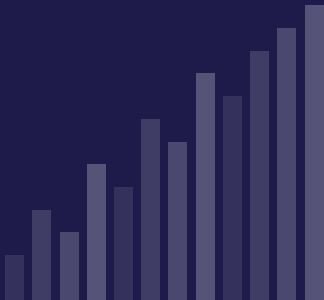

AULA 26 · BLOCO 3: TABELAS HASH E ÁRVORES

Árvores B e memória secundária

Capítulo 18 (seções 18.1 e 18.2)

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Explicar por que o acesso ao disco muda o modelo de custo
- ▶ Definir árvore B de grau mínimo t e limitar sua altura
- ▶ Inserir com divisão preventiva de nós cheios
- ▶ Medir a altura para $t = 2, 16$ e 128 e conferir $\log_t \frac{n+1}{2}$



Teoria

Discos e páginas, definição, altura, busca, divisão e inserção



Prática em C

Árvore B em C com divisão preventiva



Experimento

Altura para três graus mínimos, até 4 milhões de chaves



Conexões

Bancos de dados, sistemas de arquivos, armazenamento e SSDs

Quando o dado está no disco

- ▶ Ler uma página do disco (ou de um SSD) custa milhares de vezes mais que acessar a RAM.
- ▶ O custo que importa passa a ser o **número de páginas lidas**, não o de comparações.
- ▶ Ideia: nós **gordos**, do tamanho de uma página, com centenas de chaves. A árvore fica baixíssima.
- ▶ Uma página de 4 KB com chaves de 8 bytes guarda cerca de 500 chaves por nó.

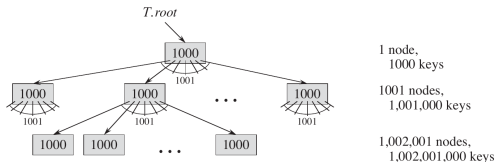


Figura 18.3 de Cormen et al. (2009) — árvore B de altura 2 com 1000 chaves por nó: mais de um bilhão de chaves

Definição

Árvore B de **grau mínimo** $t \geq 2$:

- ▶ Cada nó tem de $t - 1$ a $2t - 1$ chaves (a raiz pode ter menos), em ordem.
- ▶ Um nó interno com n chaves tem $n + 1$ filhos; as chaves separam as faixas dos filhos.
- ▶ Todas as folhas estão na **mesma profundidade**.

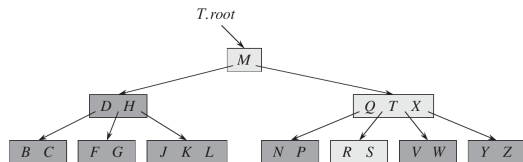
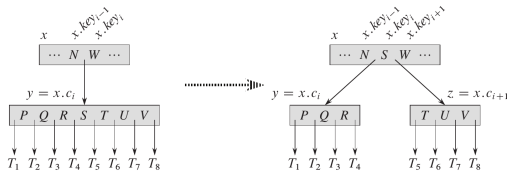


Figura 18.1 de Cormen et al. (2009) — árvore B com as consoantes do alfabeto: um nó com n chaves tem $n + 1$ filhos.

Teorema 18.1: $h \leq \log_t \frac{n + 1}{2}$.

Inserção com divisão preventiva

- ▶ A chave nova sempre entra numa **folha**.
- ▶ Na descida, todo nó **cheio** ($2t - 1$ chaves) é dividido **antes** de entrar nele: a chave do meio sobe. Assim nunca é preciso voltar.
- ▶ Se a raiz está cheia, ela é dividida e a árvore **cresce para cima** (Figura 18.6): única forma de aumentar a altura.
- ▶ $O(h)$ acessos ao disco e $O(th) = O(t \log_t n)$ de CPU.





Prática em C

- ▶ Nó com vetores de até $2t - 1$ chaves e $2t$ filhos
- ▶ B-Tree-SPLIT-CHILD e B-Tree-INSERT-Nonfull
- ▶ Teste: as letras do exercício 18.2-1 com $t = 2$

Árvore B em C (1/6)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Grau mínimo t: todo nó (menos a raiz) tem de t-1 a 2t-1 chaves.
5     Num banco de dados, cada nó é uma página do disco; aqui fica na memória. */
6  typedef struct No {
7     int n, folha;
8     char *chave;           // até 2t-1 chaves
9     struct No **filho;     // até 2t filhos
10 } No;
11
12 int t;
```

Árvore B em C (2/6)

```
14 No *novo(int folha) {
15     No *x = malloc(sizeof(No));
16     x->n = 0;
17     x->folha = folha;
18     x->chave = malloc(2 * t - 1);
19     x->filho = malloc(2 * t * sizeof(No *));
20     return x;
21 }
22
23 /* B-TREE-SPLIT-CHILD: o filho i de x está cheio; a chave do meio sobe para x. */
24 void divide(No *x, int i) {
25     No *y = x->filho[i], *z = novo(y->folha);
26     z->n = t - 1;
27     for (int j = 0; j < t - 1; j++) z->chave[j] = y->chave[j + t];
28     if (!y->folha)
29         for (int j = 0; j < t; j++) z->filho[j] = y->filho[j + t];
```

Árvore B em C (3/6)

```
30     y->n = t - 1;
31     for (int j = x->n; j > i; j--) x->filho[j + 1] = x->filho[j];
32     x->filho[i + 1] = z;
33     for (int j = x->n - 1; j >= i; j--) x->chave[j + 1] = x->chave[j];
34     x->chave[i] = y->chave[t - 1];
35     x->n++;
36 }
37
38 void insere_nao_cheio(No *x, char k) {           // B-TREE-INSERT-NONFULL
39     int i = x->n - 1;
40     if (x->folha) {
41         while (i >= 0 && k < x->chave[i]) { x->chave[i + 1] = x->chave[i]; i--; }
42         x->chave[i + 1] = k;
43         x->n++;
44         return;
    }
```

Árvore B em C (4/6)

```
45     }
46     while (i >= 0 && k < x->chave[i]) i--;
47     i++;
48     if (x->filho[i]->n == 2 * t - 1) {           // divide antes de descer: nunca volta
49         divide(x, i);
50         if (k > x->chave[i]) i++;
51     }
52     insere_nao_cheio(x->filho[i], k);
53 }
54
55 // B-TREE-INSERT: a árvore cresce pela raiz
56 No *insere(No *raiz, char k) {
57     if (raiz->n == 2 * t - 1) {
58         No *s = novo(0);
59         s->filho[0] = raiz;
```

Árvore B em C (5/6)

```
60         divide(s, 0);
61         insere_ao_cheio(s, k);
62         return s;
63     }
64     insere_ao_cheio(raiz, k);
65     return raiz;
66 }
67
68 void libera(No *x) {
69     if (!x->folha)
70         for (int i = 0; i <= x->n; i++) libera(x->filho[i]);
71     free(x->chave); free(x->filho); free(x);
72 }
```

Árvore B em C (6/6)

```
74 int main(void) {
75     char s[16];
76     if (scanf("%d", &t) != 1 || t < 2) return 1;
77     No *raiz = novo(1);
78     while (scanf("%15s", s) == 1)                // letras ou números pequenos
79         raiz = insere(raiz, s[0] > '9' ? s[0] : (char)atoi(s));
80     int altura = 0;
81     for (No *x = raiz; !x->folha; x = x->filho[0]) altura++;
82     printf("altura %d\nraiz:", altura);
83     for (int i = 0; i < raiz->n; i++)
84         printf(" %c", raiz->chave[i] > ' ' ? raiz->chave[i] : '0' + raiz->chave[i]);
85     printf("\n");
86     libera(raiz);
87     return 0;
88 }
```



Experimento

- ▶ Teorema 18.1: $h \leq \log_t \frac{n+1}{2}$
- ▶ $t = 2$ (árvore 2-3-4), 16 e 128
- ▶ n de 2^{10} a 2^{22} chaves aleatórias

Quanto o grau achata a árvore

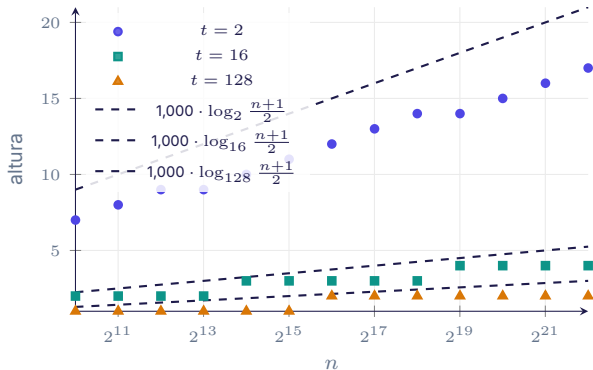
Hipótese. A altura nunca passa de $\log_t \frac{n+1}{2}$. Com $t = 128$, cada nó tem de 127 a 255 chaves: quatro milhões de chaves devem caber em **3 níveis** (2 acessos abaixo da raiz). Com $t = 2$, a árvore B é uma árvore 2-3-4 (equivalente a uma rubro-negra): altura perto de $\lg n$.

Método

1. Inserção com divisão preventiva (como no livro)
2. $t \in \{2, 16, 128\}$; mesmas chaves para os três
3. $n = 2^{10}, \dots, 2^{22}$
4. Altura medida descendo pelo primeiro filho

EXPERIMENTO

Altura para três graus mínimos



n	medido	razão
1.024	7,000	0,778
4.096	9,000	0,818
16.384	10,0	0,769
65.536	12,0	0,800
262.144	14,0	0,824
4.194.304	17,0	0,810

razão = medido $\div \log_2 \frac{n+1}{2}$

✓ sempre $\leq$ o limite

Quatro milhões de chaves: altura 17 com $t = 2$, 4 com $t = 16$ e só 2 com $t = 128$.

EXPERIMENTO

O que o experimento mostrou

A altura respeitou o teorema 18.1 em todos os casos. Com $t = 128$, quatro milhões de chaves ficam em **3 níveis**: uma busca lê 3 páginas do disco, contra 22 de uma árvore binária balanceada. É por isso que **todo** banco de dados relacional indexa com árvores B (na verdade, B+).

➤ Reproduza: `gcc -O2 aula26_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Bancos de dados
- ▶ Sistemas de arquivos
- ▶ Armazenamento chave-valor
- ▶ SSDs e cache

Onde isso aparece na computação

 Bancos de dados

PostgreSQL, MySQL (InnoDB), SQLite e Oracle indexam tabelas com árvores B+: as chaves nas folhas, ligadas em lista para consultas por intervalo.

 Sistemas de arquivos

NTFS, HFS+, APFS, Btrfs e XFS guardam diretórios e metadados em árvores B: abrir um arquivo entre milhões custa poucas leituras.

 Armazenamento chave-valor

LMDB e BoltDB são árvores B mapeadas em memória; MongoDB (WiredTiger) e CouchDB também usam árvores B.

 SSDs e cache

Em SSDs, escrever custa mais que ler: variantes como B⁺-trees e LSM-trees (aula 4) trocam leituras por menos escritas.

Para casa

Exercícios

1. (18.1-1) Por que não permitir $t = 1$?
2. (18.1-4) Qual o número máximo de chaves numa árvore B de altura h , em função de t ?
3. (18.2-1) Mostre as árvores após inserir F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E com $t = 2$.
4. (18.2-6) Com busca binária dentro de cada nó, mostre que o custo de CPU é $O(\lg n)$, independente de t .
5. Conte no experimento os nós visitados por busca e simule 10 ms por nó lido do disco. Qual t é melhor?

Leitura

Cormen et al., seções 18.1 e 18.2 (remoção: 18.3).

Código

josedihago.net/estruturas