Anotações Algoritmos

Andrey França

March 5, 2017

Contents

1 Introdução

Informalmente, um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída. Portanto, um algoritmo é uma sequência de passos computacionais que transformam a entrada na saída.

Estrutura de Dados

Uma estrutura de dados é um meio para armazenar e organizar dados com o objetivo de facilitar o acesso ou as modificações. Nenhuma estrutura de dados única funciona bem para todos os propósitos, e assim é importante conhecer os pontos fortes e as limitações de várias delas.

1.1 Eficiência

2 Estrutura de Dados

2.1 Listas

Definição 2.1. *Uma lista é um conjunto de nós, cada um desses nós armazenam um objeto ou chave (inteiros, reais, tipos definidos pelo programador etc.) e uma referência para o próximo nó.*

2.1.1 Lista com Vetor

2.1.2 Lista Encadeada

2.1.3 Lista Duplamente Encadeada

2.2 Pilha

2.2.1 Pilha com Vetor

2.2.2 Pilha Encadeada

2.3 Fila

2.3.1 Fila com Vetor

2.3.2 Fila Encadeada

2.4 Arvores

2.4.1 Arvore Binária

2.4.2 Arvore Binária de Busca

2.4.3 Arvore AVL

2.4.4 Arvore B

3 Algoritmos de Ordenação

Algoritmos de ordenação são algoritmos que colocam elementos de uma lista em uma certa ordem. Algoritmos de ordenação eficientes são importantes para

otimizar o uso de outros algoritmos (como busca por exemplo). Algoritmos de ordenção são geralmente classificados por:

- Complexidade;
- Memória;
- Estabilidade;

3.1 Selection Sort

A ordenação por seleção (do inglês, selection sort) é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os *n-1* elementos restantes, até os últimos dois elementos.

Exercício. *Escreva uma função que verifique se um vetor v[0..n-1] está em ordem crescente.*

Exemplo. *Para isto basta criar uma função que percorre todo vetor* `arr` *verificando se o elemento* `arr[i + 1] < arr[i]` *, caso o encontre algum elemento que não se encaixe nessa comparação, então retornaremos falso, isto é, o vetor não esta em orden crescente.*

Listing 1: Verificar se um vetor está em ordem crescente

```
1 bool verifica_ordem(int arr[], int len){
2     for (int i = 0; i < len; i++){
3         if (arr[i] < arr[i+1]) continue;
4         else return false;
5     }
6     return true;
7 }
```

3.1.1 Complexidade

O Selection Sort compara a cada interação um elemento com os outros, visando encontrar o menor. Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno. A complexidade deste algoritmo será sempre $\mathcal{O}(n^2)$ enquanto que, por exemplo, os algoritmos Heapsort e Mergesort possuem complexidades $\mathcal{O}(n \log n)$.

3.1.2 Implementação

Listing 2: Selection Sort implemetação

```
1 void selection_sort(int num[], int tam) {
2     int i, j, min, aux;
3     for (i = 0; i < (tam-1); i++)
```

```
4       {
5           min = i;
6           for (j = (i+1); j < tam; j++) {
7               if(num[j] < num[min])
8               min = j;
9           }
10          if (i != min) {
11              aux = num[i];
12              num[i] = num[min];
13              num[min] = aux;
14          }
15      }
16  }
```

3.2 Insertion Sort

3.3 Bubble Sort

3.4 Merge Sort

3.5 Quick Sort