Sistema de Organização e Busca de Livros Usando Árvores AVL

Ryann Flávyo Alves Honorato Lessa

13 Junho de 2025

1 - Introdução

1.1 - Objetivo

O objetivo principal deste projeto é desenvolver um **sistema de organização e busca de livros** utilizando uma **árvore AVL** para manter os dados organizados de forma eficiente e balanceada. O sistema permitirá a inserção, remoção e busca de livros, com cada livro sendo representado por atributos como **título**, **autor** e **ISBN** (ou um identificador único).

Ao adotar a **árvore AVL**, o sistema se beneficia da estrutura de **árvore binária de busca balanceada**, que garante que as operações de busca, inserção e remoção sejam feitas de maneira rápida e eficiente, com **complexidade O(log n)** para cada operação. A árvore AVL mantém um equilíbrio entre suas subárvores, o que é essencial para evitar a degradação de desempenho que ocorre em árvores binárias não balanceadas, onde a complexidade de busca pode se aproximar de **O(n)** em cenários desfavoráveis.

Além disso, o sistema será capaz de **exibir os livros** em ordem crescente de **ISBN** ou qualquer outro critério que o usuário desejar, utilizando o mecanismo de percorrimento em ordem da árvore AVL. O projeto também será modular, o que permitirá futuras expansões, como a adição de novas funcionalidades e a inclusão de filtros adicionais para busca, como **pesquisa por autor** ou **por título**.

1.2 - Motivação

A motivação por trás deste projeto está ligada à crescente necessidade de **organização e eficiência no gerenciamento de grandes volumes de dados**, especialmente em sistemas que envolvem **catálogos**, **bibliotecas** ou **base de dados de livros**. Em um cenário onde a quantidade de dados continua a crescer, é fundamental ter **estruturas de dados** que possibilitem **acesso rápido** e **eficiente** às informações. Árvores binárias de busca são comumente utilizadas para organizar dados de forma ordenada, porém, sem

mecanismos de balanceamento, elas podem sofrer com a **degradação de desempenho** quando a árvore se torna desequilibrada.

O uso da **árvore AVL** resolve esse problema, pois ela garante que a altura das subárvores de qualquer nó nunca difira em mais de 1, mantendo a árvore balanceada. Isso significa que, mesmo com **um grande número de livros**, o tempo de busca, inserção e remoção de registros permanece **logarítmico**, o que é essencial para sistemas de **alto desempenho**. A árvore AVL é particularmente útil em **ambientes que exigem agilidade nas buscas** e **alta eficiência no processamento de dados**, como **sistemas de gerenciamento de bibliotecas**, **catálogos de produtos** e **aplicações de leitura digital**.

Além disso, muitas soluções tradicionais de organização de dados (como **listas encadeadas** ou **árvores binárias simples**) não oferecem o mesmo nível de eficiência. Elas podem sofrer com a **ineficiência** nas operações quando a quantidade de dados aumenta significativamente, resultando em **tempo de execução linear**. O uso de uma árvore AVL como estrutura subjacente, por sua vez, oferece um **processamento rápido**, especialmente quando o volume de dados é alto, como é o caso de **catálogos de livros** com centenas ou milhares de entradas.

A motivação deste projeto, portanto, está em criar uma solução simples, mas **eficiente e escalável**, para o armazenamento e consulta de livros, e também em explorar a implementação de **estruturas de dados balanceadas** para resolver um problema prático de organização de dados, com benefícios em termos de **desempenho** e **manutenibilidade** do sistema.

2 - Organização do Projeto

2.1 - Estrutura Modular

O projeto foi estruturado de forma modular para facilitar a manutenção e a expansão futura. Os principais módulos são:

📁 arvore_avl.py

Propósito: Implementa a árvore AVL e suas operações (inserção, remoção, busca, balanceamento).

Fundamento Teórico: A árvore AVL é uma árvore binária de busca onde a altura das subárvores esquerda e direita de qualquer nó difere em no máximo 1. Esse balanceamento garante que as operações de busca sejam realizadas em tempo logarítmico.

Implementação: O módulo contém a classe **NO** (representando um nó da árvore) e a classe **ArvoreAVL** (representando a árvore AVL). A inserção, remoção e balanceamento de nós são realizados para garantir a estrutura balanceada da árvore. Este módulo não interage diretamente com o arquivo **livros.json**, mas manipula os dados da árvore, que posteriormente são salvos.

📁 livro.py

Propósito: Define a classe **Livro**, que contém os atributos título, autor e ISBN. Esses atributos são usados para armazenar as informações básicas sobre um livro.

Fundamento Teórico: Cada livro é representado por um objeto da classe **Livro**, com atributos que facilitam a organização e a busca. O uso dessa estrutura orientada a objetos permite encapsular as informações de cada livro, proporcionando uma maneira eficiente de manipular e organizar os dados na árvore AVL.

Implementação: A classe **Livro** é simples, com atributos de dados e métodos para manipulação de livros. A interação com o **livros.json** é feita por meio das funções de leitura e escrita presentes em **livros_class.py**, responsáveis por carregar e salvar os dados no arquivo.

📁 sistema_busca.py

Propósito: Implementar a busca eficiente de livros na árvore AVL.

Fundamento Teórico: A busca é feita utilizando o ISBN como chave. Como a árvore é balanceada, a busca é eficiente, realizando comparações de forma logarítmica.

Implementação: A função **buscar_livro** realiza a busca de livros por ISBN, título e autor na árvore AVL. Quando o usuário realiza uma busca, o sistema verifica a árvore e, se o livro for encontrado, retorna os dados, com destaque para o ISBN, título e autor. Este módulo não lida com a persistência diretamente, mas pode ser utilizado para realizar buscas enquanto os dados são carregados de **livros.json**.

📁 main.py

Propósito: Este módulo é responsável por gerenciar a interação com o usuário. Ele permite que o usuário insira livros manualmente, visualize todos os livros armazenados na árvore ou busque por livros usando ISBN, título ou autor.

Fundamento Teórico: O módulo utiliza a árvore AVL (implementada em **arvore_avl.py**) para realizar a inserção, remoção e busca de livros. A interação com o usuário é feita através da linha de comando.

Implementação: O módulo fornece um menu de opções para o usuário, e dependendo da escolha, chama as funções apropriadas para inserir, exibir, buscar ou excluir livros. Além disso, o sistema garante que os livros sejam carregados do arquivo **livros.json** ao iniciar o programa e que qualquer alteração seja salva no arquivo JSON após cada modificação.

2.2 - Execução do Sistema

A execução do sistema é realizada via linha de comando, onde o usuário pode inserir novos livros, buscar livros ou visualizar todos os livros armazenados na árvore. O módulo **main.py** gerencia a interação com o usuário e a manipulação da árvore AVL. O fluxo interno de execução do sistema é o seguinte:

- **O módulo sistema_busca.py** interpreta os parâmetros passados pelo usuário para buscar livros por ISBN, título ou autor. O sistema permite que o usuário realize buscas detalhadas, retornando os livros encontrados com base nos critérios escolhidos.

- **O módulo arvore_avl.py** executa a operação solicitada (inserir, remover ou buscar) e atualiza a árvore AVL conforme necessário, realizando operações eficientes com a árvore balanceada.

- **O módulo livro.py** gerencia a criação e manipulação dos dados dos livros, garantindo que cada livro tenha os atributos necessários (título, autor e ISBN) e seja manipulado corretamente nas operações de inserção e remoção.

- **O módulo livros.json** é utilizado para persistir os dados dos livros. O arquivo **livros.json** armazena os livros inseridos, e o sistema carrega os dados ao iniciar, utilizando a função **carregar_livros()**. Sempre que um novo livro é inserido ou excluído, o arquivo JSON é atualizado com a função **salvar_livros()**, garantindo que as alterações sejam salvas permanentemente.

3 - Funcionamento Técnico

3.1 - Estrutura de Dados Utilizadas

O sistema é baseado em duas estruturas de dados principais, que garantem a eficiência nas operações de inserção, remoção e busca dos livros:

- **Árvore AVL**: A árvore AVL é a estrutura fundamental utilizada para armazenar e organizar os livros no sistema. Trata-se de uma árvore binária de busca balanceada, onde a altura das subárvores de qualquer nó não difere em mais de 1. Isso garante que as operações de busca, inserção e remoção sejam realizadas de forma eficiente, com complexidade **O(log n)**, independentemente da ordem de inserção dos dados. Ao manter a árvore balanceada, o sistema evita a degradação de desempenho que pode ocorrer em árvores binárias não balanceadas.

    - **Inserção e Remoção**: A árvore AVL permite a inserção e remoção de livros de forma eficiente. Quando um livro é inserido, o sistema percorre a árvore para encontrar o local adequado e, se necessário, realizar rotações para manter o balanceamento. Da mesma forma, ao remover um livro, o sistema reequilibra a árvore, garantindo que as operações subsequentes permaneçam rápidas.

    - **Busca**: A busca de livros na árvore AVL é realizada de forma eficiente, onde o ISBN do livro é usado como chave. O sistema realiza a busca de forma recursiva, comparando o ISBN do nó atual com o ISBN do livro desejado, movendo-se para a subárvore esquerda ou direita conforme necessário. Essa abordagem garante que a busca seja concluída em tempo logarítmico, mesmo em árvores com um grande número de livros.

- **Objetos Livro**: Cada livro é representado como um objeto da classe **Livro**, que encapsula os dados principais de cada livro, como título, autor e ISBN. A classe **Livro** permite que os dados dos livros sejam manipulados de maneira organizada e estruturada, facilitando as operações de inserção e remoção dentro da árvore AVL.

    - **Atributos**: Os livros possuem três atributos principais:

        - **Título**: Representa o nome do livro.

        - **Autor**: Representa o autor do livro.

        - **ISBN**: Serve como identificador único do livro, garantindo que cada entrada na árvore AVL seja distinta e fácil de localizar.

    - **Armazenamento de Livros**: A lista predefinida de livros no início do sistema é carregada do arquivo JSON (livros.json), fornecendo dados iniciais para o funcionamento do sistema. Os livros carregados são inseridos na árvore AVL, mas o sistema também permite que o usuário adicione novos livros manualmente, que serão armazenados tanto na árvore quanto no arquivo JSON.

- **JSON para Persistência de Dados**: Para garantir que os dados não se percam ao fechar o sistema, os livros são armazenados em um arquivo JSON (**livros.json**). O sistema carrega os livros ao ser iniciado e os salva automaticamente quando novas inserções ou remoções de livros ocorrem. Isso garante persistência de dados, permitindo que o sistema mantenha o histórico dos livros ao longo de várias execuções.

3.2 - Algoritmo de Balanceamento

O algoritmo de balanceamento da árvore AVL é fundamental para garantir a eficiência da estrutura, mantendo a diferença de altura entre as subárvores esquerda e direita de qualquer nó em no máximo 1. Esse balanceamento é necessário para que as operações de inserção, remoção e busca sejam realizadas em tempo logarítmico, ou seja, O(log n). O balanceamento é feito por meio de rotações, que são aplicadas dependendo do tipo de desbalanceamento identificado após a inserção ou remoção de um nó.

Existem **quatro tipos principais de rotação** utilizados para restaurar o balanceamento da árvore:

1. **Rotação Simples à Direita (Direita Simples)**:

   - Este tipo de rotação é aplicado quando há um desbalanceamento causado pela inserção na **subárvore esquerda** do nó esquerdo de um nó. O desbalanceamento é denominado "Esquerda-Esquerda" (LL).

   - **Como funciona**: A rotação à direita promove a troca entre o nó atual e seu filho esquerdo. O filho esquerdo do nó se torna a nova raiz, e o nó atual passa a ser o filho direito dessa nova raiz.

2. **Rotação Simples à Esquerda (Esquerda Simples)**:

   - Este tipo de rotação é aplicado quando há um desbalanceamento causado pela inserção na **subárvore direita** do nó direito de um nó. O desbalanceamento é denominado "Direita-Direita" (RR).

   - **Como funciona**: A rotação à esquerda promove a troca entre o nó atual e seu filho direito. O filho direito do nó se torna a nova raiz, e o nó atual passa a ser o filho esquerdo dessa nova raiz.

3. **Rotação Dupla à Direita (Esquerda-Direita)**:

- Este tipo de rotação é aplicada quando o desbalanceamento ocorre devido à inserção na **subárvore direita** de um nó esquerdo. Esse desbalanceamento é denominado "Esquerda-Direita" (LR).

- **Como funciona**: Primeiro, realiza-se uma rotação à esquerda no nó esquerdo, para transformar o desbalanceamento "Esquerda-Direita" em um "Esquerda-Esquerda". Em seguida, aplica-se uma rotação simples à direita sobre o nó desbalanceado.

4. **Rotação Dupla à Esquerda (Direita-Esquerda)**:

- Este tipo de rotação é aplicada quando o desbalanceamento ocorre devido à inserção na **subárvore esquerda** de um nó direito. Esse desbalanceamento é denominado "Direita-Esquerda" (RL).

- **Como funciona**: Primeiramente, realiza-se uma rotação à direita no nó direito, para transformar o desbalanceamento "Direita-Esquerda" em um "Direita-Direita". Em seguida, aplica-se uma rotação simples à esquerda sobre o nó desbalanceado.

Aplicação das Rotações

As rotações são aplicadas após a inserção ou remoção de um nó. O algoritmo de balanceamento calcula o fator de balanceamento de cada nó, que é a diferença entre a altura das subárvores esquerda e direita. Dependendo do valor desse fator, uma das rotações descritas acima é escolhida para restaurar o equilíbrio da árvore.

- **Fator de balanceamento**:

  - Se o fator de balanceamento de um nó for **maior que 1**, significa que a subárvore esquerda está mais alta do que a direita, indicando que uma rotação à direita ou dupla à direita deve ser realizada.

- Se o fator de balanceamento de um nó for **menor que -1**, significa que a subárvore direita está mais alta, e uma rotação à esquerda ou dupla à esquerda será aplicada.

Essas rotações garantem que a árvore AVL permaneça balanceada após qualquer operação, assegurando que o tempo de execução das operações de busca, inserção e remoção se mantenha sempre logarítmico, ou O(log n).

3.3 - Fluxo de Operação

O fluxo de operação do sistema é intuitivo e eficiente, e segue uma sequência lógica de ações para garantir a correta inserção, remoção, busca e exibição de livros na árvore AVL. Abaixo está uma explicação detalhada de como as operações principais são realizadas.

Inserção de um Livro

A inserção de um livro na árvore AVL ocorre de forma recursiva e balanceada. O processo segue os seguintes passos:

1. **Verificação do Nó Raiz:**
   - A função de inserção começa verificando se a árvore está vazia (i.e., se a raiz é `None`).
   - Se a árvore estiver vazia, um novo nó (representando o livro) é criado e se torna a raiz da árvore.

2. **Inserção Recursiva:**
   - Se a árvore já contiver livros, a função percorre a árvore de maneira recursiva, comparando o ISBN do livro a ser inserido com o ISBN dos nós da árvore.
   - Dependendo dessa comparação:

- Se o ISBN do livro for menor que o do nó atual, o livro será inserido na subárvore esquerda.

- **C**aso contrário, o livro será inserido na subárvore direita.

3. **Balanceamento da Árvore:**

- Após a inserção de um novo livro, o algoritmo verifica se a árvore precisa ser balanceada.

- O balanceamento é realizado calculando o fator de balanceamento de cada nó, que é a diferença entre as alturas das subárvores esquerda e direita.

- Se o fator de balanceamento for maior que 1 ou menor que -1, isso indica que a árvore está desequilibrada. Nesse caso, uma rotação (simples ou dupla) é realizada para restaurar o equilíbrio da árvore.

4. **Retorno do Nó Balanceado:**

- Após a inserção e o balanceamento, a árvore retorna o nó raiz (ou a subárvore) balanceada, garantindo que as operações subsequentes continuem eficientes.

4 - Eficiência e Escalabilidade

A árvore AVL é projetada para oferecer um desempenho eficiente, mesmo com grandes volumes de dados. As operações de inserção, remoção e busca possuem complexidade **O(log n)**, onde **n** representa o número de livros armazenados na árvore. Isso assegura que o sistema consiga lidar com grandes volumes de livros de forma eficiente e com alta performance, independentemente do crescimento do número de entradas.

O balanceamento da árvore é a chave para esse desempenho: ele mantém a árvore balanceada ao garantir que a diferença de altura entre as subárvores esquerda e direita de qualquer nó não ultrapasse 1. Esse equilíbrio é o principal responsável por evitar a degradação de desempenho que pode ocorrer em árvores binárias não balanceadas, onde as operações podem se tornar lentas e chegar a uma complexidade **O(n)** em cenários desfavoráveis.

Esse balanceamento contínuo permite que as operações de busca, inserção e remoção sejam realizadas de forma logarítmica, mesmo quando o número de livros cresce de forma substancial.

4.1 - Eficiência

O balanceamento dinâmico da árvore AVL garante que as operações principais - **inserção**, **remoção** e **busca** - sempre mantenham desempenho logarítmico, independentemente da ordem das inserções. Com isso, mesmo em cenários onde o sistema gerencia milhares ou até milhões de livros, o tempo de execução das operações continua sendo baixo, resultando em alta eficiência.

Ademais, a estrutura da árvore permite realizar buscas eficientes por **ISBN**, **título** ou **autor**, aproveitando a ordem dos nós. A ordem crescente dos elementos na árvore possibilita que o sistema retorne rapidamente os livros conforme o critério escolhido, sem a necessidade de operações adicionais de ordenação ou filtragem. Isso também melhora a performance, especialmente em buscas recorrentes ou filtros simples.

4.2 - Escalabilidade

A escalabilidade do sistema é outra vantagem importante da árvore AVL. À medida que o número de livros aumenta, o sistema continua a operar de forma eficiente, mantendo a árvore com profundidade mínima e evitando que o tempo de execução das operações de busca, inserção e remoção se degrade. Isso garante que, mesmo com grandes volumes de dados, o sistema continue ágil e sem perda de performance. O balanceamento constante ajusta a árvore de forma dinâmica, garantindo que as operações se mantenham eficientes mesmo à medida que o sistema se expande.

Além disso, a estrutura modular do sistema facilita a adição de novas funcionalidades, como a implementação de buscas mais complexas ou a adição de novos critérios de pesquisa. A modularidade também torna o sistema facilmente extensível, permitindo melhorias como a integração com bancos de dados, que poderiam melhorar o armazenamento e a recuperação de grandes volumes de dados, ou a implementação de filtros de busca mais avançados.

4.3 - Limitações Atuais

Embora a árvore AVL seja eficiente, o sistema atual ainda apresenta algumas limitações que podem afetar tanto a performance quanto a funcionalidade, dependendo do cenário.

Armazenamento de Dados: Atualmente, os dados dos livros são mantidos em memória, o que pode se tornar um problema conforme a quantidade de livros aumenta. Quando o número de livros cresce significativamente, a memória necessária para armazená-los também aumenta, impactando a performance do sistema. Para lidar com grandes volumes de dados, como centenas de milhares de livros, seria necessário

integrar o sistema a uma solução de armazenamento persistente, como **MySQL**, **SQLite** ou **MongoDB**, o que permitiria maior eficiência e durabilidade.

Custo Computacional em Árvores Grandes: Apesar da complexidade **O(log n)** para as operações, em árvores AVL muito grandes, o custo computacional para realizar o balanceamento pode aumentar. A cada inserção ou remoção, o processo de recalcular o balanceamento e realizar as rotações necessárias pode tornar-se um fator limitante. Esse impacto é particularmente relevante quando a árvore cresce para milhões de nós, o que pode afetar o desempenho em cenários de alta carga.

Falta de Interface Gráfica de Usuário (GUI): O sistema, atualmente, é operado via linha de comando, o que pode ser um obstáculo para usuários não técnicos. A implementação de uma interface gráfica de usuário (GUI) melhoraria significativamente a experiência do usuário, tornando as interações mais intuitivas e acessíveis. Isso seria especialmente importante para facilitar a utilização do sistema por um público mais amplo, incluindo aqueles que não têm familiaridade com comandos de terminal.

Limitações na Busca Avançada: Embora o sistema permita buscas simples por ISBN, título e autor, ele não oferece suporte para pesquisas mais avançadas, como por **gênero** ou **ano de publicação**. Essas limitações podem ser superadas por meio de indexação adicional ou a utilização de outras estruturas de dados auxiliares, como **árvores B** ou **tabelas hash**. Implementar essas funcionalidades permitiria consultas mais rápidas e específicas, tornando o sistema mais robusto e flexível.

5 - Explicação do Código

5.1 - Código da Classe ArvoreAVL

```
class ArvoreAVL:
    def __init__(self):
        self.raiz = None
```

Linha 1: Definição da classe ArvoreAVL, que representa a árvore AVL.

Linha 2-3: O construtor (__init__) é chamado sempre que uma instância da árvore AVL é criada. Aqui, ele inicializa o atributo **raiz** como **None**, o que significa que a árvore começa vazia.

```
def altura(self, no):
    if not no:
        return 0
    return no.altura
```

Linha 4-6: Função **altura** que calcula a altura de um nó da árvore.

Linha 5: Se o nó for **None**, significa que não existe esse nó, então retorna **0**.

Linha 6: Caso contrário, retorna o valor de **altura** do nó, que já foi definido e mantido atualizado durante as operações de inserção/remover.

```
def balanceamento(self, no):
    if not no:
        return 0
    return self.altura(no.esquerda) - self.altura(no.direita)
```

Linha 7-9: Função **balanceamento** que calcula o fator de balanceamento de um nó.

Linha 8: Se o nó for **None**, o fator de balanceamento é .

Linha 9: Retorna a diferença entre a altura da subárvore esquerda e da subárvore direita do nó. Esse valor é fundamental para saber se a árvore precisa de balanceamento.

```
def rotacao_direita(self, raiz):
    nova_raiz = raiz.esquerda
    raiz.esquerda = nova_raiz.direita
    nova_raiz.direita = raiz
    raiz.altura = 1 + max(self.altura(raiz.esquerda), self.altura(raiz.direita))
    nova_raiz.altura = 1 + max(self.altura(nova_raiz.esquerda), self.altura(nova_raiz.direita))
    return nova_raiz
```

Linha 10-16: Função **rotacao_direita** que realiza uma rotação simples à direita em um nó para balancear a árvore.

Linha 11: O novo nó raiz será o filho esquerdo do nó atual (o que será rotacionado).

Linha 12: O filho esquerdo do nó atual (raiz) se torna o filho direito da nova raiz.

Linha 13: A nova raiz recebe o nó atual como filho direito.

Linha 14-15: As alturas dos nós são atualizadas após a rotação.

Linha 16: Retorna a nova raiz da subárvore, que agora é o nó que estava à esquerda da raiz anterior.

```
def rotacao_esquerda(self, raiz):
    nova_raiz = raiz.direita
    raiz.direita = nova_raiz.esquerda
    nova_raiz.esquerda = raiz
    raiz.altura = 1 + max(self.altura(raiz.esquerda), self.altura(raiz.direita))
    nova_raiz.altura = 1 + max(self.altura(nova_raiz.esquerda), self.altura(nova_raiz.direita))
    return nova_raiz
```

Linha 17-23: Função **rotacao_esquerda** que realiza uma rotação simples à esquerda, balanceando a árvore.

Linha 18: O novo nó raiz será o filho direito do nó atual (raiz).

Linha 19: O filho direito do nó atual (raiz) se torna o filho esquerdo da nova raiz.

Linha 20: A nova raiz recebe o nó atual como filho esquerdo.

Linha 21-22: As alturas dos nós são recalculadas após a rotação.

Linha 23: Retorna a nova raiz da subárvore, que agora é o nó que estava à direita da raiz anterior.

```
def _min_value_node(self, raiz):
    atual = raiz
    while atual.esquerda:
        atual = atual.esquerda
    return atual
```

Linha 24-27: Função privada **_min_value_node** que encontra o nó com o menor valor na subárvore (usado na remoção de um nó com dois filhos).

Linha 25-26: O algoritmo percorre a subárvore esquerda até encontrar o nó mais à esquerda, que é o menor valor.

Linha 27: Retorna o nó encontrado.

```
def remover(self, raiz, isbn):
    if not raiz:
        return raiz
```

Linha 28-29: Função **remover** que remove um nó (livro) da árvore. A função começa verificando se o nó **raiz** é **None**, ou seja, se a árvore está vazia ou o livro não existe.

Linha 29: Se o nó for **None**, retorna **None**, indicando que não há nada a remover.

```
if isbn < raiz.isbn:
    raiz.esquerda = self.remover(raiz.esquerda, isbn)
```

Linha 30: Se o ISBN do livro a ser removido for menor que o do nó atual, a busca continua na subárvore esquerda, chamando a função **remover** recursivamente.

```
elif isbn > raiz.isbn:
    raiz.direita = self.remover(raiz.direita, isbn)
```

Linha 31: Se o ISBN for maior que o do nó atual, a busca continua na subárvore direita, chamando a função **remover** recursivamente

```
else:
    if not raiz.esquerda:
        return raiz.direita
    elif not raiz.direita:
        return raiz.esquerda
```

.

Linha 32-35: Aqui, encontramos o nó a ser removido. Existem três casos:

- **Caso 1**: O nó não tem filho à esquerda. Nesse caso, o nó é removido e seu filho direito (se houver) o substitui.

- **Caso 2**: O nó não tem filho à direita. O nó é removido e seu filho esquerdo (se houver) o substitui.

```
temp = self._min_value_node(raiz.direita)
raiz.isbn = temp.isbn
raiz.titulo = temp.titulo
raiz.autor = temp.autor
```

Linha 36-39: Caso o nó tenha dois filhos, o nó a ser removido será substituído pelo sucessor (o menor nó da subárvore direita).

Linha 37-39: O ISBN, título e autor do nó substituto (sucessor) são copiados para o nó que será removido.

```
raiz.direita = self.remover(raiz.direita, temp.isbn)
```

Linha 40: A remoção do sucessor é feita recursivamente na subárvore direita, onde o nó do sucessor foi originalmente encontrado.

```
raiz.altura = 1 + max(self.altura(raiz.esquerda), self.altura(raiz.direita))
```

Linha 41: Após a remoção, a altura do nó atual é recalculada.

```
balanceamento = self.balanceamento(raiz)
```

Linha 42: O fator de balanceamento do nó é calculado para verificar se a árvore precisa de alguma rotação.

```
if balanceamento > 1 and self.balanceamento(raiz.esquerda) >= 0:
    return self.rotacao_direita(raiz)
```

Linha 43-45: Caso o fator de balanceamento seja maior que 1 e a subárvore esquerda estiver balanceada, realiza-se uma rotação simples à direita.

```
if balanceamento < -1 and self.balanceamento(raiz.direita) <= 0:
    return self.rotacao_esquerda(raiz)
```

Linha 46-48: Caso o fator de balanceamento seja menor que -1 e a subárvore direita estiver balanceada, realiza-se uma rotação simples à esquerda.

```
if balanceamento > 1 and self.balanceamento(raiz.esquerda) < 0:
    raiz.esquerda = self.rotacao_esquerda(raiz.esquerda)
    return self.rotacao_direita(raiz)
```

Linha 49-51: Caso o fator de balanceamento seja maior que 1 e a subárvore esquerda estiver desbalanceada para a direita, é realizada uma rotação dupla: primeiro à esquerda na subárvore esquerda e depois à direita no nó atual.

```
if balanceamento < -1 and self.balanceamento(raiz.direita) > 0:
    raiz.direita = self.rotacao_direita(raiz.direita)
    return self.rotacao_esquerda(raiz)
```

Linha 52-54: Caso o fator de balanceamento seja menor que -1 e a subárvore direita estiver desbalanceada para a esquerda, é realizada uma rotação dupla: primeiro à direita na subárvore direita e depois à esquerda no nó atual.

```
return raiz
```

Linha 55: Retorna a raiz (ou o nó balanceado) da árvore após a remoção e balanceamento.

5.2 - Código da Função de Busca

```
def buscar_livro(arvore, valor, tipo_busca):
```

Linha 1: Definição da função **buscar_livro**, que recebe três parâmetros:

- **arvore**: A instância da árvore AVL onde a busca será realizada.
- **valor**: O valor a ser buscado (pode ser o ISBN, título ou autor, dependendo do tipo de busca).
- **tipo_busca**: O tipo de busca, que pode ser **isbn**, **titulo** ou **autor**, determinando o critério da pesquisa.

```
if tipo_busca == 'isbn':
    livro = arvore.buscar(arvore.raiz, valor)
```

Linha 2-3: Aqui, verificamos o tipo de busca solicitado:

- Se o tipo de busca for **'isbn'**, chamamos a função **buscar** da árvore AVL, passando a raiz da árvore (**arvore.raiz**) e o valor de ISBN que queremos buscar.

- O resultado de **arvore.buscar()** será armazenado na variável **livro**, que será um objeto do tipo caso encontrado ou **None** caso não encontrado.

```
if livro:
    print(f"Livro encontrado por ISBN: {livro.titulo}, {livro.autor}, {livro.isbn}")
else:
    print("Livro não encontrado por ISBN.")
```

Linha 4-7: Verificamos se o livro foi encontrado:

- Se **livro** não for **None** (ou seja, o livro foi encontrado), exibimos as informações do livro: título, autor e ISBN.
- Caso contrário, se o livro não for encontrado, mostramos a mensagem **"Livro não encontrado por ISBN."**.

```
elif tipo_busca == 'titulo':
    livros_encontrados = []
```

Linha 8-9: Se o tipo de busca for **'titulo'**, criamos uma lista vazia chamada **livros_encontrados**, que armazenará todos os livros encontrados com o título correspondente.

```
def buscar_por_titulo_recursivo(no):
    if no:
        if valor.lower() in no.titulo.lower():
            livros_encontrados.append(no)
        buscar_por_titulo_recursivo(no.esquerda)
        buscar_por_titulo_recursivo(no.direita)
```

Linha 10-14: Aqui, definimos uma função recursiva interna chamada **buscar_por_titulo_recursivo**, que será responsável por percorrer a árvore e encontrar os livros com o título correspondente:

Linha 11: Se o nó (**no**) não for **None**, verificamos se o título do livro (**no.titulo**) contém o valor procurado (sem considerar maiúsculas ou minúsculas).

Linha 12: Se o título do livro contiver o valor procurado, adicionamos o nó à lista

.

Linhas 13-14: A função continua a busca recursivamente nas subárvores esquerda e direita.

```
buscar_por_titulo_recursivo(arvore.raiz)
```

Linha 15: Chama a função recursiva **buscar_por_titulo_recursivo** a partir da raiz da árvore para iniciar a busca por título.

```
if livros_encontrados:
    print(f"Livros encontrados com o título '{valor}':")
    for livro in livros_encontrados:
        print(f"{livro.titulo}, {livro.autor}, {livro.isbn}")
else:
    print(f"Nenhum livro encontrado com o título '{valor}'.")
```

Linha 16-21: Aqui, verificamos se algum livro foi encontrado:

- Se a lista **livros_encontrados** não estiver vazia, mostramos os livros encontrados e imprimimos suas informações (título, autor e ISBN).

- Caso contrário, se nenhum livro for encontrado, exibimos a mensagem **"Nenhum livro encontrado com o título 'valor'.".**

```
elif tipo_busca == 'autor':
    livros_encontrados = []
```

Linha 22-23: Se o tipo de busca for **'autor'**, criamos novamente a lista **livros_encontrados**, que armazenará os livros encontrados com o autor correspondente

```
def buscar_por_autor_recursivo(no):
    if no:
        if valor.lower() in no.autor.lower():
            livros_encontrados.append(no)
        buscar_por_autor_recursivo(no.esquerda)
        buscar_por_autor_recursivo(no.direita)
```

Linha 24-28: A função **buscar_por_autor_recursivo** funciona de forma semelhante à função de busca por título:

Linha 25: Se o nó não for **None**, verificamos se o nome do autor **(no.autor)** contém o valor procurado.

Linha 26: Se o autor do livro contiver o valor procurado, o nó é adicionado à lista

.

Linhas 27-28: A busca recursiva continua nas subárvores esquerda e direita.

```
buscar_por_autor_recursivo(arvore.raiz)
```

Linha 29: A função recursiva **buscar_por_autor_recursivo** é chamada a partir da raiz da árvore para iniciar a busca por autor.

```
if livros_encontrados:
    print(f"Livros encontrados com o autor '{valor}':")
    for livro in livros_encontrados:
        print(f"{livro.titulo}, {livro.autor}, {livro.isbn}")
else:
    print(f"Nenhum livro encontrado com o autor '{valor}'.")
```

Linha 30-35: Aqui, verificamos se algum livro foi encontrado:

- Se a lista **livros_encontrados** não estiver vazia, mostramos os livros encontrados e imprimimos suas informações.

- Caso contrário, se nenhum livro for encontrado, mostramos a mensagem

.

```
return
```

Linha 36: A função **buscar_livro** termina. Não é necessário retornar nada porque as informações foram exibidas diretamente.

5.3 - Código da Classe Livro

```
class Livro:
```

Linha 1: Definimos a classe **Livro**, que será responsável por armazenar as informações de cada livro (como título, autor e ISBN). Essa classe vai encapsular os dados de cada livro para que possam ser manipulados mais facilmente na árvore AVL.

```
def __init__(self, titulo, autor, isbn):
```

Linha 2: O método **__init__** é o construtor da classe, responsável por inicializar os atributos do livro. Ele recebe três parâmetros: **titulo**, **autor** e **isbn**.

- **titulo:** O título do livro.
- **autor:** O nome do autor do livro.
- **isbn:** O ISBN (ou identificador único) do livro.

```
self.titulo = titulo
self.autor = autor
self.isbn = isbn
```

Linha 3-5: Dentro do construtor, esses parâmetros são atribuídos aos atributos do objeto Livro. Isso significa que, ao criar um novo objeto da classe Livro, os dados do título, autor e ISBN serão armazenados nesse objeto específico.

```
def __str__(self):
```

Linha 6: O método especial __str__ é uma sobrecarga do método str(). Esse método define como o objeto será representado como uma string, ou seja, como ele será exibido quando você fizer um

ou converter o objeto em string.

```
return f"{self.titulo}, {self.autor}, {self.isbn}"
```

Linha 7: O método __str__ retorna uma string formatada, contendo o título, o autor e o ISBN do livro. Isso permite que, quando um livro for impresso ou exibido, ele mostre essas informações de forma legível.

5.4 - Código para Manipulação do Arquivo livros.json

O arquivo **livros.json** é utilizado para armazenar a lista de livros de forma estruturada em formato JSON.

Aqui está o código que você usa para carregar e salvar os livros no arquivo.

Função carregar_livros

```python
import json

def carregar_livros():
    try:
        with open('livros.json', 'r', encoding='utf-8') as file:
            livros = json.load(file)  # Carrega os livros do arquivo JSON
    except FileNotFoundError:  # Caso o arquivo não exista
        livros = []  # Se o arquivo não existir, inicializa uma lista vazia
    return livros
```

Importação do módulo json:

- A primeira linha importa o módulo **json**, que é necessário para carregar e salvar dados em formato JSON.

```python
try:
    with open('livros.json', 'r', encoding='utf-8') as file:
        livros = json.load(file)
```

Abrindo o arquivo livros.json:

- A função open('livros.json', 'r', encoding='utf-8') abre o arquivo livros.json no modo leitura ('r'), usando a codificação UTF-8.

- Se o arquivo existir, json.load(file) lê o conteúdo do arquivo JSON e o converte em uma estrutura de dados Python (no caso, uma lista de dicionários, onde cada dicionário representa um livro).

```
except FileNotFoundError:
    livros = []  # Se o arquivo não for encontrado, cria uma lista vazia
```

Tratamento de erro:

- Se o arquivo livros.json não for encontrado (FileNotFoundError), a função cria uma lista vazia para armazenar os livros.

```
return livros
```

Retorna a lista de livros:

- Se o arquivo for encontrado, livros será a lista de livros carregada do JSON. Caso contrário, a lista será vazia, e ela é retornada para o sistema.

Função salvar_livros

```
def salvar_livros(livros):
    with open('livros.json', 'w', encoding='utf-8') as file:
        json.dump(livros, file, indent=4, ensure_ascii=False)
```

Abertura do arquivo em modo escrita ('w'):

- A função **open('livros.json', 'w', encoding='utf-8')** abre o arquivo **livros.json** no modo escrita ('w'), permitindo escrever os dados nele. Se o arquivo já existir, ele será sobrescrito.

```
json.dump(livros, file, indent=4, ensure_ascii=False)
```

Escrita dos dados no arquivo JSON:

- A função **json.dump(livros, file, indent=4, ensure_ascii=False)** escreve os dados da lista **livros** no arquivo JSON.

- **indent=4**: Isso formata o JSON com uma indentação de 4 espaços, tornando o arquivo mais legível.

- **ensure_ascii=False:** Isso permite que caracteres especiais, como acentos e outros caracteres não-ASCII, sejam salvos corretamente no arquivo.

5.4 - Importação dos Módulos:

```
from arvore_avl import ArvoreAVL
from livros_class import carregar_livros, salvar_livros
from sistema_busca import buscar_livro
```

O código importa as funções e classes necessárias para o funcionamento do sistema:

- **ArvoreAVL:** Classe que implementa a árvore AVL.
- **carregar_livros:** Função que carrega os livros salvos no arquivo JSON.

- **salvar_livros:** Função que salva os livros no arquivo JSON.
- **buscar_livro:** Função para buscar livros por ISBN, título ou autor.

```
arvore = ArvoreAVL()  # Cria uma instância da árvore AVL
livros = carregar_livros()  # Carrega os livros do arquivo JSON
```

5.4.1 - Criação da Árvore AVL e Carregamento dos Livros

- **ArvoreAVL():** Cria a árvore AVL para armazenar os livros.
- **carregar_livros():** Carrega a lista de livros salvos no arquivo .

```
for livro in livros:
    arvore.raiz = arvore.inserir(arvore.raiz, livro["titulo"], livro["autor"], livro["isbn"])
```

5.4.2 - Inserção dos Livros na Árvore:

- A função percorre todos os livros carregados e os insere na árvore AVL.

```
while True:
    # Exibe o menu de opções
```

5.4.3 - Loop do Menu:

- O loop **while True:** mantém o programa em execução até que o usuário escolha a opção de sair (opção 7).

5.4.4 - Opções do Menu:

- **Opção 1 - Inserir Livro Manualmente**:

```
titulo = input("Digite o título do livro: ")
autor = input("Digite o autor do livro: ")
isbn = input("Digite o ISBN do livro: ")
```

- O usuário insere os dados do livro e, antes de inserir na árvore, o sistema verifica se o livro já está na árvore.
- Se o livro não existir, ele é inserido na árvore AVL, e a lista de livros é atualizada e salva.

Opção 2 - Exibir Todos os Livros:

```
arvore.exibir_em_ordem(arvore.raiz)  # Exibe os livros em ordem
```

- O método exibir_em_ordem() exibe todos os livros armazenados na árvore, percorrendo-os em ordem crescente de ISBN.

Opções 3, 4 e 5 - Buscar por ISBN, Título ou Autor:

- O sistema permite que o usuário busque livros por **ISBN**, **título** ou **autor**, utilizando a função buscar_livro(), que realiza a busca na árvore AVL.

Opção 6 - Excluir Livro:

```
livros = [livro for livro in livros if livro["isbn"] != isbn]  # Remove o livro da lista
```

- O livro é removido da lista e, em seguida, a árvore AVL é reconstruída com os livros restantes.

- Os dados atualizados são então salvos de volta no arquivo JSON.

Opção 7 - Sair do Sistema:

- O loop é interrompido e o programa é encerrado

5.4.5 - Função main():

```
if __name__ == "__main__":
    main()
```

- Isso garante que a função seja executada quando o script for executado diretamente.

6 - Conclusão

O sistema de organização e busca de livros utilizando árvores AVL oferece uma solução altamente eficiente e escalável para gerenciar catálogos de livros. A árvore AVL, por ser uma estrutura de dados balanceada, assegura operações rápidas de inserção, remoção e busca com complexidade de tempo **O(log n)**, mesmo em cenários com um grande número de livros. Isso torna o sistema ideal para ambientes que lidam com volumes crescentes de dados, como bibliotecas e catálogos de produtos, garantindo alta performance e baixo tempo de resposta.

Embora o sistema seja eficaz, ele apresenta algumas limitações que podem ser superadas com a evolução do projeto. A principal limitação é a falta de persistência de dados, já que os livros são carregados e manipulados apenas em memória. A implementação de um banco de dados persistente, como **MySQL** ou **SQLite**, poderia resolver esse problema, oferecendo uma solução mais robusta e escalável para o armazenamento dos livros, além de melhorar a performance ao lidar com grandes volumes de dados. A adição de uma interface gráfica de usuário (GUI) também poderia aprimorar a experiência do usuário, tornando o sistema mais acessível, intuitivo e fácil de usar, especialmente para aqueles não familiarizados com a linha de comando.

Adicionalmente, a inclusão de funcionalidades de busca avançada, como filtros de pesquisa por autor, título ou outros critérios personalizados, e a implementação de cache inteligente poderiam melhorar ainda mais a performance do sistema, proporcionando uma experiência de uso mais eficiente. A modularidade do sistema permite que essas novas funcionalidades sejam implementadas de forma gradual, sem comprometer a estrutura existente.

Além disso, a árvore AVL pode ser facilmente adaptada para outros cenários de uso além do gerenciamento de livros. Ela pode ser aplicada em sistemas de gerenciamento de estoque, catálogos de produtos, sistemas de

busca e outras áreas que exigem organização eficiente de dados ordenados. Essa flexibilidade amplia o impacto do projeto, tornando-o útil para diferentes tipos de sistemas.

Por fim, este projeto oferece uma base sólida para a construção de sistemas mais complexos, aplicando conceitos fundamentais de estruturas de dados e algoritmos de maneira prática e eficiente. Ele demonstra como uma árvore AVL pode ser utilizada para resolver problemas reais de gerenciamento de dados, combinando desempenho, escalabilidade e modularidade.