classificação e pesquisa de dados

visão geral

1

Estudo da Complexidade de Algoritmos

Métodos de Ordenação de Dados

Pesquisa de Dados

Organização de Arquivos

2

Problema computacional

Algoritmo

Estruturas de dados

Análise de complexidade

5

Problema computacional

Problema a ser resolvido por meio de um **computador**

Possui **entrada** e **saída** que definem o problema

Instância de um problema consiste na **atribuição específica** de valores para a **entrada**

Cada instância possui um **saída esperada**, denominada **solução**

Exemplo

Multiplicação de dois inteiros

6

conceitos

classificação e pesquisa de dados

Algoritmo

Solução para um problema

Sequência **finita** de passos descritos de forma **não ambígua**

Recebe um conjunto de dados e (entrada) **gera** uma saída

Resolve um problema quando devolve uma resposta correta para **qualquer** instância do problema

Exemplo

Multiplicação de dois inteiros a e b

7

algoritmo

Sequência de ações executáveis para obtenção de uma **solução** para determinado **tipo de problema**. (Ziviani, 2007)

Descrição de um **padrão de comportamento**, expresso em termos de um **conjunto finito de ações**. (Dijkstra, 1971)

Sequência de instruções **não ambíguas** para **solucionar** um **problema**, isto é, **obter** uma **saída esperada** para um **entrada legítima** em uma quantidade de **tempo finito**. (Levitin, 2011)

8

algumas características

algoritmo

Entrada: informações (dados) disponibilizadas para o algoritmo

Saída: informações (dados) geradas a partir do processamento

Instruções: não ambíguas

Finito: para todos os casos, o algoritmo termina após a execução de um número finito de passos

9

conceitos

classificação e pesquisa de dados

Algoritmo

Multiplique o primeiro (d=1) dígito menos significativo de b por a

Multiplique o resultado por $10^{(d-1)}$ e armazene este resultado

Multiplique o segundo (d=2) dígito menos significativo de b por a

Multiplique o resultado por $10^{(d-1)}$ e some ao resultado anterior

Repita o processo até esgotar os dígitos de b

Exemplo

Multiplicação dos inteiros $a = 123$ e $b = 321$

$1 * 123 * 10^0 + 2 * 123 * 10^1 + 3 * 123 * 10^2 = 123 + 2460 + 36900 = 39483$

10

objetivos de estudo — algoritmo

Fundamentais em Computação

Conhecimento de algoritmos permite a **modelagem** e resolução de inúmeros problemas

Projetar algoritmos para **solucionar** problemas

A **análise** de algoritmos permite **estimar** a eficiência de uma solução, ou seja, **prever** o comportamento dos algoritmos

11

metodologia — algoritmo

Analisar o problema: compreensão e entradas

Projetar o algoritmo: estruturas de dados e eficiência de tempo/espaço

Implementar a solução em uma determinada linguagem

Verificação do funcionamento com testes

13

análise — algoritmo

Analisar o comportamento da solução

análise de um **algoritmo particular**: custo de usar um dado algoritmo para resolver um problema específico

análise de uma **classe de algoritmos**: algoritmo de menor custo para resolver um problema específico

14

análise — algoritmo

Prever os recursos necessários **sem** precisar implementar
tempo - esforço computacional
espaço - demanda extra de memória

Comparar diversos algoritmos

Determinar se um algoritmo é viável

15

técnicas de análise — algoritmo

Empírica
baseada na **experimentação** e na observação

Análise assintótica

baseada em um modelo matemático

avalia a contagem do número de execuções de operações *mais significativas* (*passos básicos* ou *operações primitivas*)

visa compreender o *comportamento* do algoritmo

considera aspectos de *tempo* e *espaço*

16

técnicas de análise — algoritmo

Empírica
baseada na **experimentação** e na observação

Principais fatores

hardware

software

natureza do problema

17

técnicas de análise — algoritmo

Empírica
baseada na **experimentação** e na observação

Hardware

quantidade de processadores

ciclos por instrução

quantidade e tipos de memória

18

técnicas de análise — algoritmo

Empírica
baseada na **experimentação** e na observação

Software

sistema operacional

escalonamento de processos

compilador utilizado

linguagem de programação

19

técnicas de análise — algoritmo

Empírica

baseada na **experimentação** e na observação

Natureza do problema

execução pode variar de acordo com a entrada

ordenada crescente, decrescente e aleatória

influência do tamanho das entradas

20

técnicas de análise — algoritmo

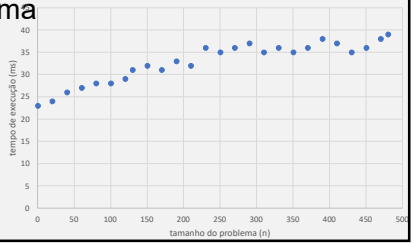
Empírica

baseada na **experimentação** e na observação

Formas de analisar

alterar os parâmetros do problema

plotar os dados para identificar o **comportamento**



21

técnicas de análise — algoritmo

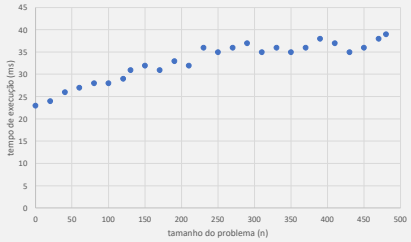
Empírica

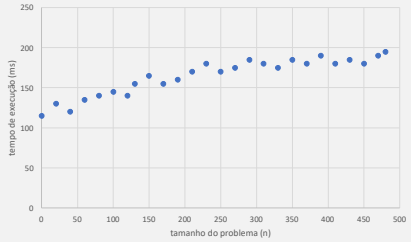
baseada na **experimentação** e na observação

Formas de analisar

Quad Core i5, 3 GHz, 4GB RAM

Duo Core i7, 1.7 GHz, 8GB RAM





22

técnicas de análise — algoritmo

Empírica

baseada na **experimentação** e na observação

Esta análise necessita de

um conjunto de testes **adequado**

o tamanho do conjunto de testes deve ser **estatisticamente significativo**

uma variação dos dados que **explore** o tipo de problema

23

técnicas de análise — algoritmo

Empírica

baseada na **experimentação** e na observação

Limitações

comparar algoritmos é **complexo**

nem sempre é possível explorar toda as instâncias de entrada

o estado do SO **altera** dinamicamente a todo instante

24

técnicas de análise — algoritmo

Empírica

baseada na **experimentação** e na observação

Exemplo

algoritmo X executou duas vezes mais rápido do que o **algoritmo Y** para uma entrada com 10000 itens

qual algoritmo é **melhor** para este problema?

25

técnicas de análise — algoritmo

Empírica

baseada na **experimentação** e na observação

Questão em aberto

Qual o **comportamento** dos algoritmos para entradas de tamanho diferente de 10000?

A análise **empírica** por si só nem sempre é suficiente para analisar o **comportamento** dos algoritmos

26

técnicas de análise — algoritmo

Análise assintótica

baseada em um **modelo matemático**

emprega um computador idealizado

independente de hardware e software

pode ser aplicada em uma representação de alto nível de um algoritmo (pseudocódigo)

não é necessário implementar o algoritmo

27

Análise assintótica

associa uma função de complexidade f a um algoritmo

$f(n)$ é a **medida do tempo** necessário para executar um algoritmo para um problema de tamanho n

$f(n)$ é a **quantidade de memória** necessária para executar um algoritmo para um problema de tamanho n

Exemplo

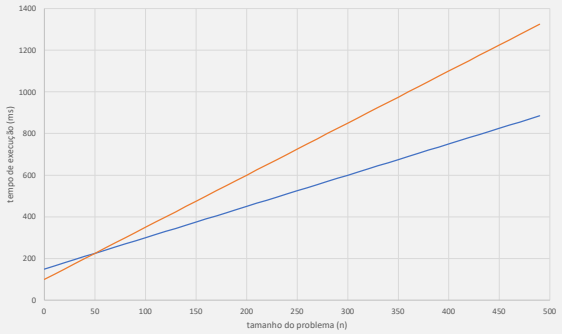
$$\boldsymbol{f(n) = c_1 n + c_2}$$

28

Análise assintótica

exemplo

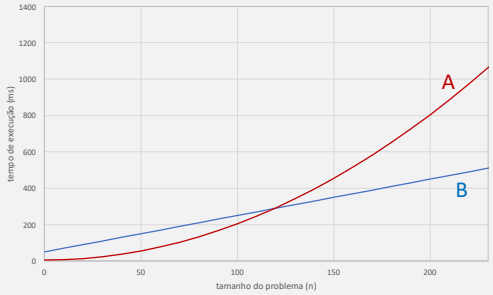
$$\boldsymbol{f(n) = c_1 n + c_2}$$



29

Análise assintótica

Qual algoritmo (A ou B) é melhor?



30

Elementos de análise

linguagem de descrição dos algoritmos

pseudocódigo

linguagens de programação

modelo computacional para execução dos algoritmos

máquina de acesso aleatório (*Random-Access Machine* - RAM)

métrica para medir o custo de execução

contagem do número de instruções

31

análise algoritmo

Modelo computacional RAM

CPU **pode** acessar uma posição aleatória da memória em apenas **uma** operação primitiva

Qualquer operação primitiva **pode** ser realizada em um número **constante** de passos **independente** do tamanho da entrada

32

análise algoritmo

RAM e **Métrica**

definir as operações primitivas (atribuições, comparações, etc.)

Exemplo

```
soma ← 5          1
calc ← soma + 10  2
soma ← soma * 2   1
resu ← soma > 2   2
                  6
```

33

análise algoritmo

RAM e **Métrica**

definir as operações primitivas (atribuições, comparações, etc.)

Exemplo

```
soma ← 5          1
calc ← soma + 10  2
se calc < 15 então  1
  soma ← soma * 2   2
então
  soma ← soma / 2   2
                    6
```

34

análise algoritmo

RAM e **Métrica**

definir as operações primitivas (atribuições, comparações, etc.)

Exemplo

```
soma ← 0                  1
para i ← 1 até 100 faça   1 + 101
  soma ← soma + i         3
  i ← i + 1               2   × 100
                          603
```

i ≤ 100

35

análise algoritmo

RAM e **Métrica**

definir as operações primitivas (atribuições, comparações, etc.)

Exemplo

```
int soma = 0;                          1
for(int i = 1; i <= 100; i++)          1 + 101
  soma = soma + i;                     4 × 100
                                       503
```

36

análise algoritmo

RAM e **Métrica**

definir as operações primitivas (atribuições, comparações, etc.)

Exemplo

1 $produto = 1$
2 **para** $i = 1$ *até* n, *incrementando* **faça**
3 $\quad produto = produto \times A[i]$
4 **devolve** *produto*

37

análise algoritmo

Exemplo

```
inteiro somar(n)
Entrada: número natural n
Saída: soma dos números naturais menores ou iguais a n
  soma ← 0                             1
  para i ← 1 até n faça                1 + n + 1
     soma ← soma + i                   4 × n
  retorna soma                         1
                                       f(n) = 5n + 4
```

38

análise algoritmo

Exemplo

```
inteiro somar(vet, tam)
Entrada: vetor vet e seu respectivo tamanho
Saída: soma dos conteúdos do vetor
  soma ← 0                             1
  para i ← 0 até tam-1 faça            1 + tam + 1
     soma ← soma + vet[i]              5 × tam
  retorna soma                         1
                                       T(n) = 6n + 4
```

39

análise algoritmo

Exemplo

```
inteiro somar(vet, n)
Entrada: vetor vet e seu respectivo tamanho
Saída: soma dos conteúdos do vetor
  soma ← 0                          1
  para i ← 0 até n-1 faça           1 + n + 1
     soma ← soma + vet[i]           5 × n
  proc(vet, n)                      ???
  retorna soma
```

40

análise algoritmo

Exemplo

```
proc(vet, n)
Entrada: vetor vet e seu respectivo tamanho
Saída: não tem
  para i ← 0 até n-1 faça           1 + n + 1
     exibir vet[i]                  (2 + 2) × n
```

T(n) = 5n + 2

41

análise algoritmo

Exemplo

```
inteiro somar(vet, n)
Entrada: vetor vet e seu respectivo tamanho
Saída: soma dos conteúdos do vetor
  soma ← 0                          1
  para i ← 0 até n-1 faça           1 + n + 1
     soma ← soma + vet[i]           (3 + 2) × n      inc
  proc(vet, n)                      5n + 2
  retorna soma                      1
```

T(n) = 11n + 6

42

análise algoritmo

Exemplo

```
inteiro menor(vet, tam)
Entrada: vetor vet e seu respectivo tamanho
Saída: menor valor presente no vetor
  menor ← vet[0]                    2
  para i ← 1 até tam-1 faça         1 + tam - 1
     se vet[i] < menor então        2 × (n – 1)
        menor ← vet[i]              2 × ???
  retorna menor
```

43

análise — algoritmo

Melhor e pior casos de execução

Dependem da **natureza** dos dados

Melhor caso ⇒ executa o **MENOR** número de instruções (**MAIS** eficiente)

Pior caso ⇒ executa o **MAIOR** número de instruções (**MENOS** eficiente)

44

análise: melhor caso — algoritmo

Qual seria um exemplo de entrada para o MELHOR CASO?

```
inteiro menor(vet, tam)
Entrada: vetor vet e seu respectivo tamanho
Saída: menor valor presente no vetor
  menor ← vet[0]                          2
  para i ← 1 até tam-1 faça               1 + tam
     se vet[i] < menor então              2 × (tam – 1)
        menor ← vet[i]                    0
                                          2 × (tam – 1)   inc
  retorna menor                           1
```

T(n) = 5n

45

análise: pior caso — algoritmo

Qual seria um exemplo de entrada para o PIOR CASO?

```
inteiro menor(vet, tam)
Entrada: vetor vet e seu respectivo tamanho
Saída: menor valor presente no vetor
  menor ← vet[0]                          2
  para i ← 1 até tam-1 faça               1 + tam
     se vet[i] < menor então              2 × (tam – 1)
        menor ← vet[i]                    2 × (tam – 1)
                                          2 × (tam – 1)   inc
  retorna menor                           1
```

T(n) = 7n - 2

46

EXERCÍCIOS — análise: melhor e pior casos

```
soma ← 0                                  1
prod ← 1                                  1
para i ← 1 até 100 faça                   1 + 101
  soma ← soma + i                         2 × 100
  prod ← prod * i                         2 × 100
                                          2 × 100   inc
```

f(n) = 704

```
for(int i = 0; i < n; i++)                1 + n + 1
   if(vet[i]%2 == 0 )                     3 × n
      vet[i] = 2 * vet[i];                0  OU  4 × n
                                          2 × n   inc
```

f(n) = 6n + 2 | f(n) = 10n + 2

47

EXERCÍCIOS — análise: melhor e pior casos

```
soma ← 0                              1
prod ← 1                              1
para i ← 1 até n faça                 1 + n + 1
  soma ← soma + i                     4 × n
para i ← 1 até n² faça                1 + (n² + 1)
  prod ← prod * i                     4 × n²
```

f(n) = 3n² + 3n + 6

```
soma ← 0                              1
para i ← 1 até n faça                 1 + (n + 1)
  para j ← 1 até n faça               1 + (n + 1)        × n
    soma ← soma + 1                   4 × n
                                      2 × n
```

f(n) = 7n² + 3n +3

48

EXERCÍCIOS — análise: melhor e pior casos

```
soma ← 0                              1
para i ← 1 até n faça                 1 + (n + 1)
  para j ← 1 até m faça               1 + (m + 1)        × n
    soma ← soma + 1                   2 × m
```

f(n) = 3nm + 3n + 3

```
soma ← 0                              1
menor ← vet[0]                        2
para i ← 1 até n-1 faça               1 + (n + 1)
  se vetor[i] < menor então           2                  × n
    menor ← vetor[i]                  0 ou 2
se menor < 100 então                  1
  para i ← 0 até n-1 faça             1 + (n + 1)        0 OU × 100
    soma ← soma + vet[i]              3 × n
```

T(n) = 2n + 5 T(n) = 403n + 205

49

análise — algoritmo

Alguns princípios para análise de algoritmos

Tempo de execução de **comandos** de atribuição, leitura ou escrita, avaliação de expressão lógica podem ser considerados constante

Tempo de execução de uma **sequência de comandos** ⇒ **maior** tempo de execução de **qualquer** comando da sequência

Tempo de execução de uma **estrutura condicional** ⇒ tempo dos comandos **dentro do corpo** da estrutura **mais** o tempo para avaliar a condição

Tempo de execução de uma **estrutura de repetição** ⇒ tempo dos comandos **dentro do corpo** da estrutura **mais** o tempo para avaliar a condição, multiplicado pelo número de iterações

Tempo de execução de **chamada** a uma **subrotina não recursiva** deve ser computado separadamente e incluído no cálculo da subrotina chamadora

50

REFERÊNCIAS — análise: melhor e pior casos

ZIVIANI, Nivio. Projeto de Algoritmos com Implementação em Java e C++. São Paulo: Thomson Learning, 2007.

LEISERSON, C. E.; STEIN, C.; RIVEST, R. L., CORMEN, T.H. AlgoritmosTeoria e Prática. Tradução da 2ª edição americana. Rio de Janeiro: Campus, 2002.

LINTZMAYER, Carla Negri; MOTA, Guilherme Oliveira. Análise de Algoritmos e de Estruturas de Dados. Notas de aula. Versão: 29/06/2022.

53