Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Identificação do aluno:
Anna Caroline Barreto Queiroz
Matrícula: 202080226

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA que tem como objetivo utilizar dados de projetos voltados para data science para a implementação e análise do desempenho dos algoritmos de ordenação Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Quicksort com Mediana de 3, counting, e HeapSort. Para esta finalidade, foi disposto um dataSet intitulado como passwords.csv contendo quatro colunas de dados, uma para o ID, outra para o comprimento da senha, a senha em si e a data em que foram criadas, como também os campos que deveriam ser utilizados para as ordenações.

O dataSet citado anteriormente necessita passar por transformações além das ordenações.

- Transformar data para o formato DD/MM/AAAA:

  A coluna que abriga o campo de datas não possui um formato padrão, para isso é necessário tratar as variações que o campo possui e transformá-las no formato especificado acima. Feito o tratamento deve-se gerar um novo arquivo com o novo formato do campo de datas, este intitulado como passwords_formated_data.csv.

- Filtrar senha pela categoria Boa e Muito Boa:

  As senhas do arquivo possuem características que nos permitem classificá-las. Na categoria de senhas boas elas necessitam ter um tamanho menor ou igual a 7 e ao mesmo tempo precisam ser compostas por todos os tipos de caracteres, por exemplo: letras (letras maiúsculas e minúsculas), número e caractere especial. Já as senhas muito boas precisam ter um tamanho maior ou igual a 8 e serem compostas por todos os tipos de caracteres, por exemplo: letras (letras maiúsculas e minúsculas), número e caractere especial.

  Assim que classificadas, torna-se necessário gerar um arquivo chamado passwords_classifier.csv com a coluna (class) com cada classe para a respectiva senha como no exemplo abaixo (Figura 1).

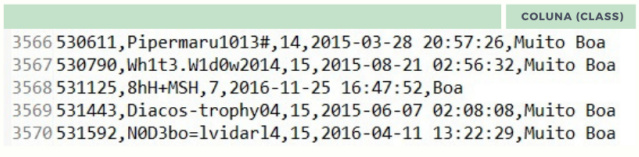

Figura 1

- Ordenações:

  As ordenações do arquivo devem considerar três campos para os sete algoritmos solicitados, o campo length (em ordem decrescente), por mês (de forma crescente) e por data completa (de forma crescente). Deve-se gerar um arquivo diferente para cada algoritmo de ordenação e casos (melhor caso, médio caso e pior caso), respeitando os nomes que foram designados para cada arquivo.

Os principais resultados obtidos com relação às transformações exigidas, foi o retorno dos arquivos na formatação esperada, tanto para a formatação das datas quanto para a classificação das senhas “boas” e "muito boas”.

Com relação aos algoritmos de ordenação, obteve-se o retorno dos arquivos ordenados de acordo com cada especificação e para cada caso de execução. Dois algoritmos apresentaram “StackOverflowException”, um estouro de pilha, sendo eles o quickSort com mediana de 3, realizando a comparação com relação ao comprimento das senhas e o quickSort, tanto por comprimento quanto por mês e ano. Isso acontece porque o quickSort necessita de um pivot e tem uma partição que não é adequada para esse tipo de dado, os campos recebidos para ordenação geram muitos valores repetidos fazendo com o que sejam feitas muitas chamadas recursivas resultando assim no estouro de pilha. Para contornar a situação foi implementado o “quicksort tree way”, que possui o método de partição mais adequado e que nos retornou a ordenação na forma esperada, assim gerando o arquivo ordenado pelo campo especificado.

2. Descrição geral sobre o método utilizado

Os testes foram realizados na IDE Eclipse e a linguagem utilizada para a implementação dos algoritmos de ordenação e transformações solicitadas foi a linguagem de programação Java. Com a finalidade de deixar as classes do projeto o mais organizadas

possível e obter resultados no projeto como um todo, utilizou-se os quatro pilares da orientação a objetos (abstração, encapsulamento, herança e polimorfismo).

Além da orientação a objeto, um dos métodos mais essenciais para obtermos resultados foi o uso da interface *Comparator*, que nos permitiu criar regras de comparação para as datas e comprimento.

Em decorrência do estouro de pilha citado no tópico 1, na ordenação do quickSort e quickSort mediana de 3, foi implementado uma versão alternativa e otimizada do quickSort, que é “quicksort tree way” que consiste em dividir os dados em três partes, os valores menores que o pivot à esquerda, os valores iguais ao pivot no meio e os valores maiores a esquerda, ao final do particionamento basta ordenar os valores menores e maiores que o pivot, assim obtivemos o resultado esperado e o arquivo ordenado como solicitado.

Para facilitar que o usuário solicitasse as ordenações de forma mais organizada e acessível foi implementado um menu, onde pode-se escolher com qual algoritmo ordenar o arquivo, por qual parâmetro (comprimento, mês/ano ou dd/mm/aaaa) e em qual caso (melhor caso, caso médio e pior caso). Com a mesma finalidade, os caminhos onde os arquivos forem gerados ficarão disponíveis na tela para o usuário.

Descrição geral do ambiente de testes

Os testes foram realizados em um DESKTOP-TRK SG2 com processador intel (R) core (TM) i7-7500U com 2.70GHz (4 CPUs) e memória de 8192MB RAM e sistema operacional Windows 10 single language 64 bits. A testagem também foi realizada no sistema operacional Linux, Umbuto.04LTS.

3. Resultados e Análise

3.1. Tabelas de comparação

Tabela de tempo (em milissegundos) da ordenação por data (DD/MM/AAAA)

Algoritmo de ordenação	Melhor caso	Médio caso	Pior caso
CoutingSort	2345	2907	4071
HeapSort	375	953	484
InsertionSort	6805	9602557	5348647
MergeSort	1027444	1060890	993479
QuickSort	11068	10075	12222
QuickSortMedianOfThree	2066	2889	2037
SelectionSort	4023898	5480821	2717290
ThreeWayQuickSort	281	522	221

Tabela de tempo (em milissegundos) da ordenação por mês e ano

Algoritmo de ordenação	Melhor caso	Médio caso	Pior caso
CoutingSort	4061	5524	7260
HeapSort	618	703	713
InsertionSort	19	8214797	4657922
MergeSort	973297	1067971	1254646
QuickSort	StackOverflowException	StackOverflowException	StackOverflowException
QuickSortMedianOfThree	2447	109762	69618
SelectionSort	3897070	2560833	2364797
ThreeWayQuickSort	141	248	159

Tabela de tempo (em milissegundos) da ordenação por comprimento de password

Algoritmo de ordenação	Melhor caso	Médio caso	Pior caso
CoutingSort	218	47	89
HeapSort	781	354	1757
InsertionSort	23	4034964	6010589
MergeSort	1054380	1174722	998271
QuickSort	StackOverflowException	StackOverflowException	StackOverflowException
QuickSortMedianOfThree	StackOverflowException	StackOverflowException	StackOverflowException
SelectionSort	35647462	2153120	2118709
ThreeWayQuickSort	353	85	74

3.2. Algoritmos mais eficientes

Os algoritmos com melhor desempenho foram CoutingSort, HeapSort e ThreeWayQuickSort com os melhores tempos conforme a análise do tempo de complexidade.

3.3. Análise geral sobre os resultados

Os algoritmos de ordenação com os piores desempenhos foram InsertionSort, SelectionSort e MergeSort, sendo o MergeSort o menos pior. O QuickSort é um algoritmo que não funciona bem quando se tem muitos valores repetidos, ele teve um bom desempenho na ordenação por DD/MM/AAAA, porém quando retiramos o campo de “dia” e deixamos apenas MM/AAAA, a quantidade de datas iguais tem um grande aumento, assim dificultando seu funcionamento e gerando um estouro de pilha. Já o QuickSort mediana de 3 consegue realizar ordenação pelo campo de mês e ano. Pelo mesmo motivo, ambos geram estouro de pilha quando ordenados pelo comprimento de senha, pois uma quantidade considerável de senhas possuem o mesmo tamanho.

O insertionSort teve um ótimo desempenho no melhor caso, levando em conta que este recebe como entrada um arquivo já ordenado, a inserção ordenada de cada elemento tem custo 0(1), então todos já estão em suas posições corretas, assim, como a ordenação é executada *n* vezes, o custo total é O(n). Vale descartar que ele não é um algoritmo muito eficiente para grandes entradas, que é o caso do arquivo aqui trabalhado, os mais indicados dentre as nossas opções de algoritmos de ordenação, seriam os de ordenação linear ou generalista, CoutingSort e HeapSort.

Como citado no parágrafo acima, os algoritmos mais indicados para o nosso arquivo seriam CoutingSort e HeapSort, que foram justamente os que apresentaram os melhores tempos de execução em melhor, médio e pior caso, tendo variações dessa posição quando ordenados por parâmetros diferentes. Por exemplo, o Heap tem melhor desempenho que o Couting quando ordenado por data (DD/MM/AAAA), já o Couting apresenta um melhor desempenho que Heap quando ordenado por mês e ano (MM/AAA), ambos nos três casos (melhor, médio e pior). Pelo comprimento o CoutingSort se sai muito melhor que Heap nos três casos.

Destarte, concluímos que através dessa ampla aplicação dos algoritmos de ordenação Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Quicksort com Mediana de 3, counting, e HeapSort em um arquivo com quase 700 mil linhas, o algoritmo que apresentou melhor desempenho em ordená-lo foi CoutingSort.