UNIVERSIDADE FEDERAL DE ITAJUBÁ

CIÊNCIA DA COMPUTAÇÃO

CONVERSÃO DE ÁRVORES 2-3-4 EM ÁRVORES RUBRO-NEGRAS: IMPLEMENTAÇÃO E ANÁLISE EXPERIMENTAL

2024008886 - Rodolfo Henrique Faria
2024004564 - Rafael Santos Pinto Batista Leite

CTCO02 - ALGORITMOS E ESTRUTURA DE DADOS II

Profa. Vanessa Cristina Oliveira de Souza

INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
UNIFEI - Itajubá

2025

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

Conversão de Árvores 2-3-4 em Árvores Rubro-Negras: Implementação e Análise Experimental

1 Introdução

A busca por eficiência no armazenamento e recuperação de dados é um dos pilares da ciência da computação. Estruturas de dados que permitem a realização de operações como busca, inserção e remoção de forma rápida são essenciais para o desenvolvimento de aplicações de alto desempenho. Dentre as estruturas mais eficientes para este fim, destacam-se as árvores de busca balanceadas.

Este trabalho explora duas dessas estruturas: as árvores 2-3-4 e as árvores Rubro-Negras. As árvores 2-3-4 são árvores B de ordem 4, onde cada nó pode armazenar de uma a três chaves e ter de dois a quatro filhos. As árvores Rubro-Negras, por sua vez, são árvores binárias de busca que mantêm o balanceamento através de um conjunto de regras baseadas na coloração (vermelho ou preto) de seus nós.

É sabido que existe uma equivalência direta entre essas duas estruturas de dados. Para cada árvore 2-3-4, existe pelo menos uma árvore Rubro-Negra correspondente que preserva a ordem e o balanceamento das chaves. Esta relação permite que uma árvore 2-3-4 seja representada por uma árvore Rubro-Negra.

O objetivo principal deste trabalho, portanto, é estudar essa equivalência e implementar um algoritmo, na linguagem de programação C, que realize a conversão de uma árvore 2-3-4 em uma árvore Rubro-Negra válida. Além da conversão, o sistema desenvolvido permitirá a manipulação de ambas as estruturas, incluindo operações de inserção e remoção de elementos. Adicionalmente, será conduzida uma análise de desempenho sobre a árvore 2-3-4, observando métricas como altura da árvore, quantidade de *splits* e blocos de memória ocupados durante operações de inserção e remoção em massa.

Nas próximas seções detalharemos o referencial teórico, a implementação, os testes e as conclusões.

2 Referencial Teórico

Em cenários de sistemas críticos, nos quais o tempo de resposta e a eficiência no acesso a dados são determinantes, estruturas de dados balanceadas tornam-se imprescindíveis. As árvores B foram introduzidas por Rudolf Bayer e Edward M. McCreight em 1970 no contexto de armazenamento em memória secundária, visando minimizar o número de acessos ao disco durante operações de busca, inserção e remoção Bayer & McCreight (1970). Diferentemente das árvores binárias de busca, as árvores B permitem que cada nó armazene múltiplas chaves e referências a filhos, reduzindo a altura da árvore e garantindo desempenho logarítmico nas operações básicas.

Uma árvore 2-3-4 é um caso particular de árvore B de ordem 4, em que cada nó interno possui entre 1 e 3 chaves e, consequentemente, de 2 a 4 filhos. Suas principais propriedades são:

- **Capacidade de chave**: cada nó armazena de 1 a 3 chaves.
- **Relacionamento pai-filho**: um nó com k chaves tem exatamente $k + 1$ referências a filhos.
- **Balanceamento estrito**: todas as folhas residem no mesmo nível, assegurando profundidade uniforme.
- **Operação de divisão (split)**: quando um nó atinge o máximo de três chaves e sofre nova inserção, ele é dividido em dois nós — a chave mediana é promovida ao nó-pai, preservando as propriedades da árvore.
- **Operação de fusão (merge)**: na remoção, se um nó ficar abaixo do limiar mínimo de chaves, ele pode se fundir com um nó-irmão ou emprestar uma chave do pai, mantendo as invariantes.

Graças a essas características, as árvores 2-3-4 exibem altura $h = O(\log_4 N)$, onde N é o número de elementos, o que as torna especialmente adequadas para aplicações que demandam acesso em bloco, como bancos de dados e sistemas de arquivos, onde o custo de I/O amortizado é crítico.

No mesmo ano de 1972, Bayer observou que certas operações em árvores 2-3-4 podiam ser reinterpretadas como manipulações de coloração em estruturas binárias, o que levou à formalização das chamadas “árvores rubro-negras” (inicialmente nomeadas de “árvores binárias B simétricas”) (Guibas & Sedgewick, 1978). Árvores rubro-negras são árvores binárias de busca balanceadas cujos nós são coloridos de vermelho ou preto, de modo a impor restrições que garantem balanceamento aproximado e eficiência logarítmica nas operações básicas.

As propriedades fundamentais de uma árvore rubro-negra são:

- **Raiz preta**: a raiz da árvore é sempre preta.
- **Cores de nós**: cada nó é vermelho ou preto.
- **Folhas pretas (nil)**: todas as folhas externas (nodos `NULL`) são consideradas pretas.
- **Filhos de nós vermelhos**: se um nó é vermelho, então ambos os seus filhos são pretos.

- **Uniformidade da altura preta**: em qualquer caminho da raiz a uma folha, o número de nós pretos (altura de preto) é constante.

Essas regras asseguram que a altura da árvore seja $h = O(\log_2 N)$, onde N é o número de elementos, e oferecem garantia de pior caso para busca, inserção e remoção (Cormen et al., 2022). O mecanismo de recolorir nós e realizar rotações simples ou duplas durante as operações mantém a árvore aproximadamente balanceada, sem a necessidade de estruturas *multiway* explícitas.

Dada a equivalência entre árvores 2-3-4 e árvores rubro-negras, é possível conceber um algoritmo de conversão que, a partir de uma árvore 2-3-4, constrói uma árvore binária rubro-negra válida, preservando todas as suas propriedades de ordenação e balanceamento. Nosso objetivo é apresentar esse algoritmo em detalhes e demonstrar seu funcionamento na prática. Para tanto, realizaremos também uma análise experimental do desempenho de uma árvore 2-3-4, conduzindo testes de inserção e remoção em massa sobre conjuntos de tamanhos e características diversas, de modo a avaliar métricas como a altura da árvore, o número de splits e merges, e o uso de memória.

3 Desenvolvimento

3.1 Árvores 2-3-4

Uma árvore 2-3-4 é uma árvore de busca multi-vias que se mantém perfeitamente balanceada, o que significa que todas as suas folhas estão no mesmo nível. O nome "2-3-4" refere-se ao número de filhos que um nó pode ter. As regras que definem sua estrutura são:

- **Nó-2:** Contém um item de dado e possui dois filhos.
- **Nó-3:** Contém dois itens de dado e possui três filhos.
- **Nó-4:** Contém três itens de dado e possui quatro filhos.

Em qualquer nó que não seja uma folha, a quantidade de filhos é sempre igual à quantidade de itens de dado mais um. A organização das chaves dentro de um nó e a relação com seus filhos seguem uma regra de ordem estrita. Para um nó com chaves K_1, K_2, ..., K_n, a subárvore à esquerda de K_1 contém apenas chaves menores que K_1; a subárvore entre K_1 e K_2 contém chaves maiores que K_1 e menores que K_2, e assim por diante. A subárvore à direita de K_n contém apenas chaves maiores que K_n.

O balanceamento da árvore 2-3-4 é mantido durante a operação de inserção através de um mecanismo chamado *split de nó*. Antes de inserir um novo item, o algoritmo percorre a árvore a partir da raiz. Se um nó cheio (um nó-4) é encontrado no caminho, ele é dividido. A divisão de um nó-4 (com chaves A, B e C) consiste em:

- Mover a chave do meio (B) para o nó pai;
- Dividir o nó-4 em dois novos nós-2: a chave A permanece no nó original e a chave C é movida para um novo nó-2, à direita do original;
- Os dois filhos mais à direita do nó-4 original são associados ao novo nó-2 da direita.

Este processo garante que, ao chegar em uma folha para realizar a inserção, sempre haverá espaço para o novo item, e a altura da árvore só aumenta quando a raiz é dividida.

3.2 Árvores Rubro-Negras

Uma árvore Rubro-Negra é uma árvore binária de busca que se mantém aproximadamente balanceada através da aplicação de um conjunto de regras baseadas em cores. Cada nó possui um atributo de cor, que pode ser vermelho ou preto. O balanceamento é garantido pela observância das seguintes regras:

- **Regra 1:** Todo nó é vermelho ou preto;
- **Regra 2:** A raiz é sempre preta;
- **Regra 3:** Se um nó é vermelho, seus filhos devem ser pretos (não podem existir dois nós vermelhos consecutivos em um caminho da raiz para uma folha);
- **Regra 4:** Todo caminho da raiz até uma folha (ou um filho nulo) deve conter o mesmo número de nós pretos.

A manutenção dessas regras durante as operações de inserção e remoção é feita através de *flips de cores* e *rotações* (simples ou duplas), que reestruturam a árvore localmente para preservar o balanceamento global.

3.3 Equivalência e Conversão

As árvores 2-3-4 e as Rubro-Negras são estruturas de dados isomórficas, ou seja, existe uma correspondência direta entre elas. Conforme descrito por Lafore (2003, p. 486), é possível transformar uma árvore 2-3-4 em uma árvore Rubro-Negra aplicando um conjunto simples de regras:

- Um nó-2 em uma árvore 2-3-4 corresponde a um único nó preto em uma árvore Rubro-Negra;
- Um nó-3 corresponde a um pai preto com um filho vermelho. A chave menor pode ser o filho esquerdo ou a maior ser o filho direito; ambas as configurações são válidas;
- Um nó-4 corresponde a um pai preto com dois filhos vermelhos. A chave do meio torna-se o pai preto.

Esta transformação garante que as regras da árvore Rubro-Negra sejam preservadas. A Regra 3 (um nó vermelho não pode ter filhos vermelhos) é garantida porque os filhos vermelhos na árvore Rubro-Negra sempre se originam de um mesmo nó-3 ou nó-4, e seus pais na árvore Rubro-Negra são sempre pretos. A Regra 4 (mesma "altura preta" em todos os caminhos) é mantida porque cada nó na árvore 2-3-4 corresponde a exatamente um nó preto na árvore Rubro-Negra, e como todas as folhas da 2-3-4 estão no mesmo nível, todos os caminhos na Rubro-Negra terão o mesmo número de nós pretos.

As operações de balanceamento também são equivalentes. O *split* de um nó-4 em uma árvore 2-3-4 corresponde a um *flip de cores* em uma árvore Rubro-Negra. As rotações em uma árvore Rubro-Negra correspondem à escolha de qual item de um nó-3 se torna o pai na transformação.

3.4 Algoritmo de Conversão

A conversão de uma árvore 2-3-4 para uma árvore Rubro-Negra se baseia na equivalência estrutural e operacional entre essas duas importantes estruturas de dados balanceadas. Embora em um primeiro momento possam parecer entidades distintas, elas são, em certo sentido, completamente equivalentes, o que permite que uma seja transformada na outra através de um conjunto de regras de mapeamento. Essa equivalência é fundamental não apenas para a compreensão teórica, mas também para a implementação prática da conversão. Historicamente, as árvores 2-3-4 precederam as árvores Rubro-Negras, que evoluíram a partir delas.

O algoritmo de conversão percorre a árvore 2-3-4, nó a nó, aplicando regras específicas de transformação para gerar a estrutura correspondente na árvore Rubro-Negra. A natureza recursiva das árvores torna um percurso recursivo o método mais natural para realizar essa conversão.

Regras de Transformação de 2-3-4 para Rubro-Negra

A transformação de uma árvore 2-3-4 em uma árvore Rubro-Negra é realizada aplicando-se três regras principais, conforme ilustrado na Figura 1:

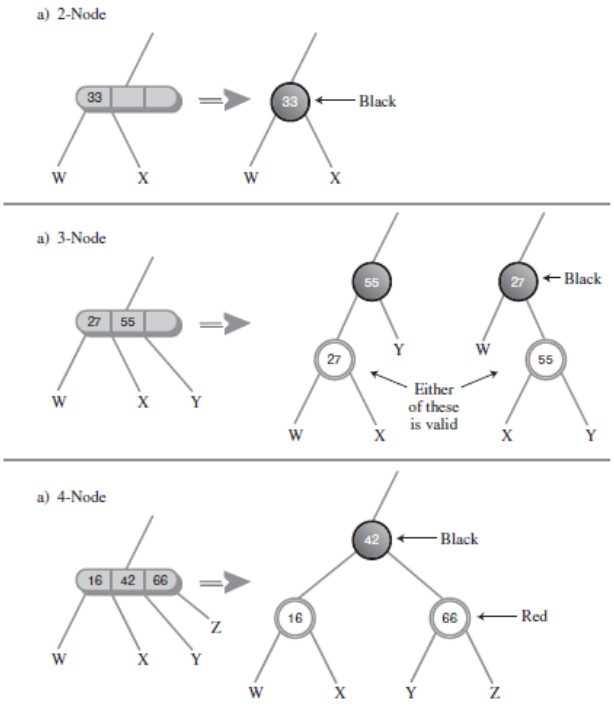

Figura 1: Regras de Transformação de Nós de uma Árvore 2-3-4 para Árvore Rubro-Negra (Figura retirada de Lafore (2017))

- **Transformação de Nó-2:** Um nó-2 em uma árvore 2-3-4, que contém um único item de dado (ex: 33), é convertido em um nó preto na árvore Rubro-Negra. Os dois filhos do nó-2 tornam-se os filhos esquerdo e direito do novo nó preto. A Figura 1 exemplifica essa conversão, onde um nó-2 com a chave 33 e filhos W e X se torna um nó preto 33 com os mesmos filhos.
- **Transformação de Nó-3:** Um nó-3 em uma árvore 2-3-4, contendo dois itens de dado (ex: '27' e '55'), é transformado em um nó pai preto e um nó filho vermelho. Os itens de dado do nó-3 são distribuídos entre esses dois novos nós. É importante notar que existem duas configurações válidas para essa transformação: a chave menor pode ser o pai preto com a chave maior como filho vermelho, ou vice-versa. A Figura 1b ilustra essas duas opções, mostrando como '27' e '55' podem ser organizados de forma a manter um pai preto e um filho vermelho. Essa flexibilidade na escolha de qual item se torna o pai está diretamente relacionada às rotações em árvores Rubro-Negras.
- **Transformação de Nó-4:** Um nó-4 em uma árvore 2-3-4, que possui três itens de dado (ex: '16', '42' e '66'), é convertido em um nó pai preto com dois nós filhos vermelhos. A chave do meio do nó-4 (ex: '42') torna-se o nó pai preto, enquanto as chaves restantes (ex: '16' e '66') se tornam os filhos esquerdo e direito do pai preto, respectivamente, sendo ambos coloridos de vermelho. Os quatro filhos originais do nó-4 são distribuídos entre os dois novos nós filhos vermelhos. A Figura 1c demonstra essa transformação.

A aplicação consistente dessas regras garante que as propriedades fundamentais das árvores Rubro-Negras sejam automaticamente satisfeitas. Isso inclui a Regra 3 (um nó vermelho não pode ter filhos vermelhos), pois os filhos vermelhos na árvore Rubro-Negra sempre se originam de um nó-3 ou nó-4 e são filhos de um nó preto. A Regra 4 (o mesmo número de nós pretos em todos os caminhos da raiz para uma folha ou filho nulo) também é mantida, uma vez que cada nó da árvore 2-3-4 corresponde a exatamente um nó preto na árvore Rubro-Negra, e todas as folhas da árvore 2-3-4 estão no mesmo nível. A Figura 2 fornece um exemplo visual de uma árvore 2-3-4 completa e sua respectiva árvore Rubro-Negra após a aplicação dessas transformações.

Equivalência Operacional: Splits e Flips de Cores

A correspondência entre as estruturas se estende às operações de balanceamento. O processo de split de nós em uma árvore 2-3-4, que ocorre durante a inserção, tem uma equivalência direta com as operações de color flip (viradas de cor) em árvores Rubro-Negras.

Quando um nó-4 é encontrado e dividido durante um percurso de descida na árvore 2-3-4 (3a), ele se transforma em dois nós-2, e a chave do meio é promovida para o nó pai (3b). Na árvore Rubro-Negra equivalente, essa operação corresponde a um color flip. Se um nó preto possui dois filhos vermelhos (3c), o color flip inverte as cores: o nó pai se torna vermelho e seus filhos se tornam pretos. Esta operação, embora altere as cores, mantém inalterado o número de nós pretos no caminho da raiz até as folhas, garantindo a Regra 4 da árvore Rubro-Negra. O resultado do color flip (Figura 3d) reflete a mesma estrutura de um nó-3 formado por um split de nó-4 na árvore 2-3-4, com um nó pai vermelho e um filho vermelho.

Equivalência Operacional: Nó-3 e Rotações

As rotações, cruciais para o balanceamento de árvores Rubro-Negras, também encontram sua contraparte na equi-

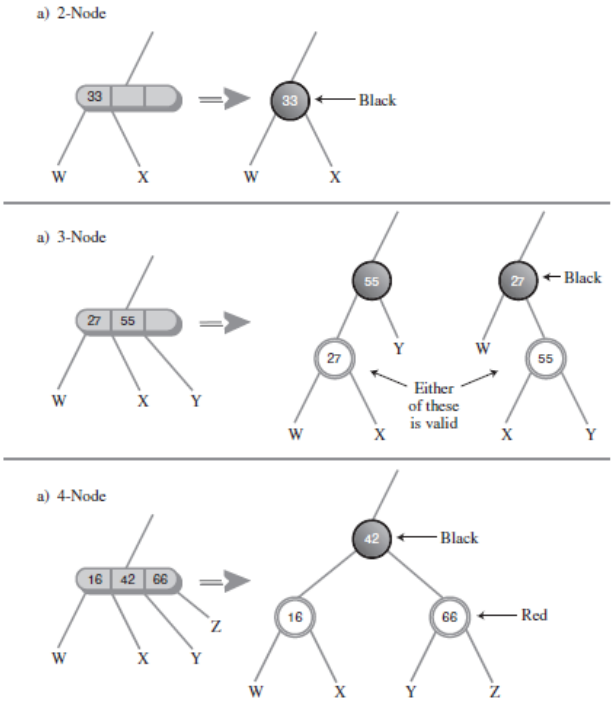

Figura 2: Exemplo Completo de Conversão de Árvore 2-3-4 para Rubro-Negra (Figura retirada de Lafore (2017))

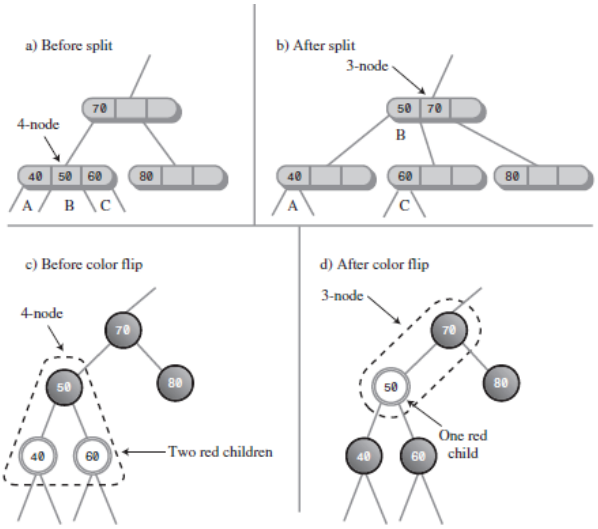

Figura 3: Equivalência entre Split de Nó-4 (2-3-4) e Color Flip (Rubro-Negra) (Figura retirada de Lafore (2017))

valência com as árvores 2-3-4. Conforme observado na transformação de um nó-3 (Figura 1b), há duas maneiras válidas de configurar o pai preto e o filho vermelho. Essas diferentes "inclinações"da conexão vermelha na árvore Rubro-Negra são importantes.

A Figura 4a mostra uma árvore 2-3-4, e as Figuras 4b e 4c apresentam duas árvores Rubro-Negras equivalentes. A diferença entre elas reside na escolha de qual dos dois itens de dado do nó-3 (aqui, o nó 70/80) se torna o pai na transformação. Se 80 é o pai (Figura 4b), a árvore resultante pode estar desbalanceada localmente. Uma rotação à direita (e mudanças de cor apropriadas) nessa árvore Rubro-Negra transforma-a na configuração balanceada mostrada na Figura 4c, onde 70 é o pai. Isso demonstra que uma rotação em uma árvore Rubro-

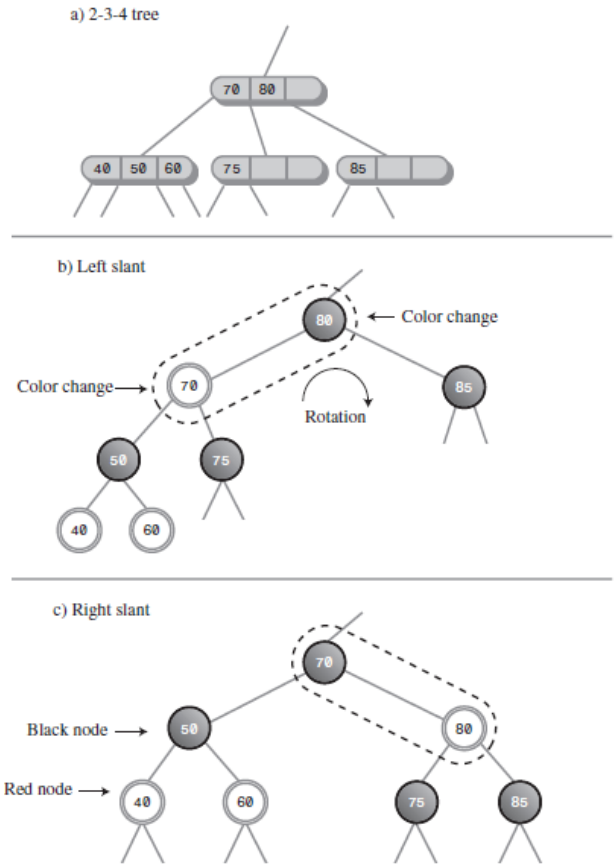

Figura 4: Equivalência entre Diferentes Configurações de Nó-3 (2-3-4) e Rotações (Rubro-Negra) (Figura retirada de Lafore (2017))

Negra é equivalente à escolha da configuração do pai ao transformar um nó-3 de uma árvore 2-3-4. Uma equivalência similar também pode ser observada para rotações duplas necessárias para "netos internos"(inside grandchildren).

3.5 Implementação

Para a implementação da função de conversão, utilizamos duas rotinas. A primeira, mostrada no Algoritmo 1, atua como auxiliar: ela aloca a árvore rubro-negra que receberá os nós da árvore 2-3-4, realiza os ajustes iniciais e, em seguida, invoca a função recursiva responsável por converter efetivamente cada nó.

O Algoritmo 1 inicia verificando se a árvore 2-3-4 de entrada é nula, retornando *NULL* nesse caso. Caso contrário, aloca uma nova árvore Rubro-Negra vazia e obtém a raiz da estrutura 2-3-4 para converter recursivamente seus nós por meio da função *converteNo*, presente no Algoritmo 2. Ao final, a raiz resultante é vinculada à nova árvore RB através de *rb_setRaiz* e a árvore completamente montada é retornada.

Como citado anteriormente, a função *converteNo*, Algoritmo 2, é recursiva e recebe dois parâmetros: um ponteiro para a árvore Rubro-Negra de destino (*arvoreRB*) e um ponteiro para o nó da árvore 2-3-4 (*no234*). No caso base, quando *no234* é nulo, a rotina retorna imediatamente a *sentinelaFolha* da árvore Rubro-Negra. Caso contrário, obtém–se *numChaves*, o número de chaves em *no234*, e seleciona-se um entre três subalgoritmos de conversão (para 1, 2 ou 3 chaves). Em

Algorithm 1: Converte234ParaRB

Data: *arv234Orig*: ponteiro para árvore 2-3-4
Result: *novaArv*: ponteiro para árvore Rubro-Negra

```
1  if arv234Orig = NULL then
2  |   return NULL;
3  end
4  novaArv ← rb_alocaArvore();
5  if novaArv = NULL then
6  |   return NULL;
7  end
8  raiz234 ← getRaiz(arv234Orig);
9  raizRB ← converteNo(novaArv, raiz234);
10 rb_setRaiz(novaArv, raizRB);
11 return novaArv;
```

cada subcaso são alocados os nós RB correspondentes, atribuídas as cores (preto ou vermelho), conectadas recursivamente as subárvores filhas e ajustados os ponteiros de pai, garantindo a equivalência estrutural entre a árvore 2-3-4 e a árvore Rubro-Negra resultante.

4 Análise da Árvore 2-3-4

Para analisar o desempenho da árvore 2-3-4 em diferentes cenários, realizamos uma avaliação em duas etapas. Na primeira, medimos o desempenho da árvore na inserção de conjuntos de dados de diferentes tamanhos: 100, 1000, 10000 e 100000 elementos. Em seguida, avaliamos a eficiência da árvore na remoção de elementos, utilizando-se um único conjunto de 10000 elementos.

Ambiente de Testes

Os experimentos foram realizados em um notebook ASUSTeK X510URR equipado com processador Intel® Core™ i5-8250U (quatro núcleos físicos, oito threads), 8 GB de memória RAM, placa gráfica integrada Intel® UHD Graphics 620 e placa dedicada NVIDIA GeForce 930MX, com 1,1 TB de armazenamento. O sistema operacional utilizado foi o Ubuntu 24.04.2 LTS (kernel 6.11.0-28-generic) executando sobre X11.

A linguagem utilizada para o desenvolvimento foi C, com o compilador `gcc` na versão 13.3.0 (Ubuntu 13.3.0-6ubuntu2~24.04). O desenvolvimento foi realizado de forma mista utilizando os ambientes *Visual Studio Code* e *CLion*.

4.1 Análise da inserção

No experimento de inserção, avaliamos métricas como o número de operações de split realizadas, a altura final da árvore e a quantidade de blocos ocupados. Para calcular a quantidade de blocos ocupados, consideramos blocos de disco de 512 bytes. Em nossa implementação, cada nó da árvore 2-3-4 tem tamanho aproximado de 64 bytes, permitindo armazenar cerca de 8 nós por bloco.

A Tabela 1 apresenta os resultados obtidos no experimento de inserção

Algorithm 2: ConverteNo

Data: *arvoreRB*: ponteiro para árvore RB, *no234*: ponteiro para nó 2-3-4
Result: *novoNo*: ponteiro para nó (ou subárvore) RB

```
1  if no234 = NULL then
2  |   return arvoreRB → sentinelaFolha;
3  end
4  numChaves ← no234 → qtd_chaves;
5  if numChaves = 1 then
6  |   novoNo ← rb_alocaNo(arvoreRB, no234 →
   |     chaves[0]);
7  |   rb_setCor(novoNo, 'P');
8  |   novoNo → esq ← converteNo(arvoreRB, no234 →
   |     filhos[0]);
9  |   novoNo → dir ← converteNo(arvoreRB, no234 →
   |     filhos[1]);
10 |   AjustaPais(novoNo);
11 |   return novoNo;
12 else if numChaves = 2 then
13 |   noPai ← rb_alocaNo(arvoreRB, no234 →
   |     chaves[1]);   rb_setCor(noPai, 'P');
14 |   noEsq ← rb_alocaNo(arvoreRB, no234 →
   |     chaves[0]);   rb_setCor(noEsq, 'V');
15 |   noPai → esq ← noEsq;
16 |   noEsq → pai ← noPai;
17 |   noPai → dir ← converteNo(arvoreRB, no234 →
   |     filhos[2]);
18 |   AjustaPai(noPai → dir, noPai);
19 |   noEsq → esq ← converteNo(arvoreRB, no234 →
   |     filhos[0]);
20 |   noEsq → dir ← converteNo(arvoreRB, no234 →
   |     filhos[1]);
21 |   AjustaPais(noEsq);
22 |   return noPai;
23 end
24 else
25 |   noPai ← rb_alocaNo(arvoreRB, no234 →
   |     chaves[1]);   rb_setCor(noPai, 'P');
26 |   noEsq ← rb_alocaNo(arvoreRB, no234 →
   |     chaves[0]);   noDir ←
   |     rb_alocaNo(arvoreRB, no234 → chaves[2]);
27 |   rb_setCor(noEsq, 'V');   rb_setCor(noDir, 'V');
28 |   noPai → esq ← noEsq;   noPai → dir ← noDir;
29 |   noEsq → pai ← noPai;   noDir → pai ← noPai;
30 |   noEsq → esq ← converteNo(arvoreRB, no234 →
   |     filhos[0]);
31 |   noEsq → dir ← converteNo(arvoreRB, no234 →
   |     filhos[1]);
32 |   AjustaPais(noEsq);
33 |   noDir → esq ← converteNo(arvoreRB, no234 →
   |     filhos[2]);
34 |   noDir → dir ← converteNo(arvoreRB, no234 →
   |     filhos[3]);
35 |   AjustaPais(noDir);
36 |   return noPai;
37 end
```

Tamanho	Quant. Splits	Altura	Quant. Blocos
100	50	5	7
1000	558	8	71
10000	5690	11	713
100000	57044	13	7133

Tabela 1: Resultados do experimento de inserção

Os resultados experimentais da operação de inserção (Tabela 1) revelam características importantes do comportamento da árvore 2-3-4 em diferentes escalas de dados.

Observa-se que a altura cresce lentamente à medida que o volume de dados aumenta: para 100 elementos, a altura é 5, enquanto para 100000 elementos ela sobe para 13, conforme previsto pela teoria de árvores B, em que a altura é muito menor do que a largura.

O crescimento segue um padrão logarítmico: há um acréscimo de aproximadamente três níveis para cada aumento de 10× no número de elementos (com uma exceção de apenas dois níveis entre 10000 e 100000). Isso comprova a boa escalabilidade da estrutura e sua eficiência amortizada em operações de inserção.

Em suma, os experimentos confirmam que a árvore 2-3-4 mantém a profundidade controlada mesmo com grandes volumes de dados, garantindo desempenho logarítmico e uso eficiente de blocos de disco.

4.2 Análise da remoção

No experimento de remoção, analisamos o desempenho da árvore 2-3-4 segundo as seguintes métricas: o número de rotações realizadas, o número de merges, a altura final da árvore após as remoções e a quantidade de blocos ocupados, seguindo a mesma lógica aplicada ao experimento de inserção. A coluna *Percentual* representa a porcentagem do total de elementos da árvore que foram removidos durante o experimento.

Percentual (%)	Qtd. Rotações	Qtd. Merges	Altura	Qtd. Blocos
10	178	526	10	709
20	306	698	11	718
35	400	1086	11	717
50	499	1573	10	712

Tabela 2: Resultados do experimento de remoção

Os resultados da operação de remoção (Tabela 2) evidenciam comportamentos de rebalanceamento distintos dos observados na inserção:

- **Frequência de merges vs. rotações:** em todos os cenários, o número de merges supera significativamente o de rotações — por exemplo, para 10 % de remoção, foram 526 merges contra 178 rotações.
- **Crescimento relativo:** à medida que a porcentagem de remoção aumenta de 10 % para 50 %, os merges crescem aproximadamente três vezes mais rápido que as rotações, refletindo o aumento de underflows nos nós.
- **Variação da altura:** a altura final da árvore não decresce monotonicamente com a remoção, devido à natureza local dos merges e rotações. Em certas condições, um merge envolvendo a raiz pode reduzir em um nível a altura total, mas a alternância entre operações com e sem efeito sobre a profundidade gera essa oscilação.

Em síntese, o experimento de remoção confirma que a estrutura 2-3-4, embora mantenha complexidade logarítmica nas operações, demanda um número crescente de merges conforme a taxa de remoção cresce, enquanto as rotações permanecem menos frequentes e mais estáveis em termos de crescimento.

5 Conclusões

Este trabalho analisou as árvores 2-3-4 e Rubro-Negras, destacando sua equivalência estrutural e operacional. A partir dessa equivalência, foi implementado um algoritmo de conversão entre as duas, e realizada uma análise de desempenho focada na árvore 2-3-4.

A análise teórica evidenciou que as árvores 2-3-4 mantêm o balanceamento perfeito por meio de operações de *split*, enquanto as Rubro-Negras utilizam colorações e rotações para garantir uma altura preta uniforme. Mostrou-se que um nó-2 corresponde a um nó preto, um nó-3 a um nó preto com filho vermelho, e um nó-4 a um nó preto com dois filhos vermelhos. Essas correspondências também se aplicam às operações de balanceamento.

Nos testes experimentais, a árvore 2-3-4 demonstrou crescimento logarítmico da altura durante inserções, mesmo com grandes volumes de dados, e uso eficiente de *splits*. Já nas remoções, observou-se que operações de *merge* ocorrem com mais frequência que rotações, crescendo em proporção ao número de remoções, devido a *underflows* nos nós. A altura final da árvore oscilou levemente, refletindo o comportamento localizado das operações de rebalanceamento.

Conclui-se que a árvore 2-3-4 é eficiente e resiliente para operações em larga escala, sendo indicada para aplicações que exigem acesso rápido e estrutura balanceada. A equivalência com árvores Rubro-Negras amplia as possibilidades de implementação e otimização, aproveitando as vantagens de ambas as estruturas.

Repositório do GitHub

Acesse o repositório no GitHub: Clique aqui

Referências

Bayer, R. & McCreight, E. (1970). Organization and maintenance of large ordered indices. Em *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control* (pp. 107–141).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to algorithms*. MIT press.

Guibas, L. J. & Sedgewick, R. (1978). A dichromatic framework for balanced trees. Em *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)* (pp. 8–21).: IEEE.

Lafore, R. (2017). *Data structures and algorithms in Java*. Sams publishing.