Algoritmos e Estrutura de Dados II

Análise de Algoritmos

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

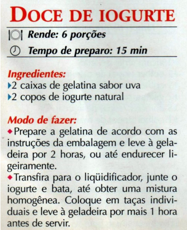
Agenda

1. Introdução
2. Tempo de Processamento
3. Exemplo: Soma Vetor
4. Exemplo: Busca Vetor
5. Exemplo: Elemento Máximo
6. Exemplo: Elemento Máximo e Mínimo
7. Comportamento Assintótico

O que é um algoritmo?

Um conjunto de instruções executáveis para resolver um problema.

- O problema é a motivação para o algoritmo.
- As instruções têm de ser executáveis.
- Geralmente existem vários algoritmos para um mesmo problema. Como escolher?
- Representação: descrição das instruções suficientes para que o leitor o entenda.



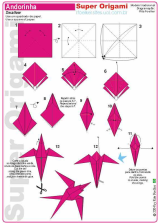
DOCE DE IOGURTE

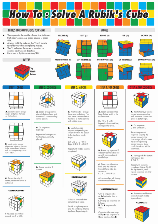
Rende: 6 porções

Tempo de preparo: 15 min

Ingredientes:
- 2 caixas de gelatina sabor uva
- 2 copos de iogurte natural

Modo de fazer:
- Prepare a gelatina de acordo com as instruções da embalagem e leve à geladeira por 2 horas, ou até endurecer ligeiramente.
- Transfira para o liqüidificador, junte o iogurte e bata, até obter uma mistura homogênea. Coloque em taças individuais e leve à geladeira por mais 1 hora antes de servir.





Como representar um algoritmo?

- Vamos usar preferencialmente pseudo-código.
- Por vezes usaremos C/C++ ou frases em português.
- O pseudo-código é baseado em linguagens imperativas.
- É "legível" e independente de qualquer compliador ou arquitetura.

Algoritmo 1: Pseudo-código

início
  $k \leftarrow 0$
  repita
    para $(j \leftarrow 0$ *até* $n-1)$ **faça**
      $i \leftarrow j-1$
      enquanto $(i > 0)$ **faça**
        $i \leftarrow i-1$
    $k \leftarrow k+1;$
  até $(k \geq 10);$

Código em C

```c
int main() {
  int k = 0, j;
  do {
    for (j = 0; j < n; j++) {
      int i = j - 1;
      while (i > 0)
        i = i - 1;
    }
    k = k + 1;
  } while (k < 10);
}
```

Tempo de Processamento

Algoritmos

Como escolher o melhor algoritmo?

Tempo de Processamento

Um algoritmo que realiza uma tarefa em 10 horas é melhor que outro que realiza em 10 dias.

Quantidade de memória necessária

Um algoritmo que usa 1MB de memória RAM é melhor que outro que usa 1GB.

Tempo de Processamento

- Medir o tempo gasto por um algoritmo **não** é uma boa opção.
  - Depende do compilador.
    - Pode preferir algumas construções ou otimizar melhor
  - Depende do hardware
    - GPU vs. CPU
    - Desktop vs. smartphone
- A solução é estudar o número de vezes que operações são executadas.

Tempo de Processamento

- Precisamos de um padrão para contagem das operações:
  - Cada operação simples (ex: $+$, -, $/$, *, Se) demora 1 passo.
  - Laços de repetição e funções não são instruções simples!
  - Cada acesso à memória custa também 1 passo
- Ao medir o tempo de execução contando o número de passos definimos uma função: $T(n)$.
- As operações estão simplificadas, mas mesmo assim isto é útil
- Ex: somar dois inteiros não custa o mesmo que dividir dois reais, mas esses valores, numa visão global, não são importantes.

Exemplo

Um problema exemplo - Soma

Exemplo: Algoritmo que soma os elementos de um vetor.

Algoritmo 2: SomaVetor

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Soma dos elementos de um vetor

```
1 início
2     soma ← 0
3     para (i ← 0 até n − 1) faça
4         soma ← soma + V[i]
5     retorna soma
```

Tempo de Processamento de SomaVetor:
$T(n) = 1 + (n+1) + n + 1 = 2n + 3$

```
Nº execuções

Linha 2: 1
Linha 3: n+1
Linha 4: n
Linha 5: 1
```

Exemplo

Tempo de Processamento

- Análise de complexidade é feita em função de n:
  - n indica o tamanho da entrada.
  - Número de elementos no vetor.
  - Número de vértices num grafo.
  - Número de linhas de uma matriz.
- Diferentes entradas podem ter custo diferente:
  - Melhor caso
  - Pior caso
  - Caso médio

Exemplo

Busca Vetor

Faça a análise do tempo de execução de um algoritmo que realiza a busca de um elemento em um vetor. O algoritmo recebe como entrada um vetor de tamanho n e retorna a posição que o elemento se encontra no vetor. Caso não encontre o elemento no vetor, o algoritmo deve retornar -1.

Exemplo

Um problema exemplo - Busca Vetor

Exemplo: Algoritmo que busca um elemento em um vetor.

Algoritmo 3: BuscaVetor

Entrada: Vetor $V[0..n-1]$, tamanho n, elemento x

Saída: Posição do elemento x no vetor V.

```
1 início
2     i ← 0
3     enquanto (i < n) faça
4         se (V[i] = x) então
5             retorna i
6         i ← i + 1
7     retorna −1
```

```
Nº execuções
Pior Caso|Melhor Caso
Linha 2: 1
Linha 3: n + 1 | 1
Linha 4: n | 1
Linha 5: 0 | 1
Linha 6: n | 0
Linha 7: 1 | 0
```

Exemplo

Um problema exemplo - Busca Vetor

- Melhor caso: $T(n) = 1 + 1 + 1 + 1 + 0 + 0 = 4$
- Pior caso: $T(n) = 1 + (n + 1) + n + 0 + n + 1 = 3n + 3$
- Caso Médio:
  - As linhas 3, 4 e 6 são executadas em média:
    - $L3 = \frac{(n+2)}{2} = \frac{n}{2} + 1$
    - $L4 = \frac{n+1}{2}$
    - $L6 = \frac{n}{2}$
  - $L5$ é executado 1 vez e $L7$ não é executada.
  - $T(n) = 1 + (\frac{n}{2} + 1) + (\frac{n+1}{2}) + 1 + \frac{n}{2} + 0 = \frac{3n}{2} + \frac{7}{2}$

Exemplo

Elemento Máximo

Faça a análise do tempo de execução de um algoritmo que realiza a busca do **maior** elemento em um vetor. O algoritmo recebe como entrada um vetor de tamanho n e retorna o valor do maior elemento presente no vetor.

Exemplo

Um problema exemplo - Max

Exemplo: Algoritmo que encontra o elemento máximo de um vetor.

Algoritmo 4: MaxVetor

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Elemento máximo do vetor

1 **início**
2 $\quad max \leftarrow V[0]$
3 $\quad$ **para** $(i \leftarrow 1$ *até* $n-1)$ **faça**
4 $\quad\quad$ **se** $(\ V[i] > max\ )$ **então**
5 $\quad\quad\quad max \leftarrow V[i]$
6 $\quad$ **retorna** max

```
Nº execuções

Pior Caso|Melhor Caso
Linha 2: 1
Linha 3: n
Linha 4: n-1
Linha 5: n-1|0

Linha 6: 1
```

- Melhor caso: $T(n) = 1 + n + (n-1) + 0 + 1 = 2n + 1$
- Pior caso: $T(n) = 1 + n + (n-1) + (n-1) + 1 = 3n$
- Caso Médio: a linha 5 executa $\frac{n-1}{2}$ vezes. $T(n) = \frac{5n+1}{2}$

Exemplo

Elemento Máximo e Mínimo

Faça a análise do tempo de execução de um algoritmo que realiza a busca do **maior** e do **menor** elemento em um vetor. O algoritmo recebe como entrada um vetor de tamanho n e retorna dois valores: o maior e o menor elemento presentes no vetor.

Exemplo

Um problema exemplo - MaxMin1

Exemplo: Algoritmo que encontra os valores máximo e minimo em um vetor.

Algoritmo 5: MaxMin1

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Elemento máximo e mínimo do vetor

```
1 início
2     max ← V[0]
3     min ← V[0]
4     para (i ← 1 até n − 1) faça
5         se ( V[i] > max ) então
6             max ← V[i]
7         se ( V[i] < min ) então
8             min ← V[i]
9     retorna (max,min)
```

```
Nº execuções

Pior Caso|Melhor Caso
Linha 2 e 3: 1

Linha 4: n
Linha 5: n-1
Linha 6: n-1|0
Linha 7: n-1
Linha 8: 0|n-1
Linha 9: 1
```

Exemplo

Um problema exemplo - MaxMin1

- Melhor caso: $T(n) = 2 + n + 3(n - 1) + 0 + 1 = 4n$
- Pior caso: $T(n) = 4n$
- Caso Médio: $T(n) = 4n$

Exemplo

Um problema exemplo - MaxMin1

Uma melhora no algoritmo MaxMin1:

- Não é necessário executar a linha 7 se a expressão da linha 5 for **verdadeira**.
- Ou seja, se $V[i] > max$, então não precisamos checar se $V[i] < min$.

Algoritmo 6: MaxMin1

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Elemento máximo e mínimo do vetor

```
1 início
2     max ← V[0]
3     min ← V[0]
4     para (i ← 1 até n − 1) faça
5         se ( V[i] > max ) então
6             max ← V[i]
7         se ( V[i] < min ) então
8             min ← V[i]
9     retorna (max,min)
```

Exemplo

Um problema exemplo - MaxMin2

Algoritmo 7: MaxMin2

Entrada: Vetor $V[0..n-1]$, tamanho n

Saída: Elemento máximo e mínimo do vetor

```
1  início
2      max ← V[0]
3      min ← V[0]
4      para (i ← 1 até n − 1) faça
5          se ( V[i] > max ) então
6              max ← V[i]
7          senão
8              se ( V[i] < min ) então
9                  min ← V[i]
10     retorna (max,min)
```

```
Nº execuções
Pior Caso|Melhor Caso
Linha 2 e 3: 1

Linha 4: n
Linha 5: n-1
Linha 6: n-1|0
Linha 7: -
Linha 8: 0|n-1
Linha 9: 0|n-1
Linha 10: 1
```

Exemplo

Um problema exemplo - MaxMin2

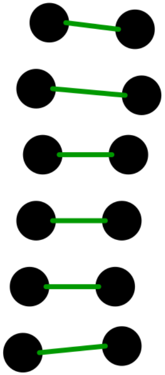
- Melhor caso: ocorre quando o vetor está em ordem crescente.
  - Nesse caso, a linha 6 é executada $n-1$ vezes, e as linhas 6, 8 e 9 zero vezes.
  - $T(n) = 2 + n + 2(n-1) + 1 = 3n + 1$
- Pior caso: ocorre quando o vetor está em ordem decrescente.
  - Nesse caso, as linhas 6 a 9 são executadas $n-1$ vezes.
  - $T(n) = 2 + n + 3(n-1) + 1 = 4n$
- Caso Médio: ocorre quando $V[i]$ é maior que *max* metade das vezes.
  - Isso faz com que as linhas 6, 8 e 9 sejam executadas $\frac{(n-1)}{2}$ vezes.

$$\begin{aligned} T(n) &= 2 + n + (n-1) + \frac{3(n-1)}{2} + 1 = 4n \\ &= 2n + 3n/2 + 2 - 1 - 3/2 + 1 \\ &= \frac{(7n-1)}{2} \end{aligned}$$

Exemplo

Um problema exemplo - MaxMin3

Dá pra fazer melhor?

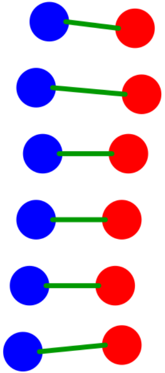


- Comparar elementos par-a-par
- Custo: $\frac{n}{2}$ comparações

Exemplo

Um problema exemplo - MaxMin3

Dá pra fazer melhor?



- Comparar elementos par-a-par
  - Custo: $n/2$ comparações
- Elementos vermelhos são maiores (*biggers*) que os azuis
- Encontrar o máximo entre os elementos vermelhos
  - Custo: $n/2$ comparações
- Elementos azuis são menores (*smallers*) que os vermelhos
- Encontrar o mínimo entre os elementos azuis
  - Custo: $n/2$ comparações

Exemplo

Um problema exemplo - MaxMin3

Algoritmo 8: MaxMin3

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Elemento máximo e mínimo do vetor

```
1  início
2      max ← −∞
3      min ← ∞
4      para (i ← 0 até n − 1 com i ← i + 2) faça
5          se ( V[i] < V[i + 1] ) então
6              s ← i; b ← i + 1
7          senão
8              s ← i + 1; b ← i
9          se (V[s] < min) então
10             min ← V[s]
11         se (V[b] > max) então
12             max ← V[b]
13     retorna (max,min)
```

```
Nº execuções
Pior Caso|Melhor Caso
L2 e L3: 1

L4:n/2 + 1
L5: n/2
L6: 2x n/2|n/4
L7: -
L8: 2 x 0|n/4
L9 e 11: 2x n/2
L10 e L12: 2x 1| n/2
L13: 1
```

Exemplo

Um problema exemplo - MaxMin3

- Pior caso:
  - Linhas 10 e 12 sempre são executadas (L10 e L12 executam $\frac{n}{2}$ vezes).
  - Elementos azuis em ordem decrescente.
  - Elementos vermelhos em ordem crescente.
  - Ex. [4 5 3 6 2 7 1 8 0 9]
- Melhor caso:
  - Linhas 10 e 12 são executadas uma única vez (L10 e L12 executam 1 vez cada).
  - Elementos azuis em ordem crescente
  - Elementos vermelhos em ordem decrescente
  - Ex. [0 9 1 8 2 7 3 6 4 5]
- Caso médio:
  - Linhas 10 e 12 são executadas em média $\frac{n}{4}$ vezes.

Exemplo

Um problema exemplo - MaxMin3

Tabela: Custo MaxMin3

Linhas	Melhor	Pior	Médio
2 e 3	2	2	2
4	$\frac{n}{2}+1$	$\frac{n}{2}+1$	$\frac{n}{2}+1$
5	$\frac{n}{2}$	$\frac{n}{2}$	$\frac{n}{2}$
6	$2\frac{n}{2}$	$2\frac{n}{2}$	$2\frac{n}{4}$
8	0	0	$2\frac{n}{4}$
9 e 11	$2\frac{n}{2}$	$2\frac{n}{2}$	$2\frac{n}{2}$
10 e 12	2	$2\frac{n}{2}$	$2\frac{n}{4}$
13	1	1	1

Exemplo

Um problema exemplo - MaxMin3

- Melhor caso:

$$T(n) = 2 + (\frac{n}{2} + 1) + \frac{5n}{2} + 2 + 1 = 3n + 6$$

- Pior caso:

$$T(n) = 2 + (\frac{n}{2} + 1) + \frac{7n}{2} + 1 = 4n + 4$$

- Caso médio:

$$T(n) = 2 + (\frac{n}{2} + 1) + \frac{6n}{2} + 1 = \frac{7n}{2} + 4$$

Exemplo

Um problema exemplo - MaxMin3

Tabela: Comparativo MaxMin

Algoritmo	Melhor Caso	Pior Caso	Caso Médio
MinMax1	4n	4n	4n
MinMax2	3n+1	5n-1	$\frac{(7n-1)}{2}$
MinMax3	3n+6	4n+4	$\frac{7n}{2}+4$

Exemplo

Um problema exemplo - MaxMin4

- O *melhor caso* é um pouco pior que o algoritmo MaxMin2.
- É possível implementar uma última melhora.
  - Em vez de *max* e *min* começaram com $-\infty$ e ∞, basta serem inicializadas com a primeira posição do vetor e descontar essa posição do laço de repetição.

Exemplo

Um problema exemplo - MaxMin4

Algoritmo 9: MaxMin4

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Elemento máximo e mínimo do vetor

1 **início**
2   **se** *V[0] > V[1]* **então**
3     $max \leftarrow V[0]$; $min \leftarrow V[1]$
4   **senão**
5     $max \leftarrow V[1]$; $min \leftarrow V[0]$
6   **para** $(i \leftarrow 2$ *até* $n-1$ *com* $i \leftarrow i+2)$ **faça**
7   **se** $(\ V[i] < V[i+1]\ )$ **então**
8       $s \leftarrow i$; $b \leftarrow i+1$
9   **senão**
10       $s \leftarrow i+1$; $b \leftarrow i$
11   **se** $(V[s] < min)$ **então**
12       $min \leftarrow V[s]$
13   **se** $(V[b] > max)$ **então**
14       $max \leftarrow V[b]$
15   **retorna** (max,min)

Nº execuções

L2 a L5: 3

L6:(n-2)/2 + 1
L7: (n-2)/2
L8: 2x (n-2)/2|(n-2)/4
L9: -
L10: 2x 0|(n-2)/4
L11 e 13: 2x (n-2)/2
L12 e L14: 2x 1|(n-2)/2
L15: 1

Exemplo

Um problema exemplo - MaxMin4

Tabela: Custo MaxMin4

Linhas	Melhor	Pior	Médio
2 a 5	3	3	3
6	$\frac{(n-2)}{2}+1$	$\frac{(n-2)}{2}+1$	$\frac{(n-2)}{2}+1$
7	$\frac{(n-2)}{2}$	$\frac{(n-2)}{2}$	$\frac{(n-2)}{2}$
8	$2\frac{(n-2)}{2}$	$2\frac{(n-2)}{2}$	$2\frac{(n-2)}{4}$
10	0	0	$2\frac{(n-2)}{4}$
11 e 13	$2\frac{(n-2)}{2}$	$2\frac{(n-2)}{2}$	$2\frac{(n-2)}{2}$
12 e 14	2	$2\frac{(n-2)}{2}$	$2\frac{(n-2)}{4}$
15	1	1	1

Exemplo

Um problema exemplo - MaxMin4

- Melhor caso:

$$\begin{aligned} T(n) &= 3 + \frac{(n-2)}{2} + 1 + \frac{(n-2)}{2} + 2\frac{(n-2)}{2} + 2\frac{(n-2)}{2} + 3 \\ &= 6\frac{(n-2)}{2} + 7 = 3n + 1 \end{aligned}$$

- Pior caso:

$$\begin{aligned} T(n) &= 3 + \frac{(n-2)}{2} + 1 + \frac{(n-2)}{2} + 2\frac{(n-2)}{2} + 4\frac{(n-2)}{2} + 1 \\ &= 8\frac{(n-2)}{2} + 5 = 4n - 3 \end{aligned}$$

- Caso médio:

$$\begin{aligned} T(n) &= 3 + \frac{(n-2)}{2} + 1 + 6\frac{(n-2)}{2} + 1 = 7\frac{(n-2)}{2} + 5 \\ &= \frac{7n}{2} - 7 + 5 = \frac{7n}{2} - 2 \end{aligned}$$

Exemplo

Um problema exemplo - MaxMin4

Tabela: Comparativo MaxMin

Algoritmo	Melhor Caso	Pior Caso	Caso Médio
MinMax1	4n	4n	4n
MinMax2	3n+1	5n-1	$\frac{(7n-1)}{2}$
MinMax3	3n+6	4n+4	$\frac{7n}{2}+4$
MinMax4	3n+1	4n-3	$\frac{7n}{2}-2$

Comportamento Assintótico

- Para valores pequenos de n, praticamente qualquer algoritmo custa pouco para ser executado.
- Logo: a escolha do algoritmo tem pouquíssima influência em problemas de tamanho pequeno.
- Estamos realmente interessados como o algoritmo se comporta conforme n aumenta indefinidamente.

FIM

Fim

Algoritmos e Estrutura de Dados II

Análise de Algoritmos

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia