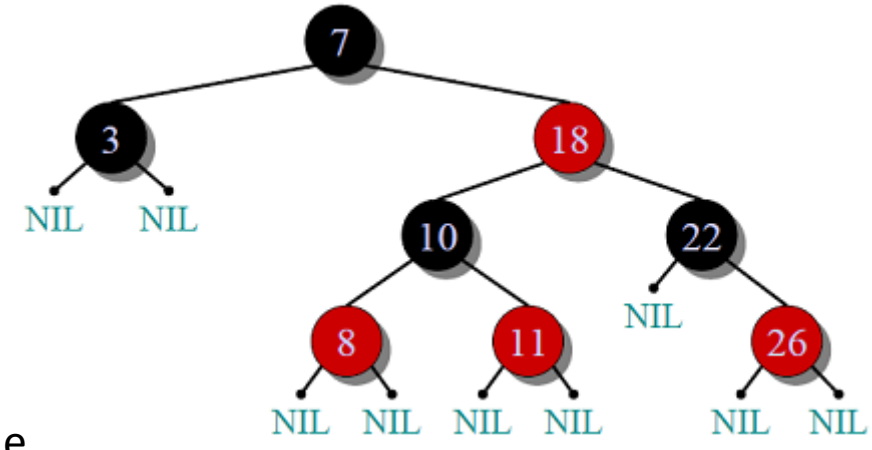
Reglas del Arbol RB

- Cada nodo es Rojo o Negro
- La raíz es negra
- Cada hoja es negra.
- No hay 2 nodos rojos adjacentes
  - Un nodo rojo no puede tener hijos ni padre rojo!
- Si un nodo es rojo, los hijos son negros.
- Para cada nodo, todas las rutas desde el nodo a las hojas tienen la misma cantidad de nodos negros.

Arboles Rojo-Negro



Los árboles rojo-negro aceptan un poco de ineficiencia en la búsqueda, a cambio de más simplicidad a la hora de insertar.

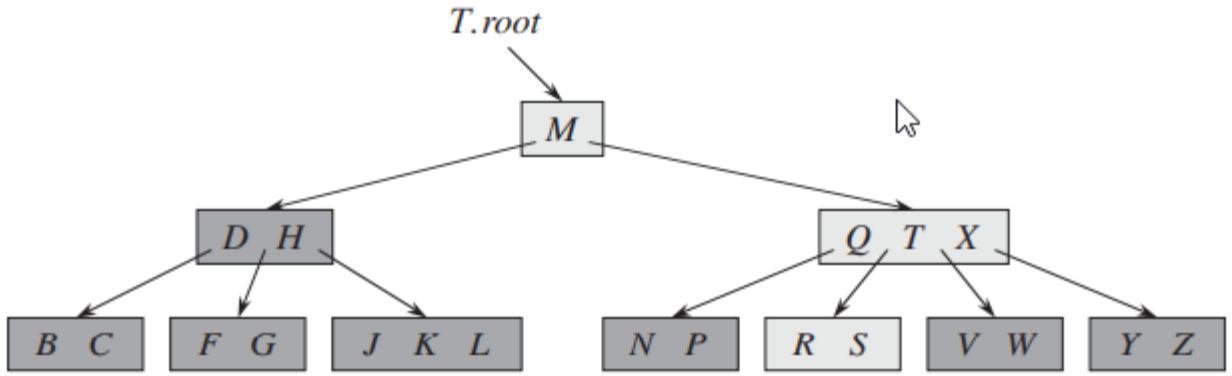


Como no tienen que tener una diferencia máxima de alturas de 1, las operaciones balance son más simples.

B-Trees

- Son árboles orientados al balance: pero pueden tener más de 2 hijos.
- Se implementan normalmente siguiendo la idea de los arboles RB: no puede haber rutas desde la raíz que sean más del doble que otras rutas.
- Son apliamente usados en almacenamiento secundario: las bases de datos usan ampliamente B-Trees.

B-Trees



Como todo árbol, tiene una raíz.

Y cada nodo contiene N llaves o elementos.

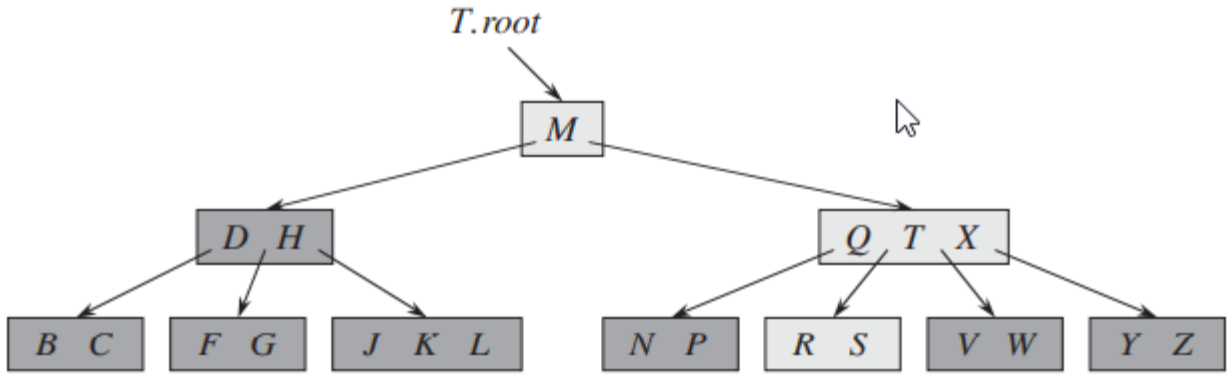
Llamaremos a esas llaves k.

Hay desde k_0…. k_{n-1}

Estas llaves están ordenadas de forma que

$k_0 < k_1 < \ldots. < k_{n-1}$

B-Trees



Cada nodo contiene N+1 punteros a sus hijos.

Un nodo “hoja” no tiene ningún “hijo”

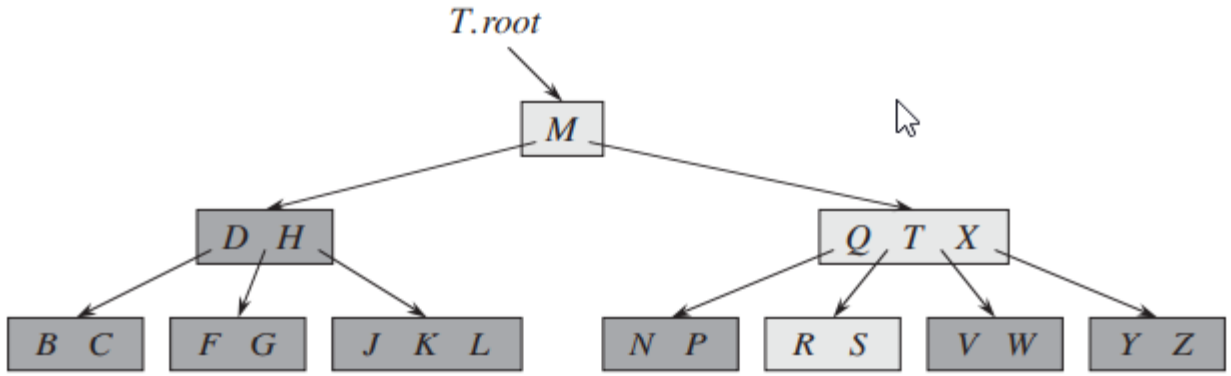
Cada nodo contiene un atributo “leaf” que es TRUE si es hoja, FALSE si no lo es.

Sea k_i una llave,

todos las llaves en el puntero ANTERIOR a k_i, son menores que k_i

y todas las llaves en el puntero POSTERIOR a k_i son mayores que k_i

B-Trees



Última regla, y la más jodida:

TODAS LAS HOJAS DEBEN ESTAR A LA MISMA ALTURA.

Todo nodo que NO sea la raíz requiere tener un número mínimo de llaves que contener, >= 2.

También pueden tener un máximo de llaves.

Esto se conoce como el GRADO MÍNIMO del B-tree

La cantidad de llaves que contiene determina el número de Nodos habilitados para sus hijos.

Operaciones de un B-Tree

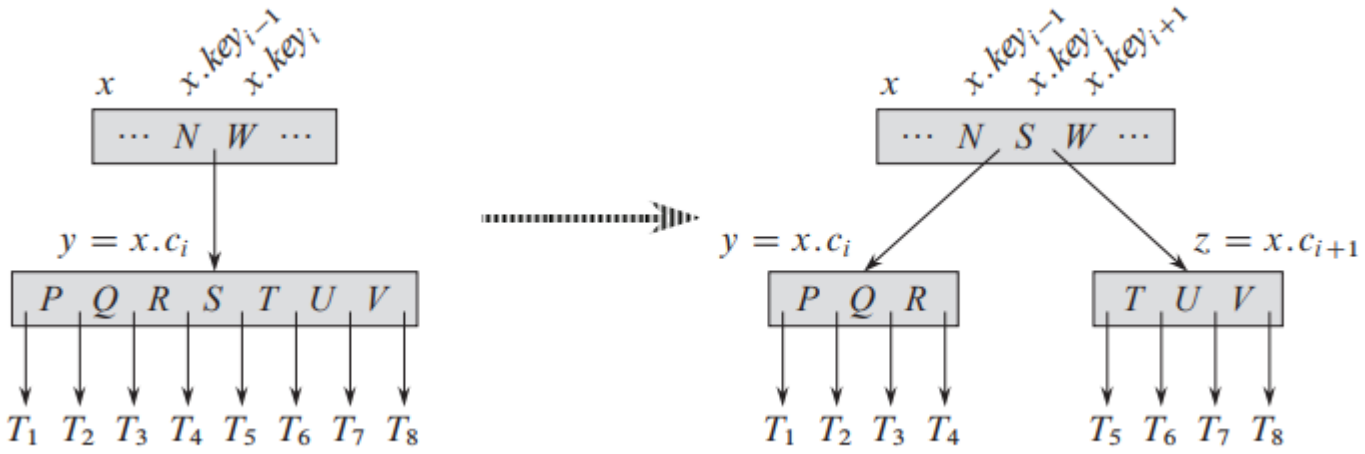
- Búsqueda
  - Es como buscar en secuencia a través de los punteros y las llaves:
    Si lo que busco es menor que la llave que proceso, voy al hijo anterior,
    si es mayor pero menor que la siguiente, voy al puntero del hijo posterior
    si no, itero a la siguiente llave.

```
B-TREE-SEARCH(x, k)
1  i = 1
2  while i ≤ x.n and k > x.key_i
3       i = i + 1
4  if i ≤ x.n and k == x.key_i
5       return (x, i)
6  elseif x.leaf
7       return NIL
8  else DISK-READ(x.c_i)
9       return B-TREE-SEARCH(x.c_i, k)
```

Operaciones en un B-tree

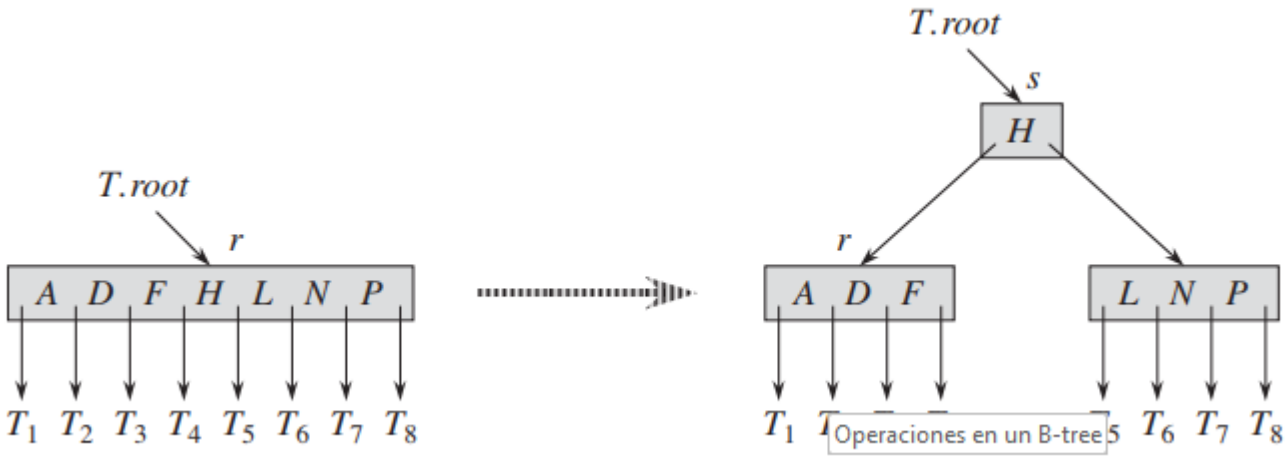
- Insertar:
  - Esto es muy complicado: no podemos solo meter un nuevo nodo entre 2 llaves: qué pasa si el nodo ya tiene el máximo de llaves? Hay que partir nodos!

    Existe de esta forma la operación SPLIT-CHILD que toma un nodo y lo “parte”



Operaciones en un B-tree

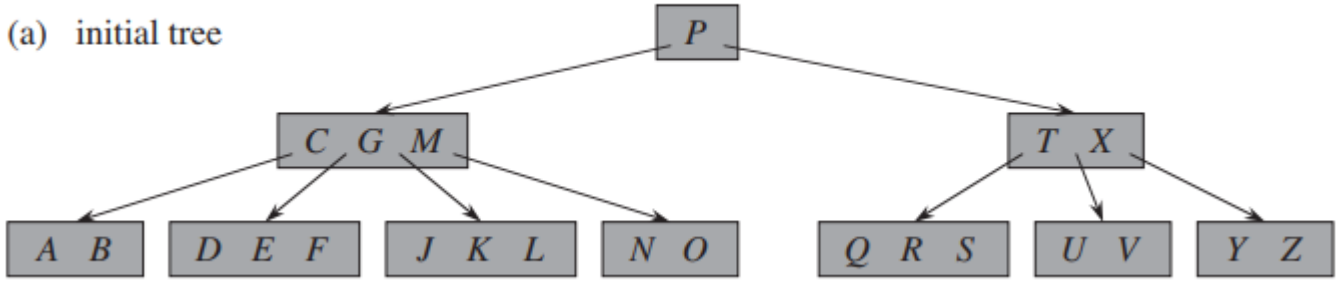
- Insertar:
  - Y si es la raíz? También hay que hacerle “split” buscando la “mediana” para procurar el equilibrio del árbol.



Operaciones en un B-tree

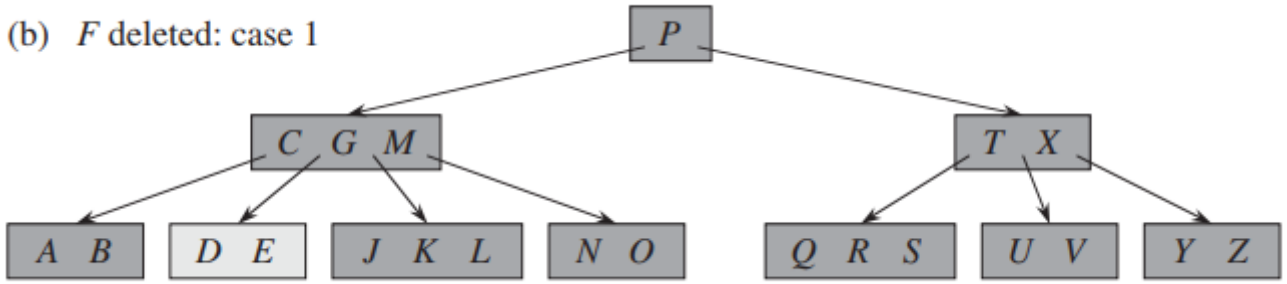
- Borrar:
  - Es todavía más complicado que insertar: porque casi siempre requiere REACOMODAR la estructura de los nodos.

    No podemos permitir que la estructura resultante incumpla las reglas del B-tree.

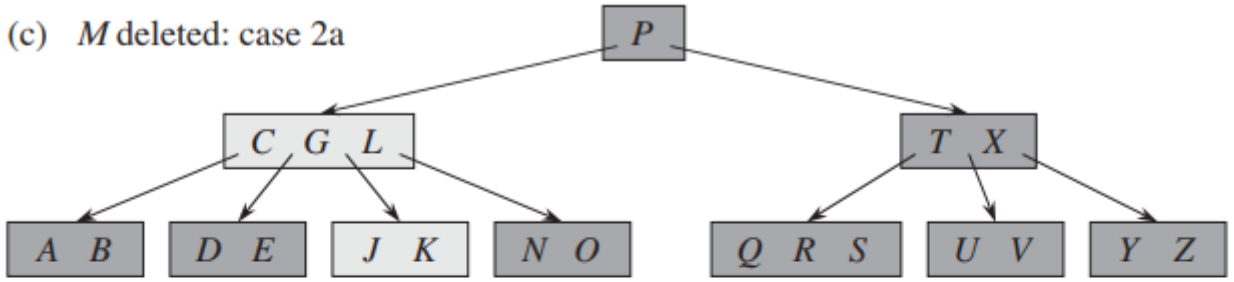


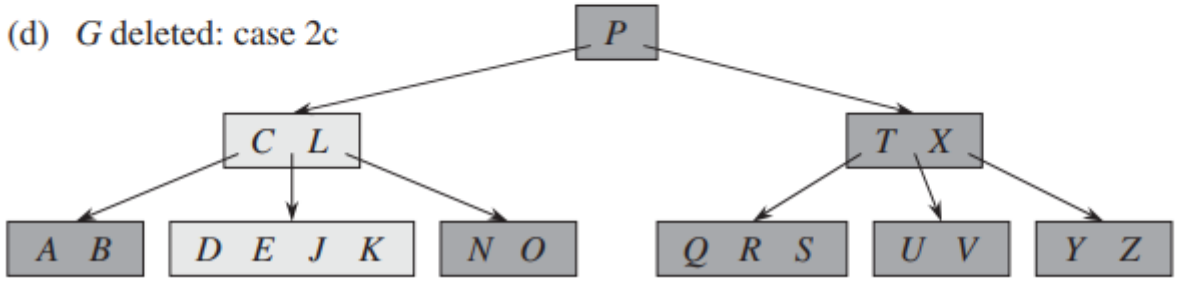
Operaciones en un B-tree

- Borrrado:
  - Caso 1



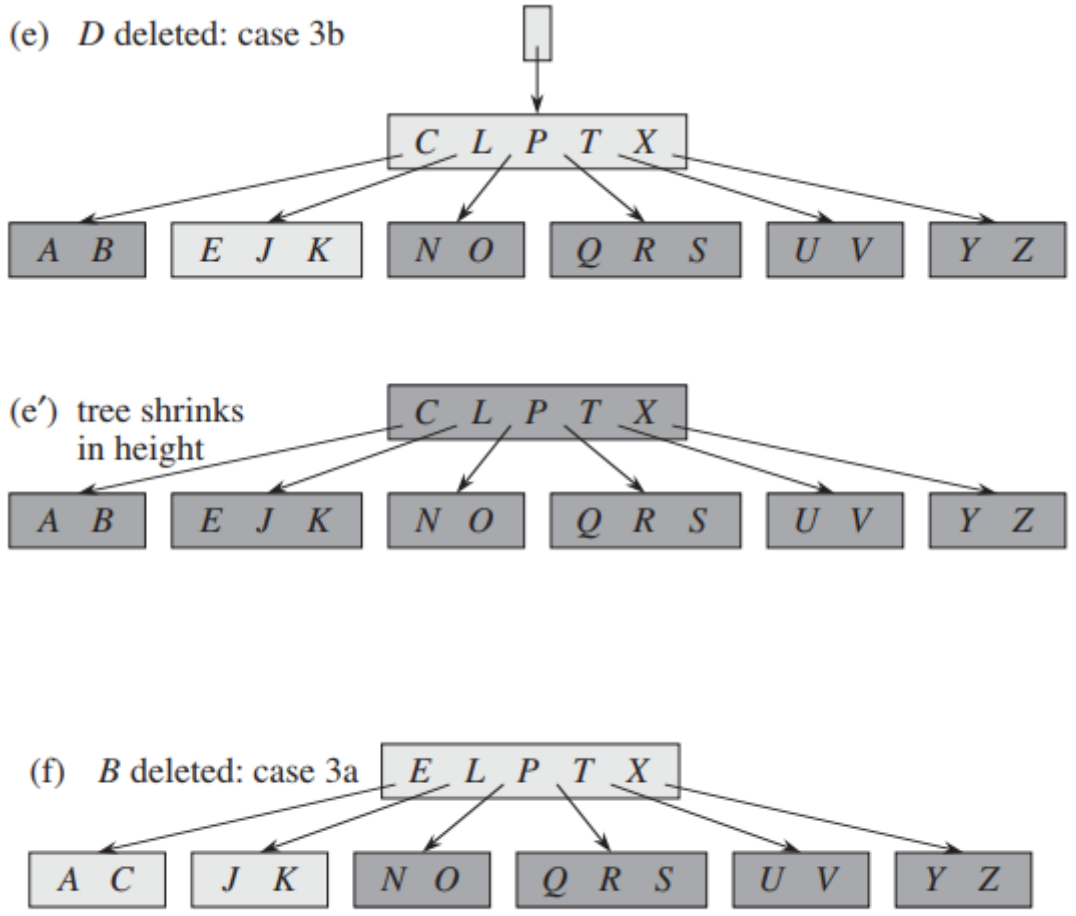
  - Caso 2:





Operaciones en un B-tree

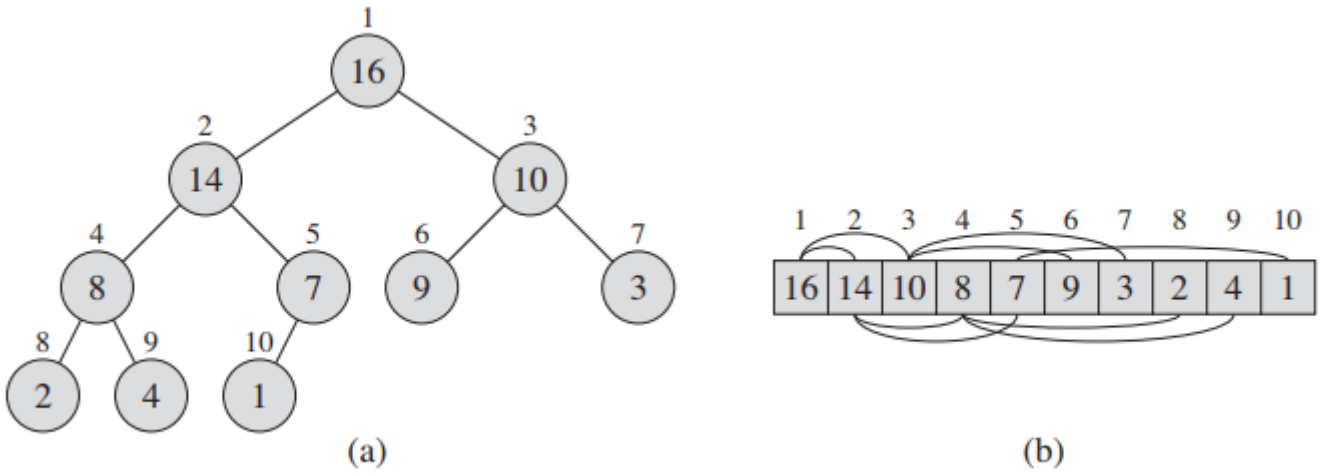
- Borrrado:
  - Caso 3



Heaps (Montón)

- Un “heap” es una estructura que podemos ver como un árbol binario o un arreglo.

  La regla de un heap es que todos los niveles MENOS EL ÚLTIMO deben estar completos y se completan de izquierda a derecha



Heaps (Montón)

- Normalmente se usan para manejar espacios en memoria, pues en lugar de buscar secuencialmente espacio libre, se puede aprovechar la “búsqueda binaria) y encontrar bloques libres más rápido.