Matheus Victor Rocha Rodrigues

RELATÓRIO
LEDA - Laboratório de Estrutura de Dados
Project Steam DataSet UPGRADE

Campina Grande - PB
15 de junho de 2024

Introdução: Neste projeto, a tarefa era implementar três novos algoritmos de ordenação para um arquivo CSV que contém informações sobre jogos, sem fazer uso de arrays. Os três algoritmos escolhidos foram Heap Sort, Shell Sort e Bucket Sort. Este relatório fornece uma justificativa para a escolha dessas estruturas de dados, explica os problemas que elas resolvem e detalha os locais onde as estruturas de dados foram introduzidas no código.

Estruturas de Dados e Algoritmos:

1. **HeapSort:**

   Justificativa: O Heapsort foi escolhido por sua eficiência e capacidade de garantir um pior caso de tempo de execução de **O(nLogn)**. Ele utiliza uma estrutura de dados de heap binário, que é eficiente para inserir e remover o maior ou menor elemento. Isso é particularmente útil em cenários onde precisamos de uma ordenação estável e com desempenho previsível.

   Problema que resolve: O Heapsort é eficaz para ordenar grandes volumes de dados de forma eficiente. No contexto deste projeto, ele resolve o problema de ordenar os jogos com base em diferentes atributos (Data de Lançamento, Preço e Número de Conquistas) sem a necessidade de armazenar todos os elementos em memória de uma vez.

   Local do Código: O Heapsort foi implementado na classe HeapSort.java

2. **ShellSort:**

   Justificativa: O Shellsort foi escolhido devido à sua simplicidade e melhor desempenho em relação ao Insertion Sort para listas maiores. Ele usa uma estratégia de "gap" que permite comparações e trocas de elementos que estão distantes uns dos outros, o que pode melhorar significativamente a eficiência de ordenação em listas parcialmente ordenadas.

   Problema que resolve: O Shellsort resolve o problema de ordenar eficientemente listas que podem ser quase ordenadas inicialmente. Ele é adequado para o contexto, onde diferentes critérios de ordenação podem produzir listas parcialmente ordenadas.

   Local do Código: O Shellsort foi implementado na classe ShellSort.java

3. **BucketSort:**

   Justificativa: O BucketSort foi escolhido por sua eficiência em lidar com dados que estão uniformemente distribuídos. Ele é particularmente útil quando os valores a serem ordenados podem ser distribuídos uniformemente em intervalos conhecidos, o que permite que o algoritmo opere em tempo linear O(n) em casos ideais.

   Problema que resolve: O BucketSort resolve o problema de ordenar grandes conjuntos de dados que têm uma distribuição uniforme. Ele é ideal para nosso projeto, onde os dados dos jogos (como preços) podem variar dentro de um intervalo conhecido.

   Local do Código: O BucketSort foi implementado na classe BucketSort.java

Conclusão: Os três algoritmos de ordenação escolhidos e implementados (Heap Sort, Shell Sort e Bucket Sort) foram selecionados com base em suas eficiências e características específicas que os tornam adequados para ordenar grandes volumes de dados sem a necessidade de armazenar todos os elementos em memória de uma vez. Cada algoritmo resolve problemas específicos de ordenação e proporciona uma abordagem robusta e eficiente para lidar com os dados no contexto do projeto.

Os locais onde as estruturas de dados foram introduzidas são claramente definidos nas implementações de HeapSort.java, ShellSort.java e BucketSort.java. Essas implementações garantem que os dados sejam lidos do arquivo CSV, ordenados conforme necessário e gravados de volta em arquivos CSV separados para cada critério de ordenação.