Laboratório de Estrutura de Dados

Relatório de Comparação entre os Algoritmos de Ordenação Elementar

Alunas:

Raquel Gomes de Vasconcelos da Silveira

Natália Maria de Araújo Lima

1. Introdução

Esse relatório corresponde a especificidades e detalhamento dos resultados obtidos no projeto da disciplina de Laboratório de Estrutura de Dados, que tem como fito implementar e analisar os algoritmos de ordenação, como Selection Sort, Insertion Sort, Merge Sort, Quicksort, QuickSort com Mediana de 3, Counting Sort, Bubble Sort, e HeapSort. Assim, utilizando de seus dados. Nesse projeto, foram analisados um conjunto de dados de arquivo csv, chamados de “passwords.csv”, que contém informações sobre as senhas, como tamanho, data e classificação.

Antes da análise dos algoritmos, foram realizadas algumas transformações nos arquivos.

- **Classificação de senhas:**

  O processo de classificação de senhas consistiu na análise dos dados presentes no arquivo “passwords.csv”, a partir da qual foi possível identificar e classificar cada senha de acordo com o tipo correspondente.Para essa classificação,foram considerados critérios como o tamanho da senha e a presença de diferentes tipos de caracteres.

  Muito Ruim: tamanho da string menor que 5, e só um tipo de caractere, por exemplo: só letra (letras minúsculas ou maiúsculas), só número ou só caractere especial.

  Ruim: tamanho da string menor ou igual a 5, e só um tipos de caracteres, por exemplo: letra (letras minúsculas ou maiúsculas), número ou caractere especial.

  Fraca: tamanho da string menor ou igual a 6, e só dois tipos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial.

  Boa: tamanho da string menor ou igual a 7, e só todos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial.

  Muito Boa: tamanho da string maior que 8, e só todos os tipos de caracteres, por exemplo: letra(letras minúsculas e maiúsculas), número ou caractere especial

  Sem Classificação: Senhas que não se qualificam com nenhuma das classificações acima.

  A partir dessas análises, foi criada uma nova coluna chamada “class” e os dados foram salvos em um novo arquivo chamado “*password_classifier.csv*”.

- **Transformar data:**

  A coluna que contém o campo de datas no arquivo “password_classifier.csv” apresenta variações em seu formato, sendo assim, necessário a mudança desse formato para realizar a análise dos dados.Dessa forma, foi efetuado um processo de tratamento que consistiu na identificação das diferentes variações de formato presente na coluna de datas, e na aplicação de métodos de manipulação nas datas para realizar a padronização das mesmas para um formato específico (DD/MM/AAAA).

- **Filtrar senha pela categoria Boa e Muito Boa:**

  Após a realização do processo de classificação de senhas,é preciso filtrar as senhas categorizadas como Boa e Muito Boa através da coluna “class” do arquivo “password_classifier.csv” e adicioná-las a um novo arquivo intitulado “passwords_classifier.csv”.

- **Ordenações:**

  Para as ordenações, são considerados sete algoritmos de ordenação, o Selection Sort, Insertion Sort, Merge Sort, Quicksort, Quicksort com Mediana de 3, Counting, e HeapSort. E o arquivo de entrada utilizado para ser ordenado é o “*passwords_formated_data.csv*”.

  Primeira ordenação: é uma ordenação decrescente, que ordena de acordo com o campo “length”, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

  Segunda ordenação: é a ordenação crescente de acordo com o mês no campo data, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

  Terceira ordenação: é a ordenação crescente do campo data, depois de ordenado ele vai gerar os arquivos de melhor, médio e pior caso para cada algoritmo.

Os principais resultados alcançados com relação a classificação de senhas e a filtragem, é o retorno de arquivos que mostram a classificação da senha, seguindo os critérios estabelecidos, e um novo arquivo que mostra a filtragem das senhas consideradas Boa e Muito Boa. Outro resultado encontrado em relação a

A transformação do campo de datas é a obtenção de um arquivo em que a coluna datas se encontra formatada de acordo com as exigências estabelecidas.

Com relação aos algoritmos de ordenação, foi obtido o retorno de todos os arquivos ordenados de acordo com cada especificação e para cada caso de execução.

2. Descrição geral sobre o método utilizado

3.

Os testes foram realizados na IDE Visual Studio Code utilizando a linguagem de programação Java.

De início foram realizados os primeiros testes nos tópicos de classificação de senha, transformação de formatação das datas e na filtragem das senhas. Onde foi analisado o uso da CPU, memória RAM e tempo de execução com o objetivo de avaliar a eficiência dessas transformações.

Para os testes realizados a partir do uso de algoritmos de ordenação, foi realizada uma comparação de todos os algoritmos em relação ao que apresenta o melhor desempenho. Também foram analisados como cada algoritmo de ordenação se comportou em relação ao melhor, médio e pior caso.

Descrição geral do ambiente de testes

Os testes foram executados em um macOS Mojave, versão 10.14.6, MacBook Air, processador 1,8GHz Intel Core i5, memória 8GB 1600 MHz DDR3, placa gráfica Intel HD Grafics 6000 1536 MB.

4. Resultados e Análise

Elaborar os resultados dos testes usando tabelas e gráficos

Para análise dos algoritmos consideramos o arquivo “passwords.csv” com apenas 1000 linhas.

Os algoritmos de ordenação analisados mostram desempenhos diferentes em relação aos campos "length", "mês" e “data”.

Ordenação pelo campo length (em nanosegundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Selection Sort	16134741 ns	10510657 ns	6538234 ns
Insertion Sort	90925 ns	15738874 ns	1734170 ns
Merge Sort	647875 ns	784755 ns	763056 ns
QuickSort	24410846 ns	388055 ns	17282313 ns
QuickSort com Mediana de três	2795306 ns	1353078 ns	713926 ns
Bubble Sort	107125 ns	6543221ns	22428973 ns
HeapSort	1754348 ns	1241135 ns	658391 ns
Counting Sort	369752 ns	783551 ns	377705 ns

Em geral, o Heap Sort parece ser bastante eficaz e consistente em todos os casos. O Quick Sort se destaca pela rapidez e velocidade nos tempos de execução do médio caso, mas tem desempenho variável em outros cenários. O Insertion Sort e Bubble Sort mostram-se ineficientes em alguns cenários, com tempos extremamente altos no médio e pior caso, exigindo mais poder computacional.

No melhor caso , o mais rápido foi o Insertion Sort com 90, 925 ns, o mais eficaz foi o Counting Sort com 369,752 ns, (não foi o mais rápido, mas oferece um bom desempenho em termos gerias nos três casos) e o mais lento foi o Quick Sort com 24, 410,846 ns. No médio caso, o mais rápido foi o Quick Sort com 388, 055 ns, o mais eficaz foi o Heap Sort com 1, 241,135 ns (oferece boa performance tanto no melhor caso quanto no pior caso). No pior caso, o mais rápido foi o Heap Sort com 658, 391 ns, o mais eficaz foi o Quick Sort Mediana de três com 713,926 ns (tempos razoáveis nos três casos, sendo uma opção equilibrada).

Ordenação pelo month (em nanosegundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Selection Sort	625528527 ns	620946454 ns	594470329 ns
Insertion Sort	1313361 ns	312188442 ns	638125286 ns
Merge Sort	6854327 ns	41817811 ns	7497945 ns
QuickSort	346967680 ns	29311958 ns	124252725 ns
QuickSort com Mediana de três	178235277 ns	63060161 ns	121273291 ns
Counting	1225070 ns	1448061 ns	1797804 ns
HeapSort	19970866 ns	41817811 ns	25923779 ns
BubbleSort	1151533 ns	642868624 ns	744282986 ns

Em termos gerais, o Counting Sort teve protagonismo no critério de eficácia e velocidade em todos os casos, mostrando-se como a melhor opção para o campo mês. O Heap Sort e Merge Sort apresentaram tempos razoáveis e podem ser considerados eficazes para uso geral. O Bubble Sort e Selection Sort apresentaram os tempos mais lentos em vários cenários exigindo mais da CPU, sendo menos eficientes para o campo mês.

No melhor caso, o mais rápido foi o Counting Sort com 1, 225, 070 ns, o mais eficaz foi Counting Sort e o mais lento foi o Selection Sort com 625, 528, 527 ns. No médio caso, o mais rápido foi o Counting Sort com 1, 448, 061 ns, o mais eficaz foi o Counting Sort e o mais lento foi o Bubble Sort com 642, 868, 624 ns. No pior caso, o mais rápido foi o Counting Sort com 1, 797, 804 ns, o mais eficaz foi o Counting Sort e o mais lento foi o Bubble Sort com 744, 282, 986 ns.

Ordenação pela data (em nanosegundos)

Algoritmos	Médio caso	Melhor caso	Pior caso
Selection Sort	586540281 ns	613754437 ns	597218498 ns
Insertion Sort	334356709 ns	1238534 ns	664301675 ns
Merge Sort	17031516 ns	7475718 ns	8664724 ns
QuickSort	13565909 ns	331785286 ns	392535233 ns

QuickSort com Mediana de três	22497724 ns	15162974 ns	22488480 ns
BubbleSort	1216724 ns	599578012 ns	609640174 ns
HeapSort	35970491 ns	20845498 ns	29082622 ns
Counting	2907998 ns	1483853 ns	1301425 ns

Em geral, o Counting Sort destaca-se como o mais rápido e eficaz em todos os casos, sendo uma excelente opção para a ordenação das datas com eficiência consistente. O Quick Sort com Mediana de três e o Merge Sort oferecem tempos razoáveis e podem ser considerados alternativas eficazes dependendo do caso específico. O Bubble Sort e Selection Sort foram os mais lentos em vários casos e sendo ineficientes comparados aos outros algoritmos.

No melhor caso, o mais rápido foi o Counting Sort com 1,483, 853 ns, o mais eficaz foi o Counting Sort, e o mais lento foi o Bubble Sort com 599, 578, 012 ns. No médio caso, o mais rápido foi o Bubble Sort com com 1, 216, 724 ns, o mais eficaz foi o Counting Sort com 2, 907, 998 ns obtendo excelente performance. No pior caso, o mais rápido foi o Counting Sort com 1, 301, 425 ns, o mais eficaz foi o Counting Sort e o mais lento foi o Bubble Sort com 609, 640, 174 ns.

Acerca da complexidade dos algoritmos utilizados para estimar o tempo de execução de um arquivo 700 mil entradas com base nos dados fornecidos para 1000 entradas.

Algoritmos	Complexidade
Selection Sort	O(n^2)
Insertion Sort	O(n^2)
Merge Sort	O(n log n)
QuickSort	no médio caso O(n log n), no pior caso O(n^2)
QuickSort com Mediana de três	melhora um pouco mas mantém no médio caso O(n log n), no pior caso O(n^2)
HeapSort	O(n log n)
BubbleSort	O(n^2)
Counting Sort	O(n+k)

Para algoritmos de complexidade **O(n^2)** e o tempo de execução para 1000 entradas é T, para 700.000 entradas o tempo estimado seria:

Tempo estimado = T x (700.000/1000)^2

Tempo estimado = T x 490000

O cálculo para algoritmos de complexidade é um pouco mais complexo e envolve o logaritmo de n. Porém, foi utilizado uma proporção direta , pois 700 x log (700) dá aproximadamente 4900 e 1000 x log (1000) dá aproximadamente 3000, a razão é cerca de 1,6. Assim , o tempo estimado seria:

Tempo estimado = T x 70000 / 1000 x 1,6

Tempo estimado = T x 1120

Algoritmos mais eficientes:

Entre os algoritmos analisados, os mais eficientes em termos de desempenho de tempo são o Counting Sort e o Heap Sort. O Counting Sort é altamente eficiente quando o intervalo de valores é pequeno, com uma complexidade linear de O(n+k), sendo ideal em cenários específicos. O Heap Sort, utiliza moderadamente em termos de CPU, especialmente em um grande conjunto de dados, não requer espaço adicional significativo para sua execução, além do espaço de entrada. A única memória adicional necessária é para armazenamento temporário de poucas variáveis, possui complexidade O(n log n).

Os algoritmos de ordenação diferem em seu uso da CPU durante a execução. Algoritmos como Selection Sort, Bubble Sort e Insertion Sort tendem a ter um uso mais intenso da CPU, enquanto algoritmos como Counting Sort têm um uso mínimo da CPU. Algoritmos como MergeSort, Quicksort e HeapSort têm um uso moderado da CPU.

Os algoritmos de ordenação que utilizam menos memória são o Selection Sort, Insertion Sort e o HeapSort. O Counting Sort também é eficiente no uso de memória, porém, requer uma quantidade de memória proporcional ao intervalo de valores a serem ordenados. O Merge Sort e o QuickSort usam mais memória, mas são eficientes em grandes conjuntos de dados. O Quicksort com Mediana de 3 é eficiente em termos de uso de memória, não exigindo uma quantidade significativa de memória adicional.

Análise geral dos resultados:

Entre os algoritmos analisados, Counting Sort e Heap Sort foram os mais eficientes em termos de desempenho de tempo em cenário de 1000 linhas. Algoritmos como Selection Sort, Bubble Sort e Insertion Sort têm um uso mais intenso da CPU, enquanto algoritmos como Counting Sort têm um uso mínimo da CPU. Os algoritmos que utilizam menos memória são Selection Sort, Insertion Sort e HeapSort, enquanto MergeSort e QuickSort usam mais memória, mas são eficientes em grandes conjuntos de dados.

5. Resultados e Análise usando estruturas de dados

Estruturas de dados utilizadas:

A estrutura de dados usando apenas arrays, junto com o algoritmo de ordenação Counting Sort foi utilizada visando a melhoria na etapa de ordenação dos dados, onde a troca de elementos é feita por meio da reorganização das referências entre os nós. Ou seja, a ausência da necessidade de copiar os elementos para um novo array ajuda no desempenho do código, principalmente pelo fato do arquivo conter uma grande quantidade de dados.

Elaborar os resultados dos testes usando tabelas:

Para análise dos algoritmos consideramos o arquivo “*passwords.csv*” com apenas 1000 linhas.

Os algoritmos de ordenação analisados mostram desempenhos diferentes em relação aos campos "length", "mês" e “data” para os algoritmos de ordenação Insertion Sort, QuickSort e Counting.

Ordenação pelo campo length

(em nanosegundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Insertion Sort	90925 ns	15738874 ns	1734170 ns
QuickSort	24410846 ns	388055 ns	17282313 ns
Counting	369752 ns	783551 ns	377705 ns

Ordenação pelo mês (em nanosegundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Insertion Sort	1313361 ns	312188442 ns	638125286 ns
QuickSort	346967680 ns	29311958 ns	124252725 ns
Counting	1225070 ns	1448061 ns	1797804 ns

Ordenação pela data (em nanosegundos)

Algoritmos	Melhor caso	Médio caso	Pior caso
Insertion Sort	334356709 ns	1238534 ns	664301675 ns
QuickSort	13565909 ns	331785286 ns	392535233 ns
Counting	2907998 ns	1483853 ns	1301425 ns

Algoritmos mais eficientes usando estrutura de dados:

O algoritmo de ordenação QuickSort, combinado com a estrutura de dados de array, foi o mais eficiente em grande volume de dados nos testes realizados. Em todos os cenários analisados (ordenação pelo campo "length", "mês" e "data"), o QuickSort apresentou tempos de execução menores em comparação com os algoritmos Counting Sort e Insertion Sort. A utilização da lista duplamente encadeada contribuiu para o desempenho superior do QuickSort, permitindo uma manipulação ágil e eficiente dos elementos durante o processo de ordenação.

Análise geral dos resultados:

O Counting Sort em comparação a 1000 linhas mostrou o algoritmo mais eficiente em todos os cenários analisados, apresentando tempos de execução consistentemente menores em comparação com o Selection Sort, Bubble Sort e o Insertion Sort. O Counting Sort obteve o melhor desempenho. Por outro lado, o Bubble Sort apresentou um desempenho inferior nos três cenários, sendo mais prejudicado no pior caso da ordenação pela data.