AULA 23

Análise amortizada

CLR 18 ou CLRS 17

Análise amortizada

Exemplo 1:

Contador binário

$k-1$		3	2	1	0	
						A

```
INCREMENT (A, k)
  i ← 0
  enquanto i < k e A[i] = 1
      faça A[i] ← 0
           i ← i + 1
  se i < k
      então A[i] ← 1
```

"Custo" = consumo de tempo =
= número de bits alterados = O(k)

Seqüência de n chamadas:

$$\underbrace{\text{INCR}\quad\text{INCR}\quad\cdots\quad\text{INCR}\quad\text{INCR}\quad\text{INCR}}_{n}$$

Custo = O(nk) Exagero!

$$
\begin{array}{cccccc}
\multicolumn{6}{c}{A} \\
5 & 4 & 3 & 2 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0
\end{array}
\qquad n = 16
$$

$A[0]$	muda	n	vezes
$A[1]$	"	$\lfloor n/2 \rfloor$	"
$A[2]$	"	$\lfloor n/4 \rfloor$	"
$A[3]$	"	$\lfloor n/8 \rfloor$	"

Custo total:

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor \; < \; n \sum_{i=0}^{\infty} \frac{1}{2^i} \; = \; 2n \; = \; \mathrm{O}(n)$$

Custo amortizado (= custo fictício =
= custo faz-de-conta) de uma operação:

$$\frac{2n}{n} \; = \; \mathrm{O}(1)$$

Este foi o **método agregado** de análise:
soma os custos de todas as operações
para determinar o custo amortizado
de cada operação

Outro método de análise: potencial

$$A_0 \xrightarrow{1^{\text{a}}\text{ op}} A_1 \xrightarrow{2^{\text{a}}\text{ op}} A_2 \longrightarrow \ldots \xrightarrow{n^{\text{a}}\text{ op}} A_n$$

A_i = estado de A depois da i^{a} operação

Custo real da i^{a} operação: $c_i = t_i + 1$
onde t_i = número de 1→0 na i^{a} operação

Energia **potencial** de A_i:

$$\begin{aligned} \Phi(A_i) &= \text{número de bits ``1''} \\ &= \Phi(A_{i-1}) - t_i + 1 \end{aligned}$$

Custo amortizado da i^{a} operação:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi(A_i) - \Phi(A_{i-1}) \\ &= c_i + \Phi(A_{i-1}) - t_i + 1 - \Phi(A_{i-1}) \\ &= c_i - t_i + 1 \\ &= (t_i + 1) - t_i + 1 \\ &= 2 \end{aligned}$$

Soma dos custos amortizados limita a soma dos custos reais pois $\Phi \geq 0$:

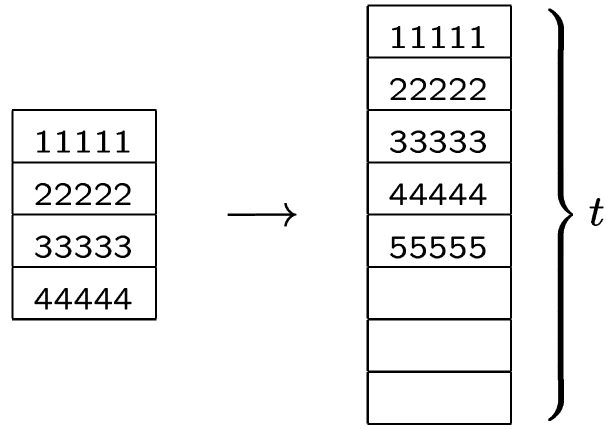
$$\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum (c_i + \Phi(A_i) - \Phi(A_{i-1})) \\
&= \sum c_i \ + \ \Phi(A_n) - \Phi(A_0) \\
&= \sum c_i \ + \ \Phi(A_n) - 0 \\
&= \sum c_i \ + \ \Phi(A_n) \\
&\geq \sum c_i
\end{aligned}$$

Conclusão: $\quad \sum c_i \ \leq \ \sum \hat{c}_i \ = \ 2n \ = \ \mathrm{O}(n)\,.$

Resumo da relação entre c, $\hat{c}$ e Φ:

$$\boxed{\hat{c}_i - c_i \ = \ \Phi_i - \Phi_{i-1}}$$

Exemplo 2: **Tabelas dinâmicas**



$n[T]$= número de itens

Inicialmente $n[T] = t[T] = 0$

TABLE-INSERT (T, x)

01 se $t[T] = 0$

02 então aloque $tabela[T]$ com 1 posição

03 $t[T] \leftarrow 1$

04 se $n[T] = t[T]$

05 então aloque $nova$ com $2\,t[T]$ posições

06 insira itens da $tabela[T]$ na $nova$

07 libere $tabela[T]$

08 $tabela[T] \leftarrow nova$

09 $t[T] \leftarrow 2\,t[T]$

10 insira x na $tabela[T]$

11 $n[T] \leftarrow n[T] + 1$

Custo = número de inserções elementares (linhas 6 e 10)

Seqüência de n TABLE-INSERTs

Custo real da i^{a} operação:

$$c_i = \begin{array}{ll} 1 & \text{se há espaço} \\ 1 + n_{i-1} & \text{se tabela cheia} \end{array}$$

$$\begin{array}{lcl} \text{onde } n_i & = & \text{valor de } n \text{ depois da } i^{\text{a}} \text{ operação} \\ & = & i \end{array}$$

Custo de uma operação $= \mathrm{O}(n)$

Custo das n operações $= \mathrm{O}(n^2)$. Exagero!

Função **potencial**: $\Phi(T) := 2\,n[T] - t[T]$

Note que $0 \leq \Phi(T) \leq t[T]$

Cálculo do custo amortizado $\widehat{c}_i$:

Se iª operação *não* causa expansão então

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= 1 + (2n_i - t_i) - (2(n_i - 1) - t_i) \\
&= 3
\end{aligned}$$

n_i, t_i, Φ_i = valores depois da iª operação

Se iª operação causa expansão então

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= n_i + (2n_i - t_i) - (2n_{i-1} - t_{i-1}) \\
&= n_i + (2n_i - 2n_{i-1}) - (2n_{i-1} - n_{i-1}) \\
&= n_i + 2n_i - 3n_{i-1} \\
&= n_i + 2n_i - 3(n_i - 1) \\
&= 3
\end{aligned}$$

Conclusão: $\widehat{c}_i = 3$ para qualquer i

O custo real das n operações é limitado pelo custo amortizado pois $\Phi \geq 0$:

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &= \sum \widehat{c}_i - \Phi_n + \Phi_0 \\
&= \sum \widehat{c}_i - \Phi_n \\
&\leq \sum \widehat{c}_i \\
&= 3n \\
&= \mathrm{O}(n)
\end{aligned}
$$

TAREFA 23

Exercício 23.A [CLR 18.1-3 18.2-2 18.3-2, CLRS 17.1-3 17.2-2 17.3-2] Uma seqüência de n operações é executada sobre uma certa estrutura de dados. Suponha que a i-a operação custa

$$\begin{array}{ll} i & \text{se } i \text{ é uma potência de 2,} \\ 1 & \text{em caso contrário.} \end{array}$$

Mostre que o custo amortizado de cada operação é O(1). Use o método da "análise agregada"; depois, repita tudo usando o método da função potencial.

Exercício 23.B [Importante]
Mostre que a função $\Phi(T) = t[T] - n[T]$ não é uma boa função potencial para a análise da tabela dinâmica T sob a operação TABLE-INSERT. Mostre que $\Phi(T) = t[T]$ também não é um bom potencial para a análise da tabela dinâmica.

Exercício 23.C [CLR 18.3-3, CLRS 17.3-3]
Considere a estrutura de dados min-heap munida das operações INSERT e EXTRACT-MIN. Cada operação consome tempo O($\lg n$), onde n é o número de elementos na estrutura. Dê uma função potencial Φ tal que o custo amortizado de INSERT seja O($\lg n$) e o custo amortizado de EXTRACT-MIN seja O(1). Prove que sua função potencial de fato tem essas propriedades.

Exercício 23.D
Descreva um algoritmo que receba um inteiro positivo n e calcule n^n fazendo não mais que $2\lg n$ multiplicações de números inteiros.

Exercício 23.E [CLR 18-2, CLRS 17-2, Busca binária dinâmica] Busca binária em um vetor ordenado consome tempo logarítmico, mas o tempo necessário para inserir um novo elemento é linear no tamanho do vetor. Isso pode ser melhorado se mantivermos diversos vetores ordenados (em lugar de um só). Suponha que queremos implementar as operações BUSCA e INSERÇÃO em um conjunto de n elementos. Seja $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$ a representação binária de n, onde $k = \lceil \lg(n+1) \rceil$. Temos vetores crescentes $A_0, A_1, \ldots, A_{k-1}$, sendo que o comprimento de A_i é 2^i. Um vetor típico A_i é relevante se $n_i = 1$ e irrelevante se $n_i = 0$. O número total de elementos nos k vetores é, portanto,

$$\sum_{i=0}^{k-1} n_i \, 2^i = n \, .$$

Cada vetor é crescente, mas não há qualquer relação entre os valores dos elementos em dois vetores diferentes.

a. Dê um algoritmo para a operação BUSCA. Dê uma delimitação superior para o consumo de tempo do algoritmo.

b. Dê um algoritmo para a operação INSERÇÃO. Dê uma delimitação superior para o consumo de tempo do algoritmo. Calcule o consumo de tempo *amortizado*.

c. Discuta uma implementação da operação REMOÇÃO.