UNIVERSIDADE FEDERAL DE ITAJUBÁ

CIÊNCIA DA COMPUTAÇÃO

ÁRVORES BALANCEADAS EM C

2024001034 - Gustavo Senador de Faria
2023012436 - Matheus Junqueira Jundurian Bolonha
2023007697 - Paulo Victor Olympio

CTCO02 - ALGORITMOS E ESTRUTURAS DE DADOS II

Prof. Vanessa Cristina

INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
UNIFEI - Itajubá

2025

UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO
CTCO02 - ALGORITMOS E ESTRUTURAS DE DADOS II

Árvores Balanceadas em C

1 Introdução

O estudo de estruturas de dados eficientes é um pilar da ciência da computação, impactando diretamente o desempenho de algoritmos e a capacidade dos sistemas em gerenciar grandes volumes de dados de maneira eficiente.

Nesse contexto, as árvores balanceadas destacam-se por manterem uma estrutura que garante operações de busca, inserção e remoção em tempo logarítmico (O(log n)), mesmo em conjuntos dinâmicos com milhares ou milhões de elementos (Cormen et al. (2012)). Ao assegurar uma altura próxima do mínimo teórico, tais estruturas buscam proporcionar eficiência e escalabilidade ao evitarem a degradação da árvore para formas lineares, característica de árvores binárias de busca não balanceadas que tornam suas operações mais ineficientes e mais custosas computacionalmente.

Dessa forma, árvores balanceadas, como as árvores 2-3-4 (árvores B de ordem 4) e árvores rubro-negras, são amplamente aplicadas em diversos sistemas computacionais. Em bancos de dados, estruturas derivadas, como árvores B e B+, otimizam o acesso a dados em disco. Já em sistemas de arquivos, elas organizam índices hierárquicos, acelerando operações de leitura e escrita, por exemplo. Compiladores e sistemas operacionais também se beneficiam dessas estruturas, utilizando árvores balanceadas para representar expressões sintáticas ou gerenciar recursos como processos e memória.

As árvores 2-3-4 são árvores de busca balanceadas caracterizadas por nós que podem armazenar 1, 2 ou 3 chaves e possuir 2, 3 ou 4 filhos, sendo essa uma estrutura que simplifica a gestão de múltiplas chaves em um único nó. Seu balanceamento é mantido por meio de operações de divisão (split) durante inserções e fusão (merge) durante operações de remoção, garantindo que a altura da árvore permaneça logarítmica, com operações de busca, inserção e remoção executadas em tempo O(log n). Essa flexibilidade de nós com múltiplos filhos reduz a frequência de reestruturações em cenários dinâmicos, como em bancos de dados, onde atualizações constantes são comuns. Além disso, a representação abstrata das árvores 2-3-4 facilita a compreensão conceitual, tornando-as ideais para modelar problemas que exigem organização hierárquica eficiente (Cormen et al. (2012)).

Por outro lado, as árvores rubro-negras são árvores binárias de busca balanceadas, onde cada nó possui uma cor distinta (vermelho ou preto) para manter as seguintes propriedades que asseguram uma altura aproximadamente logarítmica:

- Um nó é vermelho ou preto;
- A raiz é preta. (Esta regra é usada em algumas definições. Como a raiz pode sempre ser alterada de vermelho para preto, mas não sendo válido o oposto, esta regra tem pouco efeito na análise);
- Todas as folhas(nil) são pretas;
- Ambos os filhos de todos os nós vermelhos são pretos; e
- Todo caminho de um dado nó para qualquer de seus nós folhas descendentes contem o mesmo número de nós pretos.

Operações como rotações e ajustes de cor corrigem desequilíbrios durante processos de inserção e remoção, garantindo eficiência em O(log n). Sua estrutura binária é particularmente vantajosa em implementações computacionais, como em linguagem C, devido à sua compatibilidade com sistemas de memória e menor overhead em comparação com árvores de múltiplos filhos, tornando-as amplamente utilizadas em aplicações práticas, como sistemas de arquivos e bibliotecas de software, onde a simplicidade de manipulação de nós binários otimiza o desempenho.

Embora estruturalmente distintas, as árvores 2-3-4 e rubro-negras apresentam uma equivalência funcional, pois ambas garantem balanceamento e operações em tempo logarítmico. Essa equivalência reside na possibilidade de representar cada nó de uma árvore 2-3-4, com até três chaves, como uma árvore rubro-negra, onde nós vermelhos simulam as conexões internas de um nó 2-3-4, e nós pretos mantêm a estrutura balanceada. Essa correspondência permite a conversão direta entre as estruturas, preservando suas propriedades de eficiência. Este trabalho implementa, em linguagem C, um sistema que realiza essa conversão, incluindo operações de inserção, remoção e impressão, além de análises estatísticas do comportamento da árvore 2-3-4, explorando a relevância prática e teórica dessa transformação em sistemas computacionais modernos.

2 Desenvolvimento

Todo o código desenvolvido pode ser acessado no Github: https://github.com/Astrogusss/TrabalhoALG2-2/tree/main.

2.1 Algoritmo de Conversão

A conversão de uma árvore 2-3-4 para uma árvore rubro-negra é um processo fundamentado na equivalência entre essas duas estruturas. Na prática, cada tipo de nó na árvore 2-3-4 é representado por uma configuração específica de nós e cores na árvore rubro-negra:

- Nó 2 (1 chave) corresponde a um nó preto com até dois filhos;
- Nó 3 (2 chaves): corresponde a um nó preto com um filho vermelho à direita; e
- Nó 4 (3 chaves): corresponde a um nó preto com dois filhos vermelhos.

A Figura 1 apresenta as relações de equivalência entre nós da árvore 2-3-4 e configurações de nós da árvore rubro-negra.

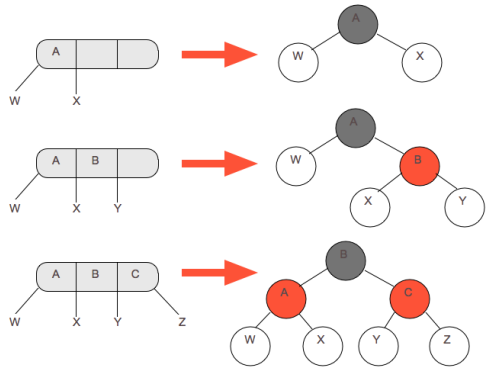

Figura 1: Relações de equivalência entre nós da árvore 2-3-4 e configurações de nós da árvore rubro-negra. Disponível em: https://dtkachenko.blogspot.com/2011/11/multi-way-trees-and-external-storage.html?m=1. Último acesso em 02/07/2025.

Dito isso, o algoritmo de conversão, representado pelo Algoritmo 1, segue uma abordagem recursiva, construindo a árvore rubro-negra a partir da raiz da árvore 2-3-4. Primeiramente, a função de conversão recebe como parâmetro dois ponteiros de raiz (um para a raiz da árvore 2-3-4 e um para a raiz da árvore rubro-negra). A partir disso, para cada nó da árvore 2-3-4, o algoritmo realiza as seguintes etapas: primeiramente, verifica a quantidade de chaves presentes no nó da árvore 2-3-4. Se o nó possui apenas uma chave (nó 2), um nó preto é criado e seus filhos são convertidos recursivamente; se o nó possui duas chaves (nó 3), o algoritmo cria um nó preto e um filho vermelho à sua direita, inserindo duas chaves e convertendo seus filhos recursivamente; e se o nó possui três chaves (nó 4), é criado um nó preto com dois filhos vermelhos, convertendo cada parte da subárvore recursivamente.

O algoritmo faz com que cada nó criado possua um ponteiro específico para o próprio pai, mantendo a integridade da estrutura da árvore recém-convertida nessa atribuição. Finalmente, cada uma das subárvores da árvore 2-3-4 é processada da mesma maneira recursivamente, garantindo que toda sua estrutura seja convertida corretamente.

Algorithm 1: Conversão de uma árvore 2-3-4 para árvore Rubro-Negra

```
1  Função converte234(raiz234, pai):
2      if raiz234 é nulo then
3          return NULL
4      end
5      RaizRubro ← NULL;
6      chaves ← obtemChaves(raiz234);
7      filhos ← obtemFilhos(raiz234);
8      quantidadeKey ← obtemQtdChaves(raiz234);
9      switch quantidadeKey do
10         case 1 do
11             RaizRubro ← novo nó com chave
                chaves[0], cor preta, pai = pai;
12             RaizRubro.esq ← converte234(filhos[0],
                RaizRubro);
13             RaizRubro.dir ← converte234(filhos[1],
                RaizRubro);
14         end
15         case 2 do
16             RaizRubro ← novo nó com chave
                chaves[0], cor preta, pai = pai;
17             vermelho ← novo nó com chave chaves[1],
                cor vermelha, pai = RaizRubro;
18             RaizRubro.esq ← converte234(filhos[0],
                RaizRubro);
19             RaizRubro.dir ← vermelho;
20             vermelho.esq ← converte234(filhos[1],
                vermelho);
21             vermelho.dir ← converte234(filhos[2],
                vermelho);
22         end
23         case 3 do
24             RaizRubro ← novo nó com chave
                chaves[1], cor preta, pai = pai;
25             vermelhoEsq ← novo nó com chave
                chaves[0], cor vermelha, pai = RaizRubro;
26             vermelhoDir ← novo nó com chave
                chaves[2], cor vermelha, pai = RaizRubro;
27             RaizRubro.esq ← vermelhoEsq;
28             RaizRubro.dir ← vermelhoDir;
29             vermelhoEsq.esq ←
                converte234(filhos[0], vermelhoEsq);
30             vermelhoEsq.dir ←
                converte234(filhos[1], vermelhoEsq);
31             vermelhoDir.esq ←
                converte234(filhos[2], vermelhoDir);
32             vermelhoDir.dir ← converte234(filhos[3],
                vermelhoDir);
33         end
34     end
35     return RaizRubro
```

2.2 Ambiente de Desenvolvimento

- **Hardware:** Intel Core i5-1135G7 e 8 GB de memória RAM;
- **Sistema Operacional:** Windows 11 64-bit;
- **Linguagem de Programação:** C; e

- **Ferramentas de Desenvolvimento:** Visual Studio Code, versão 1.100.3, GitHub.

3 Benchmarking

Após o desenvolvimento da árvore 2-3-4, foram realizados alguns testes de desempenho no intuito de avaliar e analisar o comportamento da estrutura em diferentes contextos. A princípio, foram analisados os comportamentos da árvore perante a entradas de valores inteiros não ordenados de diversos tamanhos (100, 1.000, 10.000 e 100.000 elementos). Os aspectos escolhidos para análise foram:

- Tamanho da amostra;
- Quantidade de operações de splits;
- Altura final da árvore obtida; e
- Total de blocos ocupados (no contexto desse trabalho, o total de blocos ocupados foi interpretado como a quantidade de nós obtidos pós-inserção).

Os resultados obtidos estão organizados na Tabela 1.

Tamanho da Amostra	Quantidade de Splits	Altura da Árvore	Blocos Ocupados
100	44	4	48
1000	512	6	518
10000	5213	9	5222
100000	50654	11	50665

Tabela 1: Resultados dos testes de inserção.

Além da análise de inserção, foi realizada uma série de experimentos para avaliar o comportamento da árvore 2-3-4 durante operações de remoções sucessivas. Os testes consideraram a exclusão de 10, 20, 35 e 50 porcento dos elementos de uma árvore previamente construída com 10.000 elementos. Foram registradas os seguintes dados:

- Quantidades de operações de rotações;
- Quantidade de operações de merge;
- Altura final; e
- Total de blocos ocupados após cada percentual de remoção (no contexto desse trabalho, o total de blocos ocupados foi interpretado como a quantidade de nós obtidos pós-remoção).

Os resultados estão demonstrados na Tabela 2:

Percentual de Remoção	Quantidade de Rotações	Quantidade de Merges	Altura Final	Total de Blocos Final
10	117	78	9	5076
20	361	268	9	4919
35	735	774	9	4425
50	1178	1555	9	3650

Tabela 2: Resultados obtidos dos testes de remoção.

4 Discussões

A análise dos dados extraídos da Tabela 1 (operações de inserção) revela um crescimento proporcional e consistente das métricas em relação ao aumento do número de elementos inseridos. A quantidade de splits cresce quase que linearmente com o tamanho da amostra, sendo coerente com a natureza das árvores 2-3-4, que realizam divisões (splits) para manter seu balanceamento conforme os nós se tornam cheios. Já a altura da árvore cresce de forma logarítmica, como esperado para árvores balanceadas, evidenciando o excelente desempenho estrutural da árvore 2-3-4 mesmo perante grandes volumes de dados. Com 100.000 elementos, a altura da árvore é de apenas 11, demonstrando que a estrutura consegue preservar o balanceamento de forma eficiente. Em relação ao número total de blocos ocupados (neste caso, quantidade de nós), observa-se uma correspondência quase direta com a quantidade de elementos inseridos. Isso sugere uma distribuição adequada dos elementos entre os nós, sem excessiva fragmentação ou sobrecarga da árvore.

Já os dados exibidos na Tabela 2 (operações de remoção) revelam que a altura da árvore permaneceu constante (9) independentemente do percentual de remoção aplicado, evidenciando a capacidade da estrutura da árvore de manter seu balanceamento mesmo após remoções volumosas. No entanto, observa-se um crescimento expressivo nas operações de rotações e merges à medida que o percentual de remoção aumentava. Enquanto 10 porcento de remoção amostral exigiram apenas 117 rotações e 78 merges, a exclusão de 50 porcento dos elementos acabou por demandar 1.178 rotações e 1.555 merges. Esses dados indicam que o custo computacional das operações de remoções é elevado, especialmente quando há necessidade de reestruturações cada vez mais complexas para manter a consistência da árvore. Já a quantidade de blocos ocupados pela árvore possuiu uma redução progressiva junto ao aumento das remoções, demonstrando o esvaziamento da árvore. Mesmo com 50 porcento dos dados removidos, a árvore manteve sua estrutura robusta e bem distribuída, com 3.650 blocos ainda ocupados.

De forma geral, os resultados analisados sugerem que a árvore 2-3-4 é capaz de oferecer um excelente desempenho em operações de inserção, mesmo com volumes elevados de dados, mantendo sua altura reduzida e uma distribuição equilibrada de elementos entre os blocos ocupados. Além disso, os testes demonstram que a estrutura mantém seu balanceamento de forma eficaz também durante operações de remoção, ainda que essas exijam um custo computacional mais elevado devido à realização intensiva de rotações e merges. Esses dados reforçam a robustez e a eficiência da árvore 2-3-4 como uma solução viável e eficiente para o gerenciamento dinâmico de conjuntos de dados em diversas aplicações.

5 Considerações Finais

A realização deste trabalho possibilitou não apenas o aprofundamento teórico no estudo das árvores balanceadas, mas também a aplicação prática dos conceitos por meio da implementação de um sistema funcional em linguagem C. A conversão entre árvores 2-3-4 e rubro-negras revelou-se uma excelente oportunidade para consolidar o entendimento sobre as propriedades estruturais que garantem o balanceamento e

a eficiência dessas representações, demonstrando-se didaticamente eficaz para a compreensão de como diferentes modelos de estruturação de dados podem representar o mesmo conjunto de informações com variações específicas de desempenho e simplicidade computacional. Além disso, os testes realizados demonstraram a robustez da árvore 2-3-4 frente a operações intensivas de inserção e remoção, validando seu uso em contextos que exigem desempenho consistente e estrutura estável. Ainda que as remoções apresentem maior custo interno, a árvore manteve suas propriedades de balanceamento, o que reforça sua aplicabilidade em sistemas reais.

Por fim, o trabalho contribuiu para o desenvolvimento de habilidades técnicas relevantes, como o domínio de estruturas dinâmicas em linguagem C, o uso de estratégias recursivas para manipulação de árvores e a análise crítica de desempenho por meio de benchmarking. Essa experiência será valiosa para futuros projetos que demandem a modelagem e a manipulação eficiente de grandes volumes de dados.

Referências

Cormen, T. H., Leiserson, C. E., & Clifford Stein, R. L. R. (2012). *Algoritmos: teoria e prática.* Elsevier.