Laboratório de Estrutura de Dados

Versão final do projeto da disciplina

Comparação entre estruturas de dados para ordenação

Lucas Silva Nascimento

Manoel Barros da Cruz Neto

Vitor Francisco Farias Souza

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no segundo projeto da disciplina de LEDA que trazem os resultados das ordenações efetuadas por estruturas de dados diferentes das anteriores, usando uma base de dados fornecida pelo professor, disponibilizada pela plataforma *kaggle*, contendo diversas informações sobre jogos da *steam*, sendo dados coletados por meio da *Steam Spy*, API desse ambiente. O objetivo do projeto, é efetuar as ordenações solicitadas pelo professor, utilizando de estruturas de dados desenvolvidas por nós alunos.

Por conseguinte, no projeto é implementado a transformação das datas de lançamento (campo: Release date) do formato "MM DD, AAAA" para o formato "DD/MM/AAAA", gerando um arquivo chamado "games_formated_release_date.csv". Analogamente, foi gerado um arquivo com todos os jogos compatíveis com o sistema operacional linux (campo: Linux) com base no arquivo previamente gerado (games_formated_release_date.csv). Finalmente, é feito um novo arquivo com filtro de jogos que tem suporte a língua portuguesa (campo: Supported languages) utilizando novamente o arquivo das datas de lançamento. Feitas as devidas transformações solicitadas, foram implementadas estruturas de dados para serem efetuadas as devidas ordenações:

- Ordenar pela data de lançamento de forma crescente;
- Ordenar por preço dos jogos de forma crescente;
- Ordenar pelo número de conquistas de forma decrescente.

Além disso, foram propostos três casos para os quais os testes seriam efetuados. São eles: o dataset original (sem nenhuma ordenação prévia), o dataset de maneira já ordenada e o dataset ordenado na sua forma inversa, de maneira que esteja totalmente desordenado.

Para esse projeto, foram escolhidas para serem implementadas 3 diferentes árvores binárias. Foram elas: árvore binária de busca —*bst*—, árvore AVL e árvore preto-vermelho. Com isso, tendo seu uso justificado pela sua característica de ser uma estrutura ordenada durante sua criação, onde os elementos ficam organizados da maneira desejada, conforme implementado(parâmetro de ordenação). Nesse sentido, a cada inserção de elemento em suas respectiva estrutura, ele se posiciona de maneira ordenada, não tornando necessária aplicação de métodos de ordenação para tal.

Além disso, vale ressaltar que todas as estruturas implementadas apresentaram-se mais eficientes que os métodos de ordenação propostos anteriormente. Inicialmente, a árvore binária de busca, foi a menos eficiente dentre as árvores, tendo seu desempenho justificado pela falta de balanceamento, que para os casos onde os dados a serem inseridos já estavam ordenados de alguma forma apresentou seu pior caso, com complexidade $O(n)$. Assim, para amenizar esse problema e evitar

erros como *stackoverflow,* as inserções nessa árvores foram implementadas utilizando o método iterativo, evitando recursões excessivas que geraram tal problema.

Ademais, a árvore binária de busca, demonstrou sua eficiência com relação aos métodos utilizados no projeto anterior, nos demais casos, onde apresentou seu desempenho *O(log n)*, cumprindo com todas as ordenações necessárias. Outrossim, ambas as outras árvores implementadas, são árvores que possuem estrategias de balanceamento efetivo, ou seja, excluem problemas causados pelo desbalanceamento que pode ser gerado na árvore binária de busca. Assim, não apresentando casos diferentes de *O(log n)* ao inserirem elementos e balanceando a estrutura durante sua criação. Desse modo, justificando seus usos e vatangens de aplicação.

Resumidamente, o relatório estruturou-se da seguinte forma, foi inicialmente apresentado a ideia e objetivos de implementação do projeto, com uma motivação da utilização de tais estruturas de dados. Posteriormente, seguiu-se expondo uma descrição geral sobre a implementação do projeto, descrevendo como se deu o desenvolvimento do código para conquistar os objetivos propostos, bem como, a descrição e especificação do ambiente ao qual o submeteu-se a execução do código. Por fim, foram expostos os resultados, bem como dissertado e analisado cada resultado com apresentação de tabelas, para chegar as conclusões coerentes.

2. Descrição geral sobre o método utilizado

Para o desenvolvimento desse projeto, foi criado um projeto Maven na IDE Intellij. Incrementando a API Apache Commons CSV, aplicada para leitura e escrita dos arquivos separados por vírgulas, por meio, de objetos *Writer*, *Reader*, *CSVPrinter* e *CSVParser*. Além disso, vale salientar que o projeto foi dividido em pacotes, onde, cada estrutura desenvolvida possui seu devido pacote com sua implementação, bem como, o pacote *database* que contém todos os arquivos .csv, além dos pacotes, *tranformations* e *ordenations*, que armazenam as classes responsáveis por cada operação respectivamente, por fim, o pacote *org.example* contendo a classe Main.java, sendo a única necessária a ser executada.

No pacote *transformations*, esta contida apenas uma classe que possui métodos para realizar todas as transformações e seu método “mainTranformations()” que executa cada função no momento exato para realização das transformações. Nessa etapa, foram convertidas as datas utilizando classes da biblioteca *java.time*, dividindo a fase de transformações em 4 funções principais: *convertDate(), toReleaseDate(), getLinuxGames()* e *getPortugueseGames().*

De modo que, a função *convertDate()* é utilizada pelo método *toReleaseDate()* que tem funcionamento semelhante aos demais métodos. Com isso, cada um dos métodos, exceto o que executa a conversão das datas tem seu funcionamento da seguinte forma, todo o .csv é lido e

armazenado em um objeto *CSVParser* e sobre ele é iterado até o fim de todas as linhas fazendo as devidas verificações para efetuar as transformações, tudo isso na classe criada, chamada de *Transformations.java*.

No pacote *ordenations*, existe uma interface chamada *OrdenationsInterface.java* implementada pela superclasse *Ordenations*, sendo ela, classe "pai" de todas as outras classes de ordenações que foram criadas. Nesa superclasse, encontram-se os métodos de operações com .csv e inversão de elementos, que seriam necessárias para todas as ordenações propostas. Além disso, nesse pacote existem as classes *OrdenationsAvlTree.java, OrdenationsBinarySearchTree.java* e *OrdenationsRedBlack.java*. Essas classes, tem funcionamento semelhante, uma vez que elas contêm seus respectivos métodos "main" que organizam cada etapa de processamento, onde inicialmente lê-se o .csv, inserem-se os elementos em cada árvore, percorre-se essa árvore em ordem(calculando o tempo de execução de cada operação), por fim, escreve-se um novo arquivo .csv com os dados ordenados.

Por fim, cada pacote de árvore contêm sua respectiva implementação, conforme o nome do pacote. Nesse sentido, todas as árvores possuem uma classe Node.java que representam os nós de cada árvore. Além disso, possuem as implementações de cada árvore, que para cada demanda de ordenação possui um método de inserção de elementos diferente, mudando o fator de comparação que determina a posição dos elementos na árvore, ou seja, um método de inserção por datas, um método de inserção por preços e outros de inserção por conquistas, com funcionamento de maneira contrária para ficar ordenada de maneira decrescente.

Assim, para o pleno funcionamento do código foi criada uma classe *Main.java*, contida no pacote *org.example*, que invoca o método principal de cada uma das classes de ordenação e transformação anteriores, garantindo assim que o código funcione perfeitamente. Ademais, vale ressaltar que todos os caminhos para os arquivos foram criados usando um objeto *Path*, garantindo sua portabilidade em diversos sistemas operacionais. Além disso, vale ressaltar que os elementos que possuem atributos iguais para ordenação foram colocados como filho a direita, ou seja, datas iguais, preços iguais e quantidades iguais de conquistas.

Descrição geral do ambiente de testes

Os testes foram realizados em um notebook com o sistema operacional Windows 11 (dual boot), possuindo 16gb (gigabytes) de memória ram, um processador AMD A10-9600P Radeon R5, frequência de 2.40GHz. A IDE utilizada para os testes foi o Intellij Community 2024.2.1, com o aplicativo instalado no SSD de 512gb. Vale ressaltar, que o notebook esteve a todo momento ligado na tomada e somente com a IDE em execução, assim garantindo a melhor análise possível.

3. Resultados e Análise

Resultados observados a partir dos testes realizados com árvores binárias de busca.

Tabela 1: Tempo de execução das operações com árvore binária de busca em segundos.

Árvore Binária de Busca (Em segundos)			
Tipo de ordenação	**Dados originais**	**Dados ordenados**	**Dados inversamente ordenados**
Datas	9	3533	118
Preços	145	1633	150
Conquistas	231	1248	207

Fonte: Autor.

Como foi possível observar, mesmo sendo uma árvore sem balanceamento, seus tempos de execução foram expressivamente eficientes, cumprindo com o que poderíamos esperar, uma vez que sua complexidade esperada é de *O(log n)*. Entretanto, conforme o exposto anteriormente, constatamos o seu pior desempenho quando os dados são inseridos de maneira ordenada, com isso gerando uma degeneração dessa estrutura, uma vez que ela se tornará totalmente desbalanceada, denotando sua complexidade *O(n)* nesse caso. Além disso, os dados inversamente ordenados não obtiveram essa complexidade, pois, foi utilizada uma abordagem onde os dados em que as comparações mostraram ser iguais, foram inseridos a direita, logo, não permitindo que se torne uma árvore totalmente desbalanceada a esquerda.

Resultados observados a partir dos testes realizados com árvores AVL.

Tabela 2: Tempo de execução das operações com árvore AVL em segundos.

Árvore AVL (Em segundos)			
Tipo de ordenação	**Dados originais**	**Dados ordenados**	**Dados inversamente ordenados**
Datas	2	1	1
Preços	0,406	0,296	0,234
Conquistas	0,203	0,093	0,094

Fonte: Autor.

Essa estrutura, evidencia o desempenho de uma árvore binária balanceada, mostrando que não possui um pior caso em termos de resultado. Nesse sentido, todos os resultados obtidos mostraram sua eficiência *O(log n)*, cumprindo com os resultados esperados, onde todos os tempos de execução foram mínimos. Assim, denotando a eficácia de uma árvore balanceada, que excluem piores casos, gerando estruturas já ordenadas e que podem ser acessadas com alta eficiência.

Resultados observados a partir das operações com árvores preto-vermelho.

Tabela 3: Tempo de execução das operações com árvore preto-vermelho em segundos.

Árvore Preto-Vermelho (Em segundos)			
Tipo de ordenação	**Dados originais**	**Dados ordenados**	**Dados inversamente ordenados**
Datas	2	1	2
Preços	0,484	0,422	0,781
Conquistas	0,203	0,094	0,11

Fonte: Autor.

Essa estrutura, semelhante a anterior, denotaram a eficiência de uma estrutura de árvore balanceada. Assim, possuindo baixos tempos de execução e consequentemente, alta eficiência, com complexidade de tempo de execução *O(log n)*. Além disso, foi possível observar como uma restrição de balanceamento afeta diretamente no tempo de execução até entre árvores que apresentam essa característica. Como foi possível observar, comparando os resultados entre árvores AVL e preto-vermelho.

Nesse sentido, as árvores AVL que possuem regras de balanceamento mais estritas e rígidas, apresentaram os melhores resultados, mesmo que um pouco melhor ainda que a preto-vermelho. Assim, demonstrando o poder dessa estrutura de dados. Com isso, foi comprovado que a utilização dessa estrutura se justifica pela sua eficiência e característica, que foram mais eficazes que todos os algoritmos de ordenação utilizados anteriormente, cumprindo o mesmo objetivo, uma vez que essa estrutura por sua característica é criada de maneira ordenada, ou seja, despreza a necessidade de algoritmos de ordenação para tal de forma mais eficiente, como foi possível observar pelos tempos de execução anteriormente obtidos.