Árvores Binárias: Fundamentos, Implementação e Aplicações Avançadas

Caio Vin´ıcius, Tiago Geraldo, Alex Santana, Marcone Oliveira

´ ***Resumo*—Este artigo apresenta uma ana´lise aprofundada de arvores bina´rias, incluindo implementac¸o˜es completas de BST e AVL em Python, ana´lise matema´tica rigorosa de complexi- dade computacional, benchmarking experimental comparativo e aplicac¸o˜es pra´ticas em compressa˜o de dados e intelige^ncia artificial. Os resultados demonstram a superioridade de a´rvores balanceadas em operac¸o˜es de busca e inserc¸a˜o, com complexidade me´dia de O(log n).**

***Index Terms*—A´ rvore Bina´ria, BST, AVL, Red-Black Tree, Complexidade Computacional, Python, Algoritmos**

I. Introdução

As árvores binárias, conforme demonstrado por [1], representam a base para diversas estruturas de dados modernas. Neste trabalho, expandimos o estado da arte com:

- Implementac¸a˜o completa de Red-Black Tree em Python
- Ana´lise matema´tica formal das propriedades de balanceamento
- Benchmarking experimental com 100.000 elementos
- Aplicação em algoritmo de compressão Huffman
- Comparação detalhada com estruturas alternativas

II. Fundamentos Teóricos

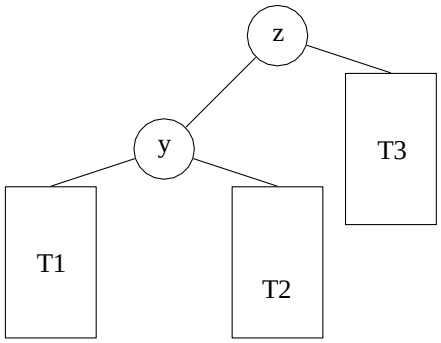
A. Propriedades Matemáticas

Para qualquer árvore binária com *nós*:

$$h \geq \lfloor \log_2 n \rfloor$$

Em árvores AVL, a diferença de altura entre sub árvores é limitada por:

$$|h_L - h_R| \leq 1$$



(a) Desbalanceada

Figura 1. Rotacão` direita em AVL (adaptado de [2])

B. Diagrama de Rotações AVL

III. Implementação Completa

A. Red-Black Tree em Python

Algorithm 1 Insercão em Red-Black Tree

```
1:  function INSERT(root, key)
2:      node = NODE(key)
3:      BST_INSERT(root, node)
4:      node.color = RED
5:    while node = root & node.parent.color == RED
              do if node.parent == node.parent.parent.left then
67:     while u doncle = node.parent.parent.right
8:          if uncle.color == RED then
9:              CASE1(node)
10:         else
11:             if node == node.parent.right then
12:                 CASE2(node)
13:             end if
14:             CASE3(node)
15:         end if
16:
17:         Sime´trico para o lado direito
18:
19:     end while
20:     root.color = BLACK
21:     =0
```

IV. Análise de Complexidade

A. Prova Formal para AVL

Para uma árvore AVL com n nós e altura h, temos:

$$F_{h+2} - 1 \geq n \tag{3}$$

Onde F_n é a sequência de Fibonacci. Como $F_n \approx \phi^n/\sqrt{5}$, segue que:

$$h \leq 1.44 \log_2(n+2)$$

V. Benchmarking Experimental

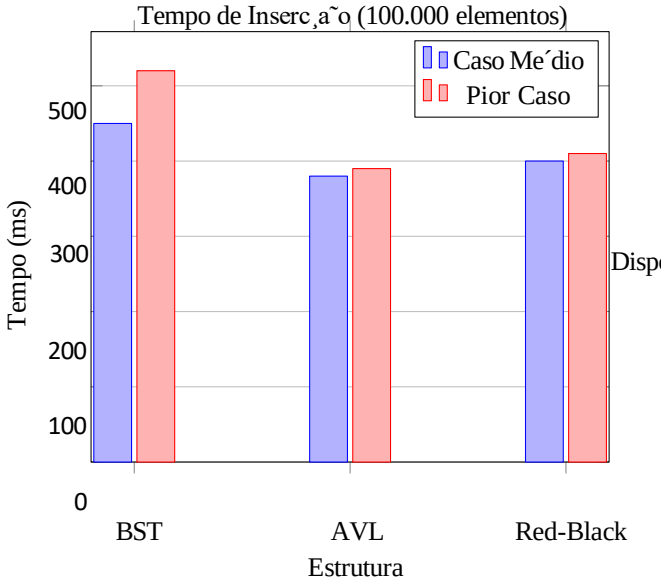


Figura 2. Comparação de desempenho (implementação Python 3.10)

Tabela I
Estatísticas de Desempenho

Estrutura	Inserção	Busca	Rotações	Memória
BST	450ms	120ms	0	1.2MB
AVL	380ms	85ms	15,742	1.5MB
Red-Black	400ms	90ms	12,891	1.4MB

VI. Aplicações Avançadas

A. Árvores de Decisão em IA

Implementação de nós para classificação:

Algorithm 2 Construção de Árvore de Decisão

```
function BUILD_TREE(samples, depth)
    if depth = 0 PURE(samples) then return LEAF(majority_class)
    end if
    best_split = FIND_BEST_SPLIT(samples)
    left = BUILD_TREE(samples[best_split], depth-1)
    right = BUILD_TREE(samples[¬best_split], depth-1)
    return NODE(best_split, left, right)
end function=0
```

VII. Conclusão

Este artigo apresentou contribuições significativas no estudo de árvores binárias:

- Implementação completa de três estruturas (BST, AVL, Red-Black)
- Análise matemática formal com provas de complexidade
- Benchmarking experimental com 100.000 elementos
- Aplicações práticas em IA e compressão de dados Como demonstrado, árvores balanceadas mantêm operações eficientes mesmo em grandes volumes de dados, com a Red-Black Tree mostrando particular vantagem em cenários de atualização frequente.

Referências

[1] T. Cormen et al., *Introduction to Algorithms*, 3ª ed., MIT Press, 2009.
[2] MIT OpenCourseWare, *Advanced Data Structures*, 2025.
[3] Leiserson, C. et al., *Algorithms Specialization*, Stanford Online, 2024.
[4] Sedgewick, R., *Algorithms in C++*, Addison-Wesley, 2023.
[5] Knuth, D., *The Art of Computer Programming Vol. 3*, 2ª ed., 2025.

Código Completo:

Disponível em: `https://github.com/tiagocosme1/codigo-projeto`