Análise dos algoritmos

Relatório elaborado como parte de um projeto da disciplina de Estrutura de Dados na Universidade Federal de Alagoas (UFAL). O trabalho busca aplicar diferentes tipos de algoritmos de ordenação e analisar sua eficiência em um dataset escolhido pela equipe.

Equipe:
Gustavo Henrique dos Santos Malaquias - 22111978
Luis Henrique Amorim da Silva - 22111977
Jairon José Tavares dos Santos – 22112583

Introdução

Neste relatório, analisaremos a eficiência dos algoritmos Quick Sort, Radix Sort e Bucket Sort aplicados a um dataset de filmes, com mais de 15000 linhas, retirado do site Kaggle. Retiramos algumas colunas que julgamos desnecessárias para a análise do algoritmo e salvamos toda a informação em um arquivo CSV. A análise de ordenação foi feita com base na coluna de votos, chamada dentro do arquivo de 'vote_count'. Definimos dois ambientes de execução para os testes de tempo, o terminal e a IDE. O teste de cada algoritmo será apresentado em forma de tabela, apresentando o tempo de execução em cada ambiente e em cada computador.

Hardware

Para a análise dos algoritmos, utilizamos 3 computadores diferentes, os quais apelidaremos conforme o nome do usuário:

Gustavo
- Processador Intel Core i7-3770 @ 3.40GHz
- 2x8GB RAM 1600MHz
- Placa de Vídeo NVIDIA GTX 1050TI
- SO: Windows 10

Jairon
- Processador Ryzen 5 3600x @ 3.60GHz
- 2x8GB RAM 3333MHz
- Placa de Vídeo NVIDIA GTX 1650 Super
- SO: Windows 10

Henrique
- Processador Ryzen 5 5600g @ 3.90 GHz
- 2x8GB RAM 3200MHz
- SO: Windows 10

Quick Sort

- Complexidade

No meio da estruturação de dados com o intuito de ordenar alguns tipos de dataframes de formas específicas e com o intuito de facilitar a vida do programador, onde uma delas é o QuickSort que possui o objetivo de otimizar a ordenação da lista.

O objetivo é dividir o array em subarrays menores de forma que todos os elementos menores que o pivô estejam à sua esquerda, e todos os elementos maiores estejam à sua direita. O algoritmo começa escolhendo um pivô e particionando o array em dois subarrays menores: um subarray contendo todos os elementos menores que o pivô, e outro subarray contendo todos os elementos maiores. Em seguida, o algoritmo é recursivamente aplicado a cada subarray menor. Isso é feito até que todos os subarrays tenham tamanho 1, o que significa que o array está ordenado.

Depois de realizar o particionamento, os dois subarrays menores podem ser recursivamente classificados. A complexidade do algoritmo Quicksort depende do método de escolha do pivô. No pior caso, o pivô é o maior ou o menor elemento do array, e cada partição contém apenas um elemento. Nesse caso, a complexidade do Quicksort é O(n^2). No entanto, em média, a complexidade do Quicksort é O(n*log n), o que o torna um dos algoritmos de ordenação mais rápidos.

- Tempo de Execução

	IDE	Terminal
Gustavo	0.96 s	0.95 s
Henrique	0.32 s	0.31 s
Jairon	0.23 s	0.27 s

Radix Sort

- Complexidade

Radix Sort é um algoritmo não comparativo que ordena os elementos de um array de acordo com seus dígitos, de forma que os elementos com os mesmos dígitos são agrupados. O algoritmo é executado para cada dígito em ordem crescente, começando pelo dígito menos significativo. O algoritmo também usa de subrotina outro algoritmo chamado Counting Sort. Para analisar a complexidade total, vamos analisar separadamente cada algoritmo, começando com o Counting Sort.

Dado n como o número de elementos a serem ordenados e k como o número máximo de valores possíveis, o algoritmo Counting Sort tem uma complexidade de O(3n+k). Simplificando e observando que, neste caso, k é igual a 10, obtemos O(n).

Já analisando o algoritmo Radix Sort, se temos d como o número de dígitos do valor máximo do campo “vote_count”, e tendo em vista que o algoritmo usa o Counting Sort, que neste caso tem complexidade de O(n), podemos concluir que a complexidade deste algoritmo é de O(d*n)

- Tempo de execução

	IDE	Terminal
Gustavo	0.07 s	0.06 s
Henrique	0.54 s	0.21 s
Jairon	4.32 s	0.92 s

Bucket Sort

- Complexidade

O Bucket Sort funciona dividindo o intervalo dos dados em um número finito de “baldes”, onde cada balde contém uma quantidade específica de elementos. Então os elementos são distribuídos nos baldes de acordo com o seu valor e, logo depois, cada balde é classificado individualmente ou os elementos podem ser distribuídos em baldes menores até que cada balde contenha apenas um elemento.

Depois que todos os baldes são classificados, os elementos são concatenados na ordem dos baldes, criando o final da lista ordenada.

A complexidade do Bucket Sort será em média $O(n + k)$, onde n é o número de elementos e k é o número de baldes. Isso ocorre porque a distribuição dos elementos em baldes de tamanhos uniformes garante que cada balde tenha, em média, aproximadamente o mesmo número de elementos, o que leva a uma redução significativa no tempo de ordenação.

No pior dos casos, a complexidade do Bucket Sort seria $O(n^2)$, onde n é o número de elementos a serem ordenados. Isso ocorre quando todos os elementos são inseridos em um único balde.

- Tempo de execução

	IDE	Terminal
Gustavo	1.86 s	1.81 s
Henrique	1.13 s	0.95 s
Jairon	0.82 s	0.57 s

Análise Geral

Analisando o tempo de execução de cada algoritmo, podemos perceber que o mais eficiente para ordenar o dataset foi o Radix Sort, seguido do Quick Sort e então, o Bucket sort, sendo o menos eficiente dentre os três.

Observando o tempo de execução na IDE (onde optamos por usar o VSCode), podemos ver também uma disparidade no tempo de execução do Radix Sort no computador do Jairon, sendo quase 5 vezes maior do que quando executado no terminal, e mais de 60 vezes maior do que quando executado na IDE no computador do Gustavo. Isso demonstra a importância da realização de testes em diferentes ambientes e diferentes hardwares para avaliar a eficiência dos algoritmos.