

# Análise de Algoritmos

**Parte destes slides são adaptações de slides  
do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.**

# Análise amortizada

CLRS 17

# Análise amortizada

Serve para analisar uma sequência de operações ou iterações onde o pior caso individual não reflete o pior caso da sequência.

Em outras palavras, serve para melhorar análises de pior caso que baseiem-se diretamente no pior caso de uma operação/iteração e que deem uma delimitação frouxa para o tempo de pior caso da sequência.

## Métodos:

- agregado
- por créditos
- potencial

# Odômetro binário

Considere um **contador binário**, inicialmente zerado, representado em um vetor  $A[0..n-1]$ , onde cada  $A[i]$  vale 0 ou 1.

**Operação:** incrementa.

INCREMENTA ( $A, n$ )

1  $i \leftarrow 0$

2 **enquanto**  $i < n$  **e**  $A[i] = 1$  **faça**

3      $A[i] \leftarrow 0$

4      $i \leftarrow i + 1$

5     **se**  $i < n$

6         **então**  $A[i] \leftarrow 1$

**Consumo de tempo no pior caso:**  $\Theta(n)$

# Odômetro binário

Considere um **contador binário**, inicialmente zerado, representado em um vetor  $A[0..n-1]$ , onde cada  $A[i]$  vale 0 ou 1.

**Operação:** INCREMENTA.

**Consumo de tempo no pior caso:**  $\Theta(n)$ .

O odômetro dá uma **volta completa** a cada  $2^n$  execuções do INCREMENTA.

Quanto tempo leva para o odômetro dar uma volta completa?

Leva  $O(n2^n)$ .

Será que é  $\Theta(n2^n)$ ?

# Odômetro binário

<i>i</i>	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

# Odômetro binário

$i$	3	2	1	0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

# Método agregado

Custo da volta completa é proporcional ao número de vezes que os bits são alterados.

bit 0 muda  $2^n$  vezes

bit 1 muda  $2^{n-1}$  vezes

bit 2 muda  $2^{n-2}$  vezes

...

bit  $n - 2$  muda 4 vezes

bit  $n - 1$  muda 2 vezes

Total de alterações de bits:  $\sum_{i=1}^n 2^i < 2 \cdot 2^n$ .

Custo da volta completa:  $\Theta(2^n)$ .

Custo amortizado por INCREMENTA:  $\Theta(1)$ .

# Análise por créditos

Atribuimos um número fixo de créditos por operação **INCREMENTA** de modo a pagar por toda alteração de bit.

**Objetivo:** atribuir o **menor número possível de créditos** que seja ainda suficiente para pagar por todas as alterações.

Relembre...

**INCREMENTA** ( $A, n$ )

1      $i \leftarrow 0$

2     **enquanto**  $i < n$  **e**  $A[i] = 1$  **faça**

3          $A[i] \leftarrow 0$

5          $i \leftarrow i + 1$

6     **se**  $i < n$

7         **então**  $A[i] \leftarrow 1$

# Análise por créditos

Atribuimos **2 créditos** por **INCREMENTA**.

**INCREMENTA** ( $A, n$ )

1  $i \leftarrow 0$

2 **enquanto**  $i < n$  **e**  $A[i] = 1$  **faça**

3  $A[i] \leftarrow 0$

4  $i \leftarrow i + 1$

5 **se**  $i < n$

6 **então**  $A[i] \leftarrow 1$

Um é usado para pagar pela alteração da linha 6.

O outro fica armazenado sobre o bit alterado na linha 6.

Há um crédito armazenado sobre cada bit que vale 1.

Alterações da linha 3 são pagas por créditos armazenados por chamadas anteriores do **INCREMENTA**.

# Análise por créditos

Atribuimos **2 créditos** por INCREMENTA.

INCREMENTA ( $A, n$ )

1  $i \leftarrow 0$

2 **enquanto**  $i < n$  **e**  $A[i] = 1$  **faça**

3  $A[i] \leftarrow 0$

4  $i \leftarrow i + 1$

5 **se**  $i < n$

6 **então**  $A[i] \leftarrow 1$

Um é usado para pagar pela alteração da linha 6.

O outro fica armazenado sobre o bit alterado na linha 6.

O número de créditos armazenados em cada instante é o número de bits que valem 1, logo é sempre não negativo.

**Custo amortizado por INCREMENTA:** 2

# Método do potencial

Seja  $\phi(A)$  o número de bits que valem 1 em  $A[0..n-1]$ .

Seja  $A_i$  o estado do contador  $A$  após o  $i$ -ésimo **INCREMENTA**.

Note que  $\phi(A_0) = 0$  e  $\phi(A_i) \geq 0$ .

Seja  $c_i$  o número de bits alterados no  $i$ -ésimo **INCREMENTA**.

Note que  $c_i \leq 1 + t_i$  onde  $t_i$  é o número de bits 1 consecutivos no final do contador  $A$ .

Note que  $\phi(A_i) - \phi(A_{i-1}) \leq 1 - t_i$ .

Seja  $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$ .

# Método do potencial

Seja  $\phi(A)$  o número de bits que valem 1 em  $A[0..n-1]$ .

Seja  $A_i$  o estado do contador  $A$  após o  $i$ -ésimo **INCREMENTA**.  
Temos que  $\phi(A_0) = 0$  e  $\phi(A_i) \geq 0$ .

Seja  $c_i$  o número de bits alterados no  $i$ -ésimo **INCREMENTA** e  $t_i$  é o número de bits 1 consecutivos no final do contador  $A$ .  
Temos que  $c_i \leq 1 + t_i$ .

Seja  $\hat{c}_i = c_i + \phi(A_i) - \phi(A_{i-1}) \leq (1 + t_i) + (1 - t_i) = 2$ .

Então o custo da volta completa é

$$c = \sum_{i=1}^{2^n} c_i = \sum_{i=1}^{2^n} \hat{c}_i + \phi(A_0) - \phi(A_{2^n}) \leq \sum_{i=1}^{2^n} \hat{c}_i \leq 2 \cdot 2^n.$$

**Custo amortizado por INCREMENTA: 2**

▷ (valor do  $\hat{c}_i$ )

# Outro exemplo: pilha

Operações básica: empilha, desempilha.

# Outro exemplo: pilha

Operações básica: empilha, desempilha.

**OP**: operação única de acesso

# Outro exemplo: pilha

Operações básicas: empilha, desempilha.

**OP**: operação única de acesso

**OP** ( $n$ )

- 1  $\triangleright$  exige que a pilha tenha  $\geq n$  elementos
- 2 desempilhe  $n$  itens da pilha
- 3 empilhe um item na pilha

# Outro exemplo: pilha

Operações básicas: empilha, desempilha.

**OP**: operação única de acesso

**OP** ( $n$ )

- 1  $\triangleright$  exige que a pilha tenha  $\geq n$  elementos
- 2 desempilhe  $n$  itens da pilha
- 3 empilhe um item na pilha

Consumo de tempo de **OP**( $n$ ) é  $\Theta(n)$ .

# Outro exemplo: pilha

Operações básica: empilha, desempilha.

**OP**: operação única de acesso

**OP** ( $n$ )

- 1  $\triangleright$  exige que a pilha tenha  $\geq n$  elementos
- 2 desempilhe  $n$  itens da pilha
- 3 empilhe um item na pilha

Consumo de tempo de **OP**( $n$ ) é  $\Theta(n)$ .

Sequência de  $m$  operações **OP**.

Consumo de tempo de uma operação no pior caso:  $\Theta(m)$ .

# Outro exemplo: pilha

Operações básicas: empilha, desempilha.

**OP**: operação única de acesso

**OP** ( $n$ )

- 1  $\triangleright$  exige que a pilha tenha  $\geq n$  elementos
- 2 desempilhe  $n$  itens da pilha
- 3 empilhe um item na pilha

Consumo de tempo de **OP**( $n$ ) é  $\Theta(n)$ .

Sequência de  $m$  operações **OP**.

Consumo de tempo de uma operação no pior caso:  $\Theta(m)$ .

Qual o consumo total das  $m$  operações no pior caso?

# Outro exemplo: pilha

Consumo de tempo de uma operação no pior caso:  $\Theta(m)$ .

Qual o consumo das  $m$  operações no pior caso?  $\Theta(m^2)$ ?

# Outro exemplo: pilha

Consumo de tempo de uma operação no pior caso:  $\Theta(m)$ .

Qual o consumo das  $m$  operações no pior caso?  $\Theta(m^2)$ ?

Em aula...

- análise por créditos

Quantos créditos damos para cada chamada de OP?

# Outro exemplo: pilha

Consumo de tempo de uma operação no pior caso:  $\Theta(m)$ .

Qual o consumo das  $m$  operações no pior caso?  $\Theta(m^2)$ ?

Em aula...

- análise por créditos  
Quantos créditos damos para cada chamada de OP?
- análise por função potencial  
Qual seria uma boa função potencial neste caso?

# Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

# Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

# Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

# Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

A cada inserção em que o vetor está cheio, antes da inserção propriamente dita, um vetor do dobro do tamanho é alocado, o vetor anterior é copiado para o novo vetor e depois é desalocado.

# Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

A cada inserção em que o vetor está cheio, antes da inserção propriamente dita, um vetor do dobro do tamanho é alocado, o vetor anterior é copiado para o novo vetor e depois é desalocado.

O custo no pior caso de uma inserção é alto, pois pode haver uma **realocação**.

# Tabelas dinâmicas

Para  $i = 0, 1, 2, \dots, n - 1$ ,

$$c_i = \begin{cases} 1 & \text{se } i \text{ não é potência de } 2 \\ i + 1 & \text{se } i \text{ é potência de } 2 \end{cases}$$

# Tabelas dinâmicas

Para  $i = 0, 1, 2, \dots, n - 1$ ,

$$c_i = \begin{cases} 1 & \text{se } i \text{ não é potência de } 2 \\ i + 1 & \text{se } i \text{ é potência de } 2 \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \dots + 2^k)$$

onde  $k = \lfloor \lg n \rfloor$ .

# Tabelas dinâmicas

Para  $i = 0, 1, 2, \dots, n - 1$ ,

$$c_i = \begin{cases} 1 & \text{se } i \text{ não é potência de } 2 \\ i + 1 & \text{se } i \text{ é potência de } 2 \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \dots + 2^k)$$

onde  $k = \lfloor \lg n \rfloor$ .

Logo  $\sum_{i=0}^{n-1} c_i = n + 2^{k+1} - 1 \leq n + 2n - 1 < 3n$ .

# Tabelas dinâmicas

Para  $i = 0, 1, 2, \dots, n - 1$ ,

$$c_i = \begin{cases} 1 & \text{se } i \text{ não é potência de } 2 \\ i + 1 & \text{se } i \text{ é potência de } 2 \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \dots + 2^k)$$

onde  $k = \lfloor \lg n \rfloor$ .

Logo  $\sum_{i=0}^{n-1} c_i = n + 2^{k+1} - 1 \leq n + 2n - 1 < 3n$ .

Custo amortizado por inserção: 3

# Análise por créditos

Chame de **velho** um item que já estava no vetor no momento da última realocação do vetor, e de **novos** os itens inseridos após a última realocação.

# Análise por créditos

Chame de **velho** um item que já estava no vetor no momento da última realocação do vetor, e de **novos** os itens inseridos após a última realocação.

Atribuimos **3 créditos por inserção**:

um é usado para pagar pela inserção do item, os outros dois são armazenados sobre o item.

# Análise por créditos

Chame de **velho** um item que já estava no vetor no momento da última realocação do vetor, e de **novos** os itens inseridos após a última realocação.

Atribuimos **3 créditos por inserção**:

um é usado para pagar pela inserção do item, os outros dois são armazenados sobre o item.

Ao ocorrer uma **realocação**, há 2 créditos sobre cada item novo no vetor, e isso é suficiente para pagar pela cópia de todos os itens do vetor para o novo vetor pois, quando o vetor está cheio, há um item novo para cada item velho.

# Análise por créditos

Chame de **velho** um item que já estava no vetor no momento da última realocação do vetor, e de **novos** os itens inseridos após a última realocação.

Atribuimos **3 créditos por inserção**:

um é usado para pagar pela inserção do item, os outros dois são armazenados sobre o item.

Ao ocorrer uma **realocação**, há 2 créditos sobre cada item novo no vetor, e isso é suficiente para pagar pela cópia de todos os itens do vetor para o novo vetor pois, quando o vetor está cheio, há um item novo para cada item velho.

Em outras palavras, o segundo crédito paga a cópia do item na primeira realocação que acontecer após a sua inserção, e o terceiro crédito paga a cópia de um item velho nesta mesma realocação.