COMPLEXIDADE DE ALGORITMOS

Eliane Cristina da Silva Oliveira[1]

Resumo

Algoritmos são processos computacionais bem definidos que tem como objetivo a realização de uma tarefa. Eles são capazes de receber uma entrada, processá-la e produzir um ou mais valores como saída. A execução de um algoritmo demanda tempo de processamento e armazenamento. A complexidade de um algoritmo tem a ver com quanto tempo e memória esse algoritmo gasta de acordo com o tamanho de sua entrada. Um mesmo problema pode ser resolvido por diferentes algoritmos e nem todos eles com a mesma complexidade.

Palavras-chave: Algoritmos. Análise. Complexidade. Ordenação.

Abstract

Algorithms are well-defined computational processes that aim to accomplish a task. They are able to take an input, process it, and produce one or more values as output. The execution of an algorithm demands processing and storage time. The complexity of an algorithm has to do with how much time and memory it spends according to the size of it's input. The same problem can be solved by different algorithms, and not all of them with the same complexity.

Keywords: *Algorithms. Analysis. Complexity. Sorting.*

1 Introdução

Na ciência da computação, algoritmos, são processos computacionais bem definidos, que recebem uma entrada, a processa e produz como saída um ou mais valores.

No geral, algoritmo, é uma ferramenta para solução de um problema bem definido e, problemas são conjuntos de dados específicos, que se deseja analisar para obter um resultado.

Comparar a complexidade de dois ou mais algoritmos serve para mostrar qual algoritmo e melhor para a solução de um problema. Ou seja, o melhor algoritmo para a solução de um problema, é aquele que consome menos tempo e espaço.

[1] Graduando em [...] pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: [eliane.oliveira11@fatec.sp.gov.br].

A Complexidade de Algoritmos, analisa um algoritmo sem considerar qualquer implementação de linguagens específicas ou hardware.

O projeto de um algoritmo deve considerar como será seu desempenho após ser implementado.

Pode-se calcular a complexidade de um algoritmo anotando o tempo que leva para sua execução em um computador real ou através de uma fórmula que dê o número exato de operações feitas pelo algoritmo para chegar no resultado, em função do tamanho da entrada. Porém ambas as duas formas são muito trabalhosas e em geral, não valem a pena. Por isso o método mais indicado para fazer esse cálculo é a “complexidade assintótica”.

2 Referencial teórico e trabalho correlatos

2.1 Complexidade assintótica

Através dessa análise, o desempenho de um algoritmo é representado através de uma curva de tendência aproximada. Nela é considerado a medida que o conjunto de dados tende ao infinito será considerado apenas o termo de maior grau da equação, ou seja, aquele que mais cresce, descartando todos os outros.

Segundo Borin [2020. P. 19] “Nosso objetivo na análise de algoritmos é encontrar uma forma de construir o algoritmo de tal forma que o aumento do tempo de execução seja o menor possível em detrimento ao crescimento do conjunto de dados de entrada.”

2.1.1 Notação Big-O

Na complexidade assintótica, é utilizada para destacar o crescimento superior do algoritmo.

Dessa forma pode-se dizer que a complexidade de um algoritmo é "Big-O de ~~O~~(n)", ou seja, no pior caso cresce em ordem de n. Dependendo do conjunto de dados de entrada, a situação mostrada nessa notação nunca será melhor do que um código.

2.2 Bubble sort

Bubble sort, é um algoritmo de ordenação por troca. Consistem em percorrer um vetor e ir trocando as posições dos elementos com o objetivo de ordenar a lista. Dessa forma de qualquer maneira é necessário percorrer o vetor inteiro no mínimo uma vez.

Neste algoritmo de ordenação, os dados armazenados são comparados em um vetor que possui tamanho *n*.

O pior caso desse algoritmo é quando o vetor está em ordem decrescente, onde a estrutura *enquanto* será executada *n* vezes. A estrutura *para* que possui n-1 iterações, também, será executada *n* vezes. Assim, $n(n\text{-}1) = n^2 - n$, será o número de comparações. Assim em a complexidade desse algoritmo será $\Theta(n^2)$, pois será necessário executar n^2 passos para cada *n* elemento que vai ser ordenado.

Esse método é recomendado apenas para quem deseja aprender e não é indicado para ser usado no desenvolvimento de aplicações profissionais.

2.3 Selection sort

Nesta ordenação os elementos do vetor, serão escolhidos e comparados com o menor ou maior, de acordo com a forma na qual se deseja ordenar, que será um número entre os que estão à direita do escolhido.

Em um vetor com n posições, cada elemento i da primeira até a penúltima posição troca de lugar com o menor elemento.

Dessa forma, pode-se mostrar através da soma dos termos, o tempo de execução:

$$T(n) = 1 + 2 + 3 + \cdots + n - 1$$

Essa fórmula possui uma progressão aritimética de razão 1 e é calculada aplicando a fórmula da soma dos termos de uma PA:

$$T(n) = \frac{(a_1 + a_n).n}{2}$$

$$T(n) = \frac{(1 + n - 1).(n - 1)}{2}$$

$$T(n) = \frac{n.(n - 1)}{2}$$

$$T(n) = \frac{n^2 - n}{2}$$

$$T(n) = \frac{n^2}{2} - \frac{n}{2}$$

A complexidade desse algoritmo é $\Theta(n^2)$ o que significa que ele exige n^2 passos para cada n elementos do vetor.

O selection sort se comportará da mesma forma para qualquer valor de entrada. Ele é ideal para ordenações em pequena escala e que os elementos estejam em ordem aleatória.

2.4 Quick Sort

Esse método de ordenação, através da recursividade divide o vetor em dois, até restar apenas um elemento. Enquanto o vetor é repartido os elementos são ordenados.

O pior caso é quando a divisão do vetor faz com que uma das partes fique com n-1 elementos e outra com apenas um elemento. Dessa forma obtém-se a fórmula:

$$T(n) = T = 1 + n^2 - n$$

Portanto, sua complexidade é de $\Theta(n^2)$ para c=2, n ≥1.

É recomendado para ordenar listas de qualquer tamanho já que é rápido e eficiente e seu funcionamento para listas grandes é muito bom, além de ordenar a lista no próprio local, não precisando de espaço adicional de armazenamento. Porém não é um algoritmo estável.

2.5 Merge Sort

Esse algoritmo divide o vetor em duas metades, através da recursividade até restar apenas um elemento a ser ordenado e intercalado.

Pode ser calculado pela fórmula:

$$T(n) = n + n \log n$$

Assim, a complexidade de pior caso é $\Theta(n \log n)$.

Apesar de ser estável e não alterar a ordem de dados iguais, possui um gasto extra de espaço de memória quando comparado aos outros.

3 Materiais e métodos ou desenvolvimento

O trabalho foi desenvolvido em duas partes para facilitar o entendimento. Na primeira parte, foi feita uma introdução para explicar o conteúdo do trabalho desenvolvendo o conceito geral de algoritmos e do que é a complexidade de algoritmos. Na segunda parte, foram analisados e explicados individualmente cada um dos quatro algoritmos de ordenação estudados e feita uma análise da complexidade de cada um. Foram utilizadas as fontes bibliográficas recomendadas pelo professor, além de pesquisas em outros materiais para que fosse possível explicar cada um dos algoritmos de forma clara e para verificar a coerência das informações apresentadas.

4 Resultados e discussão

Após o estudo e desenvolvimento de todo o conteúdo foi possível concluir que a complexidade de algoritmos é importante para desenvolver aplicações da melhor forma possível, para que o cliente possa atingir seus objetivos com o uso do sistema, sem haver problemas ou complicações e evitar mal funcionamento do software por conta de incompatibilidade com o hardware, fazendo com que o usuário tenho uma boa experiência de uso.

Fazendo uma análise individual da complexidade de cada algoritmo, pode-se concluir que, o bubble sort e o merge sort não tem seu uso recomendado, o primeiro pois sua eficiência diminui de uma forma muito grande de acordo com o aumento do número de elementos do vetor, e o segundo pois ocupa um espaço muito grande na memória o que pode gerar conflitos dependendo das características do hardware, não devendo ser utilizado no desenvolvimento de aplicações para ambientes profissionais. Em casos de ordenações em pequena escala pode-se usar o selection sort. Porém, o mais recomendado para uso em ambientes profissionais é o quick sort, pois é sua execução é rápida e eficiente para ordenações em grande escala.

Considerações finais

O trabalho propôs, como objetivo geral, analisar a complexidade dos algoritmos de ordenação estudados. Foi desenvolvido o conceito geral da complexidade de algoritmos e do método de análise utilizado para calcular a

complexidade de um algoritmo, uma breve explicação sobre cada algoritmo de ordenação estudado, e o cálculo da complexidade de cada um desses algoritmos individualmente. Pode-se dizer que o tema é algo muito importante e complexo. Complexo, pois foram necessárias muitas pesquisas e a consulta de várias fontes diferentes, pois alguns dos materiais consultados não traziam muitas informações sobre o conteúdo, principalmente sobre o merge sort, que foi o mais difícil de achar informações. Importante, pois o tema aborda questões que são de extrema importância e que devem ser consideradas no desenvolvimento de uma aplicação e na escolha de qual algoritmo usar, de acordo com o objetivo que o usuário espera atingir com o uso e o ambiente de implementação do software. Acredito que este trabalho vai ajudar futuramente, em questões profissionais, para executarmos com eficiência nosso trabalho como desenvolvedores de software, para que possamos entregar ao cliente a melhor versão do nosso produto.

Referências

ASCENCIO, Ana Fernanda Gomes; ARAÚJO, Graziela Santos de. **Estruturas de dados:** algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson Prentice Hall, 2010. Disponível em: <https://pt.b-ok.lat/book/11246646/b62426>. Acesso em: 6 junho 2022.

BORIN, Vinícius Pozzobon. **Estruturas de Dados**. Curitiba: Contentus, 2020. Disponível em: <https://pt.b-ok.lat/book/12671797/eb35fe>. Acesso em: 6 junho 2022.

PIVA JUNIOR, Dilermando et al. **Estruturas de dados e técnicas de programação**. Rio de Janeiro: Elsevier, 2014. Disponível em: <https://pt.b-ok.lat/book/2569882/b8ec9f>. Acesso em: 7 junho 2022.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro: LTC, 2015. Disponível em: <https://pt.b-ok.lat/book/3691700/554a08>. Acesso em: 7 junho 2022.

MEDIUM, Introdução à Complexidade de Algoritmos, 16/03/2019.Disponível em: <https://medium.com/nagoya-foundation/introdu%C3%A7%C3%A3o-%C3%A0-complexidade-de-algoritmos-4a9c237e4ecc#:~:text=Um%20algoritmo%20pode%20ser%20melhor,um%20programa%20executa%20suas%20computa%C3%A7%C3%B5es>. Acesso em: 6 junho 2022.

BACKES, André. (2011). Algoritmos de Ordenação. [PowerPoint de apoio, lecionado na FACOM, UFU]. Disponível em:

<https://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>. Acesso em: 7 junho 2022.