# Conversão de uma Árvore B 2-3-4 para uma Árvore Rubro-Negra

**Matheus Martins Batista**[1], **Luis Ricardo Albano Santos**[1]

[1]Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
Caixa Postal 50 – 37500-903 – Itajubá – MG – Brasil

{matmb,luisricardo}@unifei.edu.br

***Abstract.*** *This report wants to implement the conversion of multiple trees type B 2-3-4 to Binary Trees type Rubro-Negra in the C programming language. Throughout the report, the algorithm development process is exposed, its behavior in execution and the theoretical basis that supports the conversion properties. Also, the validity of the conversion and the properties provided by the literature used is verified.*

***Resumo.*** *Este relatório tem como objetivo implementar a conversão de arvores múltiplas do tipo B 2-3-4 para Árvores Binárias do tipo Rubro-Negra na linguagem de programação C. Ao longo do relatório, é exposto o processo de desenvolvimento do algoritmo, seu comportamento na execução e a fundamentação teórica que embasa as propriedades de conversão. Também, verificou-se a validade da conversão e das propriedades fornecidas pela literatura utilizada.*

## 1. Introdução

Conforme os conceitos discutidos na disciplina de Algoritmos e Estrutura de Dados II (COM112) da Universidade Federal de Itajubá, as Árvores Binárias são estrutura de dados muito eficientes para armazenar e recuperar informações. Nessa perspectiva, uma árvore binária é definida como um conjunto finito de nós que ou está vazio ou consiste de um nó chamado raiz mais os elementos de duas árvores binárias distintas chamadas de subárvores esquerda (elementos menores que a raiz) e direita do nó raiz (elementos maiores que a raiz), cada nó possui uma chave e no máximo duas subárvores. Todavia, a eficiência dessas árvores é garantida em memória primária, se considerarmos um ambiente que trabalhe com acessos na memória secundária para recuperar informações as árvores B tem um melhor desempenho. As árvores B são múltiplas. Logo, são formadas por nós que possuem mais do que dois filhos e, consequentemente, mais do que uma chave associada. Esse tipo de árvore trabalha com o conceito de ordem. Segundo os conceitos da disciplina, a ordem (m) é a quantidade máxima de filhos que um nó intermediário pode ter. Vale ressaltar o conceito de grau mínimo: um inteiro que define os limites inferiores e superiores sobre o número de chaves que um nó pode conter. As árvores B 2-3-4 são árvores múltiplas de grau mínimo 2 (t = 2). Ou seja, cada nó da árvore pode ter 2, 3 ou 4 descendentes e, consequentemente, armazenar 1, 2 ou 3 chaves por nó. Isso posto, este relatório visa relacionar e trabalhar as árvore binárias do tipo rubro-negra (RB) e as árvores B de ordem 4.

Por fim, sabe-se que árvores B 2-3-4 e as árvores rubro-negras são estruturas de dados equivalentes. Ou seja, para cada árvore 2-3-4 existe pelo menos uma árvore rubro-

negra equivalente. Sendo assim, a proposta desse trabalho é desenvolver um algoritmo que converta uma árvore 2-3-4 para uma árvore rubro-negra.

## 2. Metodologia

Para validar os aspectos acima, foi necessário consultar na bibliografia parâmetros para realizar a conversão correta dos nós presentes na árvore B para a RB. O material encontrado no livro ”Data Structures and Algorithms in Java”[Lafore 1998], apesar de ser direcionado para linguagem Java, possibilitou a compreensão da exata conversão para cada tipo de nó da árvore B. Conforme os conceitos da literatura obtida, classificou-se os nós em 3 tipos e desenvolveu-se um algoritmo recursivo de conversão dos nós da árvore B. Além disso, vale ressaltar que os algoritmos previamente desenvolvidos ao decorrer do curso de estrutura de dados da árvore B 2-3-4 e da RB foram aproveitados nesse projeto.

### 2.1. Materiais

O livro ”Data Structures and Algorithms in Java”orientou a lógica desenvolvida para conversão de cada nó. As imagens para ilustrar o problema foram retiradas de uma pergunta no site Stack Overflow[1]. Todo o programa foi implementado na linguagem C, codificado no Visual Studio Code, compilado (make fornecido) e executados no terminal bash do Ubuntu. Foram criados quatro arquivos .c: main, converte, rb e btree. Cada um possui o .h e .o correspondente. A máquina de testes utilizada possui a seguinte configuração:

- Processador: Intel® Core™ i3-4005U CPU @ 1.70GHz × 4;
- Memória: 8,00 GB;
- Sistema Operacional: Ubuntu 22.04 LTS;
- Compilador: gcc version 11.2.0 (Ubuntu 11.2.0-19ubuntu1);
- Makefile: desenvolvido pelo Luis Ricardo;
- Método de execução: comando make pelo terminal Bash.
- Observação: caso ocorra algo problema com o make fornecido, deve-se excluir os códigos objeto (.o) e tentar rodar novamente.

### 2.2. Métodos

Considerando que uma árvore B esteja criada, conforme a literatura, é necessário classificar os nós que serão convertidos e encaixados na árvore RB. O primeiro passo é instanciar uma árvore RB. O critério para classificar os nós será a ocupação. Nessa perspectiva eles podem ser:

a) Ordem 2 - o nó possui ocupação 1, basta transformá-lo num nó de cor preta e ligá-lo aos seus filhos correspondentes;
b) Ordem 3 - o nó possui ocupação 2, será convertido em nós filho e pai. O nó filho tem dois filhos próprios: W e X ou X e Y. O pai tem um outro filho: Y ou W. Não importa qual item se torna o filho e qual o pai. A criança é de cor vermelha e o pai é de cor preta. Adotou-se a primeira chave como pai. Logo, a chave mais à esquerda será transformada num nó preto e a mais à direita, num nó vermelho. O nó preto terá como filho da esquerda o primeiro nó filho da antiga árvore e como filho da direita o novo nó vermelho. Este, por sua vez, terá como filhos os dois filhos restantes da lista de filhos da árvore original;

[1] encurtador.com.br/yzIOS

c) Ordem 4 - o nó possui ocupação 3, a chave do meio será transformada num nó vermelho e terá como filhos as antigas chaves adjacentes que serão nós pretos com os antigos filhos da árvore B.

As Figuras 1, 2 e 3 ilustram os parâmetros supracitados.

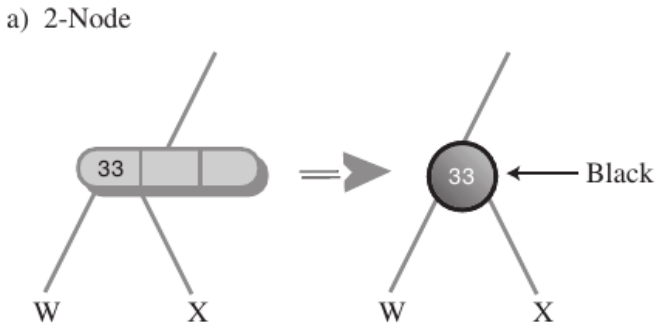


**Figura 1. Caso de Ordem 2**

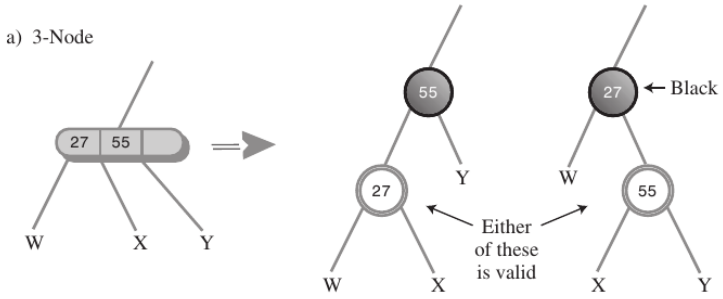


**Figura 2. Caso de Ordem 3**

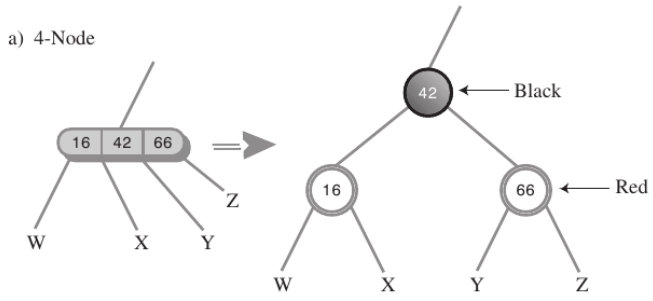


**Figura 3. Caso de Ordem 4**

A devida aplicação dessas regras garante o balanceamento automático da RB final.

Tendo em vista os fatos acima, o algoritmo de conversão (converte.c/.h) possui duas funções:

- *converte():* recebe a árvore B, instância a árvore RB, chama a função recursiva dos nós com a raiz sendo o primeiro nó atual e retorna uma RB convertida com base na árvore B;

- *converteNos()*: recebe o nó atual da árvore B, o nó atual da RB e a árvore RB. É uma função recursiva que aloca nó para o pai, verifica o caso em que o nó da árvore B se enquadra, se necessário aloca nó para os filhos, liga filho(s) e pai, determina o próximo pai e chama novamente a função para os filhos da direita e esquerda (se existirem). Os novos parâmetros passados incluem o nó atual da árvore B e nó atual da RB.

Para realizar os testes criou-se os menus com funções para manipular as árvores na main.c. Por fim, o arquivo para compilação *make* foi criado.

## 3. Resultados

Para a entrada fornecida no arquivo ”as”uma árvore B 2-3-4 é criada e preenchida. O programa cria ávore B, preenche, converte para uma RB e percorre a RB com éxito. Para confirmar os resultados obtidos foram utilizados applets de visualização para BTree [1] e RB [2]. Considerando que o pai é a primeira chave no caso de ocupação 2, houve uma pequena divergência com a ordem em nós locais, uma vez que a conversão não está sujeita ao balanceamento e sim às propriedades. Todos os nós foram corretamente coloridos e inseridos na árvore. Também, inserções e remoções foram realizadas para validar o funcionamento da RB. A RB fornecida está balanceada e funcionando corretamente.

## 4. Conclusão

A maior complexidade do algoritmo está em entender quais nós devem ser repassados para a recursão, qual deve ser o fator de parada e ligar filhos e pais. O processo de construção de um nó mostrou-se como a parcela menos complexa do programa. As orientações obtidas nas propriedades da literatura permitem a conversão correta das árvores B 2-3-4 para Rubro-Negra.

## Referências

Lafore, R. (1998). 2-3-4 trees and external storage: Transformation from 2-3-4 to red-black. In *Data Structures & Algorithms in Java*, pages 354–357. Waite Group Press: SAMS, 4 edition.

[1] `https://www.cs.usfca.edu/~galles/visualization/BTree.html`
[2] `https://www.cs.usfca.edu/~galles/visualization/RedBlack.html.`