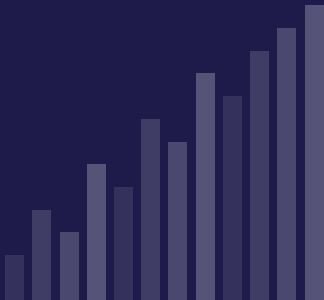

AULA 24 · BLOCO 3: TABELAS HASH E ÁRVORES

Árvores AVL

Problema 13-3 e seção 13.2 (rotações)

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Definir o fator de balanceamento AVL
- ▶ Aplicar rotações simples e duplas
- ▶ Provar que a altura de uma AVL é $O(\lg n)$ (árvores de Fibonacci)
- ▶ Medir a altura com chaves em ordem e o número de rotações por inserção



Teoria

Balanceamento, rotações, os quatro casos, altura máxima



Prática em C

AVL em C com altura em cada nó



Experimento

Altura com chaves em ordem e aleatórias; rotações por inserção



Conexões

Bancos em memória, sistemas de tempo real, linguagens e roteamento

Balanceamento de altura

- ▶ Adelson-Velsky e Landis (1962): a primeira árvore de busca autobalanceada.
- ▶ Cada nó guarda sua **altura** $h(x)$ (folha: 0; NIL: -1).
- ▶ **Propriedade AVL:** em todo nó, as alturas das subárvores esquerda e direita diferem de no máximo 1.
- ▶ Fator de balanceamento: $b(x) = h(x.esq) - h(x.dir) \in \{-1, 0, 1\}$.
- ▶ Depois de inserir, sobe-se pelo caminho atualizando alturas; onde $|b| = 2$, **rotaciona**.

Rotações

- ▶ Uma rotação troca a posição de pai e filho mexendo em $O(1)$ ponteiros.
- ▶ A propriedade de ABB se mantém.
- ▶ Uma subárvore sobe um nível, outra desce.

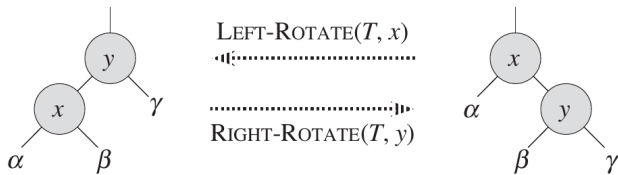


Figura 13.2 de Cormen et al. (2009) — rotação à esquerda em x e à direita em y : mudam a forma e preservam a ordem

$$\alpha < x < \beta < y < \gamma.$$

Os quatro casos

Seja x o nó mais baixo desbalanceado após inserir:

Caso	Situação	Correção
esquerda-esquerda	$b(x) = 2$, inserção na subárvore esq. do filho esq.	rotação à direita em x
direita-direita	$b(x) = -2$, na subárvore dir. do filho dir.	rotação à esquerda em x
esquerda-direita	$b(x) = 2$, na subárvore dir. do filho esq.	esquerda no filho, direita no pai
direita-esquerda	$b(x) = -2$, na subárvore esq. do filho dir.	direita no filho, esquerda no pai

Exemplos (slides antigos): inserir 7, 9, 11 (DD); 10, 6, 2 (EE); 9, 4, 7 (ED); 15, 20, 18 (DE).

Na inserção, **uma** rotação (simples ou dupla) restaura todo o caminho: a subárvore volta à altura que tinha antes.

Por que a altura é $O(\lg n)$

Seja N_h o **menor** número de nós de uma AVL de altura h (problema 13-3a). A árvore mais magra tem uma subárvore de altura $h - 1$ e outra de altura $h - 2$:

$$N_h = N_{h-1} + N_{h-2} + 1, \quad N_0 = 1, \quad N_1 = 2$$

Então $N_h = F_{h+3} - 1$ (Fibonacci), e $F_k \approx \varphi^k / \sqrt{5}$ com $\varphi = 1,618\dots$:

$$n \geq N_h \approx \varphi^h \implies h \leq \log_{\varphi} n \approx 1,44 \lg(n + 2) - 0,328$$

AVL é mais rigidamente balanceada que a rubro-negra ($1,44 \lg n$ contra $2 \lg n$): buscas um pouco mais rápidas, inserções e remoções com mais rotações.



Prática em C

- ▶ Altura guardada em cada nó; balanceia na volta da recursão
- ▶ Rotação dupla = duas rotações simples
- ▶ Testes: os quatro casos dos slides antigos

Árvore AVL em C (1/5)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Cada nó guarda a altura da sua subárvore (folha = 0; vazio = -1).
5     Balanceamento de x = altura(esq) - altura(dir), sempre em {-1, 0, 1}. */
6  typedef struct No { int chave, h; struct No *esq, *dir; } No;
7
8  int h(No *x) { return x ? x->h : -1; }
9  int maior(int a, int b) { return a > b ? a : b; }
10 void atualiza(No *x) { x->h = 1 + maior(h(x->esq), h(x->dir)); }
```

Árvore AVL em C (2/5)

```
12 // RIGHT-ROTATE: x sobe, y desce à direita
13 No *gira_direita(No *y) {
14     No *x = y->esq;
15     y->esq = x->dir;
16     x->dir = y;
17     atualiza(y);
18     atualiza(x);
19     return x;
20 }
21
22 No *gira_esquerda(No *x) { // LEFT-ROTATE: o espelho
23     No *y = x->dir;
24     x->dir = y->esq;
25     y->esq = x;
26     atualiza(x);
27     atualiza(y);
```

Árvore AVL em C (3/5)

```
28     return y;
29 }
30
31 No *balanceia(No *x) {
32     atualiza(x);
33     int b = h(x->esq) - h(x->dir);
34     if (b > 1) {                                // pesado à esquerda
35         if (h(x->esq->esq) < h(x->esq->dir))    // caso esquerda-direita: rotação dupla
36             x->esq = gira_esquerda(x->esq);
37         return gira_direita(x);
38     }
39     if (b < -1) {                                // pesado à direita
40         if (h(x->dir->dir) < h(x->dir->esq))    // caso direita-esquerda: rotação dupla
41             x->dir = gira_direita(x->dir);
```

Árvore AVL em C (4/5)

```
42     return gira_esquerda(x);
43 }
44 return x;
45 }
46
47 // insere como na ABB e rebalanceia na volta
48 No *insere(No *x, int k) {
49     if (!x) {
50         x = calloc(1, sizeof(No));
51         x->chave = k;
52         return x;
53     }
54     if (k < x->chave) x->esq = insere(x->esq, k);
55     else x->dir = insere(x->dir, k);
56     return balanceia(x);
57 }
```

Árvore AVL em C (5/5)

```
58
59 void libera(No *x) { if (x) { libera(x->esq); libera(x->dir); free(x); } }
60
61 int main(void) {
62     No *raiz = NULL;
63     int k;
64     while (scanf("%d", &k) == 1) raiz = insere(raiz, k);
65     if (!raiz) return 1;
66     printf("raiz %d, altura %d\n", raiz->chave, raiz->h);
67     libera(raiz);
68     return 0;
69 }
```



Experimento

- ▶ Chaves em ordem: a ABB comum teria altura $n - 1$
- ▶ Limite: $1,44 \lg(n + 2) - 0,328$
- ▶ Rotações por inserção, chaves aleatórias

AVL contra a ordem de chegada

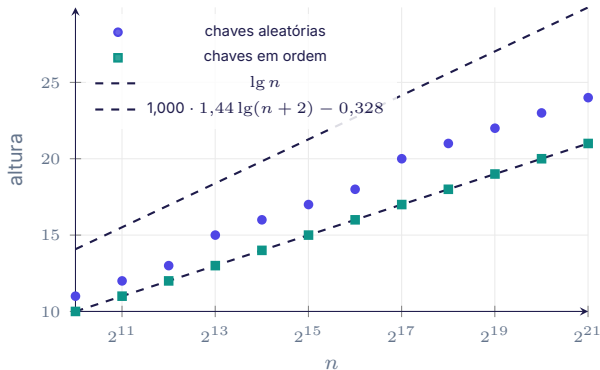
Hipótese. Com chaves em ordem, a AVL deve ficar **perfeitamente** balanceada: altura $\lg n$ (para $n = 2^k$). Com chaves aleatórias, a altura deve respeitar $1,44 \lg(n + 2) - 0,328$. E as rotações por inserção devem ser **constantes**, independentes de n .

Método

1. $n = 2^{10}, \dots, 2^{21}$
2. Verificação da propriedade AVL em todos os nós (aborta se falhar)
3. Contador de rotações simples (dupla conta 2)
4. Mesmo código para os dois tipos de entrada

🧪 EXPERIMENTO

Altura da AVL



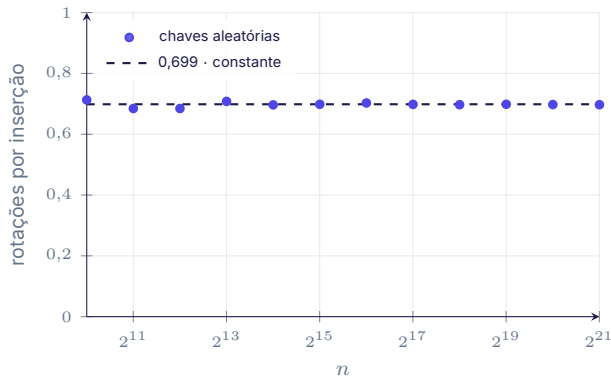
n	medido	razão
1.024	10,0	1,000
4.096	12,0	1,000
16.384	14,0	1,000
65.536	16,0	1,000
262.144	18,0	1,000
2.097.152	21,0	1,000

razão = medido $\div$ $\lg n$

✓ desvio máx. 0,0%

Em ordem: exatamente $\lg n$, uma árvore perfeita. Aleatórias: cerca de $\lg n + 3$, bem abaixo do limite.

Rotações por inserção



n	medido	razão
1.024	0,713	0,713
4.096	0,685	0,685
16.384	0,697	0,697
65.536	0,703	0,703
262.144	0,697	0,697
2.097.152	0,697	0,697

razão = medido $\div$ constante

✓ desvio máx. 0,6%

Cerca de 0,7 rotação por inserção, de mil a dois milhões de chaves: custo constante.

EXPERIMENTO

O que o experimento mostrou

Com chaves em ordem, a AVL tem altura **exatamente** $\lg n$, contra $n - 1$ da ABB comum (21 níveis contra dois milhões). Com chaves aleatórias, fica abaixo de $1,44 \lg n$, e o reequilíbrio custa só $\approx 0,7$ rotação por inserção.

>_ Reproduza: `gcc -O2 aula24_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Bancos em memória
- ▶ Tempo real
- ▶ Linguagens funcionais
- ▶ Roteamento

Onde isso aparece na computação

Bancos em memória

Bancos e caches em memória que priorizam leituras usam árvores AVL ou parecidas: a altura menor que a da rubro-negra acelera buscas.

Tempo real

Garantia de $O(\lg n)$ no pior caso, sem amortização: adequada a sistemas que não podem ter picos de latência.

Linguagens funcionais

Conjuntos e mapas imutáveis (`Data.Map` em Haskell, `Set` em OCaml) usam árvores balanceadas por tamanho ou altura, parentes da AVL.

Roteamento

Tabelas de prefixos e estruturas de intervalos em roteadores usam árvores balanceadas para manter a busca curta mesmo sob atualizações constantes.

Para casa

Exercícios

1. Insira 1, 2, ..., 15 numa AVL vazia e desenhe a árvore após cada rotação.
2. (13-3a) Prove que uma AVL com n nós tem altura $O(\lg n)$ ($N_h \geq F_h$).
3. (13-3b) Escreva $\text{BALANCE}(x)$ para $|h(x.esq) - h(x.dir)| \leq 2$.
4. (13-3d) Mostre que a inserção faz $O(1)$ rotações.
5. Implemente a remoção numa AVL. Ainda basta uma rotação? Meça.

Leitura

Cormen et al., problema 13-3 e seção 13.2; slides antigos de AVL.

Código

josedihego.net/estruturas