UNIVERSIDADE DO ESTADO DE SANTA CATARINA - UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS - CCT
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Análise da Complexidade Algorítmica das Operações de Adição e Remoção em Árvores AVL, B e Rubro-Negra

Equipe: Kelwin Efrain Bagnhuk da Silva
Leonardo Floriani Monteiro
Uriel Pacheco de Souza
Guilherme Vieira Maria
Emanuel Scharmitzel Johanson
Docente: Allan Rodrigo Leite
Disciplina: Estrutura de Dados II

JOINVILLE

2025

SUMÁRIO

1. INTRODUÇÃO

À medida que códigos tratavam com um maior número de dados, tornou-se necessário buscar estruturas de dados que diminuíssem o custo computacional da máquina. Como solução, surgiram as árvores, em especial a árvore binária. Entretanto, tal árvore ainda era limitada (como, por exemplo, o fato de lidar melhor com dados na memória principal do que dados advindos da memória externa). Para aumentar o desempenho e tratar casos específicos quanto à maneira de lidar com os dados, como inserção, remoção e percurso, foram criados outros modelos de árvores, dos quais, neste estudo, são destacados: a árvore AVL, a árvore Rubro-Negra e a árvore B, cada qual interagindo de uma maneira diferente com os dados, podendo focar-se mais na inserção ou mais na remoção dos elementos.

Com a finalidade de compreender melhor até que ponto uma árvore pode ser considerada mais eficiente em dado contexto, dentre os modelos destacados, esse estudo analisa o comportamento de cada árvore a medida que o tamanho da entrada aumenta, comparando a qualidade delas, nos quesitos inserção e remoção, por meio de gráficos com base na notação matemática Big-O.

2. OBJETIVO DO ESTUDO

O objetivo deste estudo é analisar o comportamento das árvores AVL, Rubro-Negra e B quando submetidas às operações de inserção e remoção à medida que se aumenta o tamanho da entrada de dados na respectiva árvore. Como ferramenta de análise, cria-se o gráfico comparativo entre as estruturas de dados nos quesitos inserção e remoção a partir da notação matemática Big-O.

3. EXPERIMENTOS

3.1 Métricas e Contabilização

A principal métrica utilizada para a comparação de árvores é o contabilizador de custos, considerando o custo de inserção e remoção, sendo tanto as variáveis globais quanto a função específica para cada árvore (a função "{tipo da árvore}_contaCusto (int vezes)" e as variáveis globais "{tipo da árvore}_custo_insercao" e "{tipo da árvore}_custo_remocao"). Essas métricas são necessárias pois caso considerasse apenas o custo por entrada em funções, o comportamento das funções agiria de modo linear e semelhante para a maioria das árvores. Portanto, essas métricas se aproximam da real complexidade com base na notação Big-O.

O funcionamento da função de contagem é igual para as três árvores, ela recebe um inteiro referente ao valor do custo. Usando um if, ela verifica se é uma operação de inserção ou remoção e contabiliza na variável respectiva.

Estas funções são chamadas onde acontece uma mudança de nível, a cada verificação de balanceamento e a cada efetiva operação de balanceamento. Além disso, existem diferenças nas operações de balanceamento de cada árvore, e isso é crucial para as diferenças no valor do custo para a inserção e remoção.

Na árvore Rubro-Negra são contadas as operações de rotação e verificação das cores dos nós, enquanto que na árvore AVL as 4 operações de rotação (RSE, RSD, RDE e RDD), verificação e avanço de nível são consideradas para a contagem. Já nas árvores B, é contabilizado o percurso no nó em busca de uma chave (pesquisa binária), o split de um nó, verificação de transbordo e o *merge* de nós. É importante destacar que, apesar da pesquisa binária dentro do nó não ser uma operação de balanceamento propriamente dita, esse custo foi incluído para manter a comparação justa entre as árvores, uma vez que a árvore B contém poucos níveis devido a sua propriedade

multi-chave, beneficiando-a fortemente no contexto específico em que está sendo analisado.

Foi utilizada uma quantidade de dados máxima “N”, da qual segue em passos de 500 partindo de 0 até N. A fim de aumentar a precisão do custo computacional com relação à complexidade *Big-O*, utiliza-se um número de amostras “M” para cada passo, e em seguida aplica-se uma média considerando a soma do custo total de cada amostra para cada categoria (Inserção e Remoção) dividido pelo total M:

$$\frac{(CustoTotal\ (amostra\ 0) + CustoTotal(amostra\ 1) + ... + CustoTotal(amostra\ M)}{M}$$

3.2 Aplicação das Árvores e Compilação

Em cada experimento, foi utilizado um dos tipos de árvores destacadas (Árvore AVL, Rubro-Negra ou B). Para cada uma, dentro dos códigos, são gerados valores arbitrários de entrada, limitados por quantidades de análise N (500, 1000, 1500, .., N). Para cada valor de N são aplicadas as operações de Adição e Remoção a partir da função “obter_operacoes_{tipo de árvore}”, da qual, por meio da biblioteca “arvores.h” e da conexão entre os códigos de cada árvore, pode-se obter e operar com inserções e remoções, e para cada operação é gerado um arquivo csv (“{tipo de árvore}_insercao.csv” e “{tipo de árvore}_remocao.csv”). Testam-se os limites de quantidade de análise N e para cada nova amostra até uma amostra máxima M é gerado novamente novos valores arbitrários a fim de aumentar a precisão do modelo. Após a análise de um dos modelos de árvore, segue-se o mesmo processo para os demais modelos, gerando dois arquivos para cada árvore.

Após a geração dos arquivos, utilizam-se esses arquivos em csv para serem postos em gráficos gerados a partir de um código em Python “plot.py”. No código, entra-se com uma lista dos tipos de árvores utilizados no estudo, assim como uma lista para as operações

testadas. Define-se uma função “ler_csv” que lê a primeira e segunda coluna de cada linha, sendo, respectivamente, o valor de N e o do custo computacional, isto é, o número de iterações obtidas em loops e em saltos condicionais. Em seguida, cria-se um loop para a plotagem da inserção, do qual obtém a lista de árvores, lê os valores usando a função “ler_csv”, cria o gráfico com base nos dados lidos e salva gerando um arquivo png para a inserção. Repete-se o processo lendo-se o arquivo csv quanto à remoção e aplicando os mesmos passos para a operação anterior, dessa vez gerando um arquivo png com os dados da remoção.

No caso do código implementado, foi utilizado como padrão N = 100.000 e M = 100.

4. INSTRUÇÕES DE COMPILAÇÃO

A implementação foi desenvolvida na linguagem C, com a utilização da linguagem Python para geração dos gráficos. Nesse sentido, a compilação e execução do programa foram testadas nos sistemas operacionais *Windows 10* e *Linux Mint* não ocasionando problemas de portabilidade e realizando os testes corretamente.

Para realizar a compilação do código, é necessário que todos arquivos que são componentes do código-fonte *main.c* estejam localizados no mesmo diretório. Em seguida, utilizando o terminal, ele poderá ser compilado inserindo o comando:

(Para sistemas Windows)

gcc main.c avl.c rn.c b.c -o main.exe

(Para sistemas Linux)

gcc main.c avl.c rn.c b.c -o main

Após este passo, a execução ocorre pela inserção do próximo comando:

(Para sistemas Windows)

main.exe

(Para sistemas Linux)

./main

Para que a geração automática dos gráficos funcione, é necessário ter a linguagem Python instalada juntamente com as bibliotecas *matplotlib* e *numpy.* A instalação delas pode ser feita no terminal através do comando:

pip install matplotlib numpy

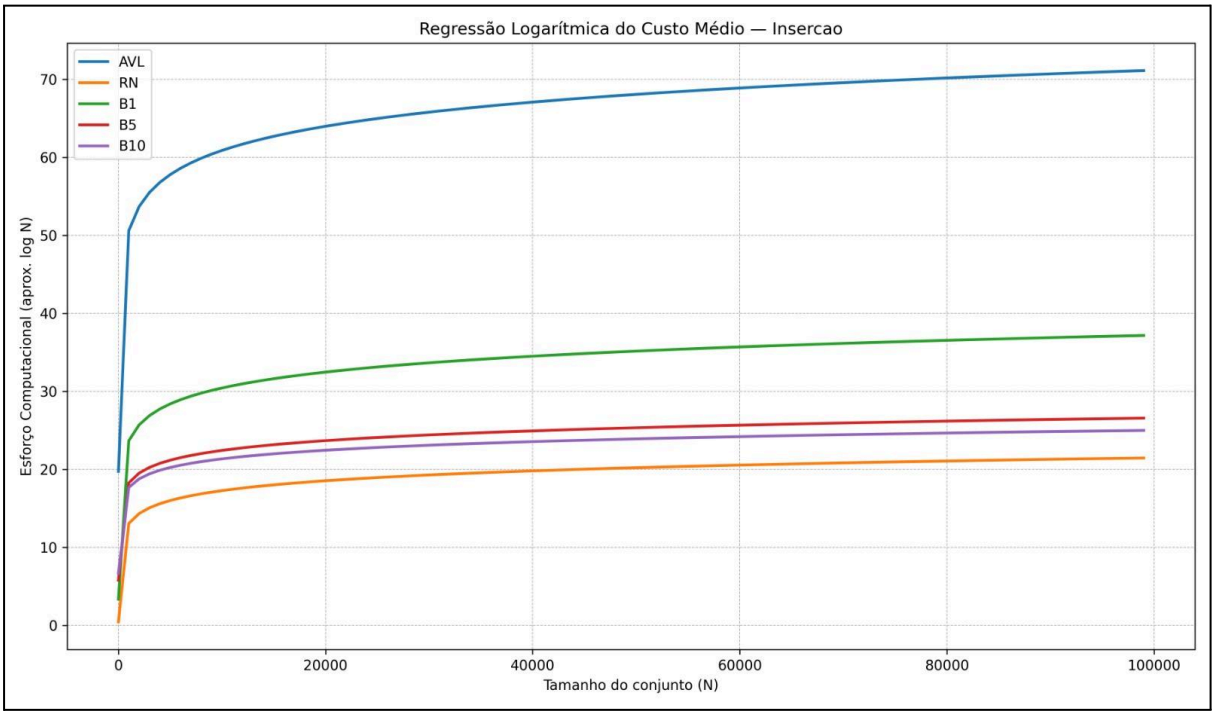
Depois da realização de todas essas etapas, a implementação começará a ser executada, gerando os dados e posteriormente exibindo os gráficos de comparação.

5. ANÁLISE DOS EXPERIMENTOS

Como descrito anteriormente, a árvore B tem menos níveis que as demais árvores devido as chaves estarem concentradas em nós ao invés de somente uma chave por nó como ocorre na Rubro-Negra e AVL. A contabilização de operações e loops seria desleal e injusta. Portanto, para tornar a comparação mais justa, foi adicionado a contabilização no loop da pesquisa binária, levando a formação de gráficos mais precisos à complexidade a eles dotadas.

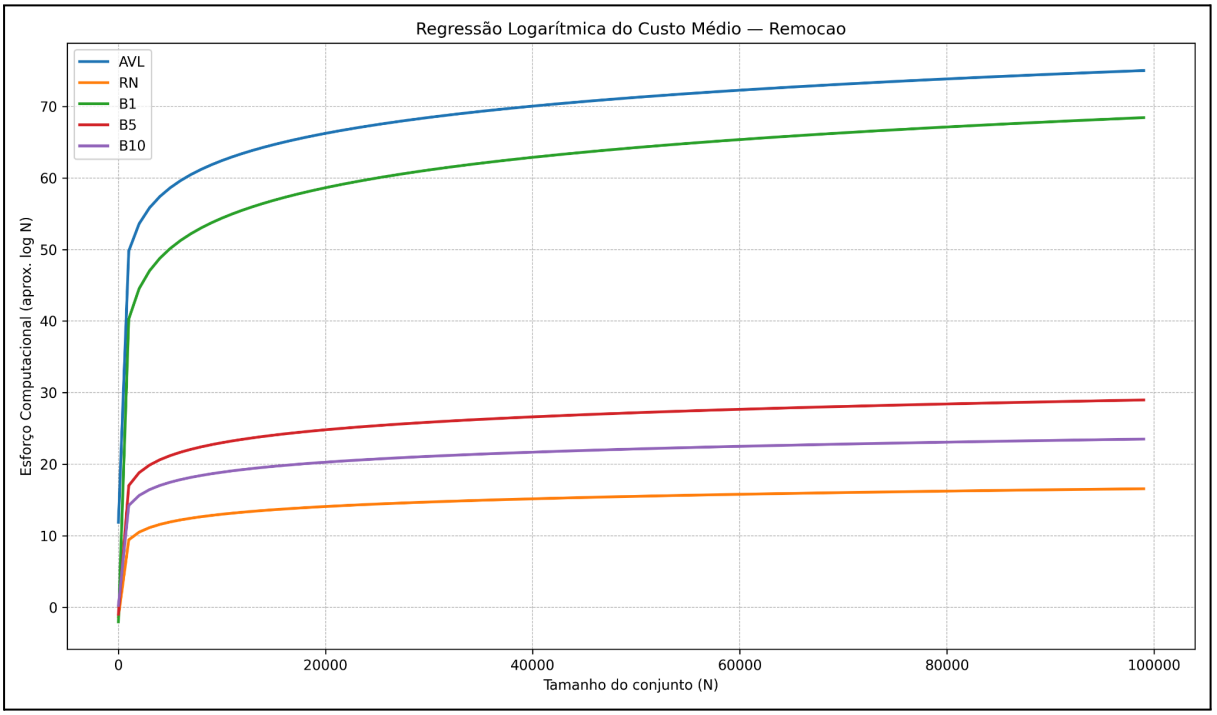
Os gráficos abaixo ilustram os custos de cada árvore para comparação entre elas. Para a maior suavidade dos gráficos foi utilizada uma estratégia de regressão logarítmica por mínimos quadrados, mitigando demasiadas oscilações nos dados e promovendo imagens mais limpas para uma melhor análise.

Figura 1 – Comparação do esforço computacional médio na operação de inserção



Fonte: Elaborado pelos autores (2025)

Figura 2 – Comparação do esforço computacional médio na operação de remoção



Fonte: Elaborado pelos autores (2025)

A ordem de mais custosa para menos custosa é: AVL, Árvore B de ordem 1, B de ordem 5, B de ordem 10 e a mais eficiente a árvore Rubro-Negra.

A árvore AVL é a mais custosa por ter um balanceamento rigoroso exigindo que a diferença de altura das sub-árvores seja no máximo 1, o que significa rotações constantes para manter a árvore balanceada, isso adiciona o custo.

Depois vem a árvore B de ordem 1, que não é derivada da binária. Por permitir apenas 2 chaves em cada nó ela precisa fazer muitas operações de *split* e *merge*, que são custosas.

Já as árvores B de ordem 5 e ordem 10 possuem nós com capacidade muito maior, 10 e 20 respectivamente, o que reduz a quantidade de operações de balanceamento necessárias.

Assim como a árvore AVL, a Rubro-Negra é uma árvore Binária porém é mais eficiente que todas as outras, porque é mais flexível, permitindo que a altura de uma sub-árvore seja muito diferente da outra, eliminando a necessidade de muitas rotações como a árvore AVL.

E geralmente as operações de balanceamento, rotações e re-colorações, são mais simples e localizadas na sub-árvore em que foi inserido ou removido um nó. O que permite o balanceamento com poucas rotações e re-colorações, assim tendo o menor custo.

Dessa forma, fica evidente que dentre as estruturas analisadas na disciplina a árvore rubro-negra é uma opção interessante para aplicações que lidam com grandes volumes de dados devido a sua engenharia rápida e econômica. Da mesma forma, fica o lembrete de que árvores derivadas das árvores binárias comuns que não utilizam regras inteligentes podem acarretar em alto custo aplicacional.

6. REFERÊNCIAS

CIFERRI, Cristina Dutra de Aguiar; PARDO, Thiago A. S. .**Árvore-B:** Remoção. São Paulo: **USP**, [S.d]. Disponível em: http://wiki.icmc.usp.br/images/b/bf/SCC0215012014p5arvoreBremocao.pdf . Acesso em: 28 out. 2025.

YU, Brian. ***Understanding B-Trees:*** *The Data Structure Behind Modern Databases. EUA: Spanning Tree, abr. 2024.* Disponível em:https://www.youtube.com/watch?v=K1a2Bk8NrYQ. Acesso em: 27 nov. 2025.

Todos os códigos e arquivos estão disponíveis no repositório do github: https://github.com/Shinerey0/Comparando-Arvores