# Problema 2 : Sistema de Catalogação de Livros

**Wesley R. dos Santos**[1]

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

***Abstract.*** *This report presents the development of problem 2 offered by the MI-Programming discipline. It aims to show the whole process for problem solving, as well as the difficulties encountered and the methods to solve them encountered through trial and error besides references used.*

***Resumo.*** *Este relatório apresenta o desenvolvimento do problema 2 ofertado pela disciplina do MI-Programação. Ele tem como objetivo mostrar todo o processo para resolução do problema, bem como as dificuldades encontradas e os métodos para solucioná-las encontradas atraves de tentativa e erro alem referencias utilizadas.*

## 1. Introdução

O **projeto Gutenberg** é um projeto mantido por milhares de voluntários que digitalizam e revisam vários livros que foram previamente publicados por editoras, e disponibilizam essa versão digital gratuitamente que podem ser acessados de forma on-line, utilizando os motores de busca do Google e do Yahoo!, e também de forma off-line utilizando bases de dados gratuitas atualizadas semanalmente no formato epub, Kindle e em formato de arquivos de texto.

Um dos utilizadores desse projeto é o **clube da leitura da Comunidade da Batata** que utiliza as bases de dados off-line com arquivos de texto, porem falta otimização no processo de busca e pesquisa de livros. Para resolver esse problema foi solicitado um protótipo de um sistema de catalogação e pesquisa de livros off-line que carregar a base de dados com um arquivo **CSV**.

Além de carregar a base de dados deve ser possível inserir um novo livro, gravar novos dados no arquivo, listar todos os autores juntamente com a quantidade de obras catalogadas, listar todos os livros de um determinado autor, listar todos os livros catalogados, achar o endereço para acesso do livro através do n de e-book, exibir todos os livros de determinado ano, e por fim excluir um livro da base de dados através do n de e-book.

Com todos os requisitos em mente, e considerando que o problema exige uma estrutura de dado com complexidade media O(log n) para busca, inserção e remoção a conclusão foi que seria necessário usar a estrutura de dados conhecida como arvore de busca binaria, uma estrutura de dados que consiste em ou não tem elemento algum (árvore vazia). Ou tem um elemento distinto, denominado raiz, com dois apontamentos para duas estruturas diferentes, denominadas sub árvore esquerda e sub árvore direita.

Como todos os requisitos do sistema precisam ser passiveis de testes para evitar qualquer problema foi necessário fazer alguns testes de unidade principalmente nos métodos da arvore que é a base que sustenta todo o projeto.

## 2. Árvore binária

Como foi dito na sessão anterior a estrutura de dados que satisfaz a condição de complexidade media **O(log n)** escolhida para fazer o sistema de catalogação de livros foi a árvore de busca binária porem no pior caso a complexidade media pode chegar a **O(n)** logo ela não satisfazia completamente a condição exigida para o problema então visando a otimização a solução encontrada foi a utilização da **Árvore AVL**, que nada mais é que uma arvore de busca binaria balanceada que devido ao balanceamento da árvore, as operações de busca, inserção e remoção em uma árvore com n elementos podem ser efetuadas em **O(log n)**, mesmo no pior caso.

## 3. Arvore AVL

As árvores AVL ou balanceadas são arvores em que para qualquer nó da arvore a diferença entre as alturas das sub arvores a esquerda e a sub arvore a direita é no máximo em modulo igual a 1. Se a arvore é dita AVL então todas as suas sub arvores também são AVL, caso essa afirmação se prove falsa significa que arvore em questão não é balanceada. O fator de balanceamento de uma arvore é dada por:

**FB(nó) = H (esquerda do nó) – H (direita do nó)**

O nó balanceado tem o fator de balanceamento igual a 1, 0 ou -1. Se FB(nó) = 1, a sub arvore a esquerda está mais alta que a sub arvore a direita, se FB(nó) = 0, altura da sub arvore a esquerda igual a sub arvore a direita e se FB(nó) = -1, sub arvore a esquerda é mais alta que a sub arvore a direita. No caso em que o FB(nó) maior que 1, a sub arvore a esquerda está desbalanceado o nó e se FB(nó) menor que -1, a sub arvore a direita está desbalanceado o nó.

No caso da arvore estar desbalanceadas ocorrem as rotações para balanceá-la. Na arvore AVL existem quatro tipos de rotação, são elas:

1. Rotação a esquerda
2. Rotação a direita
3. Rotação Direita Esquerda
4. Rotação Esquerda Direita.

## 4. Implementação da Arvore AVL

A implementação da Arvore AVL para o sistema foi feita utilizando duas classes, são elas a classe **Nó** e a classe **ArvoreAVL**. Nesse sistema foram usadas 3 arvores. A arvore para as Classes **Livro** e **Data** é feita com comparação de inteiros ja a arvore para a Classe **Autor** é feita com a comparação de string usando o método **"compareToIgnoreCase"** que compara strings disconsiderando a diferença entre caixa auta e baixa. Nessa sessão sera discutida a implementação da padrão que foi usada em mais casos.

Na classe Nó existe uma variável do tipo **Livro** chamada valor que armazenara justamente os livros do banco de dados, outras duas variáveis do tipo Nó "chamadas **"ramoEsquerdo" e "ramoDireito"** que referenciarão as sub arvores e duas variáveis do tipo inteiro chamadas **"alturaE" e "alturaD"** que guardarão a altura das sub arvores a esquerda e sub arvore a direita respectivamente. É importante salientar que todos os atributos dessa classe são **privados**, fora isso existem os métodos padrões **"getters" e "setters"** e o construtor da classe.

Na classe ArvoreAVL existe um único atributo privado do tipo Nó chamado **“raiz”** a qual se refere a origem da arvore. É nessa classe que também foram implementados todos os métodos necessários para o funcionamento da arvore. Esses metodos serão tratados nas subseções seguintes.

### 4.1. Método de Inserção

O método de inserção recebe como parâmetro um no do tipo “Nó” e um valor do tipo “Livro” e funciona da seguinte forma, existe uma condicional que verifica se o nó passado for nulo ou seja n existe, o que significa que é a raiz e que a arvore esta vazia ele cria um novo nó, se não ele pega a altura dos nó anterior a esquerda e a direita, verifica se o valor passado é menor do que o valor daquele nó, se for ele verifica se o no a esquerda do no atual é diferente de nulo e se sim usa uma chamada recursiva pra chama a própria função só que dessa vez pegando o nó a esquerda do no passado anteriormente se não ele cria um novo nó local. Se o valor passado não for menor ai ele verifica se o nó a direita dele e diferente de nulo, se sim ele faz uma chamada recursiva passando dessa vez esse nó a direita, se não ele cria um novo no nessa posição. Sempre que o novo no é criado o valor passado é atribuído a esse nó.

### 4.2. Método de Busca

O método de busca recebe como parâmetro um variável do tipo Nó e uma do tipo inteiro, retorna um Livro e ela foi implementada da seguinte forma. Um novo livro é criado com todos os seus atributo recebendo uma String “0” isso foi feito porque como será citado na sessão de explicação da classe livro todos os atributos do livro são do tipo String e para a busca na arvore binaria foi utilizado a função **“parseInt” da biblioteca “Integer”** para comparar os valores passados com o valor do n de E-book do livro, caso contrário em alguns casos a busca daria erro e encerraria o programa. Após a criação do livro existe uma condicional que checa se o valor do n de E-book do livro guardado no nó é diferente do número passado ou o novo valor do n de E-book do livro criado incialmente é igual ao valor passado. Isso foi implementado assim para **evitar que ocorresse um erro de “nullPointerExcepition”**, caso a condição seja satisfeita ele vai para outra condicional que verifica se o valor do n de E-book do livro guardado no nó é igual ao da busca se sim ele substitui o livro criado incialmente pôr o livro que está nesse nó se não, se o valor do n de E-book do livro guardado no nó é menor ele faz uma chamada recursiva dessa vez passando o nó a esquerda do nó passado anteriormente e se nenhuma das outras condições anteriores forem satisfeitas ele faz uma chamada recursiva passando dessa vez o nó a direita do nó passado anteriormente. Isso se repetira enquanto a primeira condição for satisfeita, ao final ela retornara ou o Livro encontrado ou então o livro com valores 0.

### 4.3. Método de Remoção

O método de Remoção recebe como parâmetro um variável do tipo Nó e uma do tipo inteiro e retorna o nó removido, inicialmente ela se parece um pouco com o método de busca se tratando das condicionais, ela verifica se o nó passado é nulo ou seja não existe se sim ela retorna nulo caso essa condição não seja satisfeita ela faz as verificações do n de e-book com o inteiro passado assim como no método de busca se for menor faz uma chamada recursiva como o nó a esquerda do anterior e se for maior faz uma chamada recursiva com o nó a direita. Caso nenhuma das condições anteriores seja satisfeita significa

que ela achou o nó onde está o Livro que ela estava procurando então a função verifica se os nós a esquerda e a direita desse nó são nulos, satisfeita a condição ela simplesmente atribui a esse nó o valor nulo porem se a condição se mostrar falsa ela verifica se apenas um dos lados é nulo, uma condição para cada lado, se sim ela cria um novo nó e atribui a ele o valor do no encontrado depois pega o no encontrado e atribui a ele o valor do nó oposto ao nó que satisfez a condição de estar nulo. Porém se nem o nó a esquerda nem o nó a direita do encontrado for nulo ele cria um novo no e atribuiu a ele o valor do no a esquerda do no encontrado depois coloca um "While" para repetir enquanto o valor do nó criado a direita for diferente de nulo e dentro dessa repetição faz sempre o nó criado ganhar o valo do no a direita dele. Ao terminar a repetição e a condição ser satisfeita ele seta o valor do no encontrado como o valor do nó criado e o no criado ao final se torna nulo e o ramo esquerdo do no encontrado que agora tem um novo valor é setado com uma chamada recursiva do método de remoção.

### 4.4. Método de verificar existência

O método **"existe"** retorna um booleano recebe como parâmetro um Nó um inteiro e um booleano que por padrão recebe o valor "falso". Ele foi implementado usando condicionais. A primeira condição verifica se o nó passado é igual a nulo se sim, ele retorna a variável booleana que tem como padrão o valor "falso", se o nó em questão não é nulo ele vai para uma outra verificação que verifica se o valor do n de e-book do livro do nó e diferente do inteiro passado e se o valor da variável booleana é diferente de "verdadeiro", essa condição é para evitar o erro de **"NullPointerException"**, satisfeita a condição ele vai verificar se o valor do inteiro é igual que o valor do n de e-book do nó passado, caso sim ele muda o valor da variável booleana passada para "verdadeiro", caso não ele verifica se o valor do inteiro é menor ou maior do que o valor do n de e-book do nó passado, se for menor é feita uma chamada recursiva dessa vez passando o nó a direita do nó anterior se o número inteiro passado for maior a chamada recursiva passara o nó a esquerda. Quando a primeira condição não for mais satisfeita a função retorna a variável booleana passada como parâmetro.

### 4.5. Método para calcular altura da árvore

O método **"contarBalanceamento"** recebe como parâmetro dois Nós um que recebera inicialmente como valor a raiz e o outro que recebera o valor do Nó que foi inserido ou removido considere seus nomes inicial e final respectivamente. Ela também terá duas variáveis inteiras **"AD"** e **"AE"** que corresponderão a altura da sub arvore da direita e da sub arvore a esquerda respectivamente. A primeira verificação que o método faz é se o nó inicial diferente de nulo e o nó final diferente do inicial ele fara duas verificações:

A primeira, se o nó a esquerda do inicial for diferente de nulo ele somara uma unidade no valor de "AE" e depois verificara se o nó a direita da esquerda do nó inicial for diferente de nulo ele somara uma unidade em "AD" e vai "setar" esse valor de "AD" no atributo "alturaD" do nó. Encerrado essa condicional ele vai "setar" o valor de "AE" no atributo "alturaE" do nó, após isso chamara recursivamente o método passando o nó a esquerda do inicial e termina a primeira condicional.

Para a segunda condicional, se o nó a direita do inicial for diferente de nulo ele somara uma unidade no valor de "AD" e depois verificar se o nó a esquerda da direita do nó inicial for diferente de nulo ele somara uma unidade em "AE" e vai "setar" essve valor

“AE” mp atributo “alturaE” do nó. Encerrado essa condicional ele vai “setar” o valor de “AD” no atributo “alturaD” do nó, após isso chamara recursivamente o método passando o nó a direita do inicial e termina a primeira condicional.

### 4.5.1. Método para calcular o fator de Balanceamento

O método **“FB**” recebe como parâmetro um nó e retorna um inteiro. Ele faz isso pegando os valores dos atributos “alturaE” e “alturaD” e os subtrai nessa exata ordem, retornando o resultado.

## 4.6. Métodos de rotações

Os métodos de rotações só recebem como parâmetro um Nó e não possuem retorno. No total existem 4 métodos de rotação e 1 método que interliga todos eles. São eles:

### 4.6.1. Método de rotação a direita e a esquerda

O método **“rodarDireita”** funciona da seguinte forma. Além do parâmetro do tipo Nó ela tem outras duas declarações de variáveis, sendo dois Nós **q e aux**. O no q irá receber o valor da esquerda do parâmetro, o aux recebera a direita do parâmetro a esquerda do nó q será “setada” com o parâmetro e o parâmetro recebera o valor de q.

Para o método **“rodarEsquerda”** é o inverso, o que foi colocado como esquerda no método de “rodarDireita” se tornara direita e o que foi colocado como direita se tornara esquerda.

### 4.6.2. Métodos de rotações Direita/Esquerda e Esquerda/Direita

O método “rodarDE” funciona da seguinte forma. É chamado a função “rodarDireita” passando como parâmetro o nó a direita do nó principal e depois é chamada a função “rodarEsqueda” passando como parâmetro o nó principal. Já para a função “rodarED” é chamada a função “rodarEsqueda” passando como parâmetro o nó a esquerda do nó principal e depois é chamada a função “rodarDireita” passando como parâmetro o no principal.

### 4.6.3. Método Rotacionar

O método “rotacionar” funciona da seguinte forma. É atribuído a uma variável inteira chamada balanceamento a função “FB” passando como parâmetro o nó. Após isso é feito 4 condicionais para abranger todos os casos de desbalanceamento.

Se o balanceamento seja igual a 2 e a altura do sub arvore a esquerda seja menor que 0 é chamada a função “rodarED”.

Se o balanceamento for igual a -2 e a altura da sub arvore a direita for maior que 0 é chamada à função “rodarDE”.

Se o balanceamento for igual a 2 é chamada a função “rodarEsquerda”.

Se o balanceamento for igual a -2 é chamada a função “rodarDiretia”.

### 4.7. Método para calcular altura da árvore

O método **“printarArvore”** recebe como parâmetro um nó e funciona da seguinte forma. Ele possui duas condicionais. A primeira verifica se a esquerda desse nó é diferente de nulo ou seja existe, caso a condição seja satisfeita ele faz uma chamada recursiva passando dessa vez a esquerda desse nó como parâmetro. Já a segunda verifica se a direita desse nó é diferente de nulo ou seja existe, caso a condição seja satisfeita ele faz uma chamada recursiva passando dessa vez a direita desse nó como parâmetro. Por fim ele “printa” o título do Livro que está dentro do nó.

#### 4.7.1. Método Imprimir

O método **“imprimir”** é um método implementado para **facilitar** o uso do método “printarArvore” ele não recebe parâmetro e tudo que faz é uma checagem de condição, se a arvore não está vazia ele chama o método “printarArvore” passando como parâmetro a raiz se ela está vazia ele exibi uma mensagem informando isso.

### 4.8. Método para retorno da arvore em forma de lista

O método **“retornarArvore”** é um método muito parecido com o método “printarArvore” ele também tem duas condicionais “if” onde checa se a esquerda e a direita do no passado são diferentes de nulo e caso sim faz uma chamada recursiva passando a direita e a esquerda do nó como parâmetro, isso em condicionais separadas. A diferença vem em seguida porque além do nó essa função recebe também uma lista como parâmetro e após as duas condicionais ela adiciona o valor do nó na lista ou seja, seu objetivo é copiar todos os dados da arvore para uma lista, isso foi feito para facilitar a escrita e a remoção de dados no arquivo csv.

#### 4.8.1. Método listaEscrita

O método **“listaEscrita”** é um método criado para **facilitar a chamada do método “retornarArvore” no controller**, pois com ele só tenho que me preocupar com a passagem de um parâmetro que é a lista já que dentro dele já e chamado o método “retornarArvore” passando como parâmetro a “raiz” da arvore e a própria lista passada como parâmetro da função da “listaEscrita”.

### 4.9. Informações adicionais

Essa implementação de arvore balanceada foi feita **exclusivamente** para esse sistema de catalogação de livros logo sem algumas alterações nos atributos dos métodos ela não funcionara em nenhum outro sistema.

É importante salientar que para os métodos de inserção, remoção, busca e o de verificar existência foram utilizado um conceito de polimorfismo chamada sobrecarga mais especificamente sobrecarga de tipo onde existem dois métodos com nomes iguais e a única coisa que os difere é o tipo da variável.

Por fim sobre o balanceamento, as funções “contarBalanceamento” e “rotacionar” sempre são chamadas, **nessa exata ordem**, após a inserção e a remoção justamente com o intuito de manter a arvore e suas sub arvores sempre balanceadas.

## 5. Classe presentes no pacote model

No pacote model existem três classes, são elas as classes “Livro”, “Autor” e “Data”.

A **classe livro** contém os atributos para armazenar o n de e-book, o título do livro, o nome do autor, o mês de lançamento, o ano de lançamento e o link para acesso do livro.

A **classe Autor** possui somente dois atributos que são o nome do autor e uma lista de livros que armazenara todos os livros daquele autor. Ela foi pensada assim para facilitar a implementação do requisito 5 do sistema que exigia que dado um autor todos os livros publicados por ele precisam ser listados.

A **classe Data** assim como a classe Autor possui dois atributos, são eles, a data de publicação e uma lista de livros que é utilizadas para guardar todos os livros publicados naquele anos especifico, novamente como a classe Autor ela foi implementada pensando em facilitar um dos requisitos do sistema nesse caso o requisitos 8 que pedia que dado o ano de publicação o livro juntamente como todas as suas informações precisam ser listados.

Fora os atributos as três classes possuem um construtor, e os métodos padrões “getters” e “setters”.

Pode-se observar a relação entre as classes na figura a seguir. (figura1)

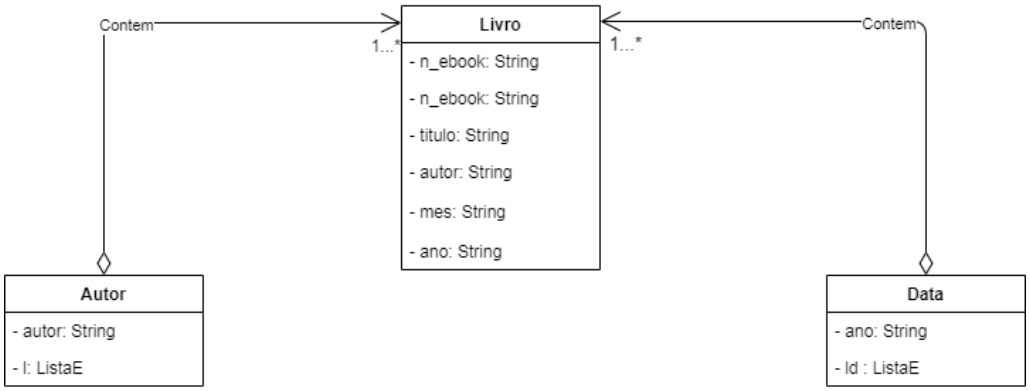


**Figure 1. Modelo conceitual**

## 6. Pacote Controller

Nesse pacote existe uma única classe chamada **”database”**, é esta classe na qual foi implementado todas as funcionalidades e requisitos do sistema. Nele tambem esta a base de dados em formato de arquivo **”CSV”**, o qual contem todos os livros do sistema.Nessa pasta também ficarão salvas todas as buscas realizadas pelo sistema em formato de arquivos de texti com o nome da busca como exigido pelo problema

. É importante salientar que em vários momentos você se deparara com a frase “verificar se existe”. A fim de sanar qualquer dúvida essa frase se refere a função “existe” que existe dentro da estrutura de dados conhecida como Arvore de busca AVL que

foi implementada nesse sistema. Esse método já foi tratado durante a explicação da implementação da arvore.

Todos os métodos que envolverem leitura, escrita e remoção de dados arquivo terão ao lado de suas declarações um comando chamado “IOException” que será explicado a seguir.

### 6.1. Classe database

A classe **“database”** por ser um **controller** não possui atributos nem métodos construtores porem possuem duas Arvores uma arvore para armazenar os objetos do tipo “Autor” e outra para os objetos do tipo “Data”. Para cumpri todos os requisitos exigidos para o sistema essa classe possui onze métodos os quais serão tratados nas subseções a seguir. Como sera informado nas

#### 6.1.1. Metodo para ler o banco de dados de texto

A função **“lerArquivo”** é uma função sem retorno que recebe como parâmetro uma arvore AVL. Por usar algumas funções do pacote IO do Java ao lado do nome da função é preciso digita o comando **“throws IOException”** para o método funcionar e por consequência qualquer outra função que chamar o método “lerArquivo” também precisara ter o comando.

Partindo agora para o funcionamento, primeiramente é instanciando um **“FileInputStream”**, basicamente o que é, dado um endereço no primeiro espaço de parâmetros ele vai procurar o arquivo se existir ele abre, se não ele criara o arquivo, o segundo parâmetro, caso o objetivo seja escrita e não leitura, onde você colocara **“true”** ou **“false”**, informa caso o arquivo já exista se é pra concatenar a nova informação ou sobrescrever respectivamente, se o espaço for deixado em branco a biblioteca assume que é para sobrescrever.

Como o objetivo dessa função é apenas leitura de arquivo foi instanciado um **“inputStreamReader”** com o nome de **“f”** e depois um **“BufferedReader”** chamado de **ler** recebendo o f como parâmetro. Além disso existem dois contadores chamados **“cont”** e **“j”**, sendo eles usados para pular as linhas desnecessárias e verificar qual dado está sendo lido no momento por uma outra variável chamada linha que esta percorrendo com um cursor a linha do arquivo respectivamente. O “j” tem o limite de 5 pois existem 6 tipos de dados dentro do arquivo e como o laço de repetição **“while”** foi iniciado a partir do 0 significa que ao chegar no 5 o cursor já leu todos os tipos de dados daquela linha. Isso se repetirá até a linha lida ser nula ou seja ao fim do arquivo e a cada repetição será criado um novo livro com aquelas informações. Esse livro então passara por uma verificação de cada um de seus dados feitos por uma estrutura de repetição “**if”** para saber se todos os dados foram preenchidos corretamente na sessão anterior, se sim ele será adicionado a arvore de livros e passara por uma verificação de autor e data se já existir o autor e a data cadastrados esses autores e datas serão procurados na arvore de “Autor” e “Data” e esse livro será adicionado a cada uma de suas listas mas se não existir ambos os itens ou um dos dois, os itens inexistentes serão criados e adicionados as suas respectivas arvores. Caso o livro não tenha todas as informações ele simplesmente não é cadastrado.

### 6.1.2. Adicionando livro a Base de dados

O método para adição de livros recebe como parâmetro uma ArvoreAVL, retorna um Livro e é bem simples, ele basicamente pede todos os dados predeterminado do livro verifica se já existe. Se o livro já está cadastrado ele ignora e mostra uma mensagem se não ele cria um novo livro adiciona na arvore e depois checa se o autor e a data de publicação já existe no sistema se sim ele procura em suas respectivas arvores e adiciona o livro a lista das classes "Autor" e "Data se não existir alguma delas ele cria uma nova instancia do tipo que não existe e o adiciona à sua respectiva arvore.

### 6.1.3. Função para escrever dados do livro na base de dados de texto

Essa função recebe como parâmetro uma ArvoreAVL e não tem retorno. Dentro dela é chamada a função "adicionarLivro" que foi explicada na subseção anterior. Como ela utiliza elementos da biblioteca IO ele teve que usar o mesmo comando que foi utilizado na função de leitura, a única diferença é que dessa vez foi utilizado o "BufferedWriter" que recebeu o nome de "escrever". Para escrever no arquivo usamos o comando append passando como parâmetro os métodos "get" da classe "Livro" concatenado e separados por ponto e vírgula para manter o padrão do arquivo CSV.

### 6.1.4. Função para grava todas as buscas feitas por sistema em arquivos de texto com o padrão CSV

A função "escreverBusca" é análoga a função "escrever" com a única diferença que em vez de receber uma ArvoreAVL ela recebe duas strings como parâmetro, respectivamente uma com o nome da busca e a outra com o conteúdo da busca. Ela tambem ultiliza um StringBuilder para concatenar o endereço com o nome da busca. Isso foi utilizado porque concatenação simples utilizando o sinal "+" estava causando erro em 3 dos métodos de busca ao chamar essa função.

### 6.1.5. Funções para buscar objetos do tipo Autor e Data nas suas respectivas Arvores AVL

Esses dois métodos serão explicados juntos porque na verdade são o mesmo método só que com tipos de retornos e o tipo de parâmetro diferentes. O funcionamento dele se baseia em checar se existe ou não o parâmetro recebido dentro das suas respectivas arvores AVL. Se existir ele procura e retorno o objeto se não exibi uma mensagem de erro.

### 6.1.6. Função para imprimir todos os autores junto com a quantidade de livros cadastrados

Essa função não tem parâmetro nem retorno, dentro dela é instanciado uma lista, depois é chamado um método exclusivo da arvore que guarda os objetos do tipo Autor. Esse método funciona assim. Dado a arvore de autores ele a percorre e os adiciona a lista que e passada como parâmetro pra função. Após isso existe um laço de repetição **"for"**

que percorre até o fim da lista obtendo autor por autor e exibindo o tamanho de suas respectivas listas de livro e chama a função **“escreverBusca”** passando como parâmetro o nome da função e o que foi exibido para criar ou alterar um dos arquivos de histórico de busca.

### 6.2. Função que lista os livros publicados pelo Autor

Essa função basicamente lê o nome do autor e o armazena em uma variável do tipo **“String**” que é passada como parâmetro para a função **“buscarAutor”** que é chamada dentro dela, guarda o retorno dessa função em uma variável do tipo “Autor”. Após isso pega a lista de livros do autor encontrado e coloca um laço de repetição para percorrer até o fim da lista obtendo os livros e exibindo seus nomes, ela também chama a função **“escreverBusca”** passando como parâmetro o nome do autor informado pelo usuário e o nome do livro. Depois fora da repetição chama a função de escrita de novo só que dessa vez passando em vez do nome do livro uma quebra de linha”.

### 6.3. Função que lista os livros publicados em determinado ano

Essa função basicamente lê a data informada pelo usuário e chama a função **“buscar-Data”** passando como parâmetro a data pesquisada. Ao achar pega a lista dessa data coloca para um laço de repetição percorrer e exibi todas as informações. Ela também chama a função **“escreverBusca”** passando como parâmetro a data e as informações do livro.

### 6.4. Função para remoção de livro

Dado o **n de E-book** do livro ele verifica se o livro existe, caso sim ele busca esse livro, na arvore de livros, pega o autor e o ano de publicação do livro e passa como parâmetro para as funções de busca de autor e data respectivamente e os armazena em variáveis. Depois pega a lista de livros daquele autor e daquela data para então percorre-las até chegar no livro para exclui-lo das duas listas. Somente após isso ele pega esse livro e exclui da arvore de livros pra ai então apagar toda a base de dados de texto e a reescrever percorrendo a arvore de livros.

### 6.5. Buscar link de acesso do livro pelo n de E-book

Essa é uma função muito simples, pois tudo que precisou ser feito é pedir o **n de E-book** ao usuário verificar se existe um livro com esse n de E-book na arvore se sim chamar a função **“buscarLivro”** passando como parâmetro o valor informado pelo usuário e mandar exibir o link de acesso do livro. Ela também chama a função **“escreverBusca”** passando como parâmetro o n de E-book e as informações do livro.

## 7. Classes de teste

Como requisitado no problema para cada item do sistema foram feitos 3 testes usando o Junit. Os métodos testados foram os métodos: **“ler arquivo”** que contém o requisito 2 do sistema, **”escrever”** que compreende os requisitos 1 e 3, o método **“autorQuantidade”** referente ao requisito 4, o método **“autorLivro”** referente ao requisitos 5, o requisitos 6 referente a listagem de livro está implícito no teste do requisitos 2 então não foi implementado separadamente. O requisitos 7 encontrasse já testados em todos os outros requisitos já que seu uso foi necessário para algumas funcionalidades, o requisitos oito

realizado pelo método **"livroAno"** também foi testado assim como o requisitos nove que na implementação recebeu o nome de **"remover"**. Os teste foram implementados de forma a checar se os livros estavam sendo totalmente carregados e excluídos da base de dados, e se havia algum retorno nulo durante o processo. Todos os testes passaram.

## 8. Conclusão

O sistema de catalogação de livros está funcionando perfeitamente desde que as informações apresentadas sejam validas e o arquivo passado siga o padrão CSV. Como é um piloto ele não é capaz de filtrar todo o tipo de dado e dependendo do endereço que a base de dados estejam no computador do usuário será necessário ir diretamente no código fonte e alterar o loca no método de leitura de escrita e de remoção, é importante salientar que apesar do método de busca está funcionando na velocidade esperada no momento de carregar o arquivo pode demorar um pouco entre as operações de leitura, impressão e inserção na arvore isso só no primeiro momento quando a base de dados é carregada pela primeira vez. Também é valido informar que o arquivo referente a base de dados precisa ter as duas primeiras linhas em branco ou com alguma informação que não seja necessária pois como o arquivo enviado tinha a primeira linha só com o nome de cada informação e a segunda em branco o projeto piloto seguiu esse padrão na sua implementação.

Os testes automatizados foram realizados mas devido a alguns problemas enfrentados foi constatado que ele só está funcionando com a versão 4.12 do JUnit por algum motivo não descoberto a versão mais recente não rodou absolutamente nenhum tipo de teste mesmo de projetos diferentes desse. Esse foi o único problema real enfrentado durante a implementação mas foi resolvido.

Só foram utilizadas referencias na hora da implementação da arvore, ela foi implementada baseadas nas informações dadas no **PDF** fornecido de forma online por [Alves ].

## References

Alves, P. R. Árvores avl.