Guilherme Lopes Inocencio

Catálogo de Livros com Árvores BST e AVL

Aparecida de Goiânia - GO

15/08/2025

Guilherme Lopes Inocencio

Catálogo de Livros com Árvores BST e AVL

Relatório de experimento apresentado como parte dos requisitos para aprovação na disciplina do o período 02/2025, ministrado pelo professor Ronaldo Del Fiaco.

UNIFAN - Centro Educacional Alfredo Nasser

Instituto de Ciências Exatas

Laboratório de Estrutura de Dados 2

Aparecida de Goiânia - GO

15/08/2025

Resumo

Este relatório apresenta o desenvolvimento de um sistema de catálogo de livros utilizando estruturas de dados baseadas em árvores. O trabalho foi dividido em duas etapas principais: a implementação de uma Árvore Binária de Busca (BST) e, posteriormente, a implementação de uma Árvore AVL, com o objetivo de comparar o desempenho entre ambas.

O sistema desenvolvido permite a inserção, busca, remoção e listagem de livros por diferentes percursos (pré-ordem, in-ordem, pós-ordem e em largura), além de possibilitar a entrada de dados via teclado ou por arquivo externo. A escolha da linguagem Python justifica-se pela clareza sintática e pela ampla utilização em contextos acadêmicos e de pesquisa.

Os resultados demonstraram que, embora a BST seja simples de implementar, sua eficiência pode ser comprometida em casos de inserções desbalanceadas. Já a árvore AVL, ao custo de realizar rotações adicionais, mantém a estrutura balanceada, resultando em um menor número de comparações em operações de busca.

Conclui-se que a prática contribuiu para a consolidação dos conceitos de árvores de busca e ressaltou a importância do balanceamento para garantir desempenho em aplicações que exigem rapidez e organização na manipulação de dados.

Palavras-chaves: Árvores Binárias de Busca, Árvore AVL, Estruturas de Dados, Python, Catálogo de Livros, Balanceamento de Árvores, Busca Binária.

Sumário

Introdução

As estruturas de dados constituem elementos fundamentais da ciência da computação, fornecendo a base para a construção de algoritmos eficientes e sistemas de grande escala. Entre essas estruturas, a árvore binária de busca (BST) destaca-se por permitir operações de inserção, busca e remoção de maneira eficiente, desde que mantida em um estado próximo ao balanceado (CORMEN et al., 2009).

Entretanto, quando a BST sofre desequilíbrio, sua altura pode crescer de forma desproporcional, degradando o desempenho das operações a níveis equivalentes aos de uma lista encadeada. Para mitigar esse problema, foi proposta a árvore AVL, uma variação balanceada que ajusta automaticamente sua estrutura a cada operação, garantindo altura logarítmica e maior previsibilidade no tempo de execução (KNUTH, 1997).

No contexto de implementação, optou-se pela linguagem Python, amplamente reconhecida por sua simplicidade sintática e expressividade. Essa característica a torna adequada tanto para o ensino de algoritmos quanto para aplicações em cenários reais, incluindo áreas de ciência de dados, inteligência artificial e desenvolvimento de sistemas (LUTZ, 2013).

O presente trabalho tem como objetivo desenvolver um sistema de catálogo de livros utilizando árvores BST e AVL como estruturas centrais de armazenamento e consulta. O sistema contempla operações fundamentais, como (1) inserção de novos registros, (2) busca eficiente por títulos, (3) remoção de livros do acervo e (4) visualização ordenada dos elementos. Além disso, busca-se evidenciar, por meio de comparações práticas, o impacto do balanceamento automático da árvore AVL em relação à BST convencional.

Dessa forma, o relatório pretende não apenas consolidar o entendimento sobre árvores de busca e seus mecanismos de balanceamento, mas também demonstrar sua aplicação em um cenário prático, servindo como base para implementações mais complexas em sistemas de indexação, recuperação de informação e organização de grandes bases de dados.

Objetivos

O objetivo principal deste trabalho é **desenvolver um sistema de catálogo de livros** fundamentado no uso de estruturas de dados do tipo *árvore binária de busca* (BST) e sua variação balanceada, a *árvore AVL*, de forma a comparar sua eficiência em diferentes operações. A proposta busca consolidar o entendimento prático sobre árvores binárias e mecanismos de balanceamento automático, demonstrando sua relevância na organização e recuperação eficiente de dados.

De maneira mais específica, o sistema desenvolvido deve contemplar:

1. **Modelagem da estrutura de dados** destinada ao armazenamento de livros, considerando os atributos fundamentais: `ID`, título e autor;
2. **Implementação das operações básicas em árvores**, abrangendo:
    - Inserção de novos registros;
    - Busca eficiente por títulos;
    - Remoção de elementos;
    - Diferentes percursos: pré-ordem, in-ordem, pós-ordem e em largura;
3. **Flexibilidade na entrada de dados**, permitindo tanto a inserção manual via teclado quanto a importação a partir de arquivos externos;
4. **Desenvolvimento da árvore AVL**, incorporando a contabilização das rotações necessárias ao balanceamento, de modo a evidenciar a complexidade adicional em comparação à BST tradicional;
5. **Análise comparativa de desempenho** entre BST e AVL, destacando o impacto do balanceamento automático nas operações de busca, inserção e remoção.

Além dos aspectos técnicos, o projeto também visa reforçar a aplicação de boas práticas de programação, como modularização do código, clareza na implementação de algoritmos e documentação adequada das etapas. Assim, o trabalho contribui tanto para a **fixação de conceitos fundamentais de estruturas de dados** quanto para a **demonstração de sua aplicabilidade em sistemas de gerenciamento de informação**, como catálogos digitais, bases bibliográficas e mecanismos de indexação.

Parte I

Metodologia

Ambiente de Desenvolvimento

- **Linguagem de Programação:** Python 3.11
- **Ambiente de Desenvolvimento:**
  - IDE: Visual Studio Code, com extensões para Python
  - Plataforma: Windows 10/11
  - Controle de versão: GitHub (para hospedagem e versionamento do código)
- **Estruturas de Dados Utilizadas:**
  - Árvore Binária de Busca (BST)
  - Árvore AVL (com balanceamento automático e contabilização de rotações)
  - Fila implementada via `collections.deque` (para percurso em largura)
- **Bibliotecas Utilizadas:**
  - `collections.deque` — suporte à fila no percurso em largura
  - `time` — medição de desempenho e tempo de execução

Procedimento de Implementação

Estrutura do Projeto

A prática foi organizada em módulos para garantir clareza e modularização do código:

- `livro.py`: define a classe `Livro`, responsável por armazenar os atributos `ID`, `título` e `autor`;
- `no.py`: define a estrutura do nó da árvore, contendo o livro e os ponteiros para subárvores esquerda e direita;
- `bst.py`: implementação da árvore binária de busca, com operações de inserção, busca, remoção e percursos;
- `avl.py`: implementação da árvore AVL, incluindo as rotações necessárias para manter o balanceamento;
- `utils.py`: funções auxiliares, como a leitura de livros a partir de arquivos externos;
- `main.py`: contém o menu interativo, utilizando a estrutura `match-case` do Python 3.11.

Inserção na Árvore Binária de Busca

A inserção segue o princípio da comparação de chaves (neste caso, o atributo `ID` do livro). Caso a posição adequada seja encontrada (nó nulo), é criado um novo nó. Se o `ID` do livro for menor, a recursão continua pela subárvore esquerda; caso contrário, pela direita.

```
def inserir(self, raiz, livro):
    if raiz is None:
        return No(livro)
    if livro.id < raiz.livro.id:
        raiz.esq = self.inserir(raiz.esq, livro)
    elif livro.id > raiz.livro.id:
        raiz.dir = self.inserir(raiz.dir, livro)
    return raiz
```

Balanceamento na Árvore AVL

A árvore AVL estende a lógica da BST, mas após cada inserção ou remoção calcula o fator de balanceamento (`altura(esq) - altura(dir)`). Se o valor for maior que 1 ou menor que -1, aplica rotações simples ou duplas:

- Rotação simples à direita;
- Rotação simples à esquerda;
- Rotação dupla à direita (esquerda-direita);
- Rotação dupla à esquerda (direita-esquerda).

Esse mecanismo garante altura logarítmica, preservando a eficiência das operações.

Percursos Implementados

Foram implementados os seguintes percursos para visualização da árvore:

- **Pré-ordem**: raiz $\rightarrow$ esquerda $\rightarrow$ direita;
- **In-ordem**: esquerda $\rightarrow$ raiz $\rightarrow$ direita (gera ordenação por `ID`);
- **Pós-ordem**: esquerda $\rightarrow$ direita $\rightarrow$ raiz;
- **Em largura**: utilizando a estrutura `deque` para fila.

Fluxo do Programa

O programa segue o seguinte fluxo principal:

1. Exibição do menu interativo para o usuário;
2. Inserção manual de livros ou carregamento a partir de arquivo;
3. Execução das operações de busca, remoção e percursos;
4. Impressão do catálogo ordenado ou balanceado (dependendo da árvore escolhida);
5. Medição de desempenho para comparação entre BST e AVL.

Parte II

Resultados

Discussão

A análise comparativa entre a Árvore Binária de Busca (BST) e a Árvore AVL permitiu observar diferenças significativas no desempenho das operações, especialmente em cenários com grandes volumes de dados (CORMEN et al., 2009). Enquanto a BST apresenta simplicidade de implementação e menor custo computacional em casos ideais (inserções aleatórias ou já balanceadas), sua eficiência pode ser severamente comprometida em situações de inserções ordenadas, aproximando-se do desempenho de uma lista encadeada (SEDGEWICK; WAYNE, 2011).

A AVL, por outro lado, mostrou-se mais robusta ao manter o balanceamento automático da estrutura. Embora esse processo demande maior custo de manutenção — devido à execução de rotações em inserções e remoções —, os experimentos demonstraram que esse investimento é compensado pelo ganho em buscas mais rápidas e previsíveis (KNUTH, 1998). Em conjuntos de dados de grande porte, esse diferencial se mostrou decisivo, resultando em menor número de comparações e maior eficiência global.

Outro aspecto relevante foi a constatação de que a AVL garante melhor desempenho em cenários de uso intensivo de consultas, enquanto a BST pode ser suficiente em aplicações de pequeno porte, onde a simplicidade da estrutura é vantajosa (GOODRICH; TAMASSIA; GOLDWASSER, 2014). Essa distinção destaca a importância da escolha da estrutura de dados adequada ao contexto da aplicação.

Além disso, a modularização do sistema em múltiplos arquivos Python contribuiu para a clareza do projeto e facilitou a comparação entre as duas implementações. Essa organização também reforça a importância de boas práticas de programação, tornando o código mais manutenível e adequado para expansões futuras (MARTIN, 2009).

Portanto, a discussão evidencia que não há uma “melhor estrutura” universal, mas sim uma adequação de cada abordagem ao problema em questão: a BST para cenários simples e a AVL para aplicações que exigem desempenho consistente mesmo diante de grandes volumes de dados (WEISS, 2012a).

Resultado

Funcionalidades do Sistema

O sistema desenvolvido contempla um conjunto de operações que demonstram, na prática, a aplicação das árvores binárias de busca (BST) e das árvores AVL no gerenciamento de um catálogo de livros. Entre as principais funcionalidades, destacam-se:

- **Inserção e remoção de livros:** permite adicionar novos registros ao catálogo ou excluir elementos existentes, mantendo a integridade da estrutura de dados.
- **Busca eficiente por ID:** possibilita a localização de um livro de forma rápida, explorando a ordenação natural das árvores.
- **Listagem em diferentes percursos:** inclui percursos clássicos de árvores, como pré-ordem, in-ordem, pós-ordem e em largura, permitindo múltiplas formas de visualização do acervo.
- **Comparação entre BST e AVL:** o usuário pode executar as mesmas operações em ambas as estruturas, evidenciando as diferenças de desempenho e organização.

Análise de Desempenho

Durante os testes práticos, observou-se que a **árvore AVL realiza um número maior de rotações** durante as operações de inserção e remoção. Entretanto, esse custo adicional é compensado pelo fato de que a AVL mantém a árvore sempre mais equilibrada, garantindo altura próxima a $\log n$.

Como consequência, verificou-se que a AVL apresentou **menor número de comparações em operações de busca**, especialmente em conjuntos de dados de maior porte. Já a BST, quando não balanceada, demonstrou degradação significativa de desempenho, aproximando-se do comportamento de uma lista encadeada em casos de inserções ordenadas (WEISS, 2012b).

Esses resultados reforçam a importância do balanceamento automático proporcionado pelas árvores AVL, ainda que com maior custo de manutenção estrutural, destacando sua aplicação em sistemas que demandam eficiência em grandes volumes de dados.

Considerações Finais

A prática desenvolvida possibilitou a consolidação dos conceitos fundamentais de estruturas de dados, com ênfase nas **árvores binárias de busca (BST)** e nas **árvores AVL**. Ao longo do trabalho, foi possível compreender não apenas a implementação dessas estruturas, mas também a análise comparativa de seus desempenhos, evidenciando os impactos do balanceamento automático na eficiência das operações de inserção, remoção e busca.

Os resultados obtidos reforçam a relevância das árvores balanceadas em cenários que envolvem grandes volumes de dados, demonstrando que o custo adicional de rotações é compensado pela manutenção da altura próxima a $\log n$, resultando em maior previsibilidade e desempenho estável.

Além do aspecto técnico, o desenvolvimento do sistema contribuiu para o aprimoramento de boas práticas de programação, modularização de código e uso de ferramentas de versionamento, competências essenciais para projetos de maior porte.

Anexo

O código-fonte utilizado neste trabalho, incluindo as implementações de um **Sistema de Catálogo de Livros com Árvores**, está disponível no repositório GitHub. Para acessar os arquivos e conferir os algoritmos utilizados, visite:

Repositório GitHub

Este repositório contém todo o material necessário para reproduzir os testes realizados e aprofundar a compreensão dos métodos abordados.

Referências

CORMEN, T. H. et al. *Introduction to Algorithms*. 3. ed. [S.l.]: MIT Press, 2009. Citado 2 vezes nas páginas 5 e 17.

GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H. *Data Structures and Algorithms in Java*. 6th. ed. [S.l.]: Wiley, 2014. Citado na página 17.

KNUTH, D. E. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. 3. ed. [S.l.]: Addison-Wesley, 1997. Citado na página 5.

KNUTH, D. E. *The Art of Computer Programming, Volume 3: Sorting and Searching*. 2nd. ed. [S.l.]: Addison-Wesley, 1998. Citado na página 17.

LUTZ, M. *Learning Python*. 5. ed. [S.l.]: O'Reilly Media, 2013. Citado na página 5.

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [S.l.]: Prentice Hall, 2009. Citado na página 17.

SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4th. ed. [S.l.]: Addison-Wesley, 2011. Citado na página 17.

WEISS, M. A. *Data Structures and Algorithm Analysis in C++*. 4th. ed. [S.l.]: Pearson, 2012. Citado na página 17.

WEISS, M. A. *Data Structures and Algorithm Analysis in Java*. 3. ed. [S.l.]: Pearson, 2012. Citado na página 19.