Laboratório de Estrutura de Dados

Segunda versão do projeto da disciplina
Comparação entre os algoritmos de ordenação elementar com estrutura de dados.

Guilherme Ribeiro Liebig

Raiff Ferreira Telecio

1. Introdução

O relatório apresenta a segunda parte da entrega do projeto de análise dos algoritmos de ordenação, dessa vez, utilizando, por meio da introdução de estruturas de dados implementares de forma manual, em conformidade com o que foi requisitado na disciplina de Laboratório de Estrutura de Dados (LEDA).

Por definição dos requisitos, é proíbido a utilização de estruturas nativas do Java, como ArrayList, LinkedList ou Stack. Para atender essa restrição, foram desenvolvidas três estruturas próprias: uma Lista Encadeada, uma Lista Duplamente Encadeada e uma Fila de Prioridade baseada em heap. Cada uma delas foi aplicada em diferentes momentos do projeto, de acordo com as características dos dados e dos algoritmos de ordenação utilizados.

O projeto segue utilizando a ideia de ordenação utilizando o dataset da Steam, tendo foco na ordenação por três critérios, sendo eles: data de lançamento, preço e número de conquistas. As novas estruturas foram integradas às classes responsáveis por cada tipo de ordenação, substituindo os arrays que foram utilizados na primeira versão.

Entre os principais resultados, podemos destacar que os algoritmos de ordenação foram adaptados com sucesso para funcionar com as estruturas de dados implementadas manualmente. Além disso, o projeto proporcionou uma compreensão mais prática sobre o controle de memória e permitiu eliminar completamente o uso de estruturas prontas da linguagem Java, como era exigido.

2. Estruturas de Dados Criadas e Justificativas

2.1 Lista Encadeada Simples:

- É uma lista linear com um ponteiro que aponta para o próximo elemento.
- A Classe implementada se chama: ListaEncadeada<T>.
- Métodos: ordenarPorSelection, ordenarPorInsertion, ordenarPorMerge, ordenarPorQuickSort, ordenarPorHeap
- Foi utilizada no seguinte arquivo: AlgoritmosDate.java e utilizando o critério de ordenação de Data de Lançamento (Release Date).

A lista encadeada foi escolhida por ter uma simples implementação e ter boa eficiência para estruturas que precisam crescer dinamicamente. Como os algoritmos testados nessa etapa, são, dentre eles, Selection, Insertion, Merge, Quick e Heap, que operam com acesso sequencial, a lista encadeada resolveu de forma eficiente o problema, facilitando operações como a troca de elementos (swap) e a divisão da lista em subpartes, especialmente em ordenações como Merge Sort.

2.2 Lista Duplamente Encadeada:

- É uma lista com ponteiros que apontam tanto para o próximo quanto para o elemento anterior.
- A Classe implementada se chama: ListaDuplamenteEncadeada<T>
- Métodos: ordenarPorSelection, ordenarPorInsertion, ordenarPorMerge, ordenarPorQuickSort, ordenarPorQuickSortMediana3, ordenarPorHeap.
- Foi utilizada no seguinte arquivo: AlgoritmosPrice.java e utilizando o critério de ordenação de Preço (Price).

A versão duplamente encadeada facilita percorrer os dados nos dois sentidos. Mostrando-se útil, principalmente para o Quick Sort com mediana de três e para o Heap Sort, que exigem manipulação mais intensa dos elementos e trocas em

posições diferentes. Essa estrutura também trouxe mais eficiência ao acessar elementos que estão mais próximos do fim da lista.

Dessa forma, permite implementação eficiente de algoritmos que exigem navegação em ambas direções e trocas frequentes de posições, como o Quick Sort com pivô na mediana.

2.3 Fila de Prioridade:

- É um Fila de prioridade implementado manualmente, baseada em heap máximo.
- A Classe implementada se chama: PriorityQueueAchievements.
- Método: ordenarPorPriorityQueue(CSVRecord[] registros).
- Seu critério de ordenação é o número de conquistas (Achievements).

A estrutura foi desenvolvida para lidar com ordenações baseadas em prioridade, nesse caso, ordenar o número de conquistas. O heap foi implementado com um vetor fixo e regras de comparação com base na coluna 26 do CSV, que representa as conquistas. Permitindo remover rapidamente o maior valor, facilitando a ordenação decrescente de forma mais eficiente.

Permite organizar os registros de forma que os elementos com maior número de conquistas sejam tratados primeiro, com inserções e remoções em tempo logarítmico.

3. Local e Contexto de Aplicação das Estruturas

As estruturas de dados implementadas foram inseridas diretamente nas classes que lidam com os algoritmos de ordenação do projeto. A escolha de onde aplicar cada estrutura foi feita com base no tipo de dado trabalhado (data, preço ou

conquistas), considerando também o comportamento dos algoritmos e a forma como os elementos precisariam ser manipulados durante o processo de ordenação.

Abaixo, explicamos onde e por que cada uma foi utilizada:

- A estrutura ListaEncadeada<T> foi aplicada na classe AlgoritmosDate.java, responsável pela ordenação por data de lançamento dos jogos. Como os algoritmos utilizados nesse módulo (como Insertion, Merge e Quick Sort) funcionam bem com acesso sequencial, a lista encadeada atendeu às necessidades do projeto de forma simples e eficiente. Ela também ajudou a evitar o uso de arrays fixos, além de facilitar operações como divisão e reorganização da lista durante a ordenação.
- Já a ListaDuplamenteEncadeada<T> foi usada na classe AlgoritmosPrice.java, voltada para a ordenação com base no preço dos jogos. Como algumas das ordenações testadas exigem muitas trocas e acessos bidirecionais (como no QuickSort com mediana de três ou no HeapSort), essa estrutura trouxe vantagens por permitir navegação tanto para frente quanto para trás. Isso tornou o código mais eficiente e evitou o retrabalho de percorrer a lista sempre desde o início.
- Por fim, a estrutura PriorityQueueAchievements, que foi implementada como uma fila de prioridade usando heap, foi utilizada na classe AlgoritmosAchievements.java. Nesse caso, o foco estava em ordenar os jogos de acordo com a quantidade de conquistas (valor localizado na coluna 26 do dataset). A estrutura se mostrou ideal para isso, já que o heap possibilita a remoção rápida dos maiores valores, o que se encaixa perfeitamente em uma ordenação decrescente.

Cada estrutura foi aplicada de forma estratégica, respeitando os limites impostos (como a proibição do uso de ArrayList e outras estruturas prontas do Java). Além

disso, a divisão dos módulos por tipo de ordenação (data, preço e conquistas) ajudou a manter o projeto organizado.

4. Aplicação das Estruturas no Projeto

A substituição das estruturas de dados nativas por implementações próprias foi feita respeitando a organização original do projeto. Ao invés de reescrever toda a lógica dos algoritmos de ordenação, o foco foi adaptar os pontos necessários, trocando os arrays pelas novas estruturas de forma pontual, sem quebrar o funcionamento do código.

Cada classe principal manteve sua função original:

- A classe AlgoritmosDate.java passou a utilizar a ListaEncadeada;
- AlgoritmosPrice.java foi adaptada para usar a ListaDuplamenteEncadeada;
- AlgoritmosAchievements.java agora trabalha com a estrutura PriorityQueueAchievements.

Essas classes continuam sendo chamadas pelos arquivos (OrdenacaoDate.java, OrdenacaoPrice.java e OrdenacaoAchievements.java), que são responsáveis por ler os dados do CSV, aplicar os algoritmos, medir o tempo e memória consumidos e salvar os resultados.

As alterações foram feitas apenas onde realmente era necessário, o que ajudou a preservar a separação entre as partes do código e facilitou a manutenção. No fim, foi possível evoluir tecnicamente a base do projeto sem comprometer sua lógica nem sua organização original.

5. Conclusão

A segunda parte do projeto foi importante para consolidar o aprendizado sobre estruturas de dados e entender, na prática, como elas se aplicam nos algoritmos de ordenação. Ao criar manualmente estruturas como listas encadeadas e fila de prioridade com heap, deu para perceber com mais clareza o impacto que essas escolhas têm no funcionamento do código e no desempenho geral do sistema.

Cada estrutura foi aplicada de forma pensada, de acordo com o tipo de dado e com o comportamento dos algoritmos usado.

Durante a implementação, foi possível entender como essas estruturas funcionam por trás. Além de manter o funcionamento do projeto original, essa etapa reforçou o entendimento prático sobre listas encadeadas e fila de prioridade, destacando como a escolha da estrutura correta influencia diretamente na eficiência e organização do código. Provando que a combinação da estrutura de dados correta com um algoritmo de ordenação adequado leva um tempo de execução do programa muito mais rápido.

No geral, a atividade contribuiu para consolidar o conteúdo teórico de estrutura de dados e demonstrou, na prática, como decisões técnicas bem pensadas fazem diferença no resultado final de um sistema.