Informação persistente (1)

Enquadramento

- Estruturas de dados em memória central desaparecem quando programa termina
- Volume dos dados pode não permitir
  - o armazenamento em memória central
  - o seu processamento sempre que é necessário aceder-lhes
- Dados persistentes, em memória secundária, requerem estruturas de dados persistentes

Condicionantes

- Acessos a memória secundária (10^{-3} s) muito mais caros que acessos à memória central (10^{-9} s)
- Transferências entre a memória central e a memória secundária processadas por páginas (4096 *bytes* é uma dimensão comum)

Informação persistente (2)

Dados em memória secundária

Estratégia

Minimizar o número de acessos a memória secundária

- Adaptando as estruturas de dados
- Usando estruturas de dados especialmente concebidas

  Em ambos os casos, procura-se tirar o maior partido possível do conteúdo das páginas acedidas

- Fazendo *cacheing* da informação

Cuidados

Garantir que a informação em memória secundária se mantém actualizada

- Operações só ficam completas quando as alterações são escritas na memória secundária

B-Trees

B-Trees

Objectivos

Grandes quantidades de informação

Armazenamento em memória secundária

Indexação eficiente

Minimização de acessos a memória secundária

B-Trees

Características (1)

São árvores

Princípios semelhantes aos das árvores binárias de pesquisa

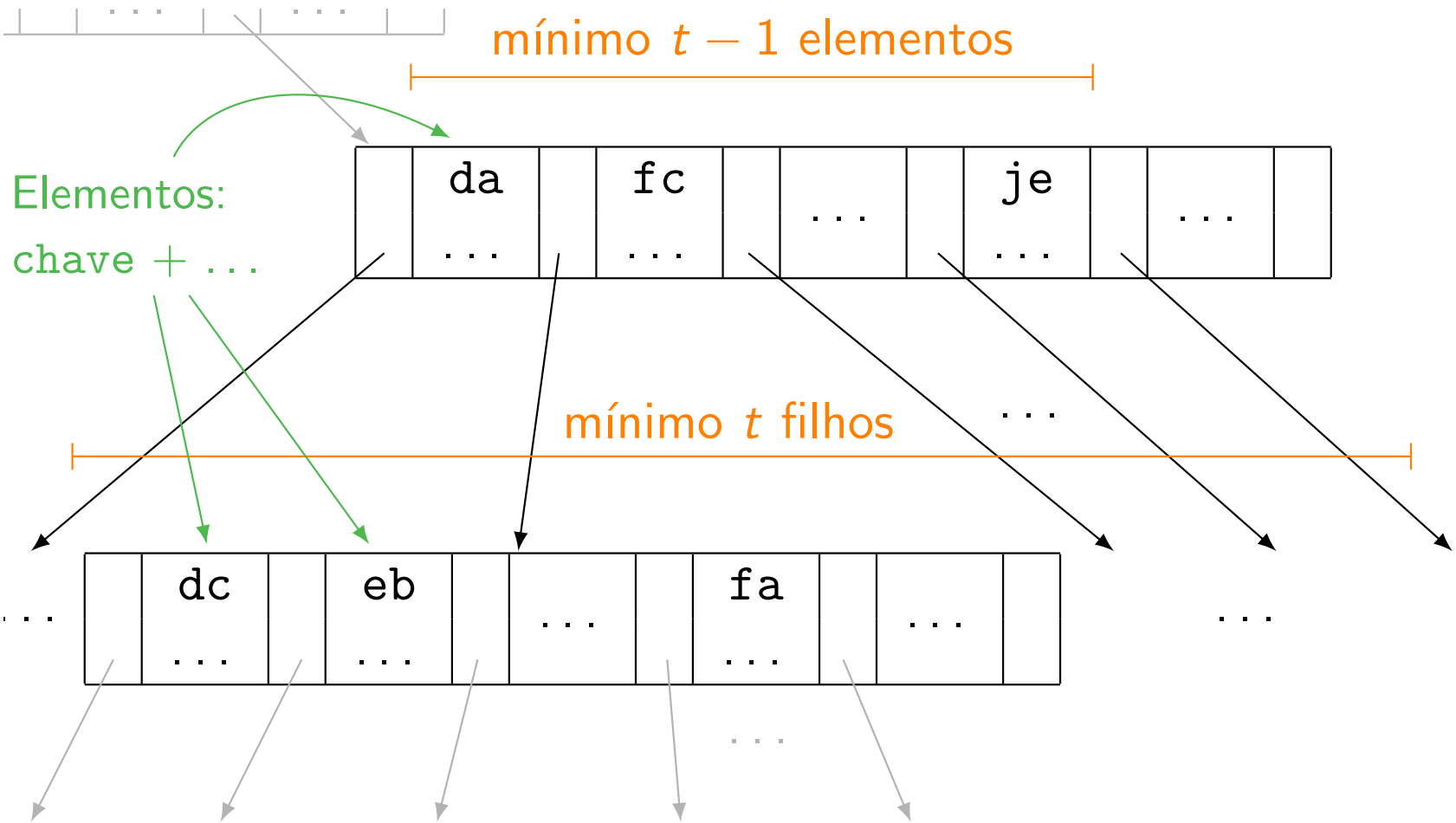
Perfeitamente equilibradas

Nós com número variável de filhos (pelo menos 2)

Nós com número variável de elementos

B-Trees

Estrutura dos nós internos (exceptuando a raiz)



B-Trees

Características (2)

Os nós internos das *B-trees* (excepto a raiz) têm, pelo menos, $t \geq 2$ filhos

t é o grau (de ramificação) mínimo de uma *B-tree*

A ordem de uma *B-tree* é $m = 2t$

Cada nó tem capacidade para $2t - 1$ elementos

Ocupação de um nó (excepto a raiz)

- entre $t - 1$ e $2t - 1$ elementos
- entre t e $2t$ filhos (excepto as folhas)

Ocupação da raiz (de uma *B-tree* não vazia)

- entre 1 e $2t - 1$ elementos
- entre 2 e $2t$ filhos (excepto se for folha)

B-Trees

Características (3)

Um nó interno com e elementos tem $e + 1$ filhos

Em todos os nós, verifica-se:

$$chave(\text{elemento}_1) \leq chave(\text{elemento}_2) \leq \ldots \leq chave(\text{elemento}_e)$$

Em todos os nós internos, verifica-se:

$$chaves(\text{filho}_1) \leq chave(\text{elemento}_1) \leq chaves(\text{filho}_2) \leq$$
$$\leq chave(\text{elemento}_2) \leq \ldots \leq chave(\text{elemento}_e) \leq chaves(\text{filho}_{e+1})$$

Todas as folhas estão à mesma profundidade

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, ...)

O valor de t depende do espaço ocupado pelos elementos e da dimensão pretendida para um nó

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 na subárvore cuja raiz é o nó x

Assume que x já está em memória quando a função é chamada

Altura máxima de uma *B-tree*

Nível	Número mínimo de nós	Número mínimo de elementos
0	1	1
1	2	$2\,(t-1)$
2	$2t$	$2t\,(t-1)$
3	$2t^2$	$2t^2(t-1)$
4	$2t^3$	$2t^3\,(t-1)$
	$\vdots$	
h	$2t^{h-1}$	$2t^{h-1}\,(t-1)$

Número de elementos de uma árvore com altura h

$$n \geq 1+\sum_{i=0}^{h-1} 2t^i(t-1) = 1+2(t-1)\frac{1-t^h}{1-t} = 1-2(1-t^h) = 2t^h-1$$

Altura de uma árvore com n elementos

$$n \geq 2t^h - 1 \quad \equiv \quad t^h \leq \frac{n+1}{2} \quad \equiv \quad h \leq \log_t \frac{n+1}{2}$$

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

$$O(t \log_t n)$$

Alturas de árvores

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