

# Projeto — Sistema de Gerenciamento de Transações Bancárias com Árvore AVL

---

**Aluno:** Cauã Borges Faria

**Disciplina:** Algoritmos e Estruturas de Dados II

**Professor:** Alexandre L. M. Levada

**Link:**

[https://colab.research.google.com/drive/1dMpmujdqRtF6c0\\_rcwqi77PsPX99gW-H?usp=sharing](https://colab.research.google.com/drive/1dMpmujdqRtF6c0_rcwqi77PsPX99gW-H?usp=sharing)

## Descrição do Problema

O projeto propõe a implementação de um **Sistema de Gerenciamento de Transações Bancárias** que precisa armazenar, buscar, remover e listar transações financeiras de forma eficiente. Como o volume de transações é alto e sua ordenação pelo tempo é necessária, utilizou-se uma **Árvore AVL** para manter as transações balanceadas, garantindo operações de inserção, remoção e busca em tempo  **$O(\log n)$** .

Cada transação possui os seguintes atributos:

- `time` (tempo único, usado como chave)
- `transaction_id` (ID aleatório)
- `amount` (valor aleatório entre 10 e 10.000)
- `transaction_type` (DEPÓSITO ou SAQUE)
- `account` (conta associada)

As funcionalidades implementadas:

1. Inserção de 5000 transações
2. Busca por 10 tempos específicos
3. Remoção de transações de tempo `0` a `500`

## 4. Impressão das transações restantes em ordem cronológica

---

### Códigos

O código completo está organizado no arquivo `main.py`, contendo:

- Classe `Transaction`
- Classe `AVLNode`
- Classe `AVLTree`
- Classe `BankingSystem`
- Função `main()` com execução do projeto e estatísticas

### Explicação do Funcionamento dos Códigos

#### Estrutura da AVL

A AVL é uma árvore binária de busca que mantém seu **balanceamento** automaticamente a cada operação de inserção ou remoção, realizando rotações quando necessário.

Cada nó (`AVLNode`) armazena uma transação e possui informações sobre suas subárvores esquerda e direita, além da altura e referência ao pai.




#### Operações:


- **Inserção:** segue as regras da BST. Após inserir, atualiza alturas e verifica o fator de balanceamento. Se necessário, executa rotações para manter a AVL válida.
- **Busca:** operação recursiva baseada na ordenação pelo `time`.
- **Remoção:** localiza o nó pelo `time`. Se encontrado:
  - Se for folha ou com um filho, remove diretamente.
  - Se tiver dois filhos, substitui pelo sucessor e remove o sucessor.
  - Após a remoção, atualiza alturas e executa rotações se necessário.
- **In-Order Traversal:** percorre a árvore pela esquerda, visita o nó e depois a direita, gerando uma lista ordenada por `time`.

---

## Resultados Obtidos

### Execução do Programa:

OPERAÇÃO	RESULTADO
Inserção de 5000 transações	✅ Concluído em <b>0.0660 segundos</b>
Altura da árvore após inserção	 <b>13</b>
Busca de 10 tempos específicos	✅ Tempo total <b>~0.000067 segundos</b>
Remoção de transações de tempo 0 a 500	✅ 501 transações removidas em <b>0.0038 segundos</b>
Altura da árvore após remoção	 <b>13</b> (permaneceu devido à estrutura da AVL)
Impressão In-Order	 4499 transações restantes, ordenadas corretamente

 *Observação 1: A altura manteve-se pois as transações removidas estavam na porção inferior esquerda da árvore e não impactaram significativamente sua altura máxima, controlada pelos nós mais recentes no lado direito.*

 *\*Observação 2: Valores referentes ao código compilado no google Colab*

---

## Considerações e Eficiência

O projeto confirmou a eficiência das operações AVL:

- **Inserção e remoção:** realizadas em tempo  **$O(\log n)$**
- **Busca eficiente**
- **Manutenção automática do balanceamento**

A **altura real da árvore** permaneceu dentro do esperado para uma AVL com 5000 nós.

 *Altura teórica máxima:*  $\approx 1.44 * \log_2(5000) \approx 17$

✅ *Altura real:* 13

---

## **Conclusão**

O sistema demonstrou ser eficiente para gerenciar grandes volumes de transações financeiras organizadas temporalmente, confirmando a adequação das Árvores AVL para cenários que exigem ordenação dinâmica e operações rápidas de inserção, busca e remoção.