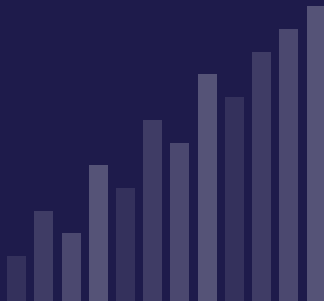

AULA 13 · BLOCO 2: DICIONÁRIOS E ESTRUTURAS LINEARES

Vetores dinâmicos e análise amortizada

Seções 17.1 e 17.4

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Fazer um vetor crescer e encolher conforme o uso
- ▶ Calcular o custo amortizado pelo método agregado
- ▶ Entender por que encolher só com $1/4$ de ocupação
- ▶ Medir $2n$ cópias contra $n^2/2000$ para duas estratégias de crescimento



Teoria

Tabela dinâmica, método agregado, expansão e contração



Prática em C

Vetor dinâmico em C com expansão e contração



Experimento

Dobrar ou somar 1000? Contando cópias



Conexões

Linguagens, bancos de dados, redes e sistemas embarcados

Quando o vetor enche

Table-INSERT: ao encher, aloque o **dobro** e copie tudo.

- ▶ Uma inserção pode custar $\Theta(n)$ (a cópia).
- ▶ Mas as cópias só acontecem nas potências de 2.
- ▶ O pior caso de **uma** operação não diz o custo de uma **sequência**.

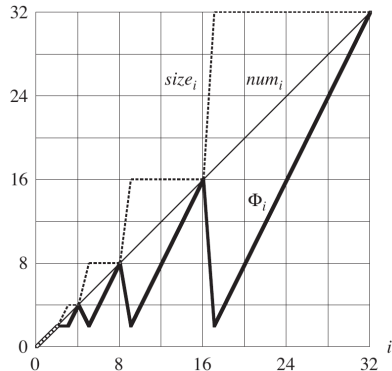


Figura 17.3 de Cormen et al. (2009) — tamanho alocado ($size_i$), ocupação (num_i) e potencial Φ_i ao longo de 32

Análise amortizada: método agregado

Seja c_i o custo da i -ésima inserção: $c_i = i$ se $i - 1$ é potência de 2 (há cópia), senão $c_i = 1$.

$$\sum_{i=1}^n c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j < n + 2n = 3n$$

- ▶ Custo **amortizado**: $3n/n = O(1)$ por inserção, **garantido** (não é caso médio: vale para qualquer sequência).
- ▶ Crescer somando uma constante c : as cópias somam $c + 2c + 3c + \dots \approx n^2/(2c)$. Quadrático, por maior que seja c .
- ▶ Os métodos contábil e do potencial (seções 17.2 e 17.3) dão o mesmo resultado com outra contabilidade.

E para encolher?

- ▶ Ideia ingênua: encolher pela metade quando a ocupação cair para $1/2$.
- ▶ Problema: com o vetor cheio, insere (dobra), remove (encolhe), insere (dobra)...: **toda** operação copia n elementos.
- ▶ Solução (seção 17.4.2): dobrar ao encher, mas só encolher quando a ocupação cair para $1/4$.
- ▶ Depois de uma mudança de tamanho, a ocupação é $1/2$: é preciso muitas operações até a próxima.
- ▶ Inserções e remoções em qualquer ordem: $O(1)$ amortizado.



Prática em C

- ▶ Expansão ao encher; contração com 1/4
- ▶ Contador de cópias embutido no vetor
- ▶ Testes: capacidade depois de inserir e remover

Vetor dinâmico em C (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* TABLE-INSERT e TABLE-DELETE: o vetor dobra quando enche e cai pela metade quando
5     fica com 1/4. Encolher só em 1/2 faria inserções e remoções alternadas copiarem
6     tudo a cada operação; com 1/4, toda operação custa  $O(1)$  amortizado. */
7  typedef struct { int *v; int n, cap; long copias; } Vetor;
8
9  void redimensiona(Vetor *T, int nova) {
10     int *w = malloc(nova * sizeof(int));
11     for (int i = 0; i < T->n; i++) w[i] = T->v[i];    // a cópia: o custo real
12     T->copias += T->n;
13     free(T->v);
14     T->v = w;
15     T->cap = nova;
16 }
```

Vetor dinâmico em C (2/3)

```
17
18 void insere(Vetor *T, int x) {
19     if (T->cap == 0) redimensiona(T, 1);
20     else if (T->n == T->cap) redimensiona(T, 2 * T->cap);
21     T->v[T->n++] = x;
22 }
23
24 void remove_ultimo(Vetor *T) {
25     T->n--;
26     if (T->cap > 1 && T->n <= T->cap / 4) redimensiona(T, T->cap / 2);
27 }
```

Vetor dinâmico em C (3/3)

```
29 int main(void) {
30     int n;
31     if (scanf("%d", &n) != 1 || n < 1) return 1;
32     Vetor T = {NULL, 0, 0, 0};
33     for (int i = 0; i < n; i++) insere(&T, i);
34     printf("capacidade %d com %d elementos (%ld cópias)\n", T.cap, T.n, T.copias);
35     if (n > 7) {
36         for (int i = 0; i < 7; i++) remove_ultimo(&T);
37         printf("depois de remover 7: capacidade %d com %d elementos\n", T.cap, T.n);
38     }
39     free(T.v);
40     return 0;
41 }
```



Experimento

- ▶ Dobrar: $\approx 2n$ cópias (amortizado $O(1)$)
- ▶ Somar 1000: $500 m(m + 1) \approx n^2/2000$ cópias
- ▶ $n = 2^k + 1$: logo após uma expansão

EXPERIMENTO

Dobrar ou somar 1000?

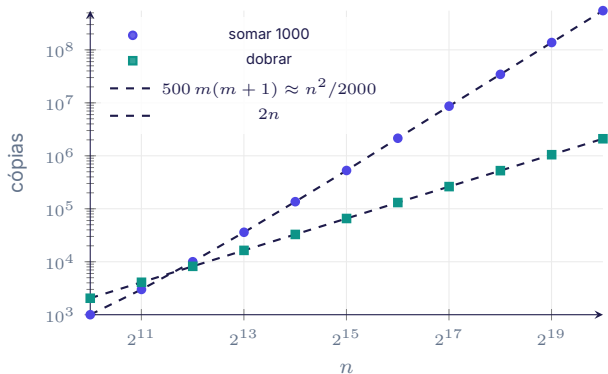
Hipótese. Dobrando, as cópias são $1 + 2 + 4 + \dots + 2^k = 2^{k+1} - 1 \approx 2n$ quando $n = 2^k + 1$: amortizado, **2 cópias por inserção**. Somando 1000 posições, há $m = \lfloor (n - 1)/1000 \rfloor$ expansões que copiam $1000, 2000, \dots, 1000m$: total $500m(m + 1) \approx n^2/2000$.

Método

1. Simulação da capacidade e das cópias (sem alocar de verdade)
2. $n = 2^{10} + 1, \dots, 2^{20} + 1$
3. Capacidade inicial: 1 (dobra) e 1000 (soma)
4. Contagem exata, sem ruído

🧪 EXPERIMENTO

Cópias para n inserções



n	medido	razão
1.025	1.000	1,000
4.097	10.000	1,000
16.385	136.000	1,000
65.537	2.145.000	1,000
262.145	34.453.000	1,000
1.048.577	549.676.000	1,000

razão = medido $\div$ $500 m(m+1) \approx n^2/2000$

✓ desvio máx. 0,0%

Para um milhão de inserções: 2 milhões de cópias dobrando, 550 milhões somando 1000.

EXPERIMENTO

O que o experimento mostrou

Dobrar custa $\approx 2n$ cópias: **amortizado** $O(1)$ por inserção. Somar uma constante custa $\approx n^2/2000$: quadrático, por maior que seja a constante. Por isso `std::vector`, `ArrayList` e as listas do Python crescem por um **fator** (1,5 ou 2), nunca por uma quantidade fixa.

>_ Reproduza: `gcc -O2 aula13_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Linguagens
- ▶ Bancos de dados
- ▶ Redes
- ▶ Sistemas embarcados

Onde isso aparece na computação

Linguagens

A lista do Python cresce cerca de 12%, o vector do C++ (GCC) dobra e o ArrayList do Java cresce 50%: fatores diferentes, todos amortizados $O(1)$.

Bancos de dados

Tabelas hash de bancos e caches (Redis) dobram de tamanho e reinserem tudo, às vezes aos poucos (*rehash* incremental) para não travar o serviço.

Redes

O TCP aumenta a janela de envio por fator e a reduz pela metade em congestionamento: crescimento multiplicativo, como o vetor dinâmico.

Sistemas embarcados

Em tempo real, um pico de $\Theta(n)$ numa cópia pode estourar um prazo: lá se prefere capacidade fixa ou crescimento sem cópia (listas de blocos).

Para casa

Exercícios

1. (17.1-3) Uma sequência de n operações em que a i -ésima custa i se i é potência de 2 e 1 senão: custo amortizado?
2. (17.4-2) Mostre que, se a ocupação está acima de $1/2$ e a operação é remoção, o custo amortizado é limitado por uma constante.
3. Mostre uma sequência de n operações com custo $\Theta(n^2)$ quando o vetor encolhe já com $1/2$ de ocupação.
4. Altere o experimento para crescer por fator 1,5 e compare com o fator 2 (cópias e memória desperdiçada).
5. Implemente uma fila em vetor circular que dobra quando enche.

Leitura

Cormen et al., seções 17.1 (método agregado) e 17.4 (tabelas dinâmicas).

Código

josedihego.net/estruturas