Análise da Ordenação por Seleção de Raiz Quadrada

Giovanni Lucas Oliveira da Silva[1]

[1]Ciência da Computação Instituto Federal de Brasília Campus Taguatinga (IFB) 72146-050 – Brasília – DF – Brazil

giovanni61540@estudante.ifb.edu.br

Abstract. *This work aims to analyze the square root selection sorting algorithm. Both an empirical approach, through practical experimentation, and an analytical analysis will be used to understand the performance and efficiency of this algorithm in different scenarios. By combining these approaches, we hope to gain a comprehensive understanding of the strengths and weaknesses of this sorting algorithm, allowing us to make informed decisions when choosing the algorithm for sorting situations.*

Resumo. *Este trabalho tem como objetivo analisar o algoritmo de ordenação por seleção de raiz quadrada. Tanto uma abordagem empírica, através de experimentação prática, quanto uma análise analítica serão utilizadas para compreender o desempenho e a eficiência desse algoritmo em diferentes cenários. Ao combinar essas abordagens, esperamos obter uma compreensão abrangente dos pontos fortes e fracos deste algoritmo de ordenação, permitindo-nos tomar decisões informadas ao escolher o algoritmo para situações de ordenação.*

1. Introdução

Algoritmos de ordenação desempenham um papel fundamental na ciência da computação, tanto em termos de teoria quanto de aplicações práticas pois organizar informações facilita a procura da mesma quando solicitada a sua utilização. O seu uso em estruturas de dados fazem com que elas se rearranje em alguma ordem, ou seja, um processo que altera a posição das informações com a estratégia de ordena-las em uma sequência.

Vetor é um tipo de estrutura de dados em que estes dados são armazenados continuamente na memória, um atraz do outro e são do mesmo tipo. O computador é uma maquina que consegue trabalhar com esse arranjo de uma forma bem rápida quando eles estão organizados em ordem do menor para o maior (ou vice-versa), e o algoritmo de ordenação por seleção de raiz quadrada pretende desepenhar essa função de manipular um vetor até que ele esteja ordenado.

Esta análise pretende comparar duas formas de implementação do algoritmo e investigar como ele se comporta em cada aplicação e avaliar o desempenho tampo por meio de análise prática, comparando seus tempos de execução em diferentes tamanhos de vetores, quanto análise teórica para compreender a complexidade e com isso entender melhor este algoritmo.

2. Conceitos preliminares

Para prosseguirmos é necessário explicar como o algoritmo irá desempenhar sua função e alguns outros pocedimentos que usaremos para desenvolver o código que usaremos para ordenar os vetores.

2.1. Algoritmo de ordenação quadrático de inserção

O algoritmo de inserção ou “Isertion Sort”, ordena uma lista começando com o segundo

elemento da lista, considerando-o como o "elemento atual". Em seguida, compara esse "elemento atual" com os elementos anteriores na lista ordenada. Se o "elemento atual" for menor do que o elemento comparado, o algoritmo move o elemento comparado uma posição para a direita para abrir espaço para o "elemento atual". Esse processo de comparação e movimento para trás na lista continua até que o "elemento atual" encontre sua posição correta na lista ordenada. Então, o algoritmo avança para o próximo elemento na lista não ordenada e repete esse processo até que todos os elementos tenham sido inseridos na lista ordenada.

O Insertion Sort é eficaz para listas pequenas ou quase ordenadas, mas pode ser menos eficiente em comparação com outros algoritmos de ordenação, como o Merge Sort ou o Quick Sort, para listas maiores, devido ao seu tempo de execução quadrático. No entanto, usaremos ele na implementação da ordenação por raiz quadrada.

2.2. Heap

Uma heap é uma estrutura de dados em forma de árvore binária que possui duas propriedades principais: a propriedade de heap e a propriedade de ordenação. Na propriedade de heap, em uma heap binária, cada nó é maior ou igual aos seus filhos (no caso de uma heap max) ou menor ou igual aos seus filhos (no caso de uma heap min).

Isso significa que o nó pai é sempre maior ou igual (ou menor ou igual) do que seus filhos, garantindo que o elemento mais significativo (o maior ou menor, dependendo do tipo de heap) esteja sempre na raiz. Na propriedade de ordenação, uma heap é uma árvore binária completa, o que significa que todos os níveis da árvore estão completamente preenchidos, exceto talvez o último nível, que é preenchido da esquerda para a direita. Isso facilita a representação da heap em uma estrutura de dados como um array.

O funcionamento básico de uma heap envolve operações de inserção e remoção. Para inserir um elemento em uma heap, o novo elemento é adicionado ao final do array e, em seguida, é "percolado para cima" ou "subido" na árvore até que a propriedade de heap seja restaurada. Isso significa que o novo elemento é trocado com seu pai até que todos os pais sejam maiores ou iguais (ou menores ou iguais, dependendo do tipo de heap) do que seus filhos. Para remover um elemento de uma heap, geralmente é removido o elemento na raiz (o elemento mais significativo) e, em seguida, o último elemento do array é movido para a raiz. Depois disso, o novo nó raiz é "percolado para baixo" ou "descido" na árvore até que a propriedade de heap seja restaurada novamente.

Essas operações de inserção e remoção garantem que a heap mantenha suas propriedades principais, permitindo que seja usada eficientemente em algoritmos como a ordenação por seleção de raiz quadrada.

2.3. O algoritmo da ordenação de raiz quadrada

O algoritmo de ordenação por seleção de raiz quadrada opera da seguinte maneira, começa dividindo logicamente o conjunto de entrada em várias partes de tamanho aproximado, cada uma contendo $\sqrt{n}$ elementos, onde n representa o tamanho total do conjunto. Se o tamanho do conjunto não for um múltiplo exato de $\sqrt{n}$, a última parte terá um tamanho diferente isto é as outras partes teram o valor inteiro de $\sqrt{n}$ e esta última tera n modulo de $\sqrt{n}$.

Em seguida, para cada uma dessas partes, o algoritmo identifica o maior item presente. Uma vez localizado o maior elemento de cada parte, ele é adicionado a um conjunto de solução. Se o maior elemento for encontrado em uma parte específica, ele é

retirado dessa parte para evitar duplicatas no conjunto de solução.

Esse processo de identificação e adição do maior elemento em cada parte é repetido até que todas as partes estejam vazias. Ao final, o conjunto de solução conterá todos os elementos do conjunto de entrada ordenados do menor para o maior.

A identificação do maior item ocorrerá de duas formas neste trabalho, usando o metodo de ordenção de inserção e o metodo heap com suas funções, para assim encontrar o maior elemento e comparar o seu desempenho das duas formas.

3. Análise empírica

Para a realização dos codigos e experimentos foi utilizada a linguagem de programação C para a implementação dos algoritmos, e para sua compilação foi utilizada a flag de otimização "-O3". Para cada uma das versões do algoritmo foi utilizado um vetor de tamanho "n", onde ele pode assumir valores de 10^4, 10^5, 10^6, 10^7 e 10^8, a população do vetor foi efetuada de forma pseudo-aleatória.

Os experimentos foram conduzidos utilizando um script "Bash" para realizar a execução dos códigos cinco vezes e realizar a média aritmética dos tempos de execução das linhas de código referente a cada ordenação proposta. Os resultados encontrados foram atribuídos em tabelas para a geração de gráficos para melhor visualização.

Tabela 1. Representação dos tempos de execução em segundos.

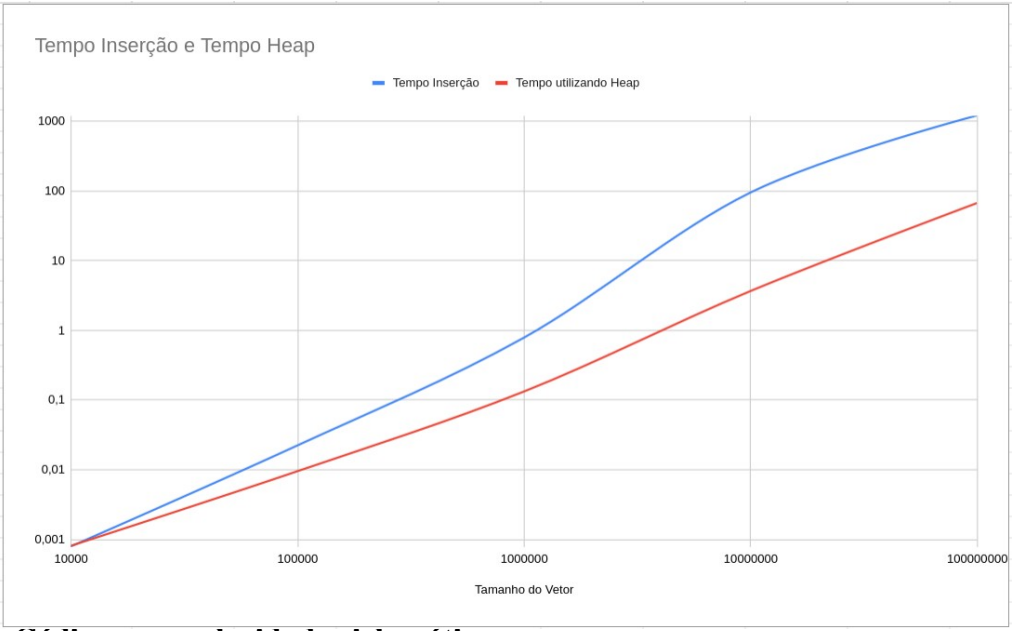
Tamanho do Vetor	Tempo Inserção	Tempo utilizando Heap
10000	0,000780	0,000797
100000	0,021991	0,009385
1000000	0,774853	0,130727
10000000	93,88395	3,624790
100000000	1.202,59	66,493118

3.1. Analise

Durante os experimentos é possível notar que mesmo utilizando o algoritmo de inserção na criação da ordenação por raiz quadrada, quando o vetor têm um tamanho de 10^4 ele é mais rápido que o que utiliza a heap devido ao tempo de criação da estrutura, e apartir deste ponto para os outros tamanhos a heap se torna mais vantajosa pois o tempo e ordenação quadrático fica mais lento que a de criação da estrutura e utilização dos metodos da heap.

Com isso é possível entender que para vetores menores o algoritmo de ordenação por seleção de raiz quadrada implementado com o insertion sort é mais adequado e para vetores maiores o tempo de funcionamento melhora substancialmente, "o algoritmo de classificação de raiz quadrada tem a menor complexidade de tempo em comparação com alguns dos algoritmos existentes, especialmente no melhor caso, e no pior caso também tem menos complexidade de tempo do que alguns dos algoritmos existentes" [Mir Omranudin Abhar e Nisha Gatuam 2019].

Figura 1. Tempo de execução dos algoritmos (tempo vs tamanho do vetor).



3.2. Código e complexidade ciclomática

O código utilizado nestes experimentos se encontra no endereço eletrônico: https://github.com/Giovanni-LOS/Sqrt_Sort, e o valor da complexidade ciclomática obtida automaticamente pelo site https://sonarcloud.io gerou que utilizando o algoritmo de inserção teve a complexidade ciclomática de 18 e a utilizando a heap chegou a 27 de valor ciclomático.

3.3. Ambiente experimental

Todos os testes foram realizados em um notebook com o sistema operacional Arch Linux x86_64 com processador Intel i7-1165G7 (8) @ 4.700GHz de décima primeira geração com 16 gigabytes de memória RAM.

References

Mir Omranudin Abnar e Nisha Gatua (2019) “Square Root Sorting Algorithm”, Em: JETIR Junho 2019, Volume 6, Problema 6, Disponivel em: https://www.slideshare.net/MirOmranudinAbhar/square-root-sorting-algorithm, acessado em Abril de 2024.