Laboratório de estrutura de Dados

Segunda versão do projeto da disciplina Comparação entre os algoritmos de ordenação elementar

Álvaro Dias, Helânio Renê e Luiz José

1. Introdução

Esse relatório corresponde as práticas testadas anteriormente na disciplina de Estrutura de dados , onde avaliamos as estruturas de dados e aplicamos 3 delas usando os métodos Sort.

Portanto usaremos dessas diferentes estruturas de dados , para demonstrar os conhecimentos acerca de estrutura de dados e do uso de métodos sort para organização de senhas.

Visão geral do projeto:

- **Introdução:** Introdução breve sobre o que foi trabalhado no projeto.

- **Descrição geral sobre o método utilizado:** Parte descritiva com base no objetivo geral, arquivos que foram utilizados, métodos do código, tabelas comparando os tempos de execução e informações acerca do ambiente de testes.

- **Resultado e análise:** Resultados e informações obtidas a partir das implementações que foram feitas no projeto.

2. Descrição geral sobre o método utilizado

Inicialmente, as planilhas com os dados foram baixadas e foram realizadas as classificações das senhas contidas no arquivo “passwords.csv” gerando um novo arquivo chamado “passwords_classifier.csv”, contendo uma nova coluna “class”, com as respectivas classificações de cada senha seguindo as regras especificadas no projeto.

Após a classificação, duas transformações foram realizadas: mudar a data para o formato

“DD/MM/AAAA” e filtrar as senhas pelas categorias ”Boa” e “Muito Boa”. Após cada transformação, dois arquivos resultantes foram gerados e nomeados como “passwords_formated_data.csv” e “passwors_classifier.csv”.

Em seguida foram implementados e utilizados todos os algoritmos de ordenação estudados na disciplina (Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Quick Sort, Counting, e Heap Sort), gerando um arquivo para cada tipo de ordenação dentro de três categorias:

Length - Ordenando as senhas pelo seu comprimento.

Month - Ordenando a tabela pelo Mês.

Date. - Ordenando pela Data inteira.

Com o objetivo de agilizar a execução dos algoritmos de ordenação e obtenção dos resultados, foi utilizado uma versão reduzida do arquivo "passwords_formated_data.csv", com um total de 10.000 senhas.

As escolhas dos tipos de Sort podem ser explicados de tal forma:

Ordenação por Contagem(Counting sort): A implementação utilizando uma Estrutura de Fila pode ser mais eficaz, uma vez que permite um acesso rápido à próxima posição de ordenação. No entanto, uma Estrutura de Lista Ligada também pode ser eficiente se for implementada adequadamente, mantendo um bom desempenho nas operações de inserção e remoção.

Ordenação por Intercalação(Merge sort): Utilizar uma Estrutura de Árvore Binária para a fusão dos elementos pode ser uma opção mais eficiente, uma vez que a estrutura da árvore permite a união eficiente de sub-listas. No entanto, uma Estrutura de Lista Ligada também pode ser utilizada com um bom desempenho, desde que seja implementada corretamente.

Ordenação por Seleção(Selection sort): O uso de uma Estrutura de Lista Ligada pode ser vantajoso para a Ordenação por Seleção, uma vez que as trocas de elementos são realizadas ajustando apenas os ponteiros da lista. No entanto, uma Estrutura de Árvore Binária ou uma Pilha também podem ser eficientes se forem implementadas corretamente.

Ordenação por Inserção(Insertion sort): Uma Estrutura de Lista Ligada pode ser uma escolha eficiente para a Ordenação por Inserção, já que a inserção de elementos é facilitada pelas operações de ligação entre os nós da lista. No entanto, uma Estrutura de Árvore Binária ou uma Pilha também podem ser utilizadas para obter um desempenho adequado.

Ordenação por Heap(Heap sort): O uso de uma Estrutura de Fila de Prioridade (Priority Queue) como estrutura de dados pode ser mais eficiente para a Ordenação por Heap, pois permite uma implementação direta do algoritmo, mantendo a propriedade do heap de forma eficiente.

Ordenação Rápida(Quick sort): Utilizar uma Pilha pode ser uma opção mais eficiente para a Ordenação Rápida, pois a pilha pode ser usada para armazenar os intervalos de partição à medida que o algoritmo avança. No entanto, uma Estrutura de Lista Ligada ou uma Estrutura de Árvore Binária também podem ser implementadas para obter um bom desempenho.

Quicksort com Mediana de 3: As mesmas considerações do Quicksort se aplicam aqui. Utilizar uma Pilha, uma Estrutura de Lista Ligada ou uma Estrutura de Árvore Binária pode ser eficiente para essa variação do algoritmo.

Os algoritmos de ordenação escolhidos nessa aplicação foram:

- **Insertion Sort:** utilizando uma lista encadeada;
- **Selection Sort:** utilizando uma árvore binária;
- **Quick Sort:** utilizando pilha.

Descrição da implementação da ferramenta (IDE) utilizada:

- A ferramenta utilizada no projeto foi o IntelliJ IDEA 2023.1.

Descrição geral do ambiente de testes:

- Processador - Intel(R) Core(™) i3-9100F CPU @ 3.60GHz
- Memória - RAM instalada 8,00GB
- Tipo de Sistema - Sistema operacional de 64 bits, processador baseado em x64
- Epecificações do Windows - Windows 10 Home versão 22H2

3. Resultados e Análise

Foi elaborada uma tabela comparando o tempo de execução dos algoritmos utilizados, para cada tipo de ordenação. Nas tabelas abaixo é possível ver os algoritmos e seus respectivos tempos de execução , tudo isso comparando com os tipos de caso(Médio caso, Melhor caso e Pior caso).

Length

-	**Médio Caso**	**Melhor Caso**	**Pior Caso**
SelectionSort	2,024s	1,191s	1,213s
InsertionSort	0,564s	0,135s	0,377s
QuickSort	0,346s	0,650s	0,242s

Month

-	**Médio Caso**	**Melhor Caso**	**Pior Caso**
SelectionSort	87,051s	85,077s	84,114s
InsertionSort	36,485s	0,140s	66,024s
QuickSort	7,053s	6,688s	6,581s

Date

-	**Médio Caso**	**Melhor Caso**	**Pior Caso**
SelectionSort	122,005s	113,068s	107,975s
InsertionSort	41,504s	0,125s	79,500s
QuickSort	0,763s	72,920s	74,405s

● Com base nos tempos de execução analisados, foi possível calcular as médias obtidas por cada algoritmo nos três casos:

Length

Algoritmos	Médias obtidas
Selection	1,071s
Insertion	0,358s
Quick	0,412s

● Com base nas médias de execução do Length e dos respectivos algoritmos, os mais eficientes estão a seguir, onde o primeiro representa o mais eficiente em relação ao tempo de execução e o

último o menos eficiente:

1. Insertion
2. Quick
3. Selection

Month

Algoritmos	Médias obtidas
Selection	85,414s
Insertion	34,216s
Quick	6,774s

● Com base nas médias de execução do Month e dos respectivos algoritmos, os mais eficientes estão a seguir, onde o primeiro representa o mais eficiente em relação ao tempo de execução e o último o menos eficiente:

1. Quick
2. Insertion
3. Selection

Date

Algoritmos	Médias obtidas
Selection	114,349s
Insertion	40,376s
Quick	49,362s

● Com base nas médias de execução do Date e dos respectivos algoritmos, os mais eficientes estão a seguir, onde o primeiro representa o mais eficiente em relação ao tempo de execução e o último

o menos eficiente:

1. Insertion
2. Quick
3. Selection

Atualização do Length , Month e Date

Length Atualizado

-	Médio Caso	Melhor Caso	Pior Caso
SelectionSort	1,178s	0,720s	0,844s
InsertionSort	1,141s	0,733s	0,335s
QuickSort	0,851s	0,619s	0,395s

Month Atualizado

-	Médio Caso	Melhor Caso	Pior Caso
SelectionSort	0,826s	0,422s	0,298s
InsertionSort	0,836s	0,370s	0,458s
QuickSort	1,266s	0,652s	0,354s

Date Atualizado

-	Médio Caso	Melhor Caso	Pior Caso
SelectionSort	0,853s	0,472s	0,313s
InsertionSort	0,853s	0,422s	0,309s
QuickSort	0,829s	0,517s	0,294s

Análise/Comentários acerca dos resultados obtidos:

Com base nos resultados obtidos pelo experimento, foi possível perceber que após a implementação da Lista Encadeada, da Pilha e da Árvore Binária, nos algoritmos de ordenação SelectionSort, InsertionSort e QuickSort houve uma melhoria bastante significativa nos seus tempos de execução em relação às suas versões anteriores, os quais demoravam muito em comparação com os outros algoritmos.