Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Trabalho Prático III

BCC202 - Estrutura de Dados I

José Carlos Brás da Silva Júnior
Milena Santana de Almeida

Professor: Pedro Henrique Lopes Silva

Ouro Preto
7 de fevereiro de 2024

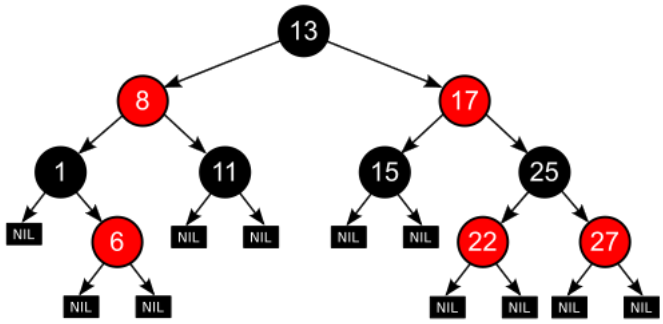
Sumário

1 Introdução

A árvore rubro-negra, também conhecida como vermelho-preto ou red-black, é um tipo de estrutura de dados balanceada que Rudolf Bayer originalmente desenvolveu em 1972. Este tipo de árvore utiliza um esquema de coloração dos nós para manter seu balanceamento. Cada nó em uma árvore rubro-negra é identificado por uma cor, geralmente representada por um bit. Apesar de possuir complexidade computacional similar à árvore AVL, a árvore rubro-negra se destaca por suas operações de inserção e remoção mais rápidas, em contrapartida de uma busca mais lenta. Após realizar inserções ou remoções, é feita uma análise das cores para identificar desbalanceamentos e realizar rotações necessárias. Além do bit extra para coloração, os nós desta estrutura contêm os campos Valor, Filho Esquerdo e Filho Direito.

Uma árvore rubro-negra deve seguir rigorosamente as seguintes regras:
1. Todos os nós são coloridos de vermelho ou preto.
2. A raiz é sempre preta.
3. Todas as folhas (nós nulos) são pretas.
4. Se um nó é vermelho, então todos os seus filhos são pretos.
5. Para qualquer nó, todos os caminhos dos nós descendentes até as folhas contêm o mesmo número de nós pretos.

Ao cumprir essas regras, podemos inferir que em qualquer caminho da raiz até uma subárvore vazia, não pode haver dois nós vermelhos consecutivos. A altura de um nó em uma árvore rubro-negra é determinada pela quantidade de nós pretos no caminho entre o nó e suas folhas descendentes (excluindo os nós nulos).



A figura acima é um exemplo de uma arvore rubro-negra.

1.1 Especificações do problema

O objetivo deste traba prático é construir uma árvore rubro-negra que siga as diretrizes mencionadas na introdução. Além disso, foi solicitado que seja implementada a função de remoção de um nó e a subsequente reestruturação da árvore para manter seu balanceamento adequado.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: Visual Studio Code. [1]
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [2]

1.3 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *VSCode*: Editor de código.

[1] Visual Studio Code está disponível em https://https://www.code.visualstudio.com/download
[2] Disponível em https://www.overleaf.com/

- *Valgrind*: ferramentas de análise dinâmica do código.

1.4 Especificações da máquina

As máquinas onde o desenvolvimento e os testes foram realizados possuem as seguintes configurações, respectivamente:

- Processador: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz.
- Memória RAM: 16Gb.
- Sistema Operacional: Windows 11 Home Single Language. (Terminal WSL, com Ubuntu.
- Processador: Intel(R) Core(TM) i5-10210U.
- Memória RAM: 8Gb.
- Sistema Operacional: Linux Ubuntu.

1.5 Instruções de compilação e execução

Para a compilação do projeto, basta digitar:

Compilando o projeto

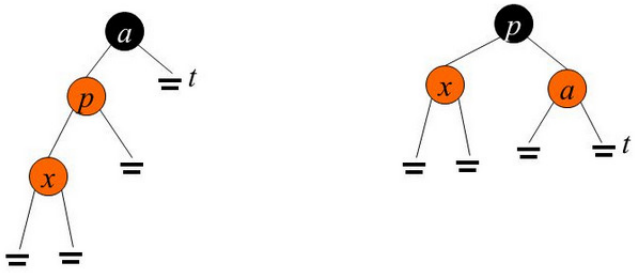
```
make
./exe < casoteste.in
```

2 Desenvolvimento

Inicialmente, copiamos os códigos apresentados durante as aulas, práticas e nos slides com o objetivo de criar pelo menos uma árvore funcional. Posteriormente, tentamos converter essa árvore em uma árvore rubro-negra. Contudo, nos deparamos com algumas dificuldades. A primeira delas ocorreu na segunda interação do usuário, onde ao inserir novas pessoas na árvore, o código não imprimia o resultado completo. Suspeitamos que o código estava alocando uma nova árvore a cada inserção, o que nos levou a modificar os escopos das funções. Após um longo período de investigação, identificamos que ao adicionar o número 2 (dois) nos arquivos .in, a árvore passou a imprimir corretamente (apesar de ainda não ser uma árvore rubro-negra). Entretanto, encontramos uma solução para esse problema sem precisar alterar os arquivos de entrada. Na função principal, logo após o loop de seleção do usuário, implementamos outro loop para imprimir as alterações, resolvendo assim o problema de impressão. O segundo obstáculo que enfrentamos foi relacionado ao processo de balanceamento, pois todas as tentativas resultavam em todos os nós sendo coloridos de preto.

2.1 Balanceamento

O processo de balanceamento é realizado através de rotações e ajustes de cores a cada inserção e remoção. O objetivo é cumprir as propriedades de uma árvore vermelho-preta, garantindo assim o equilíbrio da estrutura e mantendo um custo máximo de O(log N).

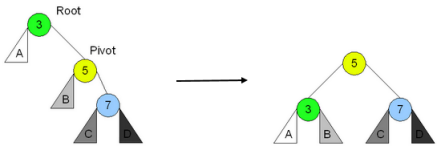


2.2 Rotações

Ao contrário das árvores AVL, que têm quatro funções de reequilíbrio, a árvore rubro-negra possui apenas duas operações de rotação: rotação para a direita e rotação para a esquerda.

2.3 Rotação para a esquerda

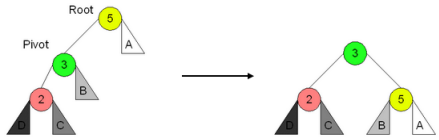
A função de rotação para a esquerda recebe um nó A com B como filho direito. Em seguida, move B para ocupar o lugar de A, tornando A o filho esquerdo de B. Por fim, o nó B assume a cor de A, enquanto A é colorido de vermelho.



(exemplo de rotação para a esquerda).

2.4 Rotação para a direita

A função de rotação para a esquerda recebe um nó A com B como filho esquerdo. Em seguida, move B para ocupar o lugar de A, tornando A o filho direito de B. Por fim, o nó B assume a cor de A, enquanto A é colorido de vermelho.

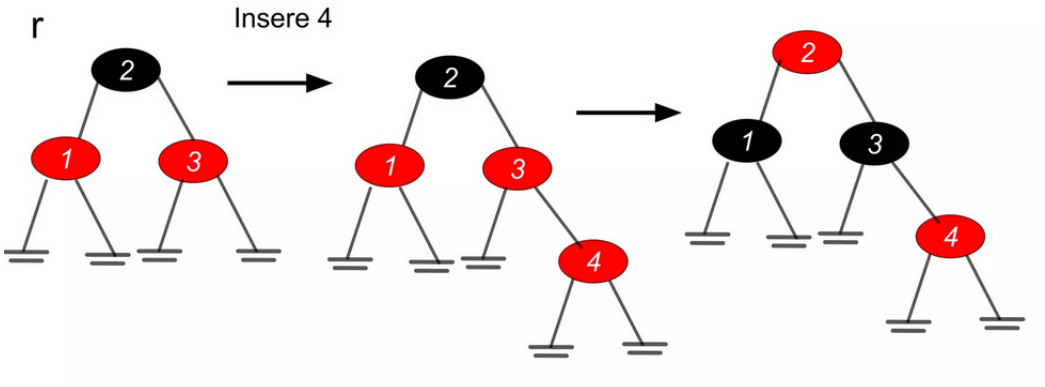


(exemplo de rotação para a direita).

2.5 Inserção

A inserção é bastante semelhante à árvore AVL, porém é crucial inserir respeitando as propriedades da estrutura rubro-negra, que são as seguintes:

- Se a raiz é igual a NULL, insira um nó Preto.
- Se a raiz é diferente de NULL, o nó inserido deve ser vermelho.



(exemplo de inserção com representação da cor dos nós balanceados).

3 Testes

3.1 Primeiro teste.

Não funcionava a impressão da segunda inserção das pessoas na árvore rubro-negra.

```
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$ ./exe < \tests/2.in
Dados inOrder:
Nome: Luisa
Idade: 21
Nome: Lucas
Idade: 22
Nome: Bruna
Idade: 25
Nome: Beatriz
Idade: 32
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$ ./exe < \tests/3.in
Dados inOrder:
Nome: Vitor
Idade: 23
Nome: Barbara
Idade: 24
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$ ./exe < \tests/5.in
Dados inOrder:
Nome: Elisa
Idade: 16
Nome: Marcos
Idade: 18
Nome: Luana
Idade: 20
Nome: Vitoria
Idade: 25
Nome: Mariana
Idade: 29
Nome: Liana
Idade: 38
Nome: Marlene
Idade: 42
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$ make
```

3.2 Segundo teste.

Todos os nós pretos porém a árvore estava balanceada e imprimindo todas as inserções.

```
Dados inOrder:
Nome: Elisa
Idade: 16
Cor: Preto
Nome: Marcos
Idade: 18
Cor: Preto
Nome: Luana
Idade: 20
Cor: Preto
Nome: Lucas
Idade: 23
Cor: Preto
Nome: Vitoria
Idade: 25
Cor: Preto
Nome: Mariana
Idade: 29
Cor: Preto
Nome: Liana
Idade: 38
Cor: Preto
Nome: Marlene
Idade: 42
Cor: Preto
Nome: Lauro
Idade: 48
Cor: Preto
Nome: Mauricio
Idade: 70
Cor: Preto
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$
```

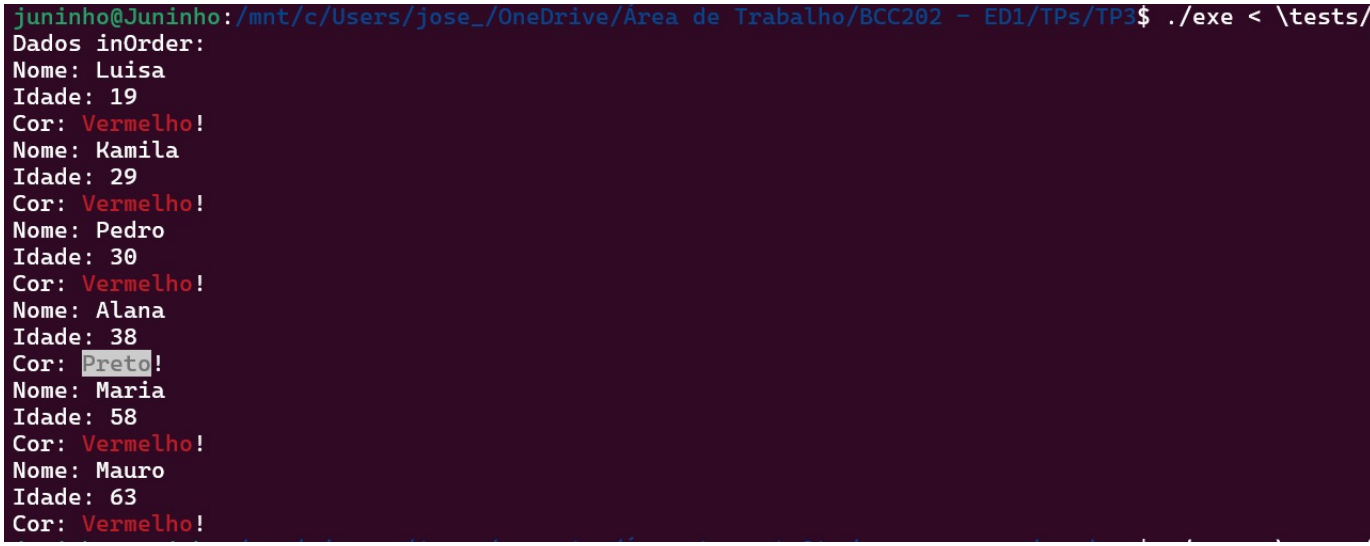
3.3 Terceiro teste.

Imprimindo a árvore balanceada e realizando a remoção da pessoa selecionada, seguida pelo balanceamento da árvore novamente.

```
Dados inOrder:
Nome: Luan
Idade: 10
Nome: Laura
Idade: 13
Nome: Lucas
Idade: 23
Nome: Ana
Idade: 24
Nome: Luisa
Idade: 29
3
Digite o nome e a idade da pessoa a ser removida:
Luan
10
Pessoa removida com sucesso!
2
Dados inOrder:
Nome: Laura
Idade: 13
Nome: Lucas
Idade: 23
Nome: Ana
Idade: 24
Nome: Luisa
Idade: 29
0
Dados inOrder:
Nome: Laura
Idade: 13
Nome: Lucas
Idade: 23
Nome: Ana
Idade: 24
Nome: Luisa
Idade: 29
juninho@Juninho:/mnt/c/Users/jose_/OneDrive/Área de Trabalho/BCC202 - ED1/TPs/TP3$
```

3.4 Quarto teste.

Imprimindo a árvore e a cor dos nós porém não segue as regras da rubro-negra.



4 Resultados

Durante a implementação e teste da árvore rubro-negra, enfrentamos dificuldades relacionadas à correta aplicação das regras de coloração, o que afetou diretamente os resultados esperados. Embora tenhamos observado que as operações básicas, como adição, remoção e busca, funcionaram perfeitamente, a estrutura da árvore não seguiu corretamente as diretrizes necessárias para uma árvore rubro-negra. Portanto, os resultados finais não atenderam às expectativas em relação à conformidade com as propriedades específicas desta estrutura de dados.

5 Considerações Finais

Exploramos como as árvores rubro-negras se equilibram automaticamente e como isso se compara com outras estruturas, como as árvores AVL. Também falamos sobre os problemas que enfrentamos ao implementá-las, como garantir que as cores e o balanceamento estejam corretos.

É interessante notar como as árvores rubro-negras são eficientes, mantendo um bom desempenho para operações básicas, como adição, remoção e busca. Mas é preciso entender bem suas regras e algoritmos para implementá-las corretamente.

Resumindo, este trabalho nos deu uma boa visão das árvores rubro-negras, desde o básico até como usá-las na prática, mostrando como elas são úteis para resolver problemas de computação de forma eficiente.