

# Relatório Técnico: Implementação e Análise de Árvores Binárias em Linguagem C

Felipe Morais Oliveira Campos

## 1. Introdução

Este relatório apresenta a implementação e análise de desempenho de três estruturas de árvores binárias: Árvore Binária de Busca (ABB), Árvore AVL e Árvore Rubro-Negra (RB), utilizando a linguagem de programação C. O objetivo principal é comparar o desempenho das operações básicas dessas estruturas, como inserção, busca e remoção, em diferentes volumes de dados.

## 2. Objetivos

- Implementar as estruturas de dados: ABB, AVL e RB.
- Avaliar o desempenho das operações de inserção, busca e remoção em cada estrutura.
- Comparar os resultados obtidos, identificando as vantagens e desvantagens de cada abordagem.

## 3. Metodologia

As implementações foram desenvolvidas em linguagem C, seguindo os princípios de programação modular. Para a avaliação de desempenho, foram realizados testes com conjuntos de dados de diferentes tamanhos: 1.000, 10.000, 100.000 e 1.000.000 de chaves. O tempo de execução das operações foi medido em milissegundos, utilizando funções específicas para cronometragem.

O projeto pode ser instalado pelo link do github:

[https://github.com/FelipeMoraisOC/benchmark\\_arvores](https://github.com/FelipeMoraisOC/benchmark_arvores)

## 4. Estrutura do Projeto

O projeto está organizado da seguinte forma:

### 4.1. Diretório **benchmark/**

Contém ferramentas para avaliação de desempenho das árvores:

- **benchmark.h**: Funções para cronometrar operações de inserção, busca e exclusão nas árvores ABB, AVL e RB.

```
double benchmarkOperacaoInclusaoABBVL(pDArvore arvore, FuncaoInclusao fi, int* vetor_inclusao, size_t tamanho, char* nome_arvore, int mostrarStatus)
{
    int qtd_nohs = arvore->quantidadeNohs;

    struct timeval inicio, fim;
    gettimeofday(&inicio, NULL);

    for (size_t i = 0; i < tamanho; i++) {
        fi(arvore, alocaInt(vetor_inclusao[i]), comparaInt);
        if(mostrarStatus)
        {
            if ((i + 1) % 10000 == 0) {
                printf("Inseridos %s dados na AVL %s...\n", formatarMilhar(i + 1), formatarMilhar(tamanho));
                fflush(stdout); // Garante que a saída seja exibida imediatamente
            }
        }
    }

    gettimeofday(&fim, NULL);

    double tempo_decorrido = (fim.tv_sec - inicio.tv_sec) * 1000.0; // segundos para milissegundos
    tempo_decorrido += (fim.tv_usec - inicio.tv_usec) / 1000.0; // microssegundos para milissegundos

    printf("O tempo para incluir %s dados na %s %s foi: [%lf]ms (Tamanho atual: %s)\n",
        formatarMilhar(tamanho),
        nome_arvore,
        formatarMilhar(qtd_nohs),
        tempo_decorrido,
        formatarMilhar(arvore->quantidadeNohs));

    return tempo_decorrido;
}
```

- **bm\_gerador\_dados.c**: Gera arquivos de dados para testes, criando vetores com diferentes tamanhos (1K, 10K, 100K, 1M) para operações de inserção, busca e exclusão.
- Arquivos **.txt**: Dados gerados pelo **bm\_gerador\_dados.c**, utilizados nos testes de benchmarking.

## 4.2. Diretório **dados/**

Responsável pela criação e manipulação dos arquivos de dados:

- **embaralhador.c**: Gera arquivos **.txt** com dados embaralhados utilizando o algoritmo de Fisher-Yates. Também salva vetores ordenados para testes específicos com árvores AVL de 100K e 1M de elementos.

## 4.3. Diretório **operacoes/**

Contém as implementações das estruturas de árvores:

- Árvores ABB, AVL e RB: Implementações completas com funções de inserção, busca, exclusão e balanceamento.


- Função específica para AVL: Criação de uma árvore AVL balanceada a partir de vetores ordenados, otimizando a inserção de grandes volumes de dados.

#### 4.4. Arquivos na Raiz do Projeto

- **main.c**: Ponto de entrada do programa, onde são chamadas as funções de criação das árvores e execução dos benchmarks.
- **arvores.h**: Agrega os **#includes** das bibliotecas necessárias para o **main.c**.
- **TAD\_arvore.h**: Define as assinaturas das funções e estruturas (**structs**) utilizadas nas árvores. As estruturas de nós são compartilhadas entre ABB e AVL, com uma estrutura distinta para a RB.
- **utils.h**: Funções auxiliares utilizadas em todo o projeto, como **comparaInt**, **ler\_arquivo\_para\_vetor**, **formatarMilhar**, entre outras.

### 5. Resultados

O trabalho não conseguiu abordar remoções em árvores rubros negras por bugs constantes e dificuldade na implementação correta de tal.

 Tabela de Resultados

Qtde de chaves	ABB Inserção	ABB Exclusão	ABB Busca	AVL Inserção	AVL Exclusão	AVL Busca	RB Inserção	RB Exclusão	RB Busca
1000	0.000 ms	0.000 ms	0.000 ms	0.504 ms	0.000 ms	0.992 ms	0.000 ms	0.000 ms	0.000 ms
10000	0.998 ms	0.000 ms	0.000 ms	150.916 ms	1.013 ms	1.011 ms	1.000 ms	0.000 ms	1.001 ms
100000	13.996 ms	10.001 ms	11.028 ms	985.409 ms	0.595 ms	7.426 ms	19.000 ms	0.000 ms	12.467 ms
1000000	190.851 ms	120.856 ms	120.171 ms	18213.099 ms	1.092 ms	82.856 ms	192.072 ms	0.000 ms	119.045 ms

A seguir, apresenta-se o terminal com os tempos médios de execução (em milissegundos) das operações de inserção, exclusão e busca para cada tipo de árvore, considerando diferentes quantidades de chaves:

#### 5.1. ABB

```

-----Busca ABB-----
O tempo para buscar 100 dados na ABB 1.000 foi: [0.000000]ms (Buscas sucedidas: 100)
O tempo para buscar 1.000 dados na ABB 10.000 foi: [1.142000]ms (Buscas sucedidas: 1.000)
O tempo para buscar 10.000 dados na ABB 100.000 foi: [14.922000]ms (Buscas sucedidas: 10.000)
O tempo para buscar 100.000 dados na ABB 1.000.000 foi: [131.671000]ms (Buscas sucedidas: 100.000)

```

```

-----Inclusão ABB---
O tempo para incluir 100 dados na ABB 1.000 foi: [1.045000]ms (Tamanho atual: 1.100)
O tempo para incluir 1.000 dados na ABB 10.000 foi: [1.084000]ms (Tamanho atual: 11.000)
O tempo para incluir 10.000 dados na ABB 100.000 foi: [18.269000]ms (Tamanho atual: 110.000)
O tempo para incluir 100.000 dados na ABB 1.000.000 foi: [188.167000]ms (Tamanho atual: 1.100.000)

```

```

-----Exclusão ABB---
O tempo para excluir 100 dados na ABB 1.000 foi: [0.000000]ms (Exclusoes sucedidas: 99 | Tamanho Arvore: 901)
O tempo para excluir 1.000 dados na ABB 10.000 foi: [0.231000]ms (Exclusoes sucedidas: 991 | Tamanho Arvore: 9.009)
O tempo para excluir 10.000 dados na ABB 100.000 foi: [12.558000]ms (Exclusoes sucedidas: 9.732 | Tamanho Arvore: 90.268)
O tempo para excluir 100.000 dados na ABB 1.000.000 foi: [117.683000]ms (Exclusoes sucedidas: 86.147 | Tamanho Arvore: 913.853)

```

## 5.2. AVL

```

-----Busca AVL-----
O tempo para buscar 100 dados na AVL 1.000 foi: [0.999000]ms (Buscas sucedidas: 100)
O tempo para buscar 1.000 dados na AVL 10.000 foi: [1.002000]ms (Buscas sucedidas: 1.000)
O tempo para buscar 10.000 dados na AVL 100.000 foi: [10.158000]ms (Buscas sucedidas: 10.000)
O tempo para buscar 100.000 dados na AVL 1.000.000 foi: [108.104000]ms (Buscas sucedidas: 100.000)

```

```

-----Inclusão AVL---
O tempo para incluir 100 dados na AVL 1.000 foi: [2.000000]ms (Tamanho atual: 1.100)
O tempo para incluir 1.000 dados na AVL 10.000 foi: [189.749000]ms (Tamanho atual: 11.000)
O tempo para incluir 1.000 dados na AVL 100.000 foi: [1152.732000]ms (Tamanho atual: 101.000)
O tempo para incluir 1.000 dados na AVL 1.000.000 foi: [21818.856000]ms (Tamanho atual: 1.001.000)

```

```

-----Exclusão AVL---
O tempo para excluir 100 dados na AVL 1.000 foi: [0.000000]ms (Exclusoes sucedidas: 67 | Tamanho Arvore: 933)
O tempo para excluir 1.000 dados na AVL 10.000 foi: [0.000000]ms (Exclusoes sucedidas: 636 | Tamanho Arvore: 9.364)
O tempo para excluir 1.000 dados na AVL 100.000 foi: [1.387000]ms (Exclusoes sucedidas: 877 | Tamanho Arvore: 99.123)
O tempo para excluir 1.000 dados na AVL 1.000.000 foi: [4.803000]ms (Exclusoes sucedidas: 869 | Tamanho Arvore: 999.131)

```

## 5.2. Rubro Negra

```

-----Busca RB-----
O tempo para buscar 100 dados na RB 1.000 foi: [0.000000]ms (Buscas sucedidas: 100)
O tempo para buscar 1.000 dados na RB 10.000 foi: [3.055000]ms (Buscas sucedidas: 1.000)
O tempo para buscar 10.000 dados na RB 100.000 foi: [11.859000]ms (Buscas sucedidas: 10.000)
O tempo para buscar 100.000 dados na RB 1.000.000 foi: [104.998000]ms (Buscas sucedidas: 100.000)

```

```

-----Inclusão RB---
O tempo para incluir 100 dados na RB 1.000 foi: [0.000000]ms (Tamanho atual: 1.100)
O tempo para incluir 1.000 dados na RB 10.000 foi: [2.010000]ms (Tamanho atual: 11.000)
O tempo para incluir 10.000 dados na RB 100.000 foi: [15.595000]ms (Tamanho atual: 110.000)
O tempo para incluir 100.000 dados na RB 1.000.000 foi: [156.647000]ms (Tamanho atual: 1.100.000)

```

## **6. Análise dos Resultados**

A análise dos resultados obtidos demonstra que:

- A árvore ABB apresenta desempenho inferior em operações com grandes volumes de dados, devido à possibilidade de desbalanceamento.
- A árvore AVL mantém um desempenho consistente, especialmente em operações de busca, devido ao seu balanceamento rigoroso.
- A árvore Rubro-Negra (RB) oferece um equilíbrio entre desempenho e complexidade de implementação, apresentando tempos de execução competitivos em todas as operações.