UNIVERSIDADE ESTADUAL DA PARAÍBA

CENTRO DE CIÊNCIA E TECNOLOGIA- CCT

CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: LABORATÓRIO DE ESTRUTURAS DE DADOS

PROFESSOR: FÁBIO LUIZ LEITE JÚNIOR

ALISSOM DA SILVA RODRIGUES

DJHONATAH WESLEY C. ALVES

MATEUS RUFINO FERREIRA

PROJETO 1: TEMA 2 - PASSWORDS

ANÁLISE COMPARATIVA DE ALGORITMOS DE ORDENAÇÃO

CAMPINA GRANDE

2024

SUMÁRIO

1. INTRODUÇÃO

Este relatório tem como objetivo demonstrar os resultados obtidos na análise comparativa de diferentes algoritmos de ordenação aplicados a um dataset com mais de 600 mil senhas, consideradas as mais comuns utilizadas. Os algoritmos selecionados para este estudo incluem Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Counting Sort, Heapsort e Quick Sort com Mediana 3. Todos os algoritmos escolhidos foram vistos em sala de aula, sendo falado pelo professor sobre sua eficiência e usabilidade.

Nosso objetivo para o que vem a seguir trata-se de explicar como foi feito as classificações, transformações, filtragens e ordenações de um conjunto de dados contendo milhares de senhas. Além disso, falaremos um pouco sobre os algoritmos utilizados e mostraremos análises feitas e deduzidas a partir da ordenação desses dados. Também citaremos sobre o ambiente de execução e como o projeto foi organizado.

2. ALGORITMOS

Seguimos o roteiro apresentado pelo professor e nele precisamos implementar e utilizar todos os algoritmos de ordenação estudados e por meio deles ordenar os registros de acordo com os atributos pelos quais os dados precisam ser ordenados. agora vamos citar os algoritmos usados e uma breve explicação de como funcionam e como implementamos.

- Selection Sort: Este é um algoritmo simples que divide a lista em duas partes: a parte ordenada e a parte não ordenada. O algoritmo funciona encontrando o menor (ou maior, dependendo da ordem de classificação) elemento na parte não ordenada e trocando-o com o elemento mais à esquerda não ordenado. Este algoritmo é mais eficaz para listas pequenas e para listas onde a troca de elementos é cara.
- Insertion Sort: Este algoritmo divide a lista em duas partes: a parte ordenada à esquerda e a parte não ordenada à direita. O algoritmo pega cada elemento da parte não ordenada e insere na posição correta na parte ordenada. Este algoritmo é eficiente para listas quase ordenadas e listas pequenas.
- Quick Sort: Este é um algoritmo de ordenação eficiente que funciona dividindo a lista em duas partes com base em um elemento pivô. Em seguida, o algoritmo ordena as duas partes de forma recursiva. Este algoritmo é eficaz para listas grandes e tem uma boa média de tempo de execução.
- Merge Sort: Este é um algoritmo de ordenação eficiente que funciona dividindo a lista em duas metades, ordenando as duas metades de forma independente e, em seguida, mesclando as duas metades ordenadas. Este algoritmo é eficaz para listas grandes e garante um tempo de execução estável.
- Counting Sort: Este é um algoritmo de ordenação não comparativo que funciona contando o número de objetos que possuem valores distintos de chave. Este algoritmo é eficaz para listas onde o intervalo de valores possíveis é limitado.
- Heapsort: Este é um algoritmo de ordenação comparativo que funciona transformando a lista em uma estrutura de dados chamada heap. O processo é repetido até que o heap esteja vazio. Este algoritmo é eficaz para listas grandes e garante um tempo de execução estável.
- Quick Sort com Mediana 3: Esta é uma variação do Quick Sort que funciona escolhendo o pivô como a mediana de três elementos da lista. Isso pode melhorar o desempenho do Quick Sort para certos tipos de listas, especialmente aquelas que já estão parcialmente ordenadas.

Cada um desses algoritmos foi aplicado a três arquivos .csv(pior caso, medio caso e melhor caso), e o tempo de ordenação é registrado no momento em que ela é realizada.

3. CASOS DE TESTE

Seguindo o passo a passo do projeto, temos as regras para classificar as senhas geração de casos de testes a partir do arquivo original "passwords.csv". nós

A partir do arquivo "password_classifier.csv", foram gerados outros com a mesma categoria, mas com uma alteração específica: a formatação da coluna de data para o formato "dd/mm/aaaa". O arquivo resultante desse processo foi denominado "passwords_formated_data.csv".

Posteriormente, com o objetivo de agilizar o processo de análise, foi construído um novo arquivo, o "passwords_test.csv", a partir do "passwords_formated_data.csv". Esse novo arquivo é significativamente menor, permitindo a obtenção de resultados em um tempo hábil e será o utilizado nas ordenações.

Com o arquivo "passwords_test.csv", foram criados arrays representando o melhor, o pior e o caso médio para cada tipo de ordenação. Essa segmentação foi orientada pelo professor, considerando diferentes critérios. Para as ordenações baseadas no mês e na data, o melhor caso foi definido como os dados em ordem crescente, o caso médio como os dados sem formatação e o pior caso como os dados em ordem decrescente. Para a análise baseada no tamanho da senha, seguiu-se o mesmo critério, no entanto, a ordenação seria em ordem decrescente, ou seja, para pior e melhor caso foi considerado o inverso dos demais.

4. AMBIENTE DE EXECUÇÃO

O projeto foi desenvolvido utilizando a IDE IntelliJ IDEA, por ja termos trabalhado com ela e por ser um conforto pessoal nosso mas oque produzimos poderia ser facilmente implementado por outro ambiente.

O projeto foi estruturado em várias classes, cada uma com funções específicas, a fim de garantir uma organização eficiente e evitar repetição de código. Para consumir os dados do arquivo "passwords.csv" e retornar novos arquivos ".csv" com as mudanças exigidas, utilizamos a classe GetVariables que também transforma um arquivo ".csv" em um array, o que é também bastante utilizado. Mais três classes foram desenvolvidas para retornar os arrays formatados para os casos de melhor, pior e médio, a partir do arquivo especificado. Para o projeto, o arquivo "passwords_test.csv" foi utilizado na construção.

Nessas classes, em contraste com o restante do código, foram usados os métodos sort e reversed, bem como o auxiliar Comparator, que fazem parte das bibliotecas do Java. Foi criada uma classe para cada algoritmo de ordenação, cada uma com métodos que realizam as ordenações baseadas no tamanho da senha, mês e data. Esses métodos podem ser facilmente chamados em qualquer parte do código, gerando as saídas e criando os arquivos ".csv" para cada ordenação. CreateFileCsv: Responsável por consumir um array e criar um arquivo ".csv" a partir dele. Cada índice nos arquivos csv representa uma string e é feita uma seleção para pegar o valor correto, sendo de grande utilizadas para os algoritmos de ordenação. RunTests: Esta classe cria instâncias para cada classe dos algoritmos de ordenação e os chama de acordo com sua categoria, seja para o tamanho da senha, mês ou data. A classe Main é onde se executa o código e lá há 3 chamadas gerais a de ordenação pelo tamanho da senha, pelo mês e pela data e engloba todos os métodos e classes criadas, proporcionando uma execução centralizada e organizada do projeto.

5. ANÁLISE COMPARATIVA DOS ALGORITMOS

Na seção a seguir, apresentamos as tabelas de resultados para cada um dos sete algoritmos de ordenação que analisamos: Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Counting Sort, Heap Sort e Quick Sort com Mediana 3. Cada tabela mostra o tempo de execução do algoritmo correspondente aos diferentes casos de teste. As tabelas permitem uma análise comparativa direta do desempenho dos algoritmos. Ao examinar os resultados, podemos observar como o tempo de execução de cada algoritmo varia com os casos de testes propostos pelo orientador.

Tabela 1 - Dados da execução de um dos nossos testes (todos os testes foram feitos na mesma execução no mesmo momento e na mesma máquina), desta vez levando em conta o tamanho da senha

Execução por tamanho da senha			
Algoritmo de Ordenação	Melhor Caso	Médio Caso	Pior Caso
Selection sort	1596ms	1810ms	1518ms
Insertion Sort	1ms	673ms	878ms
Quick Sort	877ms	137ms	355ms
Merge Sort	5ms	13ms	10ms
Counting Sort	1ms	1ms	1ms
HeapSort	12ms	13ms	7ms
Quick Sort com Mediana 3	6ms	9ms	11ms

Os resultados obtidos a partir da ordenação pelo tamanho da senha mostram que o algoritmo CountingSort é o mais eficiente, com um tempo de execução de 1 milissegundos em todos os casos. Em seguida, o algoritmo Heap Sort, Merge Sort e Quick Sort com Mediana 3 que demonstraram resultados muitos parecidos com o Heap Sort e o Merge Sort obtendo o tempo no médio caso de 13 milissegundos e o Quick Sort com Mediana 3 9 milissegundos, no pior caso o Heap Sort sai na frente com 7 milissegundos contra 10 milissegundos do Merge Sort e 11 milissegundos do Quick Sort com Mediana 3, já no melhor caso o Merge sort é mais rápido, com 5 milissegundos contra 12 milissegundos do Heap Sort e 6 milissegundos do Quick Sort com Mediana 3,.

Um destaque para o Insertion Sort que, apesar de não ter ido bem no médio e pior caso em comparação aos melhores resultados, em contrapartida, no melhor caso ele obteve o tempo de 1 milissegundo. Falando em geral do Insertion Sort Selection Sort e Quick Sort, eles ficaram absurdamente longe do resultado dos outros, principalmente o Selection sort que em todos os testes teve resultados acima dos 1500 milissegundos.

Tabela 2 - Dados da execução de um dos nossos testes (todos os testes foram feitos na mesma execução no mesmo momento e na mesma máquina), desta vez levando em conta o mês

Execução por mês			
Algoritmo de Ordenação	Melhor Caso	Médio Caso	Pior Caso
Selection sort	82703ms	81462ms	83191ms
Insertion Sort	33ms	44330ms	93828ms
Quick Sort	101190ms	7864ms	22704ms
Merge Sort	241ms	388ms	283ms
Counting Sort	66ms	52ms	52ms
HeapSort	758ms	678ms	654ms
Quick Sort com Mediana 3	685ms	675ms	687ms

Ao analisar os resultados com base no mês, logo se percebe que os tempos de maneira geral em todos os algoritmos de ordenação foram muito altos em relação aos resultados que obtemos com base no tamanho da senha. O algoritmo Counting Sort é o com melhor desempenho no médio caso e no pior caso com 52 milissegundos nos dois casos, perdendo somente para o Insertion Sort que tem o melhor tempo no melhor caso. Analisando ja por cima somente com as duas primeiras tabelas conseguimos ver que Counting Sort tem uma grande vantagem em cima dos outros e uma constância significativa até agora, se fossemos optar por um modelo real que necessitasse de ordenação até o momento o Counting Sort Seria considerado o mais ideal dentre esses algoritmos. podemos dizer nesta tabela que possui o pior desempenho, apesar dele ter 2 tempos no medio e pior caso onde ele não é o tempo de execução mais longo, ele possui uma variabilidade muito grande, o que numa situação real deixar a resolução de uma ordenação inconstante pode gerar custos imprevisíveis e às vezes desnecessários de tempo e de memoria.

Tabela 3 - Dados da execução de um dos nossos testes (todos os testes foram feitos na mesma execução no mesmo momento e na mesma máquina), desta vez levando em conta a data.

Execução por data			
Algoritmo de Ordenação	Melhor Caso	Médio Caso	Pior Caso
Selection sort	86161ms	80080ms	82745ms
Insertion Sort	56ms	40683ms	84252ms
Quick Sort	83368ms	551ms	40682ms
Merge Sort	232ms	540ms	954ms
Counting Sort	108ms	279ms	87ms
HeapSort	931ms	758ms	1002ms
Quick Sort com Mediana 3	615ms	670ms	602ms

Quando ordenado pela data completa, o algoritmo Counting Sort se destaca mais uma vez como o melhor, mas agora com uma discrepância bem menor em relaçõa aos outros algoritmos, na realidade seu tempo no melhor caso é apenas 2 vezes menor que o tempo do Merge Sort. O Merge Sort demonstra um desempenho estável, com tempos de execução de 232 milissegundos no melhor caso 540 milissegundos no medio caso e 954 milissegundos no pior caso. O Heapsort demonstra uma constância aceitável apesar do tempo ser maior do que o Counting Sort e o Merge Sort.

A melhor constância nesse teste com certeza foi do Quick Sort com mediana 3, que se fizermos a média entre os seus resultados foi de 629 milissegundos e seus dados chegam muito perto desse valor, o Selection Sort também possui uma variedade muito baixa nos dados, mas como seus dados possuem um tempo muito maior ele acaba não sendo tão bom. Agora tanto o Insertion Sort Selection Sort e Quick Sort, tiveram resultados acima dos 40 segundos

6.CONCLUSÃO

Concluímos que, ao analisar o desempenho dos algoritmos, observamos que, em termos de tempo de execução, o Counting Sort se destacou como a melhor opção em todos os cenários. O MergeSort também apresentou resultados consistentemente bons dependendo do caso e do que e onde vai ser implementado é um ótimo algoritmo levando em conta nossos resultados, em todos os testes ele ficou em 2º .

A produção dessa estrutura foi bastante interessante pois pudemos ver exatamente como alguns algoritmos se comportam dependendo da situação em que ele está, e vimos que quando a base de dados é muito grande qualquer mudança sutil que haja com a máquina ou a implementação gera resultados com diferenças que não consegue ver em base de dados pequenas que é oque normalmente usamos.