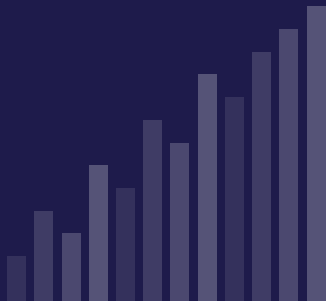

AULA 22 · BLOCO 3: TABELAS HASH E ÁRVORES

Árvores binárias de busca: percursos e consultas

Seções 12.1 e 12.2

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Enunciar a propriedade de árvore binária de busca
- ▶ Percorrer em ordem, pré-ordem e pós-ordem
- ▶ Buscar, achar mínimo, máximo, sucessor e predecessor em $O(h)$
- ▶ Medir o custo médio de busca em árvores aleatórias: $\approx 1,39 \lg n$



Teoria

Propriedade, percursos, consultas e o papel da altura



Prática em C

ABB em C: inserção, percurso, busca, mínimo, máximo e sucessor



Experimento

Custo médio de busca contra $2(1 + 1/n)H_n - 3$

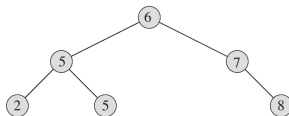


Conexões

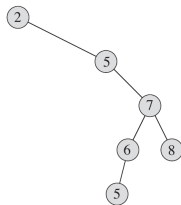
Sistemas de arquivos, compiladores, bancos de dados e geometria

A propriedade de árvore binária de busca

- ▶ Cada nó x tem *chave*, *esq*, *dir* e *pai*.
- ▶ **Propriedade:** se y está na subárvore esquerda de x , $y.chave \leq x.chave$; se está na direita, $y.chave \geq x.chave$.
- ▶ As operações custam $O(h)$, onde h é a **altura**.
- ▶ h vai de $\lceil \lg n \rceil$ (árvore cheia) a $n - 1$ (uma corrente).



(a)



(b)

Figura 12.1 de Cormen et al. (2009) — duas ABBs com as mesmas chaves: (a) altura 2; (b) altura 4, menos eficiente.

Percurso em ordem

- ▶ Imprime as chaves **em ordem**: consequência direta da propriedade.
- ▶ **Teorema 12.1:** $\Theta(n)$ para n nós.
- ▶ Pré-ordem: raiz antes das subárvores (copiar a árvore).
- ▶ Pós-ordem: raiz depois (liberar a memória, avaliar expressões).

INORDER-TREE-WALK(x)

- 1 **se** $x \neq \text{NIL}$ **então**
- 2 INORDER-TREE-WALK($x.esq$)
- 3 imprima $x.chave$
- 4 INORDER-TREE-WALK($x.dir$)

Busca, mínimo, máximo e sucessor

- ▶ Tree-SEARCH: desce à esquerda ou à direita conforme a chave: $O(h)$.
- ▶ Tree-MINIMUM: sempre à esquerda.
Tree-MAXIMUM: sempre à direita.
- ▶ Tree-Successor(x): se há subárvore direita, o mínimo dela; senão, suba até chegar por um filho esquerdo.
- ▶ **Teorema 12.2:** todas $O(h)$.

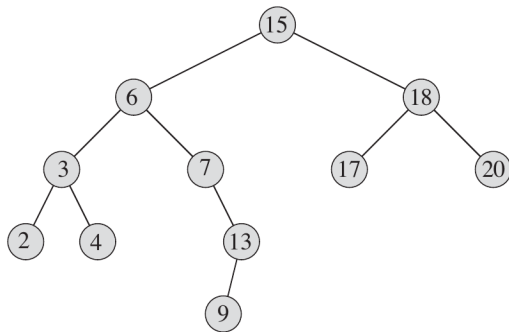


Figura 12.2 de Cormen et al. (2009) — busca por 13:
 $15 \rightarrow 6 \rightarrow 7 \rightarrow 13$. Mínimo: sempre à esquerda (2).



Prática em C

- ▶ Nós com ponteiro para o pai (necessário para o sucessor)
- ▶ Percurso em ordem recursivo; busca iterativa
- ▶ Teste com as chaves da Figura 12.2

Árvore binária de busca em C (1/4)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct No { int chave; struct No *esq, *dir, *pai; } No;
5
6 No *insere(No *raiz, int k) { // TREE-INSERT (detalhado na aula 23)
7     No *z = calloc(1, sizeof(No)), *y = NULL, *x = raiz;
8     z->chave = k;
9     while (x) { y = x; x = k < x->chave ? x->esq : x->dir; }
10    z->pai = y;
11    if (!y) return z;
12    if (k < y->chave) y->esq = z; else y->dir = z;
13    return raiz;
14 }
```

Árvore binária de busca em C (2/4)

```
15
16 // INORDER-TREE-WALK:  $\Theta(n)$ , chaves em ordem
17 void em_ordem(No *x) {
18     if (!x) return;
19     em_ordem(x->esq);
20     printf(" %d", x->chave);
21     em_ordem(x->dir);
22 }
23
24 No *busca(No *x, int k) { // ITERATIVE-TREE-SEARCH:  $O(h)$ 
25     while (x && k != x->chave) x = k < x->chave ? x->esq : x->dir;
26     return x;
27 }
```

Árvore binária de busca em C (3/4)

```
29 0
30 No *minimo(No *x) { while (x->esq) x = x->esq; return x; }
31 No *maximo(No *x) { while (x->dir) x = x->dir; return x; }
32
33 No *sucessor(No *x) {                                     // TREE-SUCCESSOR: O(h)
34     if (x->dir) return minimo(x->dir);                     // caso 1: menor da subárvore direita
35     No *y = x->pai;                                         // caso 2: sobe até virar filho esquerdo
36     while (y && x == y->dir) { x = y; y = y->pai; }
37     return y;
38 }
39
40 void libera(No *x) { if (x) { libera(x->esq); libera(x->dir); free(x); } }
```

Árvore binária de busca em C (4/4)

```
41 0
42 int main(void) {
43     No *raiz = NULL;
44     int k, ultimo = 0;
45     while (scanf("%d", &k) == 1) { raiz = insere(raiz, k); ultimo = k; }
46     if (!raiz) return 1;
47     printf("em ordem:");
48     em_ordem(raiz);
49     printf("\nmínimo %d, máximo %d\n", minimo(raiz)->chave, maximo(raiz)->chave);
50     No *x = busca(raiz, 13), *s = x ? sucessor(x) : NULL;
51     if (s) printf("sucessor de 13: %d\n", s->chave);
52     printf("última chave lida (%d) %s\n", ultimo,
53           busca(raiz, ultimo) ? "encontrada" : "ausente");
54     libera(raiz);
55     return 0;
56 }
```



Experimento

- ▶ Hipótese: $2(1 + 1/n)H_n - 3 \approx 1,39 \lg n$
- ▶ $n = 2^{10}$ a 2^{21} , média de várias árvores
- ▶ Custo de buscar uma chave = sua profundidade + 1

Quanto custa buscar numa árvore aleatória?

Hipótese. Inserir chaves em ordem aleatória equivale a montar a árvore de recursão do quicksort: cada chave é comparada com seus ancestrais. O custo médio de uma busca com sucesso é

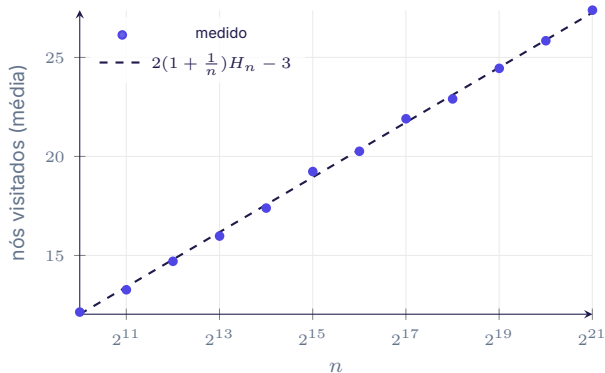
$$2\left(1 + \frac{1}{n}\right)H_n - 3 \approx 2 \ln n \approx 1,39 \lg n.$$

Método

1. Chaves de 32 bits sorteadas (xorshift)
2. Profundidade de cada nó registrada na inserção
3. $n = 2^{10}, \dots, 2^{21}$
4. Média de 10 árvores (3 para $n > 2^{17}$)

🧪 EXPERIMENTO

Nós visitados por busca



n	medido	razão
1.024	12,1	1,009
4.096	14,7	0,994
16.384	17,4	0,990
65.536	20,3	0,996
262.144	22,9	0,991
2.097.152	27,4	1,004

razão = medido $\div$ $2(1 + \frac{1}{n})H_n - 3$

✓ desvio máx. 0,9%

Com n em escala logarítmica, o custo é uma reta: cresce como $\lg n$, cerca de 1,39 nó a cada vez que n dobra.

EXPERIMENTO

O que o experimento mostrou

O custo médio segue $2(1 + 1/n)H_n - 3$ com desvio abaixo de 1%: para dois milhões de chaves, 27 nós por busca. Uma árvore aleatória é só 39% mais funda (em média) que uma perfeitamente balanceada. O perigo está nas entradas **não** aleatórias (próxima aula).

➤ Reproduza: `gcc -O2 aula22_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Sistemas de arquivos
- ▶ Compiladores
- ▶ Bancos de dados
- ▶ Geometria computacional

Onde isso aparece na computação

Sistemas de arquivos

Diretórios são árvores; ext4 e Btrfs indexam nomes em árvores de busca para achar um arquivo entre milhões sem varrer todos.

Compiladores

Árvores sintáticas são percorridas em pós-ordem para gerar código e avaliar expressões; o percurso em ordem imprime a expressão infixa.

Bancos de dados

Consultas por intervalo (BETWEEN) usam o sucessor: acha-se o início e anda-se em ordem. Hash não faz isso; árvores fazem.

Geometria computacional

Árvores k-d e de intervalos generalizam a ABB para pontos no plano: "quais restaurantes estão neste retângulo do mapa?".

Para casa

Exercícios

1. (12.1-1) Desenhe ABBs de alturas 2, 3, 4, 5 e 6 com as chaves $\{1, 4, 5, 10, 16, 17, 21\}$.
2. (12.1-3) Escreva o percurso em ordem não recursivo, com uma pilha.
3. (12.2-1) Quais sequências de busca por 363 são impossíveis numa ABB? Por quê?
4. (12.2-3) Escreva Tree-Predecessor.
5. Altere o experimento para medir a busca **sem** sucesso. A fórmula muda?

Leitura

Cormen et al., seções 12.1 e 12.2.

Código

josedihego.net/estruturas