Estructuras de datos

Arboles B *(*$B-$*tree)*[a]

Profesor: Gilberto Gutiérrez R.

Departamento de Ciencias de la Computación y

Tecnologías de la Información

Facultad de Ciencias Empresariales

Universidad del Bío-Bío

[a] Tomado del libro “Introduction to Algorithms, Second Edition”. Thomas H. Cormen et. al

Ejemplo de un $B-$tree

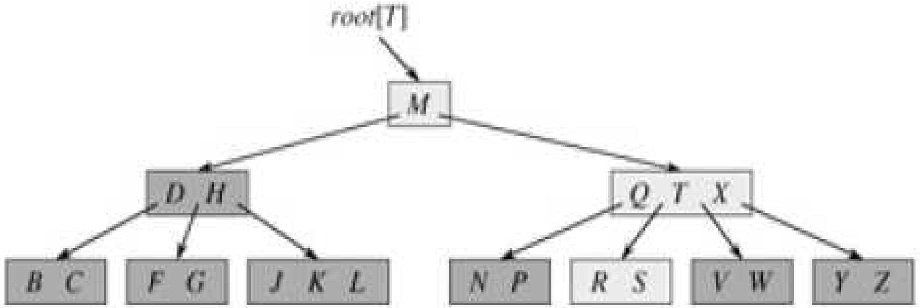


Figure 18.1: A B-tree whose keys are the consonants of English. An internal node x containing $n[x]$ keys has $n[x] + 1$ children. All leaves are at the same depth in the tree. The lightly shaded nodes are examined in a search for the letter R.

Definición de un $B-$tree

Un $B-$tree es un árbol (con raíz $root[T]$) que tiene las siguientes propiedades:

1. Cada nodo x tiene los siguientes atributos
   (a) $n[x]$, número de claves almacenadas en el nodo x
   (b) las $n[x]$ claves se encuentran ordenadas de manera ascendente, es decir, $key_1[x] \leq key_2[x], \ldots, \leq key_n[x]$
   (c) $leaf[x]$ (TRUE si x es una hoja (leaf) y FALSE si es un nodo interno (internal node))
2. cada nodo interno contiene $n[x] + 1$ punteros $c_1[x], c_2[x], \ldots, c_{n[x]+1}[x]$ a sus "hijos". Para los nodos hojas los c_i se encuentran indefinidos
3. Las claves $key_i[x]$ separan el rango de claves en cada subárbol. Si k_i es cualquiera de las claves almacenadas en el subárbol apuntado por $c_i[x]$, entonces $k_1 \leq key_1[x] \leq k_2 \leq key_2[x], \ldots, key_{n[x]}[x] \leq k_{n[x]+1}$
4. Todas las hojas se encuentran a la misma profundidad

Definición de un $B-$tree (cont.)

5 Existe un límite superior y uno inferior para la cantidad de claves que se pueden almacenar en un nodo, los cuales se expresan en términos de un entero fijo $t \geq 2$ llamado el grado mínimo de un $B-$tree $\ldots$

(a) Cada nodo diferente a la raíz debe contener al menos $t-1$ claves y al menos t hijos. Si el árbol no está vacio, la raíz debe tener al menos 1 clave.

(b) Cada nodo puede contener a lo más $2t-1$ claves. Por lo tanto cada nodo puede tener $2t$ hijos. Un nodo está lleno si contiene exactante $2t-1$ claves

Operaciones básicas sobre un $B-$tree

- Creación
- Búsqueda
- Inserción
- eliminación

Supuestos

- El nodo raíz siempre se encuentra en RAM
- DISK-READ() $-$ accesa y trae a RAM un nodo (bloque/página). Cualquier nodo que se pasa como parámetro debe haber hecho un DISK-READ.
- DISK-WRITE() $-$ Graba la información de un nodo en el disco.

Creación de un $B-$tree vacio

```
B-TREE-CREATE(T)
    x ← ALLOCATE-NODE()
    leaf[x] ← TRUE
    n[x] ← 0
    DISK-WRITE(x)
    root[T] ← x
```

$O(1)$ accesos a bloques

Búsqueda de una clave en un $B-$tree

Similar a la búsqueda en un árbol de búsqueda binaria, pero en lugar de elegir entre 2 vias, se debe elegir entre varias. El procedimiento retorna el nodo donde se encuentra la clave y la posición dentro del nodo o NIL en otro caso. La llamada al procedimiento debe ser `B-TREE-SEARCH(root[T], k)`.

```
B-TREE-SEARCH(x, k)
i ← 1
while i ≤ n[x] and k > key_i[x]
        do i ← i + 1
if i ≤ n[x] and k = key_i[x]
     then return (x, i)
if leaf [x]
     then return NIL
     else DISK-READ(c_i[x])
          return B-TREE-SEARCH(c_i[x], k)
```

El número de bloques accesados por `B-TREE-SEARCH()` es $O(h) = O(\log_t n)$

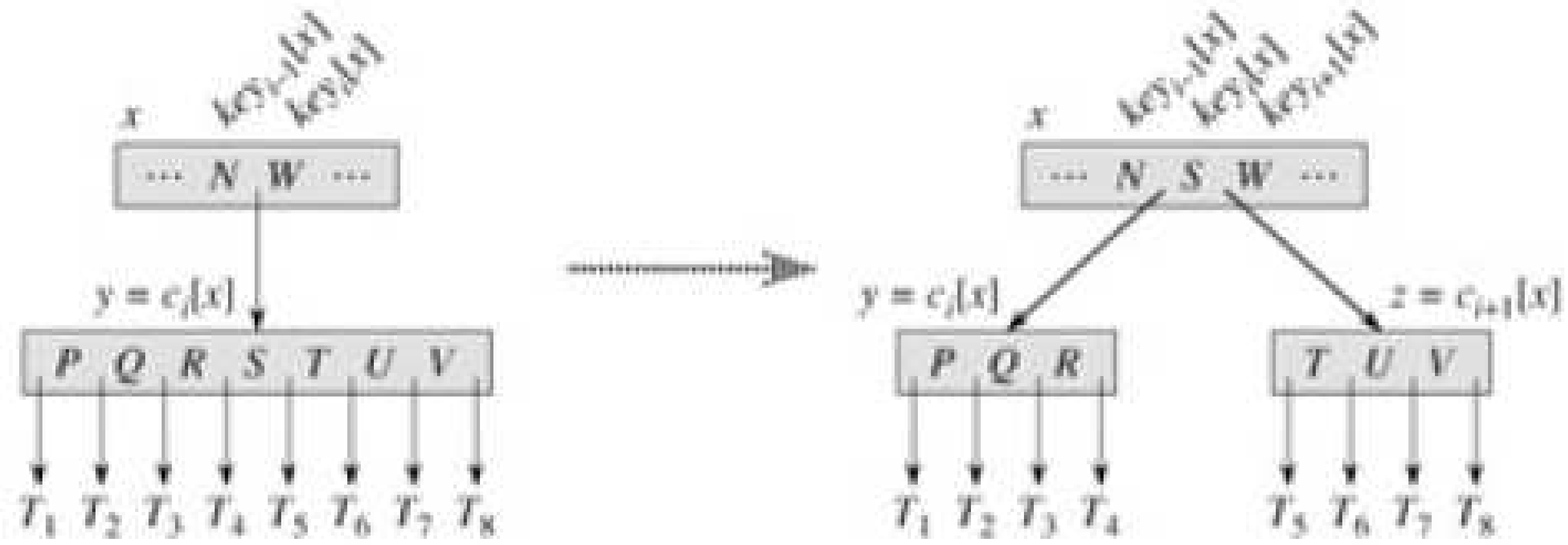
Inserción de una clave en un $B-$tree

Mucho más complicado que en un AVL, ya que al insertar una clave en un nodo este puede estar lleno y necesitamos dividirlo (split)

Split: Dividir un nod y ($2t - 1$ claves) en torno a su mediana $key_t[y]$, dejando en cada nodo $t - 1$ claves. La mediana se mueve hacia el padre de y para indicar el punto de división entre los nuevos árboles. Pero si el padre de y está lleno, entonces éste se debe dividir (split) también antes que la nueva clave sea insertada. Esto provoca que una clave se propague hacia la raíz del árbol.

Split de un nodo lleno de un $B-$tree

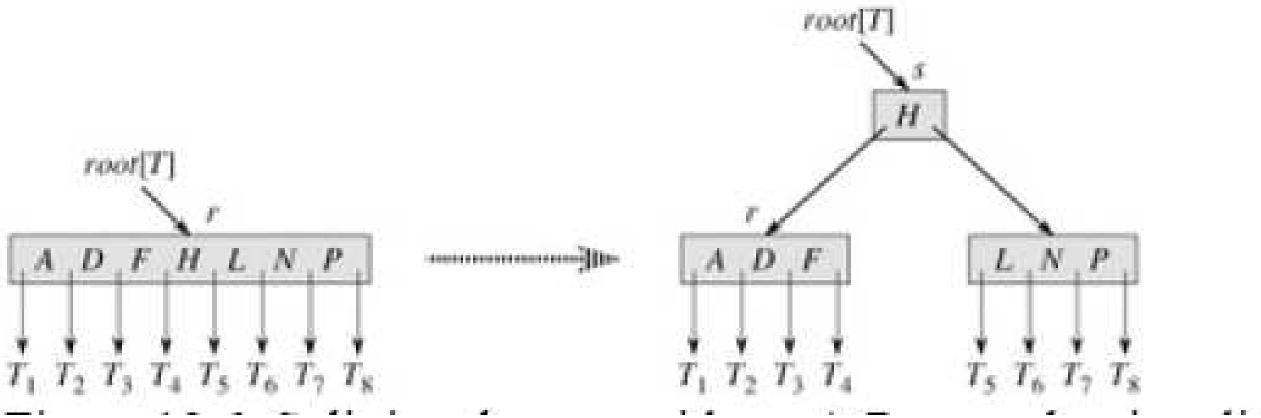
$t = 4$



Split de un nodo lleno de un $B-$tree

```
B-TREE-SPLIT-CHILD(x, i, y)
  z ← ALLOCATE-NODE()
  leaf[z] ← leaf[y]
  n[z] ← t - 1
  for j ← 1 to t - 1
        do key_j[z] ← key_{j+t}[y]
  if not leaf [y]
     then for j ← 1 to t
              do c_j[z] ← c_{j+t}[y]
  n[y] ← t - 1
  for j ← n[x] + 1 downto i + 1
        do c_{j+1}[x] ← c_j [x]
  c_{i+1}[x] ← z
  for j ← n[x] downto i
        do key_{j+1}[x] ← key_j[x]
  key_i[x] ← key_t[y]
```

Insertando una clave en un $B-$tree



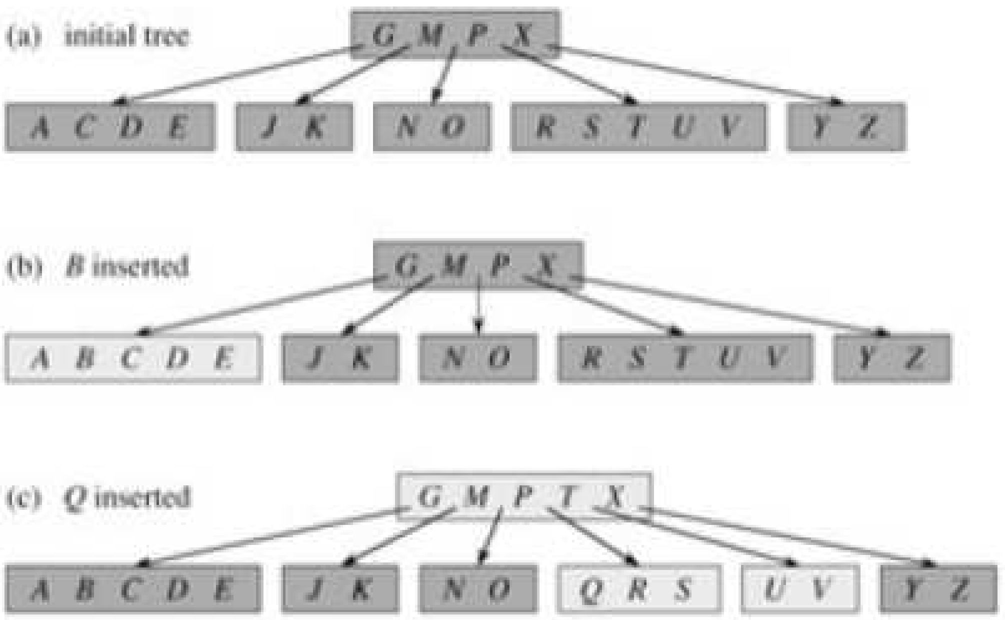
Insertando una clave en un $B-$tree

```
B-TREE-INSERT(T, k)
r ← root[T]
if n[r] = 2t - 1
   then s ← ALLOCATE-NODE()
        root[T] ← s
        leaf[s] ← FALSE
        n[s] ← 0
        c1[s] ← r
        B-TREE-SPLIT-CHILD(s, 1, r)
        B-TREE-INSERT-NONFULL(s, k)
  else B-TREE-INSERT-NONFULL(r, k)
```

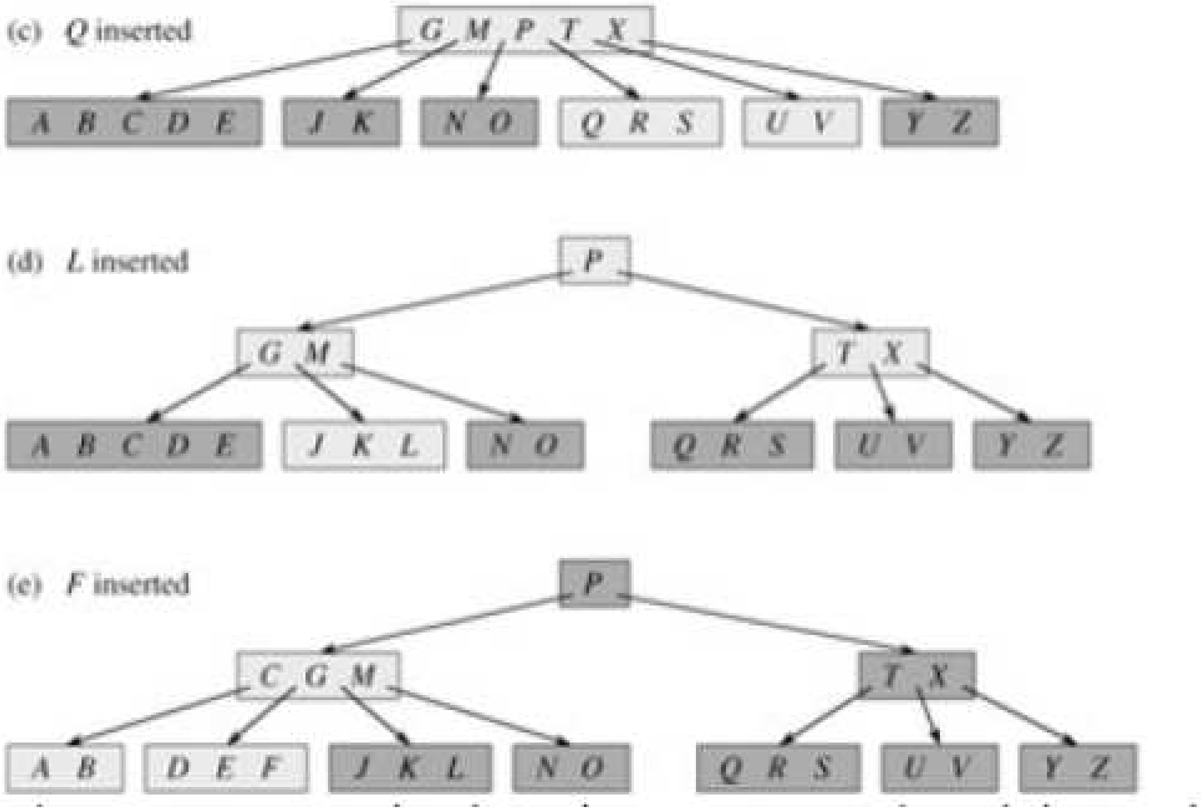
Insertando una clave en un $B-$tree

```
B-TREE-INSERT-NONFULL(x, k)
 i ← n[x]
 if leaf[x]
    then while i ≥ 1 and k < key_i[x]
             do key_{i+1}[x] ← key_i[x]
                 i ← i - 1
          key_{i+1}[x] ← k
          n[x] ← n[x] + 1
          DISK-WRITE(x)
    else while i ≥ 1 and k < key_i[x]
             do i ← i - 1
          i ← i + 1
          DISK-READ(c_i[x])
          if n[c_i[x]] = 2t - 1
            then B-TREE-SPLIT-CHILD(x, i, c_i[x])
                  if k> key_i[x]
                    then i ← i + 1
          B-TREE-INSERT-NONFULL(c_i[x], k)
```

Insertando una clave en un $B-$tree



Insertando una clave en un $B-$tree

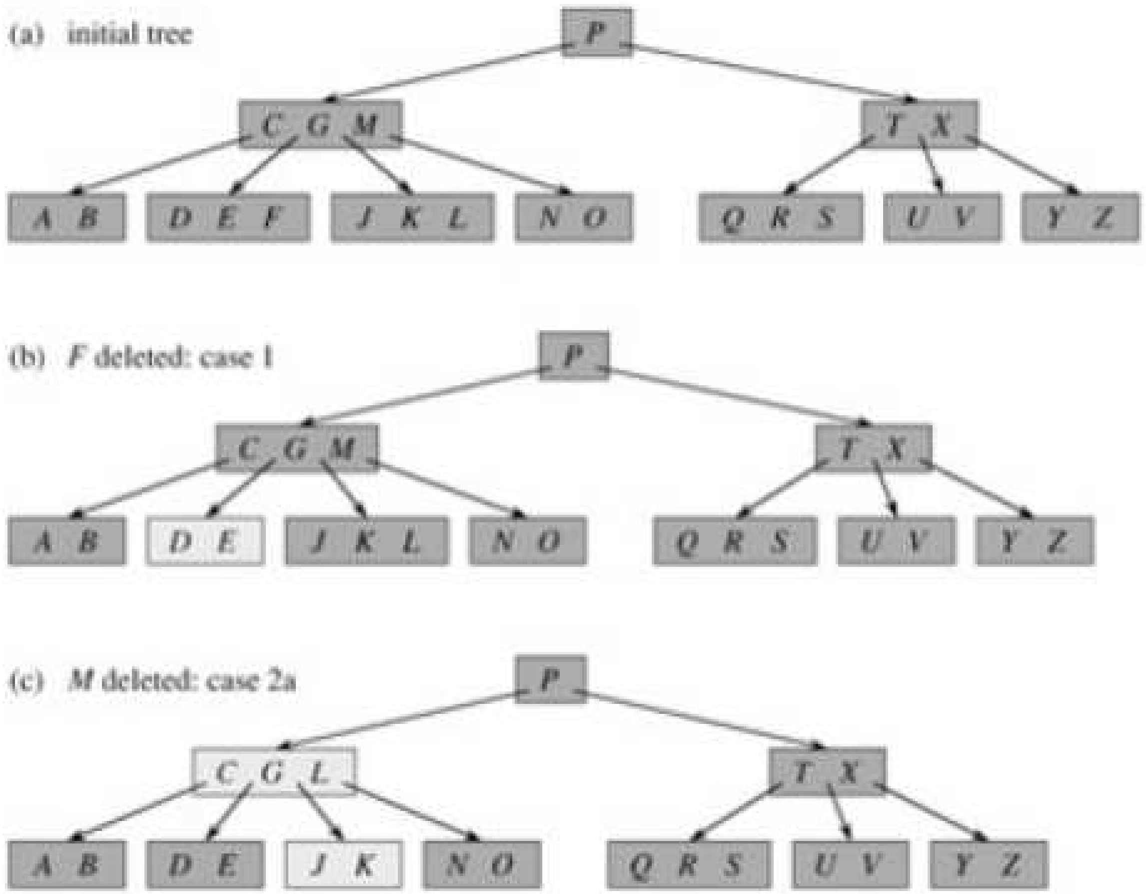


Eliminación de una clave en un $B-$tree

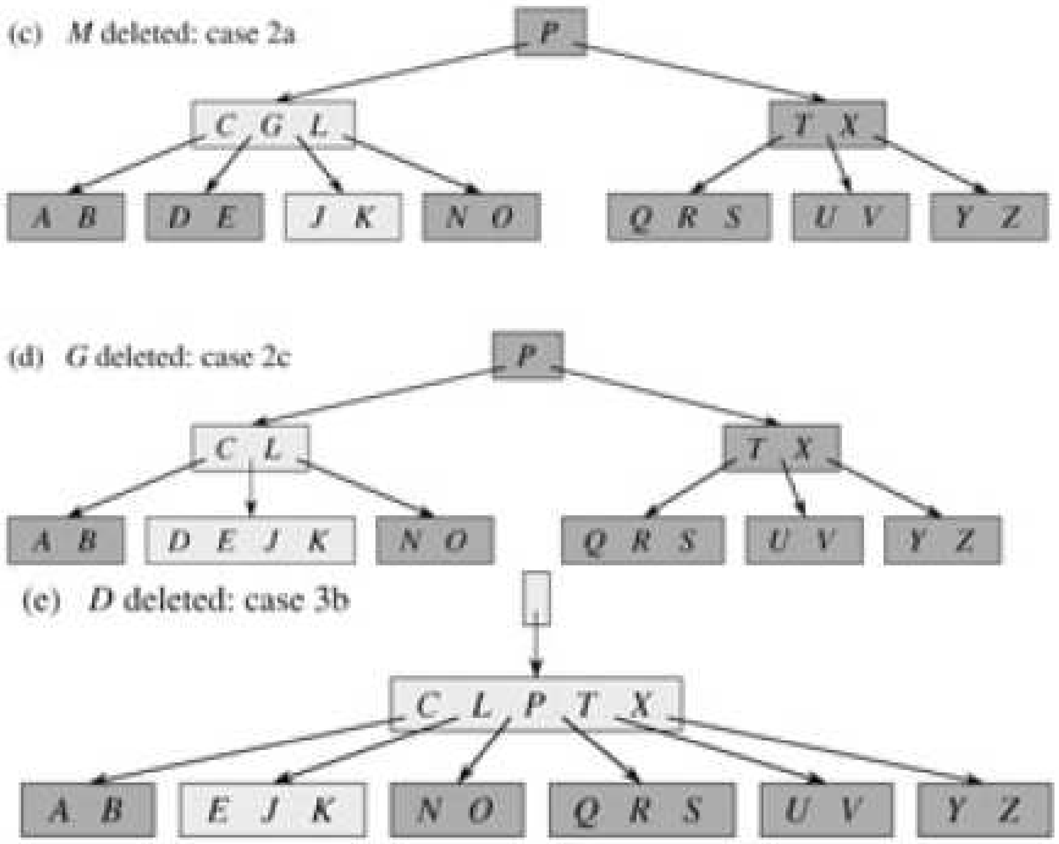
- Análoga a la inserción, pero un poco más complicado:
  - Una clave puede ser eliminada desde cualquier nodo
  - La eliminación de una clave desde un nodo interno puede requerir reordenar los nodos internos
- Al igual que la inserción, se debe garantizar que no se violen las restricciones del $B-$tree.
  - La cantidad de claves en un nodo no debe ser menor que $t - 1$.
- El procedimiento `B-TREE-DELETE` se llama para eliminar la clave k del subárbol cuya raíz es x. El procedimiento garantiza que si se llama de manera recursiva sobre un nodo x, el número de claves en x es al menos t.
- La restricción del procedimiento `B-TREE-DELETE` permite eliminar una clave en una sola pasada (desde la raíz hacia las hojas).

Eliminación de una clave en un $B-$tree

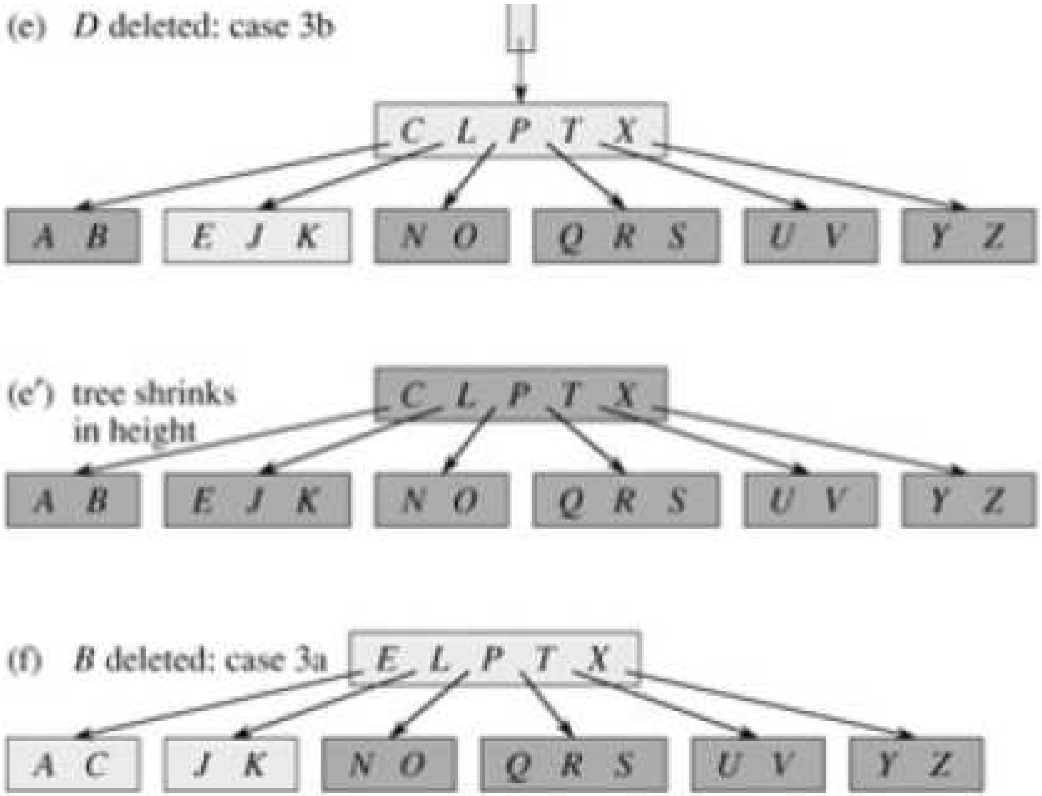
$t = 3$



Eliminación de una clave en un $B-$tree



Eliminación de una clave en un $B-$tree



Eliminación de una clave en un $B-$tree

1. Si la clave k está en el nodo x y x es una hoja, entonces eliminar la clave k de x
2. Si la clave k está en el nodo x y x es un nodo interno, hacer lo siguiente:
   (a) Si el hijo y que precede a k en el nodo x tiene al menos t claves, entonces encontrar el predecesor k' de k en el subárbol con raíz y. Recursivamente eliminar k' y reemplazar k por k' en x
   (b) Simétricamente, si el hijo z que sigue a k en el nodo x tiene al menos t claves, entonces encontrar el sucesor k' de k en el subárbol con raíz z. Recursivamente eliminar k' y reemplazar k por k' en x
   (c) En otro caso, if tanto y como z tienen sólo $t-1$ claves, mezclar k y todas las de z en y, asi que x pierde tanto la clave k como el puntero a z, e y contiene ahora $2t-1$ claves. Luego z debe liberarse y recursivamente se elimina k desde y.

Eliminación de una clave en un $B-$tree (Cont.)

3 Si la clave k no se encuentra en el nodo interno x, determinar la raíz $c_i[x]$ del subárbol apropiado que debe contener a k, si ella se encuentra en el árbol. Si $c_i[x]$ tiene sólo $t-1$ claves ejecutar los pasos $3a$ o $3b$ según sea necesario con tal de garantizar que se logra descender a un nodo que contiene al menos t claves. Luego, completar de manera recursiva sobre el hijo apropiado de x.

(a) Si $c_i[x]$ tiene solamente $t-1$ claves pero tiene un hermano con al menos t claves, dar a $c_i[x]$ una clave extra moviendo una clave desde x hacia $c_i[x]$, moviendo una clave desde el hermano izquierdo o derecho inmediato de $c_i[x]$ hacia x y moviendo el puntero al hijo apropiado desde el hermnano hacia $c_i[x]$.

(b) Si $c_i[x]$ y sus hermanos inmediato tienen $t-1$ claves, mezclar $c_i[x]$ con uno de los hermanos, y mover una clave desde x hacia el nuevo nodo mezclado transformándose en la mediana del nodo.