B-Trees

Implementação

Conteúdo de um nó	(campo)
▶ ocupação	(`n`)
▶ elementos $(2t-1)$	(`key[1..2t-1]`)
▶ filhos $(2t)$	(`c[1..2t]`)
▶ é-folha?	(`leaf`)

Um nó ocupa uma, duas páginas (do disco, do sistema de ficheiros, ...)

A raiz é mantida sempre em memória

B-TREE-CREATE(T)

```
1  x <- ALLOCATE-NODE()          // cria um novo nó
2  x.leaf <- TRUE                //   sem filhos
3  x.n <- 0                      //   nem elementos
4  DISK-WRITE(x)                 // e guarda-o em disco
5  T.root <- x                   // este nó é a raiz da
                                 // nova B-tree
```

(*Introduction to Algorithms*, Cormen et al.)

B-TREE-SEARCH(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      return (x, i)
6  if x.leaf then
7      return NIL
8  DISK-READ(x.c[i])
9  return B-TREE-SEARCH(x.c[i], k)
```

Pesquisa (recursiva) do elemento com chave k no nó x

x já está em memória quando a função é chamada

B-Trees

Comportamento da pesquisa

Altura de uma árvore com n elementos

$$h \leq \log_t \frac{n+1}{2} = O(\log_t n)$$

Número de nós acedidos no pior caso

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

Complexidade temporal da pesquisa no pior caso

$$O(t \log_t n)$$

Altura de uma árvore

Elementos	abp	*B-tree*			
		$t = 32$		$t = 64$	
	mínima	mínima	máxima	mínima	máxima
10^6	19	3	3	2	3
10^9	29	4	5	4	4
10^{12}	39	6	7	5	6

B-TREE-INSERT(T, k)

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

Inserção efectuada numa única passagem pela árvore

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

```
 1 y <- x.c[i]                              // nó a dividir (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 elementos para
 6     z.key[j] <- y.key[j + t]             // o novo nó
 7 if not y.leaf then
 8     for j <- 1 to t do                   // transfere 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 para o novo
12     x.c[j + 1] <- x.c[j]                //   filho de x
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)
```