

**INSTITUTO
FEDERAL**

Goiás

Câmpus
Formosa

Algoritmos e Estruturas de Dados

Tecnólogo em Análise e Desenvolvimento de Sistemas



Waldeyr Mendes Cordeiro da Silva

2019

Material didático para Algoritmos e Estruturas de Dados

Versão 0.1

Waldeyr Mendes Cordeiro da Silva

- Formação:
 - Bacharelado em Sistemas de Informação
 - Licenciatura em Ciências Biológicas
 - Complementação Pedagógica em Matemática
 - Especialização em Engenharia de Software
 - Especialização em Segurança da Informação
 - Mestrado em Informática
 - Doutorado em Ciências Biológicas (Bioinformática)
-  Lattes: <http://lattes.cnpq.br/2391349697609978>
-  ORCID: <https://orcid.org/0000-0002-8660-6331>

Sumário

1	Prefácio	4
2	Introdução	5
2.1	Análise de Algoritmos	6
2.1.1	Elementos de Notação Assintótica	6
3	Programação	7
3.1	Primeiros Passos em Programação	8
4	Estruturas de Dados	9
4.1	Estruturas de Dados Homogêneas e Heterogêneas	9
4.2	Algoritmos de Ordenação	11
4.3	Listas	11
4.4	Pilhas	11
4.5	Filas	11
4.6	Tabelas Hashing	11
4.7	Árvores	11
4.8	Grafos	11
4.8.1	Busca em Grafos	11
5	Aplicações	12
6	Apêndice	14



1. Prefácio

Em construção...

Este livro destina-se aos acadêmicos e demais interessados em iniciar os estudos em algoritmos e estruturas de dados.



2. Introdução

Um bonita citação...

Um algoritmo é um procedimento computacional bem definido que processa um valor ou um conjunto de dados (entrada) e produz algum valor ou conjunto de dados (saída) (CORMEN et al., 2009). Os algoritmos existem há muito tempo, mas estão presentes na sociedade moderna de uma forma nunca experimentada na história da humanidade. Quase tudo que se produz, sejam produtos ou serviços, tem alguma influência de algoritmos. O comércio eletrônico utiliza tanto algoritmos clássicos como ordenações, quanto algoritmos modernos de recomendação de produtos baseado no histórico de visitas. A segurança das senhas em qualquer sistema bancário ou *Web* é garantida por algoritmos de criptografia. Imagens de satélite, dados genômicos, reconhecimento de faces, previsão do tempo, quase tudo que se possa imaginar atualmente é influenciado direta ou indiretamente por algum algoritmo.

O estudo de algoritmos é uma demanda crescente frente aos novos desafios trazidos pelo grande volume de dados que as tecnologias modernas proporcionaram. A análise de algoritmos é uma área com questões importantes em aberto, como é o caso dos *Millennium Prize Problems*, com sete problemas matemático-computacionais em aberto e cujo prêmio é de 1 milhão de dólares por cada solução.

O propósito da análise de um algoritmo é prever seu comportamento quanto ao tempo de execução ou ao espaço em memória que irá ocupar mesmo antes de ser executado em um computador específico. Porém, muitos fatores, como o tamanho e a variedade dos dados de entrada, influenciam o algoritmo. Portanto, a análise de algoritmos provê uma aproximação, o que em muitos casos é bastante significativo.

Tipos abstratos de dados são conjuntos de valores sobre os quais é possível aplicar funções de forma homogênea através de um algoritmo. Funções e valores em conjunto, consituem um modelo matemático que pode ser empregado em problemas do mundo real (ASCENCIO; ARAÚJO, 2010). Os algoritmos são projetados em função de um tipo abstrato de dados. A representação computacional de um tipo abstrato de dados com seus tipos e operações permitidas pode ser entendida como uma **estrutura de dados**. Uma estrutura de dados é um meio para armazenar e processar dados com vistas à sua organização, acesso e modificações (CORMEN et al., 2009).

2.1 Análise de Algoritmos

2.1.1 Elementos de Notação Assintótica

3. Programação

3.1 Primeiros Passos em Programação

Programa 3.1: Meu primeiro programa em C.

```
1 #include <stdio.h>
2 void main(){
3     printf("Meu primeiro programa em C!\n");
4 }
```

4. Estruturas de Dados

4.1 Estruturas de Dados Homogêneas e Heterogêneas

Vetor como tipo abstrato de dados

Primeiro uma revisão sobre vetores. Em C uma variável que represente um vetor é um ponteiro. Portanto, quando um vetor é parâmetro para uma função o que é passado é a sua referência, ou seja, o endereço base do vetor. Vetores podem ser bi-, tri-, ou multi-dimensionais, mas abrigam o mesmo tipo de dados. O programa 4.1 mostra o vetor sendo passado para uma função que retorna o valor do meio do vetor.

Programa 4.1: Em C, vetores são passados para funções por referência.

```
1 #include <stdio.h>
2 #define TAMANHO 100
3 int getValorDoMeio(int *vetor, int tamanho);
4
5 void main() {
6     int vetor[TAMANHO];
```

```

7     int valorDoMeio;
8     for(int i=0; i<TAMANHO;i++){
9         vetor[i] = i+1;
10    }
11    valorDoMeio = getValorDoMeio(vetor,TAMANHO);
12    printf("Valor do meio: %d \n", valorDoMeio);
13 }
14
15 int getValorDoMeio(int *vetor, int tamanho){
16     printf("Valores no vetor:\n");
17     for(int i=0; i<tamanho;i++){
18         printf("%d\t",vetor[i]);
19     }
20     printf("\n");
21     return vetor[tamanho/2];
22 }
23 // Valores no vetor:
24 // 1   2   3   4   5   6   7   8   9   10
25 // 11  12  13  14  15  16  17  18  19  20
26 // 21  22  23  24  25  26  27  28  29  30
27 // 31  32  33  34  35  36  37  38  39  40
28 // 41  42  43  44  45  46  47  48  49  50
29 // 51  52  53  54  55  56  57  58  59  60
30 // 61  62  63  64  65  66  67  68  69  70
31 // 71  72  73  74  75  76  77  78  79  80
32 // 81  82  83  84  85  86  87  88  89  90
33 // 91  92  93  94  95  96  97  98  99  100
34 // Valor do meio: 51

```

Programa 4.2: Um vetor de duas dimensões mostrando apenas valores da diagonal principal.

```

1  #include <stdio.h>
2  #define TAMANHO 4
3  void main(){
4      int vetor[TAMANHO][TAMANHO];
5      for (int l=0; l<TAMANHO; l++){
6          for (int c=0; c<TAMANHO; c++){
7              printf("Valor [%d][%d]: ", l,c);
8              scanf("%d", &vetor[l][c]);
9          }
10     }
11
12     for (int l=0; l<TAMANHO; l++){
13         for (int c=0; c<TAMANHO; c++){
14             if(l == c){
15                 printf("[%d][%d]:%d\t",l,c,vetor[l][c]);
16             }
17         }
18         printf("\n");
19     }
20 }
21 // Valor [0][0]: 1
22 // Valor [0][1]: 2
23 // Valor [0][2]: 2
24 // Valor [0][3]: 2
25 // Valor [1][0]: 2
26 // Valor [1][1]: 1
27 // Valor [1][2]: 2
28 // Valor [1][3]: 2
29 // Valor [2][0]: 2
30 // Valor [2][1]: 2
31 // Valor [2][2]: 1
32 // Valor [2][3]: 2
33 // Valor [3][0]: 2
34 // Valor [3][1]: 2
35 // Valor [3][2]: 2
36 // Valor [3][3]: 1
37 // [0][0]:1
38 // [1][1]:1
39 // [2][2]:1

```

40 // [3][3]:1

Strings em C

Em C, uma *string* é um vetor de caracteres e cada *string* termina com um caracter *NULL*. Uma constante *string* é definida dentro de aspas em que o caractere *NULL* é automaticamente incluído. Por exemplo, a string "IFG" é um vetor de 4 elementos. O programa 4.1 mostra um exemplo de *string* em C.

Programa 4.3: Em C, *strings* são vetores de caracteres..

```
1 #include <stdio.h>
2 #define TAMANHO 100
3 void main(){
4     char vetor[TAMANHO] = "Aqui vai uma frase bastante longa para servir de exemplo!";
5     int qtd_de_letras_a_na_string = 0;
6     for (int i=0; i<TAMANHO; i++){
7         if (vetor[i] == 'a' || vetor[i] == 'A'){
8             qtd_de_letras_a_na_string++;
9         }
10    }
11    printf("Quantidade de letras \'a\' ou \'A\' na string: %d\n",
12           qtd_de_letras_a_na_string);
13 }
```

// Quantidade de letras 'a' ou 'A' na string: 8

4.2 Algoritmos de Ordenação

4.3 Listas

4.4 Pilhas

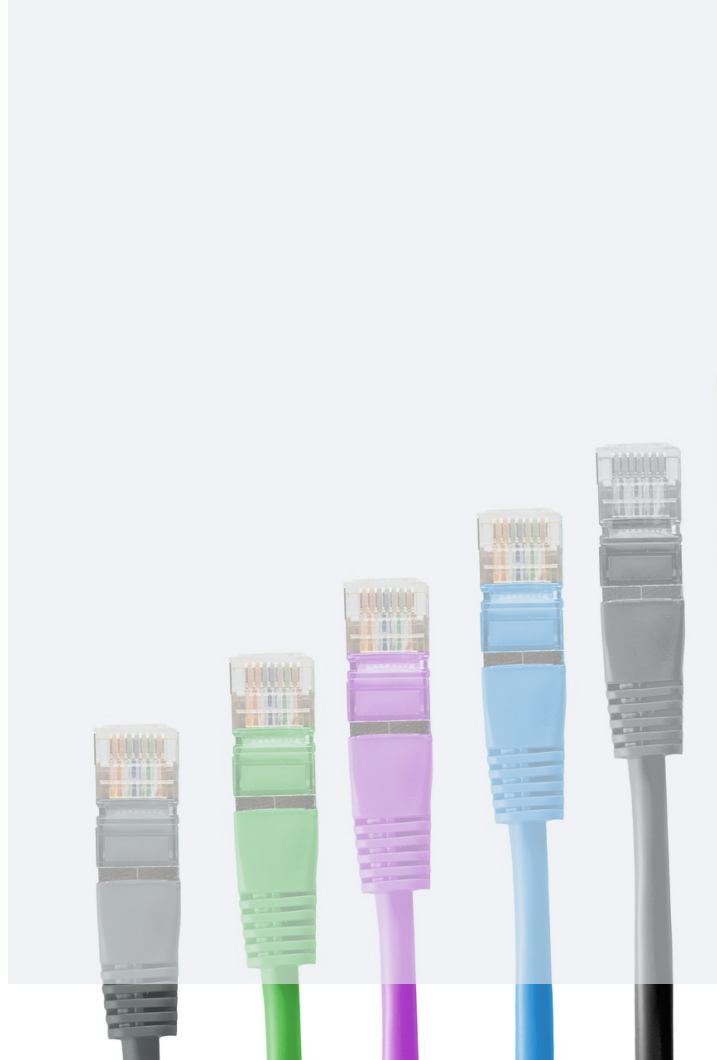
4.5 Filas

4.6 Tabelas Hashing

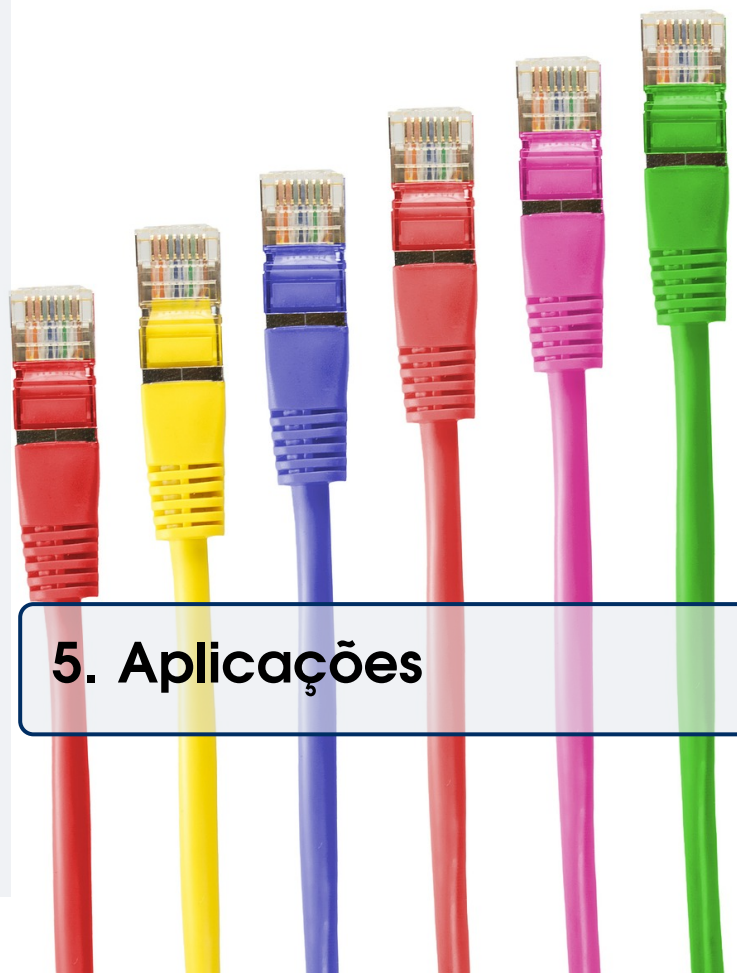
4.7 Árvores

4.8 Grafos

4.8.1 Busca em Grafos



5. Aplicações



Referências Bibliográficas

ASCENCIO, Ana Fernanda G.; ARAÚJO, Graziela S. de. Estruturas de Dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. **São Paulo: Perarson Prentice Halt**, v. 3, 2010.

CORMEN, Thomas H. et al. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

6. Apêndice