
Análise de Algoritmos

Aula 01

Prof. Marco Aurélio Stefanos

`marco@dct.ufms.br`

Ramal 7507 Sala 9

`www.dct.ufms.br/~marco`

Departamento de Computação e Estatística

Universidade Federal de Mato Grosso do Sul

Agradecimentos aos Profs José Coelho Pina Jr., Edson Cáceres e Paulo Fiofiloff por
cederem parte do material do curso.

Análise de Algoritmos – p. 1

Programa

- Ordenação

Análise de Algoritmos – p. 2

Programa

- Ordenação
- Programação Dinâmica

Análise de Algoritmos – p. 2

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso

Análise de Algoritmos – p. 2

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso
- Algoritmos em Grafos

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso
- Algoritmos em Grafos
- Análise Amortizada
- String Matching

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso
- Algoritmos em Grafos
- Análise Amortizada

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso
- Algoritmos em Grafos
- Análise Amortizada
- String Matching
- Problemas NP-Completo

Programa

- Ordenação
- Programação Dinâmica
- Método Guloso
- Algoritmos em Grafos
- Análise Amortizada
- String Matching
- Problemas NP-Completo
- Outros tópicos

Avaliações e Datas

- P1 24/09
- P2 19/11
- PO 26/11
- EF 03/12
- $MP = (P1+P2)/2$
- $MA = 0,8MP+0,2Listas$
- $MF = (MA+EF)/2$

Bibliografia

1. Cormen, Leiserson, Rivest e Stein, *Introduction to Algorithms*, 2a. Ed., MIT Press, 2001
2. Goodrich e Tamassia, *Algorithm Design - Foundations, Analysis and Internet Examples*, John Wiley & sons, 2002
3. Sedgewick, *Algorithms in C*, 3a. Ed., Addison-Wesley, 1998
4. Viziani, *Projeto de Algoritmos com Implementações em Pascal e C*, 2a. Ed., Thompson, 2004
5. Manber, *Algorithms: A Creative Approach*, Addison-Wesley, 1989

Análise de Algoritmos

Estudo teórico de desempenho de algoritmos e seus recursos computacionais

Razões para análise matemática dos algoritmos

- Comparar diferentes algoritmos
- Predição sobre desempenho
- Estabelecer limites para os algoritmos

Algoritmos

- Seqüência finita de passos bem definidos para resolver um problema
- Pseudo-código - precisão - correção
- Entrada: Especificação dos dados a serem processados (instância)
- Saída : Especificação do resultado produzido a partir da entrada
- Programa - Implementação de um algoritmo em alguma linguagem de programação

Estudar algoritmos e eficiência?

- Algoritmos ajuda a entender **escalabilidade**
- Eficiência ajuda a entender a fronteira entre o possível e o impossível
- Algoritmos dá **ferramentas** para estudarmos o comportamento dos programas
- Aprendizado sobre eficiência pode ser levado para outros recursos
- O certo e o justo!

O Problema da Ordenação

Entrada: Seqüência de números $\langle a_1, a_2, \dots, a_n \rangle$

Saída: $\langle a'_1, a'_2, \dots, a'_n \rangle$ permutação da entrada tal que $\langle a'_1 \leq a'_2 \leq \dots \leq a'_n \rangle$

Exemplo

Entrada: 3 4 9 2 4 8

Saída: 2 3 4 4 8 9

Exemplo

3 4 9 2 4 8

Exemplo

3 4 9 2 4 8

3 4 9 2 4 8

Exemplo

3 4 9 2 4 8

3 4 9 2 4 8

3 4 9 **2** 4 8
← ←

2 3 4 **9** **4** 8
← ←

Exemplo

3 4 9 2 4 8

3 4 9 2 4 8

3 4 9 **2** 4 8
← ←

Exemplo

3 4 9 2 4 8

3 4 9 2 4 8

3 4 9 **2** 4 8
← ←

2 3 4 **9** **4** 8
← ←

2 3 4 4 **9** **8**
← ←

Exemplo

3 4 9 2 4 8
3 4 9 2 4 8
3 4 9 2 4 8
2 3 4 9 4 8
2 3 4 4 9 8
2 3 4 4 8 9 Ok!

Análise de Algoritmos – p. 9

InsertionSort

Insertion-Sort $A[1..n]$

```
1:  $j = 2$ 
2: while  $j \leq n$  do
3:    $k = A[j]$ 
   { Insira  $A[j]$  na seqüência ordenada  $A[1..j-1]$  }
4:    $i = j - 1$ 
5:   while  $i > 0$  E  $A[i] > k$  do
6:      $A[i+1] = A[i]$ 
7:      $i = i - 1$ 
8:    $A[i+1] = k$ 
9:    $j = j + 1$ 
```

Análise de Algoritmos – p. 10

Correção de Algoritmos

- Um algoritmo está **correto** se para qualquer entrada válida ele pára e produz a saída desejada.
- Provas automáticas de algoritmos não são possíveis de serem feitas.
- Há técnicas práticas e formalismos que ajudam a mostrar a correção de algoritmos.

Vamos a um exemplo ...

Análise de Algoritmos – p. 11

Correção do InsertionSort

Relação **invariante** k :

(i_0) na linha 2 vale que: $A[1 \dots j-1]$ é crescente.

a_1	a_2	$\dots\dots$	a_{j-1}	a_j	a_{j+1}	$\dots\dots$	a_n
-------	-------	--------------	-----------	-------	-----------	--------------	-------

Supondo que a invariante vale. Concluimos que o algoritmo está correto.

No início da última iteração temos que $j = n + 1$, concluimos que $A[1..n]$ é crescente.

Exerc. Demonstrar invariante!

Análise de Algoritmos – p. 12

Correção de algoritmos iterativos

Em geral, a demonstração de correção de um algoritmo iterativo identifica-se uma invariante e os passos consistem em:

1. verificar que a invariante vale no início da primeira iteração
2. Mostrar que
Se a invariante vale no **início** então ela vale no **final**
3. Concluir que, se a invariante vale no início da última iteração junto com a condição de parada implicam na correção do algoritmo.

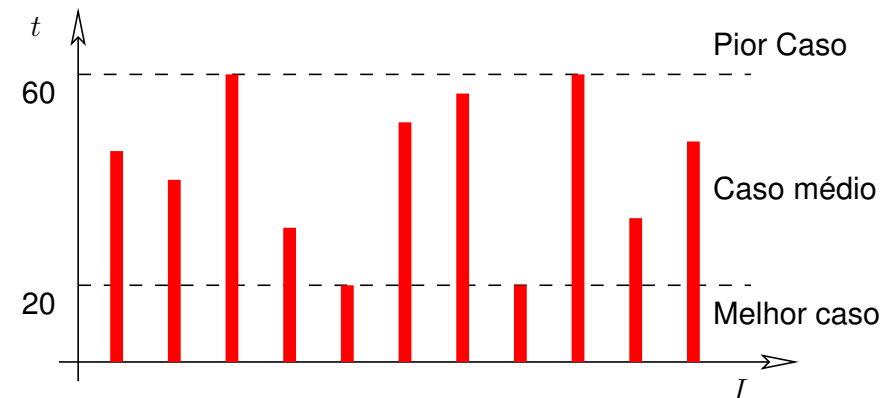
Análise de um algoritmo

- Empírica: medir tempo gasto para várias entradas. Dependente da máquina
- Analítico: Número de recursos primitivos o algoritmo utiliza
 - Eficiência: Tempo de execução e/ou espaço gasto
 - função da entrada de tamanho n
 - diferentes entradas do mesmo tamanho podem ter diferentes tempos de resposta

Tipos de análises

- de Pior Caso (realista)
 - $T(n)$ Tempo máximo do algoritmo para qualquer entrada de tamanho n
- de Caso Médio (difícil)
 - $T(n)$ Tempo esperado do algoritmo sobre todas as entradas de tamanho n
 - Análise de probabilidade das entradas
- de Melhor Caso (pouco importante)
 - $T(n)$ Tempo mínimo do algoritmo para qualquer entrada de tamanho n

Tempo de execução



Análise do InsertionSort

InsertionSort $A[1..n]$

$j = 2$	c_0	1
while $j \leq n$ do	c_1	n
$k = A[j]$	c_2	$n - 1$
$i = j - 1$	c_3	$n - 1$
while $i > 0$ E $A[i] > k$ do	c_4	$\sum t_j$
$A[i + 1] = A[i]$	c_5	$\sum t_{j-1}$
$i = i - 1$	c_6	$\sum t_{j-1}$
$A[i + 1] = k$	c_7	$n - 1$
$j = j + 1$	c_8	$n - 1$
	Custo	Vezes

Análise de Algoritmos – p. 17

Análise do InsertionSort

Seja $t_j :=$ número de vezes que o laço interno é executado para cada valor de j

$$T(n) = c_0 + c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j$$

$$+ (c_6 + c_7) \sum_{j=2}^n (t_j - 1) + c_8(n - 1) + c_9(n - 1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_8 + c_9)n - (c_2 + c_4 + c_8 + c_9 - c_0)$$

$$+ c_5 \sum_{j=2}^n t_j + (c_6 + c_7) \sum_{j=2}^n (t_j - 1)$$

Análise de Algoritmos – p. 18

Análise do InsertionSort

- Melhor caso $t_j = 1, j = 2, \dots, n$ (elementos já ordenados)

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8 + c_9)n - (c_2 + c_4 + c_5 + c_8 + c_9 - c_0)$$

$$T(n) = an + b$$

Análise de Algoritmos – p. 19

Análise do InsertionSort

- Melhor caso $t_j = 1, j = 2, \dots, n$ (elementos já ordenados)

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8 + c_9)n - (c_2 + c_4 + c_5 + c_8 + c_9 - c_0)$$

$$T(n) = an + b$$

- Pior caso $t_j = j, j = 2, \dots, n$ (elementos em ordem decrescente)

$$T(n) = (c_1 + c_2 + c_4 + c_8 + c_9)n - (c_2 + c_4 + c_8 + c_9 - c_0) + c_5 \frac{(n-1)(n+2)}{2} + (c_6 + c_7) \frac{(n-1)n}{2}$$

$$T(n) = (c_5 + c_6 + c_7)n^2/2 + (c_1 + c_2 + c_4 + c_8 + c_9 + \frac{c_5 - c_6 - c_7}{2})n - (c_2 + c_4 + c_5 + c_8 + c_9 - c_0)$$

$$T(n) = an^2 + bn + c$$

Análise de Algoritmos – p. 19