

# Capítulo VIII

## Complexidade Amortizada

(Amortized Analysis)

# Complexidade Amortizada

**Objetivo:** analisar o **custo de uma sequência** de operações numa estrutura de dados (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.

# Pilha com MultiDesempilha

```
void push( E element );           // Pior caso:  $\Theta(1)$ .

E pop( );                         // Pior caso:  $\Theta(1)$ .

void multiPop( int k ) {          // Pior caso:
    while ( !this.isEmpty() && k > 0 ) { // onde  $s$  é o número de
        E element = this.pop();        // elementos na pilha.
        k--;
    }
}
```

# Pilha com MultiDesempilha

```
void push( E element );           // Pior caso:  $\Theta(1)$ .

E pop( );                         // Pior caso:  $\Theta(1)$ .

void multiPop( int k ) {          // Pior caso:  $\Theta(\min(s, k))$ 
    while ( !this.isEmpty() && k > 0 ) { // onde  $s$  é o número de
        E element = this.pop();        // elementos na pilha.
        k--;
    }
}
```

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de `push`, `pop` e `multiPop`, numa pilha inicialmente vazia?

# Contador Binário

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

|     |     |     |     |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 0   | 0   | 1   |
| 0   | 0   | 1   | 0   |
| 0   | 0   | 1   | 1   |
| 0   | 1   | 0   | 0   |
| ... | ... | ... | ... |
| 1   | 1   | 1   | 1   |
| 0   | 0   | 0   | 0   |

# Contador Binário

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

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de `increment`, num contador inicialmente a zero?

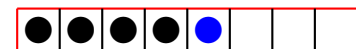
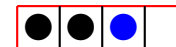
|     |     |     |     |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 0   | 0   | 1   |
| 0   | 0   | 1   | 0   |
| 0   | 0   | 1   | 1   |
| 0   | 1   | 0   | 0   |
| ... | ... | ... | ... |
| 1   | 1   | 1   | 1   |
| 0   | 0   | 0   | 0   |

# Tabela Dinâmica

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

```
void insert( E element ) {    // Pior caso:
    if ( table == null )      // s é o número de elementos na tabela.
        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;
}
```

/

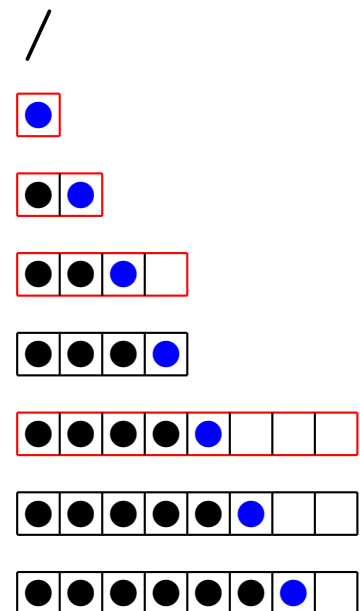


# Tabela Dinâmica

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

```
void insert( E element ) { // Pior caso:  $\Theta(s)$ , onde
    if ( table == null ) //  $s$  é o número de elementos na tabela.
        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;
}
```

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de **insert**, numa tabela inicialmente vazia?





# Métodos Existentes

- Há três métodos (com algumas variantes).
  - **Agregação.**
  - **Contabilidade.**
  - **Potencial:** é o mais versátil e o único que será estudado.
- 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.**

# Ideia Geral do Método do Potencial

- Define-se uma **função potencial**  $\Phi$ , que atribui a cada ED  $D$  um número real  $\Phi(D)$ .
- Prova-se que a **função potencial**  $\Phi$  verifica algumas propriedades.
- Sejam:
  - $D$  uma estrutura de dados,
  - $c$  o custo real da operação efetuada sobre  $D$  e
  - $D'$  a estrutura de dados resultante.

O **custo amortizado** da operação, que se denota por  $\hat{c}$ , é:

$$\hat{c} = c + \Phi(D') - \Phi(D).$$

- O custo amortizado de uma operação pode ser superior, igual ou inferior ao custo real da operação.

# Justificação do Método do Potencial (1)

- Para cada  $i = 1, 2, \dots, n$ , 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

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

- o **custo total amortizado** (da sequência de  $n$  operações) é

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

# Justificação do Método do Potencial (2)

- É necessário garantir que **o custo total amortizado nunca é inferior ao custo total real**. Como o custo total amortizado é

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

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

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

- Na prática, define-se  $\Phi$  de forma a que:

$$\textbf{(P1)} \quad \Phi(D_0) = 0; \text{ e}$$

$$\textbf{(P2)} \quad \Phi(D_i) \geq 0, \text{ para qualquer } i \geq 1.$$

E diz-se que  $\Phi$  é uma função potencial **válida**.

# Aplicação do Método do Potencial

- Define-se uma **função potencial**  $\Phi$ , que atribui a cada ED  $D$  um número real  $\Phi(D)$ .
- Prova-se que a **função potencial**  $\Phi$  é **válida**, i.e., que  $\Phi$  verifica as propriedades:

**(P1)**  $\Phi(D_0) = 0$ , onde  $D_0$  é a ED inicial (acabada de criar); e

**(P2)**  $\Phi(D) \geq 0$ , para qualquer ED  $D$  (podendo-se excluir  $D_0$ ).

- Calcula-se o **custo amortizado**  $\hat{c}$  de cada operação com a equação:

$$\hat{c} = c + \Phi(D') - \Phi(D)$$

onde  $c$  é o custo real da operação,  $D$  é a estrutura de dados **antes** da operação e  $D'$  é a estrutura de dados **depois** da operação.

# Pilha com MultiDesempilha

```
void push( E element );           // Pior caso:  $\Theta(1)$ .

E pop( );                         // Pior caso:  $\Theta(1)$ .

void multiPop( int k ) {          // Pior caso:  $\Theta(\min(s, k))$ 
    while ( !this.isEmpty() && k > 0 ) { // onde  $s$  é o número de
        E element = this.pop();        // elementos na pilha.
        k--;
    }
}
```

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de `push`, `pop` e `multiPop`, numa pilha inicialmente vazia?

# Potencial — Pilha (1)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

A função  $\Phi$  é **válida**:

**(P1)**  $\Phi(P_0) = 0$ , onde  $P_0$  é a pilha inicial,  
porque  $P_0$  é uma pilha vazia (tem zero elementos).

**(P2)**  $\Phi(P) \geq 0$ , para qualquer pilha  $P$ ,  
porque o número de elementos em  $P$  não pode ser negativo.

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

O custo amortizado de uma operação é  $\hat{c} = c + \Delta\Phi$ .

# Potencial — Pilha (2)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

| Operação     | Custo Real<br>$c$ | Dif. de Potencial<br>$\Phi(P') - \Phi(P)$ | Custo Amortizado<br>$\hat{c} = c + \Delta\Phi$ |
|--------------|-------------------|-------------------------------------------|------------------------------------------------|
| push         | 1                 |                                           |                                                |
| pop          | 1                 |                                           |                                                |
| multiPop $k$ | $\min(k, s)$      |                                           |                                                |

**Notação:**  $s$  antes /  $s'$  depois da operação



# Potencial — Pilha (3)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

| Operação            | Custo Real<br>$c$ | Dif. de Potencial<br>$\Phi(P') - \Phi(P)$ | Custo Amortizado<br>$\hat{c} = c + \Delta\Phi$ |
|---------------------|-------------------|-------------------------------------------|------------------------------------------------|
| <b>push</b>         | 1                 | $(s + 1) - s = 1$                         | $1 + 1 = 2 \quad O(1)$                         |
| <b>pop</b>          | 1                 |                                           |                                                |
| <b>multiPop</b> $k$ | $\min(k, s)$      |                                           |                                                |

**Notação:**  $s$  antes /  $s'$  depois da operação

# Potencial — Pilha (4)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

| Operação            | Custo Real<br>$c$ | Dif. de Potencial<br>$\Phi(P') - \Phi(P)$ | Custo Amortizado<br>$\hat{c} = c + \Delta\Phi$ |
|---------------------|-------------------|-------------------------------------------|------------------------------------------------|
| <b>push</b>         | 1                 | $(s + 1) - s = 1$                         | <b>2</b> $O(1)$                                |
| <b>pop</b>          | 1                 | $(s - 1) - s = -1$                        | <b>0</b> $O(1)$                                |
| <b>multiPop</b> $k$ | $\min(k, s)$      |                                           |                                                |

**Notação:**  $s$  antes /  $s'$  depois da operação

# Potencial — Pilha (5)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

| Operação            | Custo Real<br>$c$ | Dif. de Potencial<br>$\Phi(P') - \Phi(P)$ | Custo Amortizado<br>$\hat{c} = c + \Delta\Phi$ |
|---------------------|-------------------|-------------------------------------------|------------------------------------------------|
| <b>push</b>         | 1                 | $(s + 1) - s = 1$                         | 2 $O(1)$                                       |
| <b>pop</b>          | 1                 | $(s - 1) - s = -1$                        | 0 $O(1)$                                       |
| <b>multiPop</b> $k$ | $\min(k, s)$      | $-\min(k, s)$                             | 0 $O(1)$                                       |

**Notação:**  $s$  antes /  $s'$  depois da operação

**Diferença de Potencial de **multiPop**  $k$ :**

$$s' - s = (s - \min(k, s)) - s = -\min(k, s)$$

# Potencial — Pilha (6)

Seja  $P$  uma pilha qualquer e seja  $s$  o número de elementos em  $P$ .

$$\Phi(P) = s.$$

| Operação            | Custo Real<br>$c$ | Dif. de Potencial<br>$\Phi(P') - \Phi(P)$ | Custo Amortizado<br>$\hat{c} = c + \Delta\Phi$ |
|---------------------|-------------------|-------------------------------------------|------------------------------------------------|
| <b>push</b>         | 1                 | $(s + 1) - s = 1$                         | 2 $O(1)$                                       |
| <b>pop</b>          | 1                 | $(s - 1) - s = -1$                        | 0 $O(1)$                                       |
| <b>multiPop</b> $k$ | $\min(k, s)$      | $-\min(k, s)$                             | 0 $O(1)$                                       |

**Notação:**  $s$  antes /  $s'$  depois da operação

**Conclusões:** A complexidade amortizada do **push**, do **pop** e do **multiPop** é  $O(1)$ . A complexidade de uma sequência de  $n$  operações de **push**, **pop** e **multiPop**, numa pilha inicialmente vazia, é  $O(n)$ .

# Pilha — Exemplo

|            | Estado da Pilha   | Custo Real da Operação |       | Custo Amortizado da Operação |       |
|------------|-------------------|------------------------|-------|------------------------------|-------|
|            |                   |                        | Total |                              | Total |
| Criação    |                   |                        |       |                              |       |
| push $e_1$ | $e_1$             | 1                      | 1     | 2                            | 2     |
| push $e_2$ | $e_1 e_2$         | 1                      | 2     | 2                            | 4     |
| push $e_3$ | $e_1 e_2 e_3$     | 1                      | 3     | 2                            | 6     |
| pop        | $e_1 e_2$         | 1                      | 4     | 0                            | 6     |
| push $e_4$ | $e_1 e_2 e_4$     | 1                      | 5     | 2                            | 8     |
| push $e_5$ | $e_1 e_2 e_4 e_5$ | 1                      | 6     | 2                            | 10    |
| mPop 4     |                   | 4                      | 10    | 0                            | 10    |
| push $e_6$ | $e_6$             | 1                      | 11    | 2                            | 12    |
| mPop 3     |                   | 1                      | 12    | 0                            | 12    |
| push $e_7$ | $e_7$             | 1                      | 13    | 2                            | 14    |

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

# Contador Binário

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

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de `increment`, num contador inicialmente a zero?

|     |     |     |     |
|-----|-----|-----|-----|
| 0   | 0   | 0   | 0   |
| 0   | 0   | 0   | 1   |
| 0   | 0   | 1   | 0   |
| 0   | 0   | 1   | 1   |
| 0   | 1   | 0   | 0   |
| ... | ... | ... | ... |
| 1   | 1   | 1   | 1   |
| 0   | 0   | 0   | 0   |

# Potencial — Contador (1)

Seja  $C$  um contador qualquer e seja  $u$  o número de UNS em  $C$ .

$$\Phi(C) = u.$$

A função  $\Phi$  é **válida**:

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

**(P2)**  $\Phi(C) \geq 0$ , para qualquer contador  $C$ ,  
porque o número de UNS em  $C$  não pode ser negativo.

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

O custo amortizado de uma operação é  $\hat{c} = c + \Delta\Phi$ .

# Potencial — Contador (2)

Seja  $C$  um contador qualquer e seja  $u$  o número de UNS em  $C$ .

$$\Phi(C) = u.$$

| Operação   | Custo Real | Dif. de Potencial    | Custo Amortizado           |
|------------|------------|----------------------|----------------------------|
| increment  | $c$        | $\Phi(C') - \Phi(C)$ | $\hat{c} = c + \Delta\Phi$ |
| incrementa | $k + 1$    |                      |                            |
| anula      | $k$        |                      |                            |

$k$  é o número de UNS que passam a ZERO

**Notação:**  $u$  antes /  $u'$  depois da operação



# Potencial — Contador (3)

Seja  $C$  um contador qualquer e seja  $u$  o número de UNS em  $C$ .

$$\Phi(C) = u.$$

| Operação   | Custo Real | Dif. de Potencial    | Custo Amortizado           |
|------------|------------|----------------------|----------------------------|
| increment  | $c$        | $\Phi(C') - \Phi(C)$ | $\hat{c} = c + \Delta\Phi$ |
| incrementa | $k + 1$    | $-k + 1$             | 2                          |
| anula      | $k$        |                      |                            |

$k$  é o número de UNS que passam a ZERO

**Notação:**  $u$  antes /  $u'$  depois da operação

**Diferença de Potencial quando incrementa:**

$$u' - u = (u - k + 1) - u = -k + 1$$

# Potencial — Contador (4)

Seja  $C$  um contador qualquer e seja  $u$  o número de UNS em  $C$ .

$$\Phi(C) = u.$$

| Operação   | Custo Real | Dif. de Potencial    | Custo Amortizado                                           |
|------------|------------|----------------------|------------------------------------------------------------|
| increment  | $c$        | $\Phi(C') - \Phi(C)$ | $\hat{c} = c + \Delta\Phi$                                 |
| incrementa | $k + 1$    | $-k + 1$             | $\left. \begin{array}{c} 2 \\ 0 \end{array} \right\} O(1)$ |
| anula      | $k$        | $-k$                 |                                                            |

$k$  é o número de UNS que passam a ZERO

**Notação:**  $u$  antes /  $u'$  depois da operação

**Diferença de Potencial quando anula:**

$$u' - u = (u - k) - u = -k$$

# Potencial — Contador (5)

Seja  $C$  um contador qualquer e seja  $u$  o número de UNS em  $C$ .

$$\Phi(C) = u.$$

| Operação   | Custo Real | Dif. de Potencial    | Custo Amortizado                                           |
|------------|------------|----------------------|------------------------------------------------------------|
| increment  | $c$        | $\Phi(C') - \Phi(C)$ | $\hat{c} = c + \Delta\Phi$                                 |
| incrementa | $k + 1$    | $-k + 1$             | $\left. \begin{array}{c} 2 \\ 0 \end{array} \right\} O(1)$ |
| anula      | $k$        | $-k$                 |                                                            |

$k$  é o número de UNS que passam a ZERO

**Notação:**  $u$  antes /  $u'$  depois da operação

**Conclusões:** A complexidade amortizada do increment é  $O(1)$ . A complexidade de uma sequência de  $n$  operações de increment, num contador inicialmente a zero, é  $O(n)$ .

# Contador — Exemplo

|         | Estado<br>do<br>Contador | Custo Real<br>da<br>Operação | Total | Custo Amortizado<br>da<br>Operação | Total |
|---------|--------------------------|------------------------------|-------|------------------------------------|-------|
| Criação | 0 0 0                    |                              |       |                                    |       |
| incr.   | 0 0 1                    | 1                            | 1     | 2                                  | 2     |
| incr.   | 0 1 0                    | 2                            | 3     | 2                                  | 4     |
| incr.   | 0 1 1                    | 1                            | 4     | 2                                  | 6     |
| incr.   | 1 0 0                    | 3                            | 7     | 2                                  | 8     |
| incr.   | 1 0 1                    | 1                            | 8     | 2                                  | 10    |
| incr.   | 1 1 0                    | 2                            | 10    | 2                                  | 12    |
| incr.   | 1 1 1                    | 1                            | 11    | 2                                  | 14    |
| incr.   | 0 0 0                    | 3                            | 14    | 0                                  | 14    |
| incr.   | 0 0 1                    | 1                            | 15    | 2                                  | 16    |

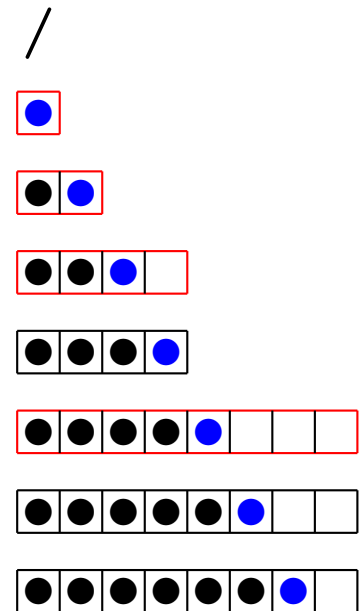
○ **custo total amortizado** nunca é inferior ao **custo total real**.

# Tabela Dinâmica

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

```
void insert( E element ) { // Pior caso:  $\Theta(s)$ , onde
    if ( table == null ) //  $s$  é o número de elementos na tabela.
        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;
}
```

Qual é a complexidade (no pior caso) de uma sequência de  $n$  operações de **insert**, numa tabela inicialmente vazia?



# Potencial — Tabela (1)

Seja  $T$  uma tabela qualquer e sejam  $s$  o número de elementos em  $T$  e  $l$  a capacidade de  $T$ .

$$\Phi(T) = 2s - l.$$

A função  $\Phi$  é **válida**:

**(P1)**  $\Phi(T_0) = 0$ , onde  $T_0$  é a tabela inicial,  
porque  $T_0$  tem 0 elementos e capacidade 0.

**(P2)**  $\Phi(T) \geq 0$ , para qualquer tabela  $T$  exceto a inicial.  
Como o fator de ocupação da tabela é superior a 0.5,

$$\begin{aligned} s &> \frac{1}{2} l \\ 2s &> l \\ \Phi(T) &> 0. \end{aligned}$$

# Potencial — Tabela (2)

Seja  $T$  uma tabela qualquer e sejam  $s$  o número de elementos em  $T$  e  $l$  a capacidade de  $T$ .

$$\Phi(T) = 2s - l.$$

| Operação    | Custo Real | Dif. de Potencial    | Custo Amortizado           |
|-------------|------------|----------------------|----------------------------|
| insert      | $c$        | $\Phi(T') - \Phi(T)$ | $\hat{c} = c + \Delta\Phi$ |
| não expande | 1          |                      |                            |
| expande     | $s + 1$    |                      |                            |

**Notação:**  $(s, l)$  antes /  $(s', l')$  depois da operação

# Potencial — Tabela (3)

Seja  $T$  uma tabela qualquer e sejam  $s$  o número de elementos em  $T$  e  $l$  a capacidade de  $T$ .

$$\Phi(T) = 2s - l.$$

| Operação      | Custo Real | Dif. de Potencial    | Custo Amortizado           |
|---------------|------------|----------------------|----------------------------|
| <b>insert</b> | $c$        | $\Phi(T') - \Phi(T)$ | $\hat{c} = c + \Delta\Phi$ |
| não expande   | 1          | 2                    | 3                          |
| expande       | $s + 1$    |                      |                            |

**Notação:**  $(s, l)$  antes /  $(s', l')$  depois da operação

**Diferença de Potencial quando não expande:**

$$\begin{aligned}(2s' - l') - (2s - l) &= (2(s + 1) - l) - (2s - l) \\ &= 2s + 2 - l - 2s + l = 2\end{aligned}$$



# Potencial — Tabela (4)

Seja  $T$  uma tabela qualquer e sejam  $s$  o número de elementos em  $T$  e  $l$  a capacidade de  $T$ .

$$\Phi(T) = 2s - l.$$

| Operação      | Custo Real | Dif. de Potencial    | Custo Amortizado                                          |
|---------------|------------|----------------------|-----------------------------------------------------------|
| <b>insert</b> | $c$        | $\Phi(T') - \Phi(T)$ | $\hat{c} = c + \Delta\Phi$                                |
| não expande   | 1          | 2                    | $\left. \begin{matrix} 3 \\ 3 \end{matrix} \right\} O(1)$ |
| expande       | $s + 1$    | $2 - s$              |                                                           |

**Notação:**  $(s, l)$  antes /  $(s', l')$  depois da operação

**Diferença de Potencial quando **expande** (e  $s = l$ ):**

$$\begin{aligned}
 (2s' - l') - (2s - l) &= (2(s + 1) - 2l) - (2s - l) \\
 &= 2s + 2 - 2l - 2s + l = 2 - l = 2 - s
 \end{aligned}$$

# Potencial — Tabela (5)

Seja  $T$  uma tabela qualquer e sejam  $s$  o número de elementos em  $T$  e  $l$  a capacidade de  $T$ .

$$\Phi(T) = 2s - l.$$

| Operação      | Custo Real | Dif. de Potencial    | Custo Amortizado                                          |
|---------------|------------|----------------------|-----------------------------------------------------------|
| <b>insert</b> | $c$        | $\Phi(T') - \Phi(T)$ | $\hat{c} = c + \Delta\Phi$                                |
| não expande   | 1          | 2                    | $\left. \begin{matrix} 3 \\ 3 \end{matrix} \right\} O(1)$ |
| expande       | $s + 1$    | $2 - s$              |                                                           |

**Notação:**  $(s, l)$  antes /  $(s', l')$  depois da operação

**Conclusões:** A complexidade amortizada do **insert** é  $O(1)$ . A complexidade de uma sequência de  $n$  operações de **insert**, numa tabela inicialmente vazia, é  $O(n)$ .

# Tabela — Exemplo

|            | Estado da Tabela |      | Custo Real da Operação |       | Custo Amortizado da Operação |       |
|------------|------------------|------|------------------------|-------|------------------------------|-------|
|            | length           | size |                        | Total |                              | Total |
| Inicial    | 0                | 0    |                        |       |                              |       |
| ins. $e_1$ | 1                | 1    | 1                      | 1     | 3                            | 3     |
| ins. $e_2$ | 2                | 2    | 2                      | 3     | 3                            | 6     |
| ins. $e_3$ | 4                | 3    | 3                      | 6     | 3                            | 9     |
| ins. $e_4$ | 4                | 4    | 1                      | 7     | 3                            | 12    |
| ins. $e_5$ | 8                | 5    | 5                      | 12    | 3                            | 15    |
| ins. $e_6$ | 8                | 6    | 1                      | 13    | 3                            | 18    |
| ins. $e_7$ | 8                | 7    | 1                      | 14    | 3                            | 21    |
| ins. $e_8$ | 8                | 8    | 1                      | 15    | 3                            | 24    |
| ins. $e_9$ | 16               | 9    | 9                      | 24    | 3                            | 27    |

○ **custo total amortizado** nunca é inferior ao **custo total real**.

Complexidade no PIOR CASO de  
 $U$  operações de **reunião** e  $R$  operações de **representante**  
(com  $n$  elementos)

|                                                     |             |                       |
|-----------------------------------------------------|-------------|-----------------------|
| Reunião por <b>Nível</b> (código Altura) ou Tamanho | $\Theta(1)$ | } $O(k \alpha(k, n))$ |
| Representante com Compressão do Caminho             | $O(\log n)$ |                       |

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

Complexidade no PIOR CASO de  
 $U$  operações de **reunião** e  $R$  operações de **representante**  
 (com  $n$  elementos)

|                                                     |             |                       |
|-----------------------------------------------------|-------------|-----------------------|
| Reunião por <b>Nível</b> (código Altura) ou Tamanho | $\Theta(1)$ | } $O(k \alpha(k, n))$ |
| Representante com Compressão do Caminho             | $O(\log n)$ |                       |

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

**Resultados:** a complexidade amortizada do **find** (com compressão do caminho) e do **union** (por nível ou por tamanho) é  $O(\alpha(k, n))$ , se se realizarem  $k \geq n$  operações.

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  $k \geq n$ .