Capítulo VII

Complexidade Amortizada

Complexidade Amortizada

- **Objetivo:** analisar o **custo de uma sequência** de operações numa ED, **no pior caso**.

- **Interesse:** mostrar que, embora alguma operação possa ser "cara", o custo total da sequência de operações é "baixo".

- **Não é** a complexidade no caso esperado (que indica o custo médio, considerando todas as distribuições da entrada). **Não envolve** probabilidades.

Contador Binário

```
void increment( int[] counter )   // Pior caso:
{                                 // c é a capacidade do vetor.
    int pos = 0;
    while ( pos < counter.length  &&  counter[pos] == 1 )
    {
        counter[pos] = 0;
        pos++;
    }
    if ( pos < counter.length )
        counter[pos] = 1;
}
```

Contador Binário

```
void increment( int[] counter )   // Pior caso: Θ(c), onde
{                                 // c é a capacidade do vetor.
    int pos = 0;
    while ( pos < counter.length  &&  counter[pos] == 1 )
    {
        counter[pos] = 0;
        pos++;
    }
    if ( pos < counter.length )
        counter[pos] = 1;
}
```

Custo, no pior caso, (com ingenuidade) de uma sequência de n operações (de increment), num contador inicialmente a zero:

Contador Binário

```
void increment( int[] counter )   // Pior caso: Θ(c), onde
{                                 // c é a capacidade do vetor.
    int pos = 0;
    while ( pos < counter.length  &&  counter[pos] == 1 )
    {
        counter[pos] = 0;
        pos++;
    }
    if ( pos < counter.length )
        counter[pos] = 1;
}
```

Custo, no pior caso, (com ingenuidade) de uma sequência de n operações (de increment), num contador inicialmente a zero: $O(nc)$.

Tabela Dinâmica

```
// int currentSize,   E[] table (preenchida de 0 a currentSize − 1).
void insert( E element )        // Pior caso:
{                               // s é o número de elementos na tabela.
    if ( table == null )
        table = new E[1];
    else if ( currentSize == table.length )
    {
        E[] newTable = new E[ 2 * currentSize ];
        System.arraycopy(table, 0, newTable, 0, currentSize);
        table = newTable;
    }
    table[ currentSize++ ] = element;
}
```

Tabela Dinâmica

```
// int currentSize,   E[] table (preenchida de 0 a currentSize − 1).

void insert( E element )        // Pior caso: Θ(s), onde
{                               // s é o número de elementos na tabela.
    if ( table == null )
        table = new E[1];
    else if ( currentSize == table.length )
    {
        E[] newTable = new E[ 2 * currentSize ];
        System.arraycopy(table, 0, newTable, 0, currentSize);
        table = newTable;
    }
    table[ currentSize++ ] = element;
}
```

Custo, no pior caso, (com ingenuidade) de uma sequência de n operações (de insert), numa tabela inicialmente vazia:

Tabela Dinâmica

```
// int currentSize,   E[] table (preenchida de 0 a currentSize − 1).

void insert( E element )        // Pior caso: Θ(s), onde
{                               // s é o número de elementos na tabela.
    if ( table == null )
        table = new E[1];
    else if ( currentSize == table.length )
    {
        E[] newTable = new E[ 2 * currentSize ];
        System.arraycopy(table, 0, newTable, 0, currentSize);
        table = newTable;
    }
    table[ currentSize++ ] = element;
}
```

Custo, no pior caso, (com ingenuidade) de uma sequência de n operações (de insert), numa tabela inicialmente vazia: $O(n^2)$.

Métodos Existentes

- Há três métodos (com algumas variantes).
  - **Agregação**: é o menos versátil e não será estudado.
  - **Contabilidade**.
  - **Potencial**: é o mais versátil.
- Em todos os métodos, calcula-se um majorante do custo total da sequência de operações.

  A esse majorante, chama-se **custo total amortizado**.
- Ao verdadeiro custo total, chama-se **custo total real**.
- **O custo total amortizado nunca é inferior ao custo total real.**

Método da Contabilidade

- Atribui-se um custo amortizado $\widehat{c}$ a cada operação, que pode ser superior, igual ou inferior ao custo real c.

- Se o custo amortizado $\widehat{c}$ for superior ao custo real c, a diferença $\widehat{c} - c$, chamada crédito, é associada a um objeto na ED.
  Posteriormente, o crédito é usado para "pagar" operações cujo custo amortizado é inferior ao custo real.

- **O custo total amortizado nunca é inferior ao custo total real.**
  Ou seja, **o crédito total acumulado nunca é negativo.**
  Para qualquer sequência de n operações, $\left(\sum_{i=1}^{n} \widehat{c}_i - \sum_{i=1}^{n} c_i\right) \geq 0$.

- As operações podem ter complexidades amortizadas diferentes.

Contabilidade — Contador (1)

Operação	Custo Real	Custo Amortizado
atribuir ZERO	1	0 $O(1)$
atribuir UM	1	2 $O(1)$
increment	k	2 $O(1)$

(k é o número de atribuições efetuadas.)

Cada UM tem 1 crédito ("como pré-pagamento da sua passagem a ZERO").

Cada operação de increment tem custo amortizado 2, porque atribui um UM (no máximo).

Contabilidade — Contador (2)

	Estado do Contador	**Crédito Total Acumulado**
Inicial	00000	0
increment	00001	$0 + (2 - 1) = 1$
increment	00010	$1 + (2 - 2) = 1$
increment	00011	$1 + (2 - 1) = 2$
increment	00100	$2 + (2 - 3) = 1$
increment	00101	$1 + (2 - 1) = 2$
increment	00110	$2 + (2 - 2) = 2$
increment	00111	$2 + (2 - 1) = 3$
increment	01000	$3 + (2 - 4) = 1$

Contabilidade — Contador (3)

Operação	Custo Real	Custo Amortizado
atribuir ZERO	1	0 $O(1)$
atribuir UM	1	2 $O(1)$
increment	k	2 $O(1)$

(k é o número de atribuições efetuadas.)

O crédito total acumulado nunca é negativo: é o número de UNS no vetor.

Conclusões: a complexidade de uma sequência de n operações de increment, num contador inicialmente a zero, é $O(n)$;

a complexidade amortizada do increment é $O(1)$.

Contabilidade — Tabela (1)

Operação	Custo Real	Custo Amortizado
atribuir elemento novo	1	3 $O(1)$
insert	k	3 $O(1)$

(k é o número total de atribuições efetuadas.)

Cada elemento na tabela tem 2 créditos ("como pré-pagamento da sua transferência para outra tabela e como pré-pagamento da transferência de outro elemento que já foi transferido alguma vez").

Cada operação de insert tem custo amortizado 3, porque atribui apenas um elemento novo.

Contabilidade — Tabela (2)

	Dimensão da Tabela	**Crédito Total Acumulado**
Tabela Cheia	$s = 2^0$	$1 + 1$
Tabela Cheia	$s = 2^k$	$1 + s$
insert e_1	$s = 2^k + 1$	$1 + 1 * 2$
insert e_2	$s = 2^k + 2$	$1 + 2 * 2$
insert e_3	$s = 2^k + 3$	$1 + 3 * 2$
.....	.....	.....
insert e_{2^k}	$s = 2^k + 2^k$	$1 + 2^k * 2$
Tabela Cheia	$s = 2^{k+1}$	$1 + s$

Contabilidade — Tabela (3)

Operação	Custo Real	Custo Amortizado
atribuir elemento novo	1	3 $O(1)$
insert	k	3 $O(1)$

(k é o número total de atribuições efetuadas.)

O crédito total acumulado nunca é negativo: é $1 + 2i$, onde i é o número de elementos inseridos na tabela após a última expansão.

Conclusões: a complexidade de uma sequência de n operações de insert, numa tabela inicialmente vazia, é $O(n)$;

a complexidade amortizada do insert é $O(1)$.

Método do Potencial (1)

- Como no método da contabilidade, cada operação tem um custo amortizado, que pode ser superior, igual ou inferior ao custo real.

- Como no método da contabilidade, as operações podem ter complexidades amortizadas diferentes.

- Em vez de se associar "crédito" aos objetos, associa-se um valor à ED, chamado potencial.

- Ou seja, define-se uma função potencial Φ, que atribui a cada ED D um número real $\Phi(D)$.

Método do Potencial (2)

- Para cada $i = 1, 2, \ldots, n$, sejam:
  - D_0 a ED inicial;
  - D_i a ED depois da operação i; e
  - c_i o custo real da operação i.

  Então:
  - o **custo amortizado da operação** i é

$$\widehat{c}_i \;=\; c_i + \Phi(D_i) - \Phi(D_{i-1});$$

  - o **custo total amortizado** é

$$\begin{aligned} \textstyle\sum_{i=1}^{n} \widehat{c}_i \;&=\; \textstyle\sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1}) \right) \\ &=\; \left(\textstyle\sum_{i=1}^{n} c_i\right) + \Phi(D_n) - \Phi(D_0). \end{aligned}$$

Método do Potencial (3)

- **O custo total amortizado nunca é inferior ao custo total real.**

  Como o custo total amortizado é

$$\left(\sum_{i=1}^{n} c_i\right) + \Phi(D_n) - \Phi(D_0),$$

  basta assegurar que, para qualquer $i = 1, 2, \ldots, n$:

$$\Phi(D_i) \geq \Phi(D_0).$$

- Na prática, define-se Φ de forma a garantir:

  (P1) $\Phi(D_0) = 0$; e

  (P2) $\Phi(D_i) \geq 0$, para qualquer i.

  E diz-se que Φ é uma função potencial **válida**.

Potencial — Contador (1)

Seja C um contador qualquer e seja 1_C o número de UNS em C.

$$\Phi(C) = 1_C .$$

A função Φ é **válida**:

(P1) $\Phi(C_0) = 0$, onde C_0 é o contador (inicial) só com ZEROS.

(P2) $\Phi(C) \geq 0$.

Portanto, o custo total amortizado nunca será inferior ao custo total real.

O custo amortizado da operação i é:

$$\widehat{c}_i = c_i + \Phi(C_i) - \Phi(C_{i-1}).$$

Potencial — Contador (2)

Seja C um contador qualquer e seja 1_C o número de UNS em C.

$$\Phi(C) = 1_C\,.$$

Operação	**Custo Real**	**Dif. de Potencial**	**Custo Amortizado**
increment	c_i	$\Phi(C_i) - \Phi(C_{i-1})$	$\widehat{c}_i = c_i + \Delta\Phi$
incrementa	$k+1$	$-k+1$	2 $\quad O(1)$
anula	k	$-k$	0 $\quad O(1)$

(k é o número de UNS que passam a ZERO.)

Conclusões: a complexidade de uma sequência de n operações de increment, num contador inicialmente a zero, é

Potencial — Contador (2)

Seja C um contador qualquer e seja 1_C o número de UNS em C.

$$\Phi(C) = 1_C\,.$$

Operação	**Custo Real**	**Dif. de Potencial**	**Custo Amortizado**
increment	c_i	$\Phi(C_i) - \Phi(C_{i-1})$	$\widehat{c}_i = c_i + \Delta\Phi$
incrementa	$k+1$	$-k+1$	2 $\quad O(1)$
anula	k	$-k$	0 $\quad O(1)$

(k é o número de UNS que passam a ZERO.)

Conclusões: a complexidade de uma sequência de n operações de increment, num contador inicialmente a zero, é $O(n)$;

a complexidade amortizada do increment é $O(1)$.

Potencial — Tabela (1)

Seja T uma tabela qualquer e sejam s_T o número de elementos em T e c_T a capacidade de T.

$$\Phi(T) \;=\; 2\,s_T \;-\; c_T\,.$$

A função Φ é **válida**:

(P1) $\Phi(T_0) = 0$, onde T_0 é a tabela (inicial)

com 0 elementos e capacidade 0.

(P2) $\Phi(T) \geq 0$, nos outros casos.

Como o fator de ocupação da tabela é superior a 0.5,

$$\begin{aligned} s_T &> \tfrac{1}{2}\,c_T \\ 2\,s_T &> c_T \\ \Phi(T) &> 0\,. \end{aligned}$$

Potencial — Tabela (2)

Seja T uma tabela qualquer e sejam s_T o número de elementos em T e c_T a capacidade de T.

$$\Phi(T) = 2\,s_T - c_T\,.$$

Operação	**Custo Real**	**Dif. de Potencial**	**Custo Amortizado**
insert	c_i	$\Phi(T_i) - \Phi(T_{i-1})$	$\widehat{c}_i = c_i + \Delta\Phi$
não expande	1	2	3 $O(1)$
expande	$s+1$	$2-s$	3 $O(1)$

(**Notação:** (s, c) antes / (s', c') depois da operação.)

Se expande (ou seja, quando $s = c$):

- $\Phi(T_{i-1}) = 2\,s - c = s$.
- $\Phi(T_i) = 2\,s' - c' = 2\,(s+1) - 2\,c = 2\,s + 2 - 2\,c = 2$;

Potencial — Tabela (3)

Seja T uma tabela qualquer e sejam s_T o número de elementos em T e c_T a capacidade de T.

$$\Phi(T) \;=\; 2\,s_T \;-\; c_T\,.$$

Operação	**Custo Real**	**Dif. de Potencial**	**Custo Amortizado**
insert	c_i	$\Phi(T_i) - \Phi(T_{i-1})$	$\hat{c}_i = c_i + \Delta\Phi$
não expande	1	2	3 $O(1)$
expande	$s+1$	$2-s$	3 $O(1)$

(**Notação:** (s, c) antes / (s', c') depois da operação.)

Conclusões: a complexidade de uma sequência de n operações de insert, numa tabela inicialmente vazia, é

Potencial — Tabela (3)

Seja T uma tabela qualquer e sejam s_T o número de elementos em T e c_T a capacidade de T.

$$\Phi(T) = 2\,s_T - c_T\,.$$

Operação	**Custo Real**	**Dif. de Potencial**	**Custo Amortizado**
insert	c_i	$\Phi(T_i) - \Phi(T_{i-1})$	$\widehat{c}_i = c_i + \Delta\Phi$
não expande	1	2	3 $O(1)$
expande	$s+1$	$2-s$	3 $O(1)$

(**Notação:** (s, c) antes / (s', c') depois da operação.)

Conclusões: a complexidade de uma sequência de n operações de insert, numa tabela inicialmente vazia, é $O(n)$;

a complexidade amortizada do insert é $O(1)$.

Complexidade no PIOR CASO

de U operações de **união** e R operações de **representante**

(com n elementos)

União por Nível ou por Tamanho $\quad \Theta(1)$

Representante com Compressão do Caminho $\quad O(\log n)$

$\Big\}\ O(k\ \alpha(k,n))$

se $k = U + R \geq n$ [Tarjan 75].

Complexidade no PIOR CASO

de U operações de **união** e R operações de **representante**

(com n elementos)

União por Nível ou por Tamanho $\quad \Theta(1)$

Representante com Compressão do Caminho $\quad O(\log n)$

$\Big\}\ O(k\ \alpha(k,n))$

se $k = U + R \geq n$ [Tarjan 75].

Resultados: a complexidade de uma sequência de k operações de find (com compressão do caminho) e union (por nível ou por tamanho), numa partição com n elementos, acabada de criar, é $O(k\ \alpha(k,n))$, se $k \geq n$;

a complexidade amortizada do find e do union é $O(\alpha(k,n))$.