Sistema de Catalogação de Livros: Uma Aplicação Desenvolvida Em Java

Everton Bruno Silva dos Santos[1]

[1]Universidade Estadual de Feira de Santana (UEFS)
Caixa Postal 44036-900 – Feira de Santana – BA – Brazil
Av. Transnordestina, s/n, Novo Horizonte

evertonbrunosds@gmail.com

Resumo. *O Clube da Batata, visando auxiliar o projeto do Projeto Gutemberg solicitou a elaboração de um sistema capaz de ler e processar os dados contidos em arquivos CSV disponíveis em sua base de dados, de modo que, a quantidade de dados não fosse capaz de influenciar no tempo e processamento da aplicação, para tal, foi utilizada uma estrutura de complexidade O(log n). Essa solução carrega, grava e processa os dados contidos nos arquivos CSV visando facilitar a manipulação feita pelo usuário dessa ferramenta a medida em que desejar. Assim, quando atendidas todas as condições de uso descritas ao longo do relatório, o software apresenta na tela um menu contendo uma série de opções que podem ser utilizadas para exibir e manipular dados forma precisa e coerente, sendo uma solução eficaz para o problema proposto.*

1. Introdução

O hábito de ler é praticado pelo homem desde a antiguidade. Assim, com o passar dos séculos, essa prática tem se modernizado a medida em que sofre influências da tecnologia. Nesse sentido, visando atender aos praticantes da leitura, o Projeto Gutemberg surge como uma ferramenta tecnológica que dentre outros objetivos pretende compartilhar conhecimento por meio da disponibilização de livros em formato de mídia digital online e sem custos aos seus utilizadores. Portanto, devido essa característica, o projeto possui uma certa flexibilidade, uma vez que, os livros podem ser acessados em qualquer lugar e a qualquer momento. Ademais, outro ponto relevante é que no *web site* do projeto até então, constam um total de mais de 60 mil livros que podem ser facilmente encontrados pelos motores de busca.

Este relatório, tem por objetivo descrever a aplicação que foi desenvolvida na linguagem de programação Java para efetuar a exibição de informações contidas em arquivos CSV que podem ser encontrados na base de dados da Comunidade da Batata, essa que tem efetuado a catalogação dos livros contidos no *web site* do Projeto Gutemberg em arquivos do tipo citado citado. Ademais, a aplicação é não só capaz de manipular arquivos previamente criados, mas é também de gera-los e altera-los a medida em que o usuário desejar, mesmo que dentro das possibilidades contidas na implementação do projeto desenvolvido.

Além disso, sabendo-se que a Comunidade da Batata em um projeto anterior havia sofrido com problemas de desempenho na manipulação de arquivos CSV, conveniou-se que a solução deste problema poderia se dar por meio de uma estrutura de média complexidade O(log n). Desse modo, espera-se que a aplicação possa proporcionar uma maior fluidez para inserir, buscar e remover dados.

2. Metodologia

Ao longo do desenvolvimento do projeto houve um considerável número de sessões em sala, essas que puderam contribuir significativamente para a implementação desse projeto. Nesse aspecto, dentre todas a discussões a mais relevante trata do núcleo do problema, que nesse caso refere-se ao desempenho do sistema, logo, desvendar do que se trata uma estrutura de média complexidade O(log n) foi algo extremamente essencial.

No decorrer do desenvolvimento da aplicação, utilizando-se da abordagem "dividir para conquistar", foi efetuada a divisão do processo de implementação em etapas de menor complexidade, processo estabelecido da seguinte forma:

1. Compreensão de complexidade de algoritmo.
2. Elaboração da árvore AVL.
3. Elaboração dos métodos responsáveis por carregar e gravar arquivos.
4. Compreensão e utilização do polimorfismo.

Compreensão de complexidade de algoritmo: ao longo das sessões a árvore AVL apresentou-se como uma ótima alternativa afim de solucionar o problema de desempenho enfrentado pela Comunidade da Batata em seu primeiro sistema. Contudo, antes de mais nada, foi preciso compreender o por que essa estrutura é considerada de complexidade O(log n).

Assim, no que se refere a complexidade de algoritmos deve se levar em consideração que buscar, remover e até mesmo inserir elementos em uma estrutura que possa conte-los é uma atividade que requer passos. Afinal, envolve determinar onde se encontra dado elemento. Frente a esse situação, temos dois modos convencionais, um envolve passos lineares e um outro, passos que pretendem "dividir para conquistar" a medida em que reduz pela metade o tempo a execução de uma inserção, busca ou remoção.

Nesse sentido, quando um algoritmo deve percorrer todos os elementos de uma estrutura afim de determinar onde se encontra um dado elemento, estamos falando de uma estrutura que efetua esse processo de determinação de forma linear. Com isso, assumimos que caso o elemento seja o último dentre todos os demais, o processo será consideravelmente lento a medida em que o número de elementos aumentar. Sendo assim, estamos falando de uma estrutura de complexidade O(n). Segundo [Deitel 2017], diz-se que um algoritmo como esse exige um total de n-1 comparações, uma vez que tem um tempo de execução linear é afetado pelo número de elementos.

Contudo, uma estrutura que possui a complexidade O(log n) tem por característica a capacidade de reduzir pela metade o número de comparações a cada passo que efetua. Logo, afeta relativamente menos o tempo de execução se comparado com um algoritmo de complexidade O(n). Nesse caso, pode-se se afirmar que em uma estrutura que contenha um número de 500 milhões de elementos levará em seu pior caso, um total de 30 comparações e caso o número de elemento dobre, será necessário apenas mais uma comparação para determinar um elemento no pior caso. Segundo [Deitel 2017], isso resulta em uma complexidade O(log n), que em uma busca binária é dita como tempo de execução logarítmico.

Elaboração da árvore AVL: A primeira vista, é tentadora a possibili-

dade de implementação de uma árvore binária se compararmos sua simplicidade de implementação com as de algumas outras. Contudo, ainda que, segundo [Lafore 2004], uma árvore binária de busca tenha a complexidade O(log n), é notório que sua eficiência apresente-se problemática no caso dos elementos serem inseridos em ordem crescente ou decrescente, algo que irá torna-la desbalanceada aumentando seu tempo de busca consideravelmente, uma vez que, segundo [Lafore 2004], o tempo gasto na procura de um nó depende da quantidade de níveis que estão sobre ele.

Frente a esse problema, surge uma árvore similar a binária de busca, mas com o diferencial capaz de garantir que sua eficiência não seja prejudicada devido a remoções ou inserções. Sendo assim, de modo geral a árvore AVL, é nada mais é que uma árvore binária de busca, contudo é capaz de manter seus filhos tanto à direita quanto à esquerda em uma diferença constante entre 1 e -1 nível de altura. Com isso, conclui-se que a principal diferença entre ambas as estruturas situa-se no balanceamento que ocorre de forma automática a cada inserção ou remoção, sendo essas as únicas operações capazes de interferir na diferença de altura entre os filhos.

Visando promover a possibilidade de balanceamento da árvore, houve a necessidade de criar outros métodos além dos que são comuns em árvores binárias de busca. Esses outros métodos a serem criados, correspondem aos algoritmos capazes de realizar o cálculos de altura e de balanceamento, este último fornece informação essencial para um outro método responsável por austar a diferença de altura entre os filhos de uma árvore. O ajuste da diferença de altura se dá por meio não só de rotações simples, mas também de rotações duplas, sejam essas a direita ou a esquerda. Quanto aos métodos básicos de uma árvore aos quais foram referenciados anteriormente são: inserção, busca e remoção.

Elaboração dos métodos responsáveis por carregar e gravar arquivos: pretendendo atender ao público que encontra-se desconectado da rede mundial de computadores, a Comunidade da Batata disponibiliza em seu site os arquivos CSV que alimentam sua base de dados, esses arquivos podem ser baixados para fins de utilização *offline*, arquivos CSV são basicamente arquivos de texto cujos dados são intercalados por sinal de "ponto e vírgula".

Tendo isso em mente, o principal objetivo da aplicação desenvolvida é ter a capacidade de carregar e gravar esse tipo de arquivo, dai que então surgiu a necessidade de procurar meios de atender a essas demandas. No que diz respeito ao carregamento de arquivos, foi utilizada a classe *BufferedReader* que segundo [Deitel 2017], "permitem o armazenamento em *buffer* para fluxos baseados em caractere", ademais esse tipo de fluxo utiliza caracteres *unicode*, o que nos dá a possibilidade de trabalhar com caracteres acentuados.

Quanto ao carregamento de arquivos, outro ponto relevante é que foram utilizadas outras duas classes, sendo elas: *InputStreamReader* e *FileInputStream*. A primeira delas visa respectivamente fornecer dados ao *BufferedReader* que segundo a [Oracle 2016b], "atua como ponte de fluxos de *bytes* para fluxos em caracteres". Já a segunda classe, também segundo a [Oracle 2016a], possibilita a obtenção de *bytes* de um dado arquivo em um sistema de arquivos.

Assim, por meio desses mecanismos, a aplicação pode ser capaz de carregar sequencialmente cada linha do arquivo CSV contendo a base de dados. Ademais, cada um

dos dados intercalados por sinal de “ponto e vírgula” tiveram que ser cortados linha à linha por meio do comando *split*, visando acessa-los separadamente.

No que tange a gravação de arquivos, o processo foi semelhante ao processo de carregamento, com o diferencial que para isso foram utilizadas outras duas classes, sendo elas: *PrintWriter* e *FileOutputStream*. Quanto a *PrintWriter*, ela é basicamente uma classe que nos permite gravar caracteres em fluxo, do mesmo modo que *FileOutputStream* nos permite efetuar a criação de arquivos, como demonstrado por [Deitel 2017].

Compreensão e utilização do polimorfismo: um conceito contido na programação orientada a objetos que pode dinamizar o processo de desenvolvimento de um *software* é o polimorfismo. Em síntese, ele visa fazer com que um objeto possa comportar-se como algum outro sem a necessidade de reimplementação de código. Basicamente, o polimorfismo permite a uma subclasse herdar os métodos de uma superclasse, como demonstra [Deitel 2017].

No projeto desenvolvido, essa possibilidade polimórfica foi utilizada na implementação de um objeto que herdou da árvore AVL seus métodos, contudo essa subclasse contém em si a possibilidade de carregar e gravar arquivos. A ideia de aplicar o polimorfismo nesse ponto em específico surgiu visando reduzir o tempo contido no processo de inserir dados carregados em outra estrutura para então inseri-los na árvore.

Ademais, esse objeto conta com a disponibilidade de converter as *strings* em livros no momento da inserção de elementos na árvore, bem como, fazer o processo contrário no momento de salvar os dados da árvore em arquivo. Algo relevante nisso, é que a árvore manteve-se cumprindo apenas com seu propósito de receber qualquer tipo de dado, precisando apenas de uma chave para inseri-los, busca-los e remove-los.

Em outro momento, foi necessário criar uma estrutura capaz de comportar-se como uma *ArrayList* de *strings* dentro da aplicação, algo que constitui-se como mais um objeto polimórfico. Desse modo, essa subclasse conta com outros métodos adicionais, esses que compreendem a possibilidade de buscar por elementos, carregar e salvar dados de arquivos. Assim, foi possível atender a demanda de salvar em arquivo, os dados exibidos ao usuário que são resultantes de buscas realizadas no sistema.

3. Resultados e discussão

Seguindo o padrão de desenvolvimento de projeto MVC *(model, view, control)*, todas as classes citadas até aqui foram direcionadas ao pacote *util*, enquanto todas as demais criadas logo após essas, foram direcionadas a algum dos três pacotes do MVC. Dentro do pacote *view* foram colocadas todas as classes que manipulam informações visuais, a medida em que, a classe *Ebook* foi direcionada ao *model*, e por fim, os controladores foram direcionados ao *control*.

A classe *Application* que visa por meio do controle de outras classes realizar cada um dos requisitos foi direcionada ao pacote *control* pela razão óbvia de se tratar de um controlador. E dentro dela, são manipulados dois objetos principais: o *EbookTreeStream* classe que herda os métodos de uma árvore AVL, mas com o diferencial armazenar livros, carregar e gravar arquivos, enquanto que o outro objeto é *StringStream* que basicamente herda de uma *ArrayList* de *strings* seus métodos, mas conta também com o diferencial de buscar, carregar e gravar dados em arquivos.

Algo relevante nesse projeto, diz respeito a organização no momento de salvar arquivos resultante de buscas realizadas no sistema, esses dados são gravados em uma pasta cujo nome é *SearchResults*. Ainda que a estrutura de diretórios do Windows seja divergente da utilizada no Linux e no MAC, o sistema é capaz de perceber em qual sistema está sendo executado afim de utilizar uma estrutura de diretórios condizente com o sistema operacional, de modo a evitar falhas.

Ademais, ao cadastrar um novo livro o usuário deve digitar apenas informações referentes ao título da obra, nome do autor, mês e ano de publicação, bem como o URL onde o livro digital encontra-se disponível, quanto ao número de identificação do mesmo é gerado automaticamente pelo sistema utilizando como referência o maior número registrado.

O projeto descrito até então atende acada um dos requisitos propostos de forma clara. Ao longo dos testes, o sistema se mostrou efetivo em suas funcionalidades e de fácil uso, ademais, quanto aos testes automatizados, todas as classes foram testadas por meio de uma única classe, sendo essa correspondente ao controlador principal *Application*. Levando em conta que todos os métodos, bem como todas as classes foram projetadas afim de atender algum requisito, ao elaborar testes extensivos para cada um dos mesmos como se pede no 10º tópico do problema, pode-se afirmar que todos os métodos foram testados.

Com os testes pode-se concluir que a aplicação apresenta falhas ao carregar arquivos CSV que não sigam o padrão exibido na descrição do problema, sendo assim, linhas contendo os dados que não são esperados pode provocar falhas. Além disso, a primeira linha do arquivo onde possui descrições do que se trata cada trecho ente "ponto e vírgula" também deve ser removida.

Cada um dos testes foi elaborado visando capturar os dados contidos na última versão do arquivo CSV disponibilizado pelo tutores, logo, arquivos que não contenham os dados buscados nos testes podem provocar falhas, fazendo com que algum deles não seja aprovado. Outro ponto relevante diz respeito aos ferramentas utilizadas para a construção desse projeto, o mesmo foi construído utilizando o JDK8, juntamente com o NetBeans8.2.

4. Conclusões

Apresentada ao longo desse relatório e construída na linguagem de programação java, a solução para o problema proposto possui uma interface caracterizada pelo uso de linhas de comando. Ademais, tendo origem em arquivos de texto do tipo CSV, todos os dados trabalhados são carregados, gravados, processados e exibidos pelo programa através da memória temporária do computador.

Assim, o software construído soluciona o problema proposto utilizando-se de recursos presentes na linguagem Java. Desse modo, nos algoritmos que compreendem a solução do problema foram utilizadas classes e métodos não só que estão disponíveis na API do Java, tendo em vista que, algumas das estruturas foram criadas pelo próprio desenvolvedor.

A forma como carrega, grava, processa e exibe informações ao usuário de modo preciso, leve, detalhado e fluido constitui o que há de melhor na solução implementada. Essa que, por meio de suas funcionalidades pode constituir-se ferramenta útil ao Clube

da Batata no que diz respeito a exibição organizada das informações contidas em seus arquivos CSV.

Referências

Deitel, H. (2017). *Java: Como Programar*. Pearson.

Lafore, R. (2004). *Estrutura de Dados Algoritmos em Java*. Ciência Moderna.

Oracle (2016a). Fileinputstream java plataform. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html>. Acesso em: 14 Jan 2020.

Oracle (2016b). Inputstreamreader java plataform. Disponível em: <https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html>. Acesso em: 14 Jan 2020.