B-TREE-INSERT(T, k)

```
 1 r <- T.root
 2 if r.n = 2t - 1 then              // se a raiz está cheia...
 3     s <- ALLOCATE-NODE()
 4     T.root <- s                   // cria uma nova raiz...
 5     s.leaf <- FALSE
 6     s.n <- 0
 7     s.c[1] <- r                   // de que a antiga é filho
 8     B-TREE-SPLIT-CHILD(s, 1)      // e explode a antiga raiz
 9     B-TREE-INSERT-NONFULL(s, k)
10 else
11     B-TREE-INSERT-NONFULL(r, k)
```

A inserção é efectuada numa única passagem pela árvore

B-TREE-SPLIT-CHILD(x, i)

```
 1 y <- x.c[i]                              // nó a explodir (filho i)
 2 z <- ALLOCATE-NODE()                     // novo filho i+1
 3 z.leaf <- y.leaf
 4 z.n <- t - 1
 5 for j <- 1 to t - 1 do                   // transfere metade dos
 6     z.key[j] <- y.key[j + t]             // elementos para o novo nó
 7 if not y.leaf then
 8     for j <- 1 to t do                   // e metade dos filhos
 9         z.c[j] <- y.c[j + t]
10 y.n <- t - 1
11 for j <- x.n + 1 downto i + 1 do         // abre espaço em x para o
12     x.c[j + 1] <- x.c[j]                 // novo filho
13 x.c[i + 1] <- z
14 for j <- x.n downto i do                 // abre espaço para o
15     x.key[j + 1] <- x.key[j]             // elemento a promover
16 x.key[i] <- y.key[t]
17 x.n <- x.n + 1
18 DISK-WRITE(y)
19 DISK-WRITE(z)
20 DISK-WRITE(x)
```

B-TREE-INSERT-NONFULL(x, k)

```
 1 i <- x.n
 2 if x.leaf then  // se está numa folha, insere o elemento
 3     while i >= 1 and k < x.key[i] do
 4         x.key[i + 1] <- x.key[i]
 5         i <- i - 1
 6     x.key[i + 1] <- k
 7     x.n <- x.n + 1
 8     DISK-WRITE(x)
 9 else              // senão, desce para o filho apropriado
10     while i >= 1 and k < x.key[i] do
11         i <- i - 1
12     i <- i + 1
13     DISK-READ(x.c[i])
14     if x.c[i].n = 2t - 1 then     // o filho está cheio?
15         B-TREE-SPLIT-CHILD(x, i)
16         if k > x.key[i] then
17             i <- i + 1
18     B-TREE-INSERT-NONFULL(x.c[i], k)
```

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 na posição i . . .

1. . . . e é uma folha

   Remove o elemento

2. . . . 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 agora está o elemento a remover)

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, terminada a remoção, a raiz fica vazia e não é folha

- o seu (único) filho passa a ser a nova raiz e a altura diminui

B-TREE-DELETE(T, k)

```
1 r <- T.root
2 B-TREE-DELETE-SAFE(r, k)          // remove o elemento
3 if r.n = 0 and not r.leaf then    // raiz vazia com filho?
4     r <- r.c[1]
5     FREE-NODE(T.root)
6     T.root <- r                   // o filho é a nova raiz
```

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
 6         B-TREE-DELETE-FROM-LEAF(x, i)                  // Caso 1
 7     else
 8         B-TREE-DELETE-FROM-INTERNAL-NODE(x, i)         // Caso 2
 9 else if not x.leaf then
10     B-TREE-DELETE-FROM-SUBTREE(x, i)                   // Caso 3
```

B-TREE-DELETE-FROM-LEAF(x, i)

```
1 for j <- i to x.n - 1 do                                // Caso 1
2     x.key[j] <- x.key[j + 1]                            // Caso 1
3 x.n <- x.n - 1                                          // Caso 1
4 DISK-WRITE(x)                                           // Caso 1
```

B-TREE-DELETE-FROM-INTERNAL-NODE(x, i)

```
 1 y <- x.c[i]
 2 DISK-READ(y)
 3 if y.n > t - 1 then                                  // Caso 2a
 4     x.key[i] <- B-TREE-DELETE-MAX(y)                 // Caso 2a
 5     DISK-WRITE(x)                                    // Caso 2a
 6 else
 7     z <- x.c[i + 1]
 8     DISK-READ(z)
 9     if z.n > t - 1 then                              // Caso 2b
10         x.key[i] <- B-TREE-DELETE-MIN(z)             // Caso 2b
11         DISK-WRITE(x)                                // Caso 2b
12     else
13         B-TREE-MERGE-CHILDREN(x, i)                  // Caso 2c
14         B-TREE-DELETE-SAFE(x.c[i], k)                // Caso 2c
```

B-TREE-DELETE-FROM-SUBTREE(x, i)

```
 1 y <- x.c[i]
 2 DISK-READ(y)
 3 if y.n = t - 1 then
 4     borrowed <- FALSE
 5     if i > 1 then
 6         z <- x.c[i - 1]
 7         DISK-READ(z)
 8         if z.n > t - 1 then                                  // Caso 3a
 9             B-TREE-BORROW-FROM-LEFT-SIBLING(x, i)            // Caso 3a
10             borrowed <- TRUE                                 // Caso 3a
11         else
12             m <- i - 1
13     if not borrowed and i <= x.n then
14         z <- x.c[i + 1]
15         DISK-READ(z)
16         if z.n > t - 1 then                                  // Caso 3a
17             B-TREE-BORROW-FROM-RIGHT-SIBLING(x, i)           // Caso 3a
18             borrowed <- TRUE                                 // Caso 3a
19         else
20             m <- i
21     if not borrowed then                                     // Caso 3b
22         B-TREE-MERGE-CHILDREN(x, m)                          // Caso 3b
23         y <- x.c[m]                                          // Caso 3b
24 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

Altura $\quad h = O(\log_t n)$

Complexidade temporal das operações

pesquisa, inserção, remoção

$$O(th) = O(t \log_t n)$$

Número de nós acedidos (nas operações acima)

$$O(h) = O(\log_t n)$$