Dener Vieira Ribeiro (3872), Germano Barcelos dos Santos (3873)

# Avaliando algoritmos de ordenação em diferentes estruturas de dados

Brasil

2019, 07/11

Dener Vieira Ribeiro (3872), Germano Barcelos dos Santos (3873)

# Avaliando algoritmos de ordenação em diferentes estruturas de dados

Prof. Thais R. M. Braga Silva
Algoritmos e Estruturas de Dados I

Universidade Federal de Viçosa - Campus Florestal – UFV-CAF

Ciência da Computação

Graduação

Brasil

2019, 07/11

# Sumário

# Introdução

Ordenar números é uma tarefa cotidiana para os humanos, porém só conseguimos ordenar, com facilidade, uma pequena quantidade de número. Para conseguir realizar a ordenação em grandes conjuntos de números e com diversas comparações recorremos aos recursos computacionais.

Nessa apresentação, realizaremos ordenação em textos e em bibliotecas implementados por duas estuturas de dados - lista encadeada e lista por vetor - utilizando dois métodos: *Selection Sort* e o *Quick Sort*.

# 1 Desenvolvimento

## 1.1 Problema

A tarefa a ser realizada era ordenar textos comparando a primeira letra de cada texto e ordenar bibliotecas comparando o tamanho de cada uma. Para realizar testes foi necessário a geração aleatória dos dados.

## 1.2 Estruturação do Problema

Para resolver o problema foi nos pedido para que criássemos 3 Tipo Abstrato de Dados (TADs): TAD Palavra, TAD Texto, TAD Biblioteca. Além disso houve a necessidade da criação das duas estruturas de dados: lista encadeada e lista com vetor; cada uma possui os TADs correspondentes.

Como a Biblioteca possui uma lista de Texto e a lista de Texto possui uma lista de Palavras, foi visto que as operações eram realizadas em cascatas, ou seja, para inserir um texto em uma biblioteca era preciso criar uma palavra primeiro.

Para que pudéssemos executar algumas operações com as estruturas de dados implementamos algumas funções como: criar, inserir, remover. Utilizamos a operação de inserir no momento de gerar automaticamente os dados para compor os TADs.

A geração[1] automática dos dados foi dada por uma função da biblioteca *time.h*

```
(char) ((rand() % 26) + (rand() % 2 ? 'A' : 'a'));
```

Código: 1.1 – geração de letra automática

### 1.2.1 Explicação das bibliotecas

Separamos nosso código em diversas bibliotecas (.h) para que o código ficasse mais enxuto e de mais fácil manutenção. A divisão foi feita por responsabilidade de cada biblioteca, portanto existe uma biblioteca para a interface usuário-software, métricas, ordenação, implementação da lista com vetor, implementação da lista com vetor.

Em primeiro lugar, na biblioteca interface usuário-software foi criado um TAD para mapear opções do usuário de acordo com o menu fornecido através do sistema.

O TAD de métricas foi feito para mapear as métricas, informações, pedidas de cada algoritmo de ordenação, como: tempo de execução, movimentações, comparações.

[1] Esse algoritmo simplesmente considera o tabela ASCII para gerar os dados.

Cada algoritmo possui sua biblioteca, sendo assim, existe o *selectionSort.h* e o *quicksort.h* cada um com a implementação para acada estrutura de dados.

Há ainda, as bibliotecas das estruturas que foram implementadas, especialemente, para operações essenciais de cada estrutura.

### 1.2.2 Dificuldades na implementação

Para que houvesse um maior aproveitamento da memória foi nos pedido que na implementação da ordenação da lista encadeada trocassem os ponteiros e não os valores contidos neles, o que gerou uma grande dificuldade especialmente para implementar o quick sort. Além disso, tivemos problemas de pilha, um erro que foi difícil de achar, estávamos usando um vetor estático, o que estourava a memória da pilha e apontava *segmentation fault*. Além disso, tivemos que reimplementar a lista encadeada transformando-a em uma lista duplamente encadeada, alterando todas as funções próprias de sua biblioteca.

## 1.3 Explicação do Algoritmo

Algoritmo de ordenação é um passo que muda a ordem dos dados de acordo com a comparação desejada.

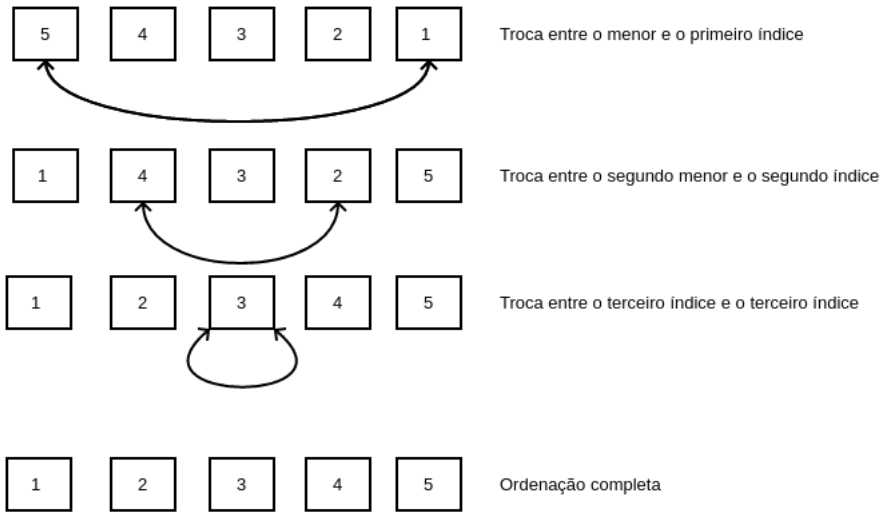
### 1.3.1 Selection Sort

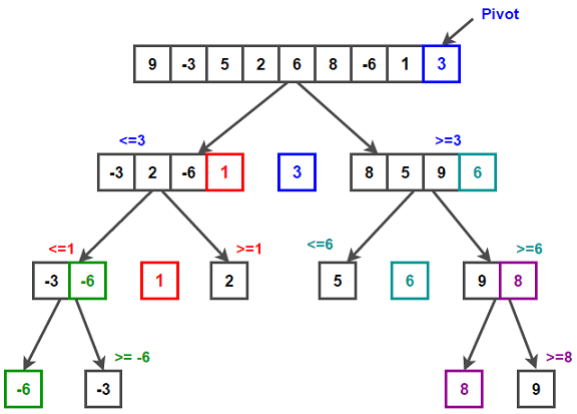
O Selection Sort ou Algoritmo de Seleção é um método simples, porém eficaz para um pequeno conjunto de dados. O nome implica o processo do algoritmo, o qual utiliza a escolha do menor número sempre dentre o vetor/lista.

Para cada número no vetor o algoritmo procura linearmente o menor valor dentre todos os valores do vetor, percorrendo n*n vezes. $O(n^2)$, sendo $n$ o tamanho do vetor.

### 1.3.2 Quick Sort

No entanto, o quick sort é um método mais robusto, opera bem para uma maior quantidade de dados. O funcionamento é da seguinte forma: há uma escolha randômica ou não do pivô, para fins explicativos adotaremos o pivô como sendo o último índice do array. Depois da definição do pivô ocorre o processo da partição, uma separação a esquerda de dados menores que o pivô e os dados maiores ficam a direita do pivo. A partir disso, ocore o que chamamos de loop invariante: a configuração do array será essa até a ordenação ser completada, logo ocorre a separação entre esquerda e direita na primeira partição e assim por diante, sempre deixando os dados menores que o pivô a esquerda e os dados maiores a direita.




Nesse algoritmo, há a ideia de dividir o vetor em partes menores e além disso o algoritmo percorre em pré-ordem criando uma árvore implicando em $O(nlogn)$, sendo $n$ o tamanho do vetor.

### 1.3.3 Os algoritmos aplicado ao problema

Para aplicá-los ao nosso problema foram feitas poucas alterações, apenas no código de comparação. Para o TAD Texto, percorremos a lista do TAD Palavra e avaliamos se a primeira letra da palavra era maior/menor que o dado de referência. Para ordenar a biblioteca, percorremos cada texto avaliando o tamanho do texto.

## 1.4 Tempo de Execução

Para medir o tempo do algoritmo utilizamos a biblioteca `time.h` e a função `clock()` usando os ciclos do cloks do sistema para medir com precisão quantos segundos o algoritmo gastou determinando a solução do problema. Testamos o algoritmo de acordo com a tabela abaixo:

### 1.4.1 Tempo de Execução para ordenação do TAD Texto

| Número de palavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|
| 10 | 4950 | 99 | 0.0 |
| 1000 | 499500 | 999 | 0.029 |
| 10000 | 49995000 | 9999 | 5.78 |
| 100000 | - | - | - |

Tabela 1 – Tempo de Execução do algoritmo Selection Sort para TAD Textos implementado com vetor

| Número de palavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|
| 10 | 4950 | 99 | 0.0 |
| 1000 | 499500 | 999 | 0,035 |
| 10000 | 49995000 | 9999 | 3.318 |
| 100000 | - | - | - |

Tabela 2 – Tempo de Execução do algoritmo Selection Sort para TAD Textos implementado com lista duplamente encadeada

| Número de palavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|
| 10 | 1290 | 201 | 0.001 |
| 1000 | 18200 | 3480 | 0.003 |
| 10000 | 242000 | 51100 | 0.011 |
| 100000 | 30900000 | 673000 | 1.55 |

Tabela 3 – Tempo de Execução do algoritmo Quick Sort para TAD Textos implementado com Vetor

| Número de palavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|
| 10 | 1370 | 178 | 0.001 |
| 1000 | 17200 | 3100 | 0.001 |
| 10000 | 243000 | 46500 | 0.01 |
| 100000 | 30300000 | 627000 | 1.85 |

Tabela 4 – Tempo de Execução do algoritmo Quick Sort para TAD Textos implementado com Lista duplamente encadeada

### 1.4.2 Tempo de Execução para ordenação do TAD Biblioteca

| Número de textos | MinPalavras/MaxPalavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|---|
| 100 | 1/100 | 4950 | 99 | 0.0 |
| 1000 | 50000/100000 | 499500 | 999 | 0.0 |
| 10000 | 1/100 | - | - | |
| 100000 | 50000/100000 | - | - | |

Tabela 5 – Tempo de Execução do algoritmo Selection Sort para TAD Biblioteca implementado com vetor

### 1.4.3 Tempo de Execução para ordenação do TAD Biblioteca

| Número de textos | MinPalavras/MaxPalavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|---|
| 100 | 1/100 | 4950 | 99 | 0.0 |
| 1000 | 50000/100000 | 499500 | 999 | 0.0001 |
| 10000 | 1/100 | - | - | |
| 100000 | 50000/100000 | - | - | |

Tabela 6 – Tempo de Execução do algoritmo Selection Sort para TAD Biblioteca implementado com lista duplamente encadeada

| Número de palavras | MinPalavras/MaxPalavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|---|
| 10 | 1/100 | 1490 | 160 | 0.0 |
| 1000 | 50000/100000 | 1560 | 179 | 0.0 |
| 10000 | 1/100 | 3130000 | 577000 | 0.035 |
| 100000 | 50000/10000 | - | - | |

Tabela 7 – Tempo de Execução do algoritmo Quick Sort para TAD Biblioteca implementado com vetor

| Número de palavras | MinPalavras/MaxPalavras | Comparação | Movimentação | Tempo (segundos) |
|---|---|---|---|---|
| 10 | 1/100 | 1389 | 190 | 0.0 |
| 1000 | 50000/100000 | 1440 | 179 | 0.0 |
| 10000 | 1/100 | 3110000 | 622000 | 0.12 |
| 100000 | 50000/10000 | - | - | - |

Tabela 8 – Tempo de Execução do algoritmo Quick Sort para TAD Biblioteca implementado com lista duplamente encadeada

Concluindo, a utilização do algoritmo Quick Sort é muito mais vantajosa para casos onde o número de itens a ser ordenado é grande. Por outro lado, para poucos itens, é possível perceber que o algorítmo Selection Sort se destaca. As diferenças de implementações entre vetor e lista duplamente encadeada não tiveram grande impacto no tempo de execução.

## 1.5 Configuração do Hardware

Para executar os testes utilizamos a seguinte configuração:

- Processador: Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz 2.60GHz
- Memória RAM: 7.89GB
- Sistema Operacional: Windows 10 Enterprise

# 2 Conclusão

Em resumo, os algoritmos devem ser aplicados com extremo cuidado em determinadas situação. Todos os programadores deveriam analisar cuidadosamente o caso necessário para sua aplicação, pois para casos com entradas pequenas não há a necessidade de usar o QuickSort, porém para casos maiores o Selection Sort se torna inviável.