

APOSTILA COMPLETA: Algoritmos e Complexidade

Material Didático Universitário - Resumo Teórico e Prático

Autor: Prof. Vagner Cordeiro

Disciplina: Algoritmos e Complexidade

Curso: Sistemas de Informação

Carga Horária: 80h

ÍNDICE GERAL

PARTE I: FUNDAMENTOS TEÓRICOS

1. Conceitos Fundamentais
2. Análise de Complexidade
3. Notação Big O
4. Estruturas de Dados

PARTE II: ALGORITMOS CLÁSSICOS

5. Algoritmos de Ordenação
6. Algoritmos de Busca
7. Estruturas Dinâmicas
8. Árvores e Grafos

CONCEITOS FUNDAMENTAIS

O que é um Algoritmo?

Definição Formal:

Um algoritmo é uma sequência **finita** e **bem definida** de instruções que resolve um problema computacional específico.

Características Essenciais:

1. **Finitude:** Termina em tempo finito
2. **Definição:** Cada passo é claramente especificado
3. **Entrada:** Zero ou mais valores de entrada
4. **Saída:** Um ou mais valores de saída
5. **Efetividade:** Cada passo deve ser executável



Exemplo Simples: Encontrar o Maior

ANÁLISE DE COMPLEXIDADE

Por que Analisar Complexidade?

Exemplo Prático:

Problema: Buscar um nome em uma lista telefônica com 1 milhão de entradas

Busca Linear: 500.000 comparações em média (50% da lista)

Busca Binária: 20 comparações máximo ($\log_2 1.000.000 \approx 20$)

Diferença: 25.000x mais rápido!

Notação Matemática

Big O - Limite Superior:

$$f(n) = O(g(n)) \iff \exists c, n_0 > 0 : f(n) \leq c \cdot g(n), \forall n \geq n_0$$

Interpretação: $f(n)$ cresce no máximo tão rápido quanto $g(n)$



EXEMPLOS TEÓRICOS SIMPLES

Exemplo 1: Análise Passo a Passo

```
// Somar elementos de um array
int somar_array(int arr[], int n) {
    int soma = 0;                // 1 operação
    for (int i = 0; i < n; i++) { // n iterações
        soma += arr[i];          // 1 operação por iteração
    }
    return soma;                 // 1 operação
}
```

Cálculo de Complexidade:

- Inicialização: 1 operação
- Loop: n iterações \times 1 operação = n operações
- Retorno: 1 operação

💡 EXEMPLOS PRÁTICOS DO DIA A DIA

📱 WhatsApp: Busca de Contatos

```
// Como o WhatsApp encontra seus contatos rapidamente
typedef struct {
    char nome[100];
    char telefone[20];
    int id;
} Contato;

// Busca linear - O(n)
Contato* buscar_contato_linear(Contato lista[], int n, char* nome) {
    for (int i = 0; i < n; i++) {
        if (strcmp(lista[i].nome, nome) == 0) {
            return &lista[i];
        }
    }
    return NULL;
}
```

```
// Na prática: WhatsApp usa hash table - O(1)
```

ESTRUTURAS DE DADOS FUNDAMENTAIS

Array vs Lista Ligada

Array (Vetor):

```
int numeros[100]; // Memória contígua, acesso O(1)

// Vantagens:
// ✓ Acesso direto: numeros[50] em O(1)
// ✓ Menos memória: apenas os dados
// ✓ Cache-friendly: dados próximos na memória

// Desvantagens:
// ✗ Tamanho fixo
// ✗ Inserção no meio: O(n) - precisa mover elementos
```

Lista Ligada:

```
typedef struct No {
```

ALGORITMOS DE ORDENAÇÃO

Bubble Sort - $O(n^2)$

Analogia: Como organizar cartas na mão

```
void bubble_sort_simples(int arr[], int n) {  
    // Para cada posição  
    for (int i = 0; i < n-1; i++) {  
        // Compara elementos adjacentes  
        for (int j = 0; j < n-i-1; j++) {  
            if (arr[j] > arr[j+1]) {  
                // Troca se estiver fora de ordem  
                int temp = arr[j];  
                arr[j] = arr[j+1];  
                arr[j+1] = temp;  
            }  
        }  
    }  
}
```


ÁRVORES BINÁRIAS

Conceitos Fundamentais

Definição: Estrutura hierárquica onde cada nó tem no máximo 2 filhos

```
typedef struct No {  
    int dados;  
    struct No* esquerda;  
    struct No* direita;  
} No;
```

Propriedades Matemáticas:

- Altura mínima: $\log_2(n)$ (árvore completa)
- Altura máxima: $n-1$ (árvore degenerada)
- Número de folhas: $\lceil n/2 \rceil$ (aproximadamente)

Árvore Binária de Busca (BST)

LISTA DE CONCEITOS TEÓRICOS

Complexidade Computacional

1. Big O Notation

- Limite superior assintótico
- Ignora constantes e termos de baixa ordem
- Foca no crescimento para $n \rightarrow \infty$

2. Big Ω Notation

- Limite inferior assintótico
- Melhor caso possível para um problema

3. Big Θ Notation

- Limite exato assintótico

LISTA DE CONCEITOS PRÁTICOS

Implementação em C

1. Gerenciamento de Memória

```
// Alocação dinâmica
int* arr = malloc(n * sizeof(int));
// Sempre verificar se malloc retornou NULL
if (arr == NULL) {
    printf("Erro de alocação!\n");
    return -1;
}
// Liberar memória
free(arr);
arr = NULL; // Boa prática
```

2. Ponteiros


```
int x = 10;
```

RESUMO EXECUTIVO

Quadro Resumo de Complexidades

Estrutura/Algoritmo	Acesso	Busca	Inserção	Remoção	Espaço
Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Lista Ligada	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Pilha	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Fila	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
BST	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
Hash Table	N/A	$O(1)^*$	$O(1)^*$	$O(1)^*$	$O(n)$

*Complexidade média, pior caso pode ser $O(n)$

 Esta apostila serve como referência rápida para conceitos fundamentais de algoritmos e estruturas de dados. Para aprofundamento, consulte o material detalhado das aulas específicas.

Última atualização: 27 de agosto de 2025