B-Trees — Remoção do elemento com chave k (1)

Remoção do elemento efectuada numa única passagem pela árvore

Se o nó corrente contém o elemento . . .

1. . . . e é uma folha

   Remove o elemento

2. . . . na posição i e é um nó interno

   a. se o filho i tem mais do que $t - 1$ elementos

      - substitui o elemento a remover pelo seu predecessor, que é removido da subárvore com raiz c_i

   b. senão, se o filho $i + 1$ tem mais do que $t - 1$ elementos

      - substitui o elemento a remover pelo seu sucessor, que é removido da subárvore com raiz c_{i+1}

   c. senão

      - funde os filhos i e $i + 1$
      - continua a partir do novo filho i (onde o elemento a remover foi parar)

B-Trees — Remoção do elemento com chave k (2)

Se o nó corrente não contém o elemento

3. Se o nó corrente não é folha, seja i o índice do filho que é raiz da subárvore onde o elemento poderá estar

   Se o filho i tem mais do que $t - 1$ elementos

   - continua a partir do filho i

   Se o filho i tem $t - 1$ elementos

   a. se algum dos irmãos esquerdo ou direito de i tem mais do que $t - 1$ elementos
      - transfere um elemento para o filho i, por empréstimo de um irmão nessas condições
      - continua a partir do filho i

   b. senão
      - funde o filho i com o irmão esquerdo ou direito
      - continua a partir do nó que resultou da fusão

Se a raiz ficou vazia (sem elementos) e não é folha

- o seu único filho passa a ser a nova raiz

B-TREE-DELETE(T, k)

```
1 r <- T.root
2 B-TREE-DELETE-SAFE(r, k)
3 if r.n = 0 and not r.leaf then
4     r <- r.c[1]
5     FREE-NODE(T.root)
6     T.root <- r
```

B-TREE-DELETE-SAFE(x, k)

```
 1 i <- 1
 2 while i <= x.n and k > x.key[i] do
 3     i <- i + 1
 4 if i <= x.n and k = x.key[i] then
 5     if x.leaf then                                          // Caso 1
 6         for j <- i to x.n - 1 do                            // Caso 1
 7             x.key[j] <- x.key[j + 1]                        // Caso 1
 8         x.n <- x.n - 1                                      // Caso 1
 9         DISK-WRITE(x)                                       // Caso 1
10     else
11         y <- x.c[i]
12         DISK-READ(y)
13         if y.n > t - 1 then                                 // Caso 2a
14             x.key[i] <- B-TREE-DELETE-MAX(y)                // Caso 2a
15             DISK-WRITE(x)                                   // Caso 2a
16         else
17             z <- x.c[i + 1]
18             DISK-READ(z)
19             if z.n > t - 1 then                             // Caso 2b
20                 x.key[i] <- B-TREE-DELETE-MIN(z)            // Caso 2b
21                 DISK-WRITE(x)                               // Caso 2b
22             else
23                 B-TREE-MERGE-CHILDREN(x, i)                 // Caso 2c
24                 B-TREE-DELETE-SAFE(x.c[i], k)               // Caso 2c
25 else if not x.leaf then
       ...
```

B-TREE-DELETE-SAFE(x, k) (cont.)

```
25 else if not x.leaf then
26     y <- x.c[i]
27     DISK-READ(y)
28     if y.n = t - 1 then
29         borrowed <- FALSE
30         if i > 1 then
31             z <- x.c[i - 1]
32             DISK-READ(z)
33             if z.n > t - 1 then                                  // Caso 3a
34                 B-TREE-BORROW-FROM-LEFT-SIBLING(x, i)            // Caso 3a
35                 borrowed <- TRUE                                 // Caso 3a
36             else
37                 m <- i - 1
38         if not borrowed and i <= x.n then
39             z <- x.c[i + 1]
40             DISK-READ(z)
41             if z.n > t - 1 then                                  // Caso 3a
42                 B-TREE-BORROW-FROM-RIGHT-SIBLING(x, i)           // Caso 3a
43                 borrowed <- TRUE                                 // Caso 3a
44             else
45                 m <- i
46         if not borrowed then                                     // Caso 3b
47             B-TREE-MERGE-CHILDREN(x, m)                          // Caso 3b
48             y <- x.c[m]                                          // Caso 3b
49     B-TREE-DELETE-SAFE(y, k)
```

B-TREE-MERGE-CHILDREN(x, i)

```
 1 y <- x.c[i]                          // fusão do filho i
 2 z <- x.c[i + 1]                      // com o i+1
 3 y.key[t] <- x.key[i]
 4 for j <- 1 to t - 1 do               // muda conteúdo de
 5     y.key[t + j] <- z.key[j]         // c[i+1] para c[i]
 6 if not y.leaf then
 7     for j <- 1 to t do               // incluindo filhos
 8         y.c[t + j] <- z.c[j]
 9 y.n <- 2t - 1                        // c[i] fica cheio
10 for j <- i + 1 to x.n do
11   x.key[j - 1] <- x.key[j]
12 for j <- i + 2 to x.n + 1 do
13   x.c[j - 1] <- x.c[j]
14 x.n <- x.n - 1
15 FREE-NODE(z)                    // apaga c[i+1] antigo
16 DISK-WRITE(y)
17 DISK-WRITE(x)
```

(NOTA: Os nós `x`, `x.c[i]` e `x.c[i + 1]` já foram lidos para memória)

B-TREE-BORROW-FROM-LEFT-SIBLING(x, i)

```
 1 y <- x.c[i]                          // irmão esquerdo
 2 z <- x.c[i - 1]                      // do nó i é o i-1
 3 for j <- t - 1 downto 1 do           // abre espaço para
 4     y.key[j + 1] <- y.key[j]         // a nova 1ª chave
 5 y.key[1] <- x.key[i - 1]
 6 x.key[i - 1] <- z.key[z.n]
 7 if not y.leaf then
 8     for j <- t downto 1 do           // abre espaço para
 9         y.c[j + 1] <- y.c[j]         // o novo 1º filho
10     y.c[1] <- z.c[z.n + 1]
11 y.n <- t
12 z.n <- z.n - 1
13 DISK-WRITE(z)
14 DISK-WRITE(y)
15 DISK-WRITE(x)
```

(NOTA: Os nós `x`, `x.c[i - 1]` e `x.c[i]` já foram lidos para memória)

B-TREE-BORROW-FROM-RIGHT-SIBLING(x, i)

Exercício

B-TREE-DELETE-MAX(x)

Exercício

(o nó x tem mais do que $t - 1$ elementos;
a função devolve o elemento removido)

B-TREE-DELETE-MIN(x)

Exercício

(o nó x tem mais do que $t - 1$ elementos;
a função devolve o elemento removido)

B-Trees

Resumo

Árvore com grau de ramificação mínimo t e com n elementos

Complexidade temporal das operações

pesquisa, inserção, remoção

$$O(t \log_t n)$$

Número de nós acedidos (nas operações acima)

$$O(\log_t n)$$