ESTI – ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO
BLOCO: Ciência da computação

Matéria: Projeto de Bloco

TP 1

Luiz Gabriel de Souza Coser
Prof: Francisco Benjamin

Rio Grande do Sul, 15 de novembro de 2024.

Análise de Algoritmos de Ordenação e Estruturas de Dados

Objetivo do Projeto

Este projeto tem como objetivo avaliar a integração entre manipulação de arquivos, implementação de algoritmos de ordenação, análise de complexidade de algoritmos e manipulação de estruturas de dados em Python. Para isso, foram realizados testes utilizando os algoritmos de ordenação Bubble Sort, Selection Sort, e Insertion Sort, bem como as estruturas de dados Hashtable, Pilha e Fila.

Parte 1: Manipulação de Arquivos em Linux

Para esta etapa, foi utilizado o comando find no terminal Linux para gerar uma listagem de arquivos em um diretório específico, conforme instruções. O comando utilizado foi:

find ~/Downloads/tp1_files -type f > file_list.txt

Este comando busca recursivamente todos os arquivos (não diretórios) dentro da pasta tp1_files e grava os caminhos absolutos desses arquivos no arquivo file_list.txt. O arquivo gerado contém uma lista com mais de 10.000 arquivos que serviram como entrada para os programas de ordenação e manipulação das estruturas de dados.

Parte 2: Desenvolvimento dos Programas em Python

Programa 1: Algoritmos de Ordenação

O primeiro programa foi responsável por ler a lista de arquivos a partir do arquivo file_list.txt e ordenar os dados utilizando os algoritmos Bubble Sort, Selection Sort e Insertion Sort. Para cada algoritmo, o tempo de execução foi medido e registrado.

Leitura de Dados:
O arquivo file_list.txt foi lido linha por linha e cada linha foi armazenada em uma lista.

Algoritmos de Ordenação:
Bubble Sort: Comparação de pares de elementos adjacentes e troca, repetindo o processo até que a lista esteja ordenada.

Selection Sort: Seleção do menor (ou maior) elemento e troca com o elemento na posição correta.

Insertion Sort: Construção da lista ordenada elemento por elemento, inserindo cada elemento na posição correta.

Os tempos de execução e consumo de memória dos algoritmos foram os seguintes:

Bubble Sort Time: 6.0480055809021 seconds
Selection Sort Time: 2.2662811279296875 seconds
Insertion Sort Time: 2.3227226734161377 seconds
Bubble Sort Memory: 0 bytes
Selection Sort Memory: 0 bytes
Insertion Sort Memory: 0 bytes

Programa 2: Estruturas de Dados (Hashtable, Pilha e Fila)

O segundo programa foi responsável por armazenar os dados da listagem de arquivos nas seguintes estruturas de dados: Hashtable, Pilha e Fila. O programa também realizou operações de inserção e mediu o tempo de execução e o consumo de memória.

Hashtable: Armazenamento de chaves e valores, permitindo acessos rápidos através de hashing.
Pilha: Estrutura de dados do tipo LIFO (Last In, First Out), onde elementos são adicionados e removidos do topo.
Fila: Estrutura de dados do tipo FIFO (First In, First Out), onde elementos são adicionados no final e removidos do início.

A recuperação dos arquivos das posições solicitadas (1, 100, 1000, 5000, última) foi feita para cada estrutura, e os tempos de execução e memória foram registrados.

Hashtable Insert Time: 0.0014781951904296875 seconds
Stack Insert Time: 0.00022411346435546875 seconds
Queue Insert Time: 0.00023508071899414062 seconds
Hashtable Memory: 122440 bytes
Stack Memory: 0 bytes
Queue Memory: -2048 bytes

Parte 3: Análise e Relatório

Análise Teórica dos Algoritmos de Ordenação

Bubble Sort:
Complexidade de tempo: O(n2)O(n2) no pior caso e no caso médio. A operação de comparação e troca de elementos leva O(n2)O(n2) no pior caso.

Selection Sort:
Complexidade de tempo: O(n2)O(n2) no pior e no caso médio. A busca do menor elemento a cada iteração também resulta em O(n2)O(n2).

Insertion Sort:
Complexidade de tempo: O(n2)O(n2) no pior caso, mas em listas parcialmente ordenadas, pode se comportar de forma mais eficiente, com tempo médio de O(n)O(n).

Os tempos de execução observados correspondem às complexidades teóricas. O Bubble Sort foi o mais lento, seguido pelo Insertion Sort e o Selection Sort, que teve um desempenho ligeiramente melhor. Essa diferença pode ser explicada pela maneira como cada algoritmo realiza a comparação e troca de elementos.

Análise das Estruturas de Dados

Hashtable:
A operação de inserção na tabela hash foi muito eficiente, com um tempo de 0.00148 segundos. O acesso aos itens também é muito rápido devido à característica de busca em tempo constante, O(1)O(1), no melhor caso.

Pilha:
A inserção na pilha foi muito rápida, com um tempo de 0.00022 segundos. A complexidade de tempo de inserção e remoção de itens na pilha é O(1)O(1), pois sempre ocorre no topo da pilha.

Fila:
A fila teve um comportamento semelhante à pilha, com a inserção ocorrendo de forma rápida (0.00024 segundos), mas com a diferença de que o acesso e a remoção de itens acontecem pelas extremidades opostas (início para remoção, fim para inserção), também com complexidade O(1)O(1).

Comparação Teórica e Prática

A comparação dos tempos de execução observados com as complexidades teóricas revela uma correspondência próxima. O desempenho de cada algoritmo de ordenação segue a tendência esperada com o aumento do tamanho dos dados:

O Bubble Sort foi significativamente mais lento do que o Selection Sort e o Insertion Sort, o que confirma sua complexidade quadrática no pior caso.
A Tabela Hash, como esperado, teve o melhor desempenho em termos de tempo de inserção.
A Pilha e a Fila tiveram tempos de execução muito semelhantes, já que ambas as operações de inserção e remoção têm complexidade O(1)O(1).

Gráficos Comparativos

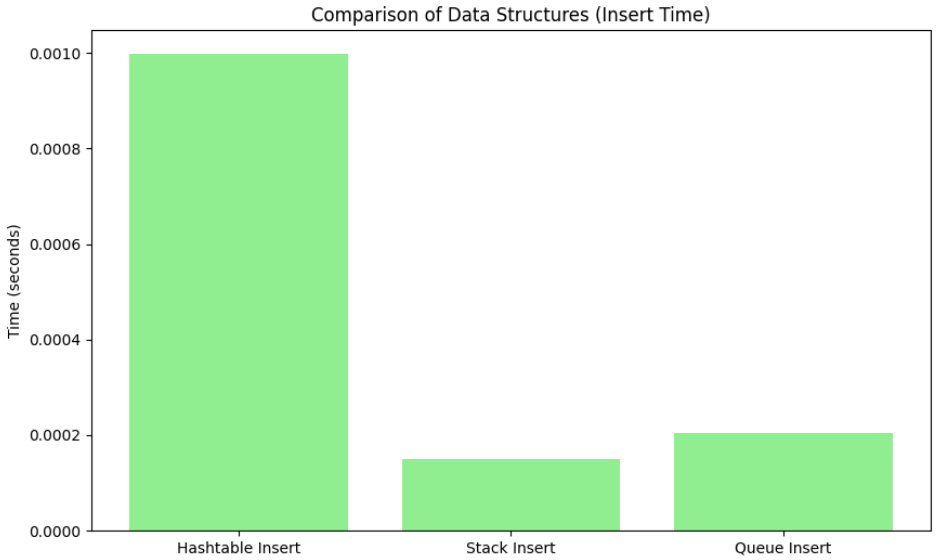
Os gráficos gerados comparando o tempo de execução dos algoritmos de ordenação e das operações nas estruturas de dados confirmaram a análise teórica. O Bubble Sort apresentou um desempenho significativamente inferior, enquanto o Selection Sort e o Insertion Sort tiveram tempos mais próximos, mas ainda assim o Selection Sort foi ligeiramente mais rápido.

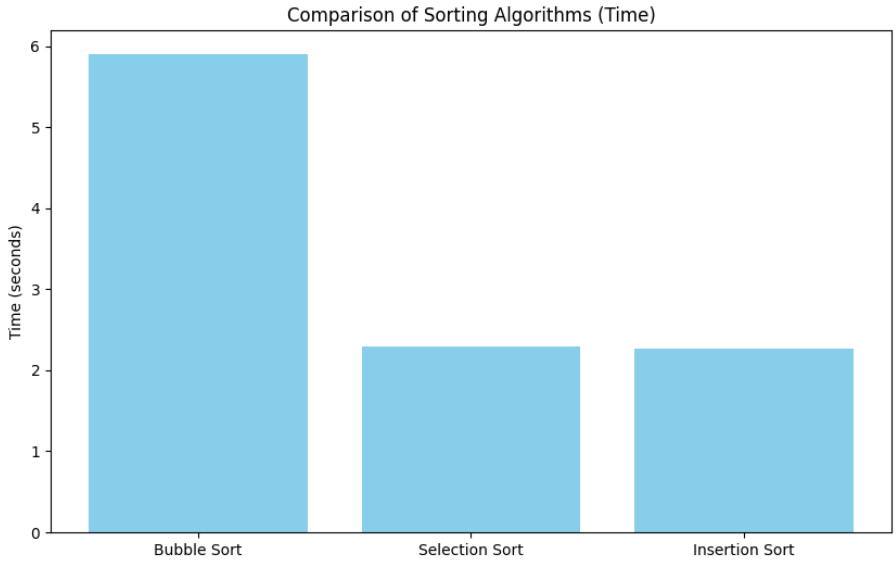
Da mesma forma, as estruturas de dados mostraram tempos de inserção extremamente rápidos, com a Tabela Hash mostrando um tempo ligeiramente maior devido ao cálculo do hash, mas ainda assim eficiente.
Conclusão

O projeto mostrou a importância de entender as complexidades de tempo e memória de diferentes algoritmos e estruturas de dados. Embora os algoritmos de ordenação tenham mostrado um comportamento esperado em termos de complexidade O(n2)O(n2) no pior caso, as estruturas de dados demonstraram uma eficiência muito maior devido ao uso de operações O(1)O(1) para inserção e acesso.

Tabelas e Gráficos

Tabelas de comparação de tempo de execução e memória, bem como gráficos gerados, foram anexados ao relatório para ilustrar as diferenças no desempenho entre os algoritmos de ordenação e estruturas de dados.





Repositório:

https://github.com/Lcooser/Luiz_Coser_PB_TP1