Comparação entre algoritmos de ordenação e Busca em um grande volume de dados da Covid-19

Igor Pereira Nascimento[1], **Karoline Batista Silva**[1], **Ygor Borges Arêda**[1]

[1]Instituto de Educação Superior de Brasília - IESB

igorpg13@gmail.com, karolinebatistasilv@gmail.com, ygorareda@gmail.com

Abstract. *This article proposes to test and analyze the execution time of ordering and search algorithms in a database about Covid-19. Using sequential and binary search algorithms with Insertion Sort, Bubblesort, Selection Sort, Quicksort, Merge Sort and Heapsort.*

Resumo. *Este artigo tem a proposta de testar e analisar o tempo de execução de algoritmos de ordenação e busca em uma base de dados sobre a covid-19. Algoritmos de busca sequencial e binária com Insertion Sort, Bubblesort, Selection Sort, Quicksort, Merge Sort e Heapsort.*

1. Introdução

Devido à alta demanda de busca de informações sobre a COVID-19 pelas autoridades de saúde e científicas, por ser uma doença de padrão desconhecido, apresenta-se uma grande quantidade de dados sobre os infectados. Para tratar esta devida quantidade de informação, torna-se necessário a criação de uma estrategia eficiente para a manipulação desses dados, utilizando-se procedimentos computacionais, de tal forma que economize tempo de busca. Uma das necessidades é a ordenação dos dados de forma específica, como quantidade de casos que vieram a óbito, pacientes internados e recuperados.

2. Objetivo

2.1. Objetivos Gerais

Analisar e comparar a performance e a eficiência de algoritmos de ordenação e algoritmos de busca em uma base de dados com uma grande quantidade de informações relacionadas ao COVID-19. Algumas ferramentas que podem ser utilizadas para a realização dessas buscas, são, por exemplo, os algoritmos de busca binária e busca sequencial.

2.2. Objetivos específicos

Neste trabalho pretende-se implementar os algoritmos para ordenação: Insertion sort, Merge sort, Quicksort, Bubble sort, Selection sort e Heapsort; juntamente com algoritmos de busca sequencial e busca binária. Efetuando-se testes para verificar seus tempos de processamento e viabilidade em cada tipo de busca, que serão alcançados através da simulação de busca de casos em uma base de dados referente à COVID-19, disponibilizada publicamente.

3. Referencial teórico

3.1. Insertion sort

Seu funcionamento se dá por comparação e inserção direta. A medida que o algoritmo varre a lista de elementos, o mesmo os organiza, um a um, em sua posição mais correta, onde o elemento a ser alocado (k) terá, a sua esquerda um valor menor (k-1), e, de maneira similar, à sua direita um valor maior (k+1). Este algoritmo possui complexidade O(n) no melhor e no pior caso.

3.2. Merge Sort

Sua ideia básica consiste em dividir e conquistar: primeiro divide-se o problema em vários subproblemas para então resolver esses subproblemas através da recursividade e após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas. Este algoritmo possui complexidade O(n log n) no melhor e no pior caso.

3.3. Quicksort

O quicksort é um algoritmo recursivo e consiste em colocar um vetor em ordem crescente. Assim como o merge sort, utiliza o método de dividir e conquistar e é um algoritmo linearítmico O(n log n), no melhor caso, quando o conjunto de elementos está dividido igualmente, e quadrático O(n) no pior caso, quando o conjunto de elementos está desbalanceado ou quando o array estiver ordenado.

3.4. Bubble sort

No algoritmo bubble sort, comparam-se dois elementos e trocam-se suas posições se o segundo elemento é menor do que o primeiro. É um dos algoritmos mais simples levando em consideração sua implementação, mas não é o mais performático se comparado com outros. Este algoritmo possui complexidade O(n) no melhor caso, quando o conjunto de elementos se encontra ordenado, e complexidade O(n) em seu pior caso, quando o conjunto de elementos se encontra na ordem inversa à desejada.

3.5. Selection Sort

A ordenação por selection sort envolve trocar o menor(ou maior) elemento na lista com o elemento no início da lista, em seguida, trocar o segundo menor elemento para a segunda posição e, assim por diante, trocar os restantes (n-1) elementos até os dois últimos elementos. Este algoritmo possui complexidade O(n) no melhor e no pior caso.

3.6. Busca Binária

O algoritmo de busca binária utiliza um método diferente da busca sequencial. Envolve primeiro a comparação do elemento pesquisado com o elemento central do conjunto, de tal modo que a metade possa ser ignorada, reduzindo assim o intervalo de pesquisa. Para que essa comparação ocorra, o conjunto de elementos deve estar previamente organizado em ordem crescente, então verifica-se se o elemento buscado é maior ou menor que o elemento centralizado, de forma que a busca comece em uma das duas metades. Este algoritmo possui complexidade O(1) no melhor caso, se o elemento estiver centralizado, e O(log n) no pior caso, o elemento não existe.

3.7. Busca Sequencial

Esta é uma maneira mais fácil de pesquisar. Dado um conjunto de variáveis, de qualquer tipo, que existam no programa, verificando a começar pelo primeiro elemento, um a um, se é o mesmo que você está procurando. Assim que for encontrado o elemento, seu índice será retornado imediatamente e a pesquisa será concluída. Se o elemento não existir no conjunto, o programa retornará -1. Este algoritmo tem complexidade O(1) no seu melhor caso, quando o elemento se encontra na primeira posição, e O(n) no pior caso, quando o elemento se encontra na última posição.

4. Metodologia

Neste trabalho foram implementadas, dentro de um código, diversas funções essenciais para se fazer a verificação proposta. A princípio, é executada uma função para ler um arquivo de dataset, que contém os dados relacionados aos casos de Covid-19 no Brasil. As informações contidas no dataset, são: cidade, data, população, população 2019, casos confirmados, casos confirmados 100k, ultima data, ultima taxa de mortes, ultimas mortes, estado, novos confirmados, novas mortes. Foram utilizados nos testes de ordenação e busca, somente: população, casos confirmados e últimas mortes. Foram analisados cerca de 250 mil dados no geral, compreendidos no dataset, lidos como um vetor de dados tipo 'string'. Para a ordenação, os dados foram convertidos em dados do tipo 'int' e armazenados em um vetor, passado como parâmetro para as funções de ordenação. Para as funções de busca os dados foram ordenados e, em seguida, realizadas as buscas, sequencial e binária.

5. Implementação Computacional

Os testes de implementação dos algoritmos de busca e ordenação, foram realizados em um Desktop com 8GB de Memória RAM, Processador Intel(R) Pentium(R) CPU G4560 3.5GHz.

6. Conclusão

Para analisar o tempo de execução entre as buscas e ordenações, proposto neste trabalho, foram simuladas opções de buscas ordenadas em um arquivo dataset de casos de Covid-19 no Brasil, utilizando funções de ordenação e de busca. Foram encontradas dificuldades de execução do código, devido ao alto consumo de memória de alguns algoritmos, como Bubblesort, Selection sort, Insertion sort. Enquanto os algoritmos que tiveram melhor desempenho nas simulações de ordenação, foram Quicksort e Merge Sort. As buscas tiveram resultados semelhantes, contudo a busca binária se sobressaiu nos resultados.

References

GOODRICH, Michael T.; TAMASSIA, Roberto; GOLDWASSER, Michael H. Data Structures and Algorithms in Python. Hoboken: Wiley, 2013.

LAUREANO, Marcos. Estrutura de Dados com Algoritmos e C. Rio de Janeiro: Brasport, 2008.

TENEMBAUM, Aaron M.; AUGENSTEIN, Moshe J; LANGSAM, Yedidyah. Estrutura de Dados Usando C. São Paulo: MAKRON Books, 1995.