Laboratório de Estrutura de Dados

Projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Arthur Lincoln da Paz Cristovão, Gustavo Genésio Rodrigues e Philipe de Oliveira Tavares

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA que tem como objetivo estudar o uso dos algoritmos de ordenação, assim como a análise geral dos resultados que serão obtidos com a utilização desses algoritmos. Para isso, foi disponibilizado um dataset que contém informações dos jogos das principais ligas europeias de futebol, cuja aplicação dos algoritmos de ordenação deverá ser feita a partir dos campos de informações do arquivo disponibilizado pelo dataset.

Desse modo, deverá ser feito algumas transformações no arquivo (matches.csv), gerando outros a partir do original. Assim, o arquivo original será filtrado, permanecendo os campos especificados na página do projeto da disciplina.

Após filtrar o arquivo, gerando um novo chamado matches_T1.csv, esse deve ser utilizado para obter outro com um novo campo (full_date) chamado matches_T2.csv. Em seguida, o matches_T2.csv deverá ser filtrado, para obtenção de um novo arquivo matches_F1.csv, com os jogos, apenas, da liga de futebol inglês (Premier League). E como última transformação no arquivo antes da aplicação dos algoritmos de ordenação, será criado um matches_F2.csv filtrando os jogos com o público acima de 20.000 (vinte mil) pagantes.

Ao fim, depois de gerar outros arquivos através da realização de transformações, será feita a aplicação dos algoritmos de ordenação.

Na ordenação das informações do arquivo, o algoritmo com pior resultado em questão de tempo foi o bubble sort para organização por strings, com um tempo de execução de 141,3 segundos (2,21 minutos) para ordenar todo o arquivo. E, o algoritmo que apresentou o menor tempo de execução foi o heap sort com uma média de 16,6 milissegundos nas ordenações de todos os casos que envolviam inteiros.

2. Descrição geral sobre o método utilizado

Os testes foram realizados no ambiente de desenvolvimento integrado Visual Studio Code, com os códigos feitos na linguagem de programação Java, com JDK (Kit de desenvolvimento Java) na versão 17.3.1.

Foi feito através de programação orientada a objetos, com o uso de classes para o controle do código que organiza todos os arquivos requisitados pelo projeto.

O package contém 5 classes para o funcionamento do código:

- FormataArquivo.java (corrige as vírgulas e conta as linhas do arquivo em csv);
- OrdenacaoInt.java (algoritmos de ordenação para inteiros);
- OrdenacaoStr.java (algoritmos de ordenação para strings);
- PausaCodigo.java (feito para que os arquivos pedidos não sejam criados de uma só vez);
- ProjetoEDA.java (Classe onde o código irá ser executado).

Descrição geral do ambiente de testes

Os testes foram feitos numa máquina com processador AMD Ryzen 3 3200U com placa integrada de 2.6GHz, memória RAM instalada de 4GB, com o sistema operacional Windows 10, tipo de arquitetura operacional de 64 bits e com armazenamento HD de 1TB.

Na IDE utilizada, foi criada uma pasta para a produção dos códigos e para a produção do projeto, foi criada outra pasta “ProjetoGAP”, onde esse título vai servir como o nome do pacote que irá conter as classes do código.

3. Resultados e Análise

Tabela de tempo (em milissegundos) da ordenação pelo público pagante (attendance):

Algoritmos	Pior Caso	Caso Médio	Melhor Caso
Bubble Sort	2236	2948	0
Insertion Sort	1354	649	1
Selection Sort	844	620	588
Quick Sort	420	52	716
Quick Sort (Med 3)	204	82	97
Merge Sort	54	74	70
Heap Sort	11	20	23
Couting Sort	39	6	15

Tabela de tempo (em milissegundos) da ordenação pela data:

Algoritmos	Pior Caso	Caso Médio	Melhor Caso
Bubble Sort	2227	1943	1
Insertion Sort	1333	572	0
Selection Sort	607	407	479
Quick Sort	564	6	236
Quick Sort (Med 3)	16	4	5
Merge Sort	27	32	42
Heap Sort	10	20	16
Couting Sort	61	44	70

Tabela de tempo (em milissegundos) da ordenação pelo local (venue):

Algoritmos	Pior Caso	Caso Médio	Melhor Caso
Bubble Sort	122636	125903	141213
Insertion Sort	91793	28934	15
Selection Sort	48348	36055	35591
Quick Sort	6749	861	261
Quick Sort (Med 3)	444	365	325
Merge Sort	86	95	50
Heap Sort	33	162	100

Para as comparações dos algoritmos de ordenação por meio de strings foi usado o método compareToIgnoreCase, o que agravou o gasto de memória e consequentemente o tempo de execução, mas nada muito fora do comum, a ordenação mais longa nesse caso ocorreu na ordenação do melhor caso do bubble sort, que levou cerca de dois minutos e vinte segundos para ocorrer, e esse método só foi usado na ordenação pelo campo 'venue', não foi necessário usar na ordenação da data pois foi feito um tratamento para que as ordenações por inteiro também funcionassem para o caso das datas, o que ocorreu foi que o campo 'full_date' que estava no formato DD/MM/AAAA fosse invertido para AAAAMMDD (sem as barras), e depois, com o método Integer.parseInt, é convertido esse valor para inteiro e depois podemos ordenar normalmente sem a necessidade de chamadas de funções externas nas comparações. É passado para as funções de ordenação uma matriz de inteiros com linhas de duas posições, em uma posição das linhas está o conteúdo AAAAMMDD a ser ordenado e na outra posição está o índice da linha, as comparações serão feitas considerando o índice 0 da linha, ou seja o AAAAMMDD, mas as trocas serão

de toda linha da matriz, o índice 1 da linha será importante na hora da gravação dos arquivos, é por ele que saberemos em que momento gravar a linha no arquivo ordenado.

3.1. Algoritmos mais eficientes

O Heap, o Merge, o Quick Sort com mediana de três e o counting sort foram os algoritmos que demonstraram melhor desempenho, pois têm complexidade linear e apresentaram ótimos resultados até mesmo para o pior caso, sendo o Merge Sort o algoritmo que teve a maior consistência nos resultados, com poucas variações entre médio, melhor e pior caso para todas as colunas a serem ordenadas.

3.2. Análise geral sobre os resultados

Os algoritmos de ordenação Bubble, Insertion e Selection, como já era esperado, foram os que tiveram os piores desempenhos, com tempos que ultrapassam a casa dos minutos mesmo com uma base de dados tão pequena, são algoritmos eficientes mas que podem ter seu uso descartado para base de dados grandes, pelo fato de sua complexidade quadrática, isso não é interessante para nossa análise de qual algoritmo teve os melhores resultados, seu uso fica restrito a base de dados curtas onde a variação de tempo no uso dos algoritmos lineares é quase irrisória, pois, apesar de sua maior complexidade, a sua implementação é muito mais simples, com funções curtas de serem escritas.

o Heap Sort teve desempenho melhor que o Merge nas ordenações por inteiro, entretanto, nas ordenações por String ele demonstrou uma variação muito grande entre o médio e os demais casos, o que prejudicou sua avaliação final e o que poderia ser um empecilho caso a base de dados fosse muito maior e não fosse possível para nós prever a qual nível de organização se encontra essa base de dados, ele é muito bom para base de dados grandes, mas mostra um melhor desempenho em relação aos demais algoritmos lineares quando já temos uma breve organização dos dados, ou seja, quando se trata de uma base de dados semi-ordenada.

Outro algoritmo linear que demonstrou bons resultados foi o Quick Sort, mas nem perto de desempenhos tão bons quanto o Merge e o Heap, um ponto positivo a se destacar é a pouca variação entre o melhor e o médio caso, principalmente nas ordenações por String, e quando levamos o consideração o método da mediana de 3 para escolha do pivô, ele tem um ganho de desempenho enorme em relação ao pior caso principalmente, com poucas variações para a ordenação por string, mesmo com o agravante do uso de funções externas que consomem mais a memória da máquina, por isso é mais perceptível sua melhor eficiência em relação ao Quick Sort normal quando olhamos para o pior caso, com ganho de tempo quase dez vezes superior em média entre todas as ordenações.