AULA 17

Mais análise amortizada

CLR 18 ou CLRS 17

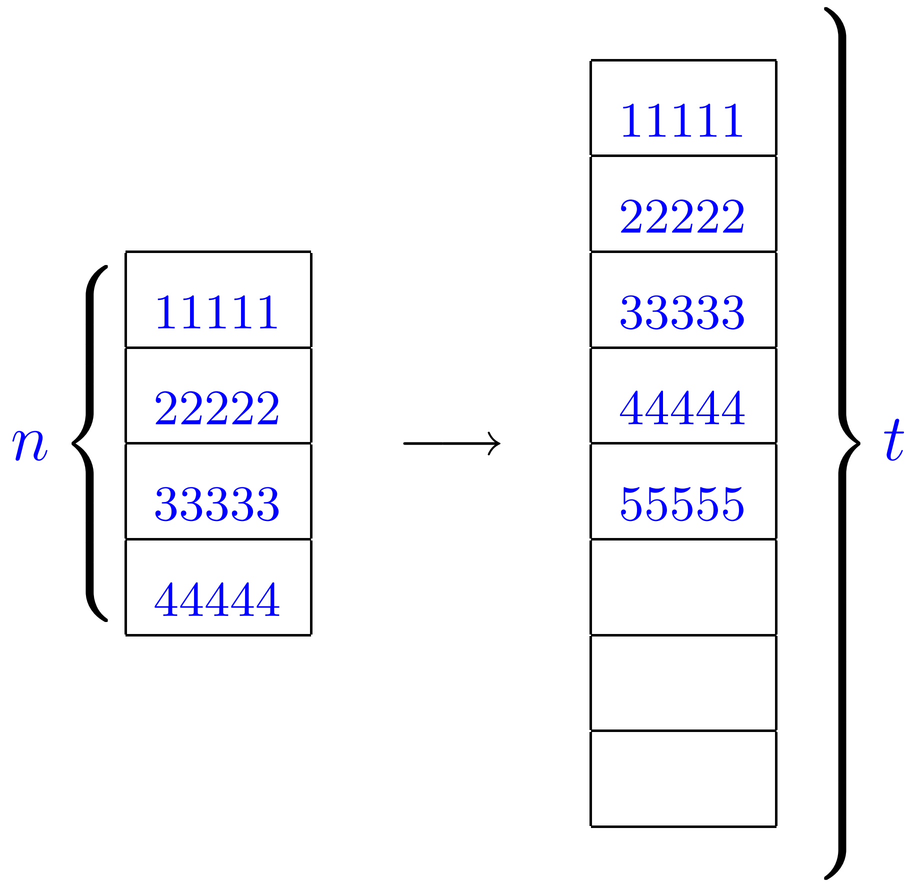
Análise amortizada

Análise amortizada $=$ análise do consumo de tempo de uma seqüência de operações

Usada nos casos em que

o consumo de tempo no pior caso de n operações é
menor
que n vezes o consumo de tempo no pior caso de uma operação

Tabelas dinâmicas



$n[T]$= número de itens $\quad$ $t[T]$= tamanho de T

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

Inserção

Insere um elemento x na tabela T

TABLE-INSERT (T, x)

1 **se** $t[T] = 0$
2 **então** aloque $tabela[T]$ com 1 posição
3 $t[T] \leftarrow 1$
4 **se** $n[T] = t[T]$
5 **então** aloque $nova\text{-}tabela$ com $2\,t[T]$ posições
6 insira itens da $tabela[T]$ na $nova\text{-}tabela$

---

7 $t[nova\text{-}tabela] \leftarrow 2\,t[T]$
8 libere $tabela[T]$
9 $tabela[T] \leftarrow nova\text{-}tabela$
10 insira x na $tabela[T]$

---

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

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

Seqüência de m TABLE-INSERTS

$$T_0 \xrightarrow{1^{\rm a}\text{ op}} T_1 \xrightarrow{2^{\rm a}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\rm a}\text{ op}} T_m$$

T_i = estado de T depois da $i^{\rm a}$ operação

Custo real da $i^{\rm a}$ operação:

$$c_i = \begin{cases} 1 & \text{se há espaço} \\ n_i & \text{se tabela cheia} \end{cases}$$

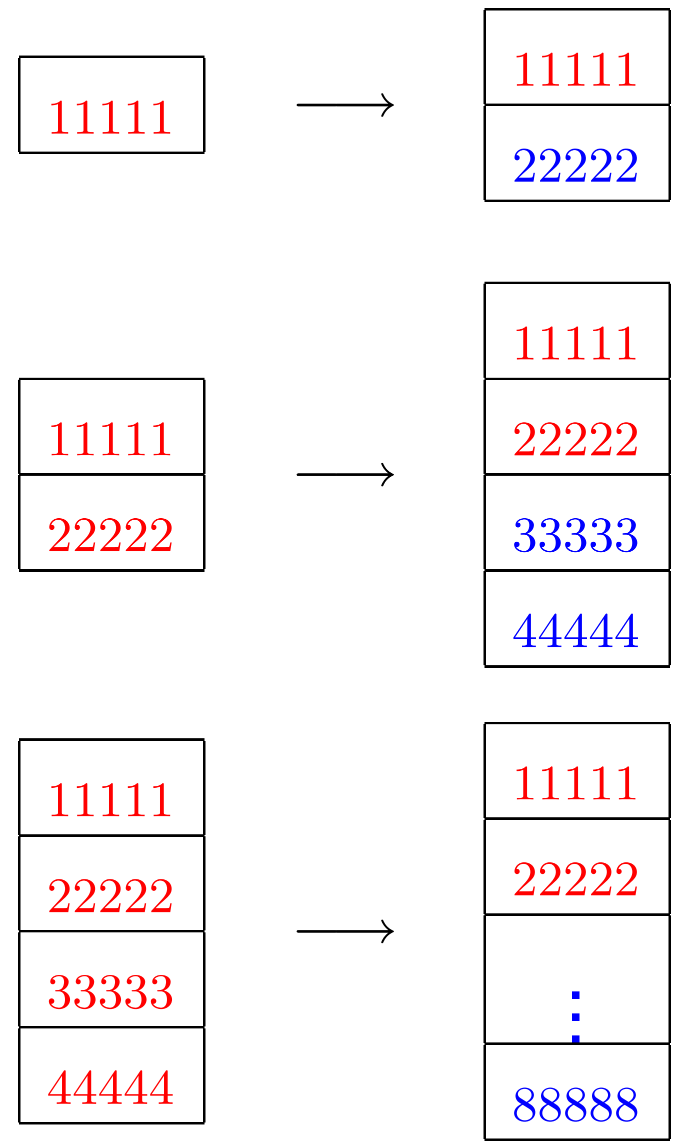
onde n_i = valor de $n[T]$ depois da $i^{\rm a}$ operação
= i

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

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

Exemplo

operação ($n[T]$)	$t[T]$	custo
1	1	1
2	2	$1+1$
3	4	$1+2$
4	4	1
5	8	$1+4$
6	8	1
7	8	1
8	8	1
9	16	$1+8$
10	16	1
16	16	1
17	32	$1+16$
33	64	$1+32$



Custo amortizado

Custo total:

$$\sum_{i=1}^{n} c_i \;=\; m + \sum_{i=0}^{k} 2^i \;=\; n + 2^{k+1} - 1 \;<\; n + 2m - 1 \;<\; 3m$$

onde $k = \lfloor \lg(m-1) \rfloor$

Custo amortizado:

$$\frac{3m}{m} \;=\; 3 \;=\; \Theta(1)$$

Conclusões

O custo de uma seqüência de m execuções do algoritmo TABLE-INSERT é $\Theta(m)$.

O custo amortizado do algoritmo TABLE-INSERT é $\Theta(1)$.

Método de análise agregada

- m operações consomem tempo $T(m)$
- custo médio de cada operação é $T(m)/m$
- custo amortizado de cada operação é $T(m)/m$
- defeito: no caso de mais de um tipo de operação, o custo de cada tipo não é determinado separadamente

Método de análise contábil

TABLE-INSERT (T, x)

$credito \leftarrow credito + 3$

se $t[T] = 0$

  então aloque $tabela[T]$ com 1 posição

    $t[T] \leftarrow 1$

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

  então aloque $nova\text{-}tabela$ com $2\,t[T]$ posições

    insira itens da $tabela[T]$ na $nova\text{-}tabela$

    $custo \leftarrow custo + n[T]$

    libere $tabela[T]$

    $tabela[T] \leftarrow nova\text{-}tabela$

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

insira x na $tabela[T]$

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

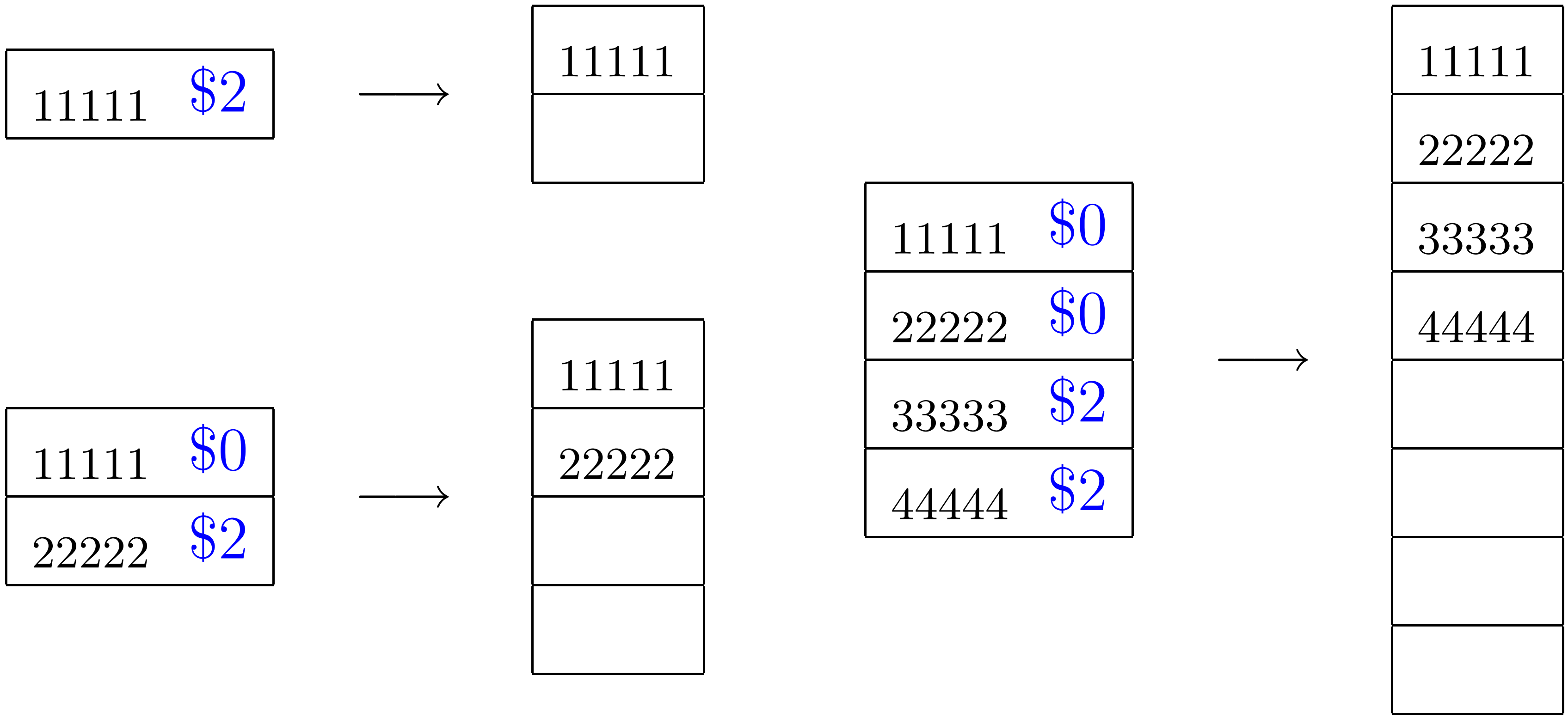
$custo \leftarrow custo + 1$

Método de análise contábil

Invariante: soma créditos $\geq$ soma custos reais

$n[T]$	$t[T]$	custo	crédito	saldo
1	1	1	3	2
2	2	$1+1$	3	3
3	4	$1+2$	3	3
4	4	1	3	5
5	8	$1+4$	3	3
6	8	1	3	5
7	8	1	3	7
8	8	1	3	9
9	16	$1+8$	3	3
10	16	1	3	5
16	16	1	3	17
17	32	$1+16$	3	3

Método de análise contábil



Método de análise contábil

Pague	\$1	para inserir um novo elemento
guarde	\$1	para eventualmente mover o novo elemento
guarde	\$1	para mover um elemento que já está na tabela

Custo amortizado por chamada de TABLE-INSERT: $\leq$ \$3

Seqüência de m chamadas de TABLE-INSERT.

Como \$ armazenado nunca é negativo,

$$\begin{aligned} \text{soma custos reais} &\leq \text{soma custos amortizados} \\ &= 3m \\ &= \mathrm{O}(m) \end{aligned}$$

Método de análise contábil

- cada operação paga seu custo real
- cada operação recebe um certo número de créditos (chute de custo amortizado)
- balanço nunca pode ser negativo

  soma créditos $\geq$ soma custos reais

  créditos não usados são guardados para pagar operações futuras.
- custo amortizado da operação $\leq$ número médio de créditos recebidos
- custo amortizado de cada tipo de operação pode ser determinado separadamente

Método de análise potencial

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

$n[T]$	$t[T]$	custo	$\Phi(T)$	$\Delta\Phi$	custo$+\Delta\Phi$
1	1	1	1	$+1$	2
2	2	$1{+}1$	2	$+1$	3
3	4	$1{+}2$	2	0	3
4	4	1	4	$+2$	3
5	8	$1{+}4$	2	-2	3
6	8	1	4	$+2$	3
7	8	1	6	$+2$	3
8	8	1	8	$+2$	3
9	16	$1{+}8$	2	-6	3
10	16	1	4	$+2$	3
16	16	1	16	$+2$	3
17	32	$1{+}16$	2	-14	3

Método de análise potencial

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

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

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

Se i^{a} operação não causa expansão então

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Método de análise potencial

Se i^{a} operação causa expansão então

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

Conclusão: $\hat{c}_i = 3$ para qualquer $i \geq 2$

Método de análise potencial

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

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

Método de análise potencial

- método contábil visto como energia potencial
- potencial associado à estrutura de dados

Seqüencia de INSERT e DELETE

Seqüência de operações TABLE-INSERT e TABLE-DELETE

$$\underbrace{\text{I} \quad \text{I} \quad \text{D} \quad \text{I} \quad \text{I} \quad \text{D} \quad \text{I} \quad \text{D} \quad \text{D} \quad \text{I} \quad \text{D} \quad \text{D} \quad \text{I} \quad \text{I}}_{m}$$

Custo total de uma seqüência de TABLE-INSERT e TABLE-DELETE?

Remoção

Remove um elemento x da tabela T

TABLE-DELETE (T, x) $\quad \rhd$ supõe x na $tabela[T]$

1 remova x da $tabela[T]$

2 $n[T] \leftarrow n[T] - 1$

3 **se** $n[T] < t[T]/\mathbf{2}$ $\quad \rhd$ tabela está “vazia”?

4 **então** aloque $nova\text{-}tabela$ com $t[T]/\mathbf{2}$ posições

5 insira itens da $tabela[T]$ na $nova\text{-}tabela$

6 $t[nova\text{-}tabela] \leftarrow t[T]/2$

7 $n[nova\text{-}tabela] \leftarrow n[T]$

8 libere $tabela[T]$

9 $tabela[T] \leftarrow nova\text{-}tabela$

Custo $=$ número de remoções e inserções elementares (linhas 1 e 5)

Seqüência de INSERT e DELETE

Seqüência de operações TABLE-INSERT e TABLE-DELETE

$$\underbrace{\underbrace{\text{I} \quad \cdots \quad \text{I} \quad \text{I}}_{m/2} \quad \text{I} \quad \text{D} \quad \text{D} \quad \text{I} \quad \text{I} \quad \text{D} \quad \text{D} \quad \text{I} \quad \text{I} \quad \text{D} \quad \text{D}}_{m}$$

Se $m = 2^k$, então custo total da seqüência:

$$\Theta(\frac{m}{2}) + \frac{m}{4}\,\Theta(\frac{m}{2}) = \Theta(m^2)$$

Remoção

Remove um elemento x da tabela T

TABLE-DELETE (T, x) $\triangleright$ supõe x na $tabela[T]$

1 remova x da $tabela[T]$

---

2 $n[T] \leftarrow n[T] - 1$
3 **se** $n[T] < t[T]/\mathbf{4}$ $\triangleright$ tabela está “vazia”?
4 **então** aloque $nova\text{-}tabela$ com $t[T]/\mathbf{2}$ posições
5 insira itens da $tabela[T]$ na $nova\text{-}tabela$

---

6 $t[nova\text{-}tabela] \leftarrow t[T]/\mathbf{2}$
7 $n[nova\text{-}tabela] \leftarrow n[T]$
8 libere $tabela[T]$
9 $tabela[T] \leftarrow nova\text{-}tabela$

Custo $=$ número de remoções e inserções elementares
(linhas 1 e 5)

Seqüência de n operações

$$T_0 \xrightarrow{1^{\text{a}}\text{ op}} T_1 \xrightarrow{2^{\text{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Custo real da i^{a} operação se for TABLE-INSERT:

$$c_i = \begin{cases} 1 & \text{se há espaço} \\ n_i & \text{se tabela cheia} \end{cases}$$

onde n_i = valor de $n[T]$ depois da i^{a} operação

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

Seqüência de m operações

$$T_0 \xrightarrow{1^{\mathrm{a}}\text{ op}} T_1 \xrightarrow{2^{\mathrm{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\mathrm{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Custo real da i^{a} operação se for TABLE-DELETE:

$$c_i = \begin{cases} 1 & \text{se } n_{i-1} > t_{i-1}/\mathbf{4} \\ 1 + n_i & \text{se } n_{i-1} = t_{i-1}/\mathbf{4} \end{cases}$$

onde n_i = valor de $n[T]$ depois da i^{a} operação

e t_i = valor de $t[T]$ depois da i^{a} operação

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

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

Método de análise potencial

$$T_0 \xrightarrow{1^{\text{a}}\text{ op}} T_1 \xrightarrow{2^{\text{a}}\text{ op}} T_2 \longrightarrow \cdots \xrightarrow{m^{\text{a}}\text{ op}} T_m$$

T_i = estado de T depois da i^{a} operação

Energia potencial de T_i:

$$\Phi(T_i) = \begin{cases} 2\,n_i - t_i, & \text{se } n_i \geq t_i/\mathbf{2} \\ t_i/2 - n_i, & \text{se } n_i < t_i/\mathbf{2} \end{cases}$$

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

n_i, t_i, Φ_i = $n[T_i], t[T_i], \Phi(T_i)$ depois da i^{a} operação

Custo amortizado TABLE-INSERT (1)

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se i^{a} operação é TABLE-INSERT, $n_{i-1} \geq t_{i-1}/\mathbf{2}$ e não houve expansão, então

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (2)

Se i^{a} operação é TABLE-INSERT, $n_{i-1} \geq t_{i-1}/\mathbf{2}$ e houve expansão, então

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (3)

Se i^{a} operação é TABLE-INSERT, $n_{i-1} < t_{i-1}/\mathbf{2}$ e $n_i < t_i/\mathbf{2}$, então não houve expansão e

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (t_i/2 - n_i) - (t_{i-1}/2 - n_{i-1}) \\ &= 1 + (t_i/2 - t_i/2) - (n_i - (n_i - 1)) \\ &= 0\,.\end{aligned}$$

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-INSERT (4)

Se i^{a} operação é TABLE-INSERT e $n_{i-1} < t_{i-1}/\mathbf{2}$ e $n_i \geq t_i/\mathbf{2}$, então não houve expansão e

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-DELETE (1)

c_i = custo real da i^{a} operação.

Custo amortizado da i^{a} operação:

$$\hat{c}_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Se i^{a} operação é TABLE-DELETE e $n_{i-1} < t_{i-1}/\mathbf{2}$ e não houve contração, então

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Custo amortizado TABLE-DELETE (2)

Se i^{a} operação é TABLE-DELETE e $n_{i-1} < t_{i-1}/\mathbf{2}$ e houve contração, então

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

n_i, t_i, Φ_i = valores depois da i^{a} operação

Quando $n_{i-1} \geq t_{i-1}/\mathbf{2}$, o custo amortizado da operação é TABLE-DELETE também é uma constante [CLRS 17.4-2].

Conclusões

O custo de uma seqüência de m execuções do algoritmos TABLE-INSERT e TABLE-DELETE é $\Theta(m)$.

O custo amortizado dos algoritmos TABLE-INSERT e TABLE-DELETE é $\Theta(1)$.

Class ArrayList

O Paulo (peas) me mostrou o trecho que foi copiado de

`http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html`

"... Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost. ..."

O Paulo também disse que na documentação do STL do `C++` tem algo semelhante.

Exercícios

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 $\mathrm{O}(1)$. 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 $\mathrm{O}(\lg n)$, onde n é o número de elementos na estrutura. Dê uma função potencial Φ tal que o custo amortizado de INSERT seja $\mathrm{O}(\lg n)$ e o custo amortizado de EXTRACT-MIN seja $\mathrm{O}(1)$. Prove que sua função potencial de fato tem essas propriedades.

Outro exercício

Exercício 23.D [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.

Mais um exercício

Exercício 23.E

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