Moroni Neres Vieira

Relatório da Implementação do Projeto de Árvores Rubro Negras

Natal-RN

Novembro de 2018

Moroni Neres Vieira

Relatório da Implementação do Projeto de Árvores Rubro Negras

Relatório do trabalho da disciplina de Estrutura de Dados do curso de Pós-graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte, como atividade avaliativa para obtenção da nota da disciplina.
Professor: Dr. Bruno Motta de Carvalho

UFRN - Universidade Federal do Rio Grande do Norte

CCET - Centro de Ciências Exatas e da Terra

PPgSC - Programa de Pós-Graduação em Sistemas e Computação

Mestrado Acadêmico em Sistemas e Computação

Natal-RN

Novembro de 2018

Resumo

Neste trabalho é abordado a implementação de uma árvore rubro negra com a linguagem de programação C++. O software utilizado para fazer a implementação foi o sublime v 3.1.1. Contém três arquivos principais: struct.h contém a estrutura básica de uma árvore binária adicionando com o atributo cor. O arquivo functions.h contém as funções básicas para inclusão e remoção de um nó na árvore assim como funções auxiliares (Checagem, Impressão, dentre outras). O arquivo main.cpp contém o menu, a verificação inicial da entrada do arquivo e as chamadas às outras funções.

Lista de ilustrações

Sumário

Introdução

Árvore rubro-negra (Red-Black tree) é uma estrutura de dados de programação criada em 1972 com o nome de árvores binárias simétricas. Como as árvores binárias comuns às rubro-negras possuem um conjunto de operações (inserção, remoção, busca), porém são geralmente mais eficientes devido ao fato da rubro-negra estar sempre balanceada. Este balanceamento se dá justamente pela característica que dá nome a árvore, que vem de um bit extra em cada nodo que determina se esta é "vermelha"ou "preta"dentro do conjunto de regras que rege a árvore. Além desde bit, cada nodo também conta com os campos dados do nodo, filho esquerdo do nodo, filho direito do nodo e pai do nodo.

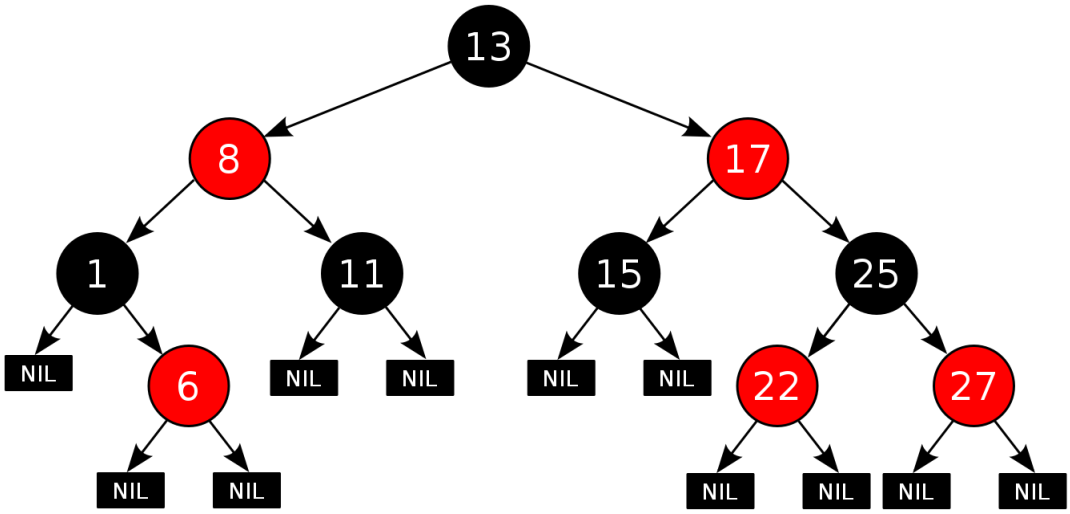

Figura 1 – Estrutura de uma árvore rubro negra

1.1 Propriedades da árvore rubro-negra

Uma árvore é rubro-negra se satisfazer as seguintes condições:

1. Todo nó é vermelho ou preto.
2. A raiz é preta.
3. Toda folha (NIL) é preta.
4. Se um nó vermelho, então ambos os seus filhos são pretos.
5. Para cada nó, todos os caminhos desde um nó até as folhas descendentes contêm o mesmo número de nós pretos.

A cada operação (inserção ou remoção) essas propriedades são verificadas e são feitas alterações com operações de rotação e ajuste de cor para que a árvore mantenha suas sempre essas regras.

As árvores rubro-negras constituem um entre muitos esquemas de árvores de pesquisa que são balanceadas com o objetivo de garantir que as operações básicas de conjuntos dinâmicos demorem o tempo O(lg n) no pior caso.

1.2 Rotações

Uma rotação é uma operação realizada na árvore para garantir seu balanceamento. Na rubro-negra pode ser feita a direita e a esquerda, onde são alterados os nodos rotacionados.

Para executar uma operação de rotação, utiliza-se o método de troca para direita ou esquerda dos nós. Quando fazemos uma rotação à esquerda em um nó x, supomos que seu filho da direita y não é nil[T]. A rotação à esquerda "faz o pivô"em torno da ligação de x para y. Ela faz de y a nova raiz da subárvore, tendo x como filho da esquerda de y e o filho da esquerda de y como filho da direita de x.

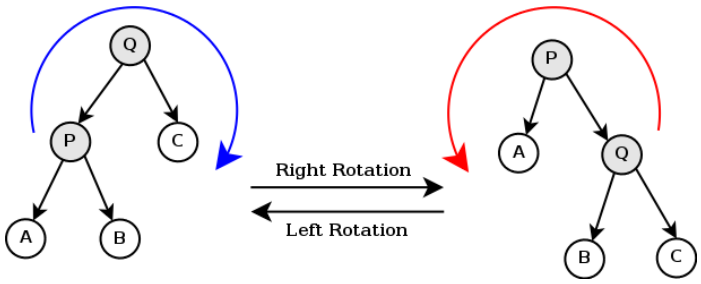

Figura 2 – Representação de rotação de uma árvore rubro negra

1.3 Inserções

A inserção de um nó em uma árvore rubro negra de n nós podem ser realizada no tempo O(lg n).

Ao inserir-se um elemento em uma árvore rubro-negra, esta é comparada com os elementos e alocada em sua posição conforme a regra 2. Ao inserir um elemento ele é sempre da cor vermelha (exceto se for o nodo raiz). A seguir a árvore analisa se o antecessor da folha. Se este for vermelho será necessário alterar as cores para garantir as propriedades da árvore.

A figura abaixo mostra a inserção dos nós em uma árvore rubro negra.

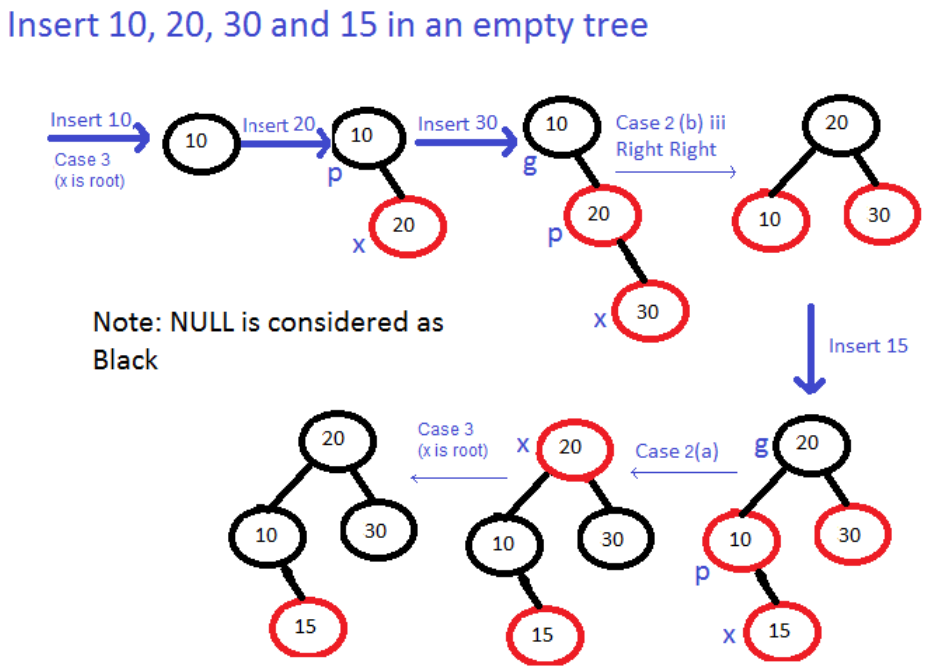

Figura 3 – Representação da inserção de nó em uma árvore rubro negra

1.4 Remoções

A remoção de um nó é ligeiramente mais complicada do que a inserção de um nó na árvore. A eliminação tem um tempo de demora de O(lg n).

1.4.1 caso 1: O irmão *w* de x é vermelho

Se o irmão do nó x é vermelho e tem dois filhos pretos. Pode-se trocar as cores de w e $x.p$ sem violação das propriedades e depois executar uma rotação para esquerda em $x.p$

1.4.2 Caso 2: O irmão *w* de *x* é preto e os filhos de *w* são pretos

Neste caso o filhos de w são pretos e w também é preto. Retira-se um preto de x e também de w, deixando x com apenas um preto e deixando w vermelho. Para compensar a remoção de um preto de x e de w, adiciona-se um preto extra a $x.p$, que era originalmente vermelho ou preto, verifica-se novamente as propriedades até que sejam satisfeitas.

1.4.3 Caso 3: O irmão de *w* de *x* é preto, o filho à esquerda de *w* é vermelho e o filho à direita de *w* é preto

Neste caso pode-se permutar as cores de w e de seu filho à esquerda $w.esquerda$ e então executar uma rotação para a direita em w sem violar qualquer das propriedades. O novo irmão w agora é um nó preto com um filho à direita vermelho.

1.4.4 Caso 4: O irmão de *w* de *x* é preto e o filho à direita de *w* é vermelho

Este caso ocorre o irmão w do nó x é preto e o filho à direita de w é vermelho. Deve-se fazer algumas alterações de cores e executa uma rotação para a esquerda em $x.p$, pode-se remover o preto extra em x, tornando-se unicamente preto, sem violar qualquer das propriedades da árvore.

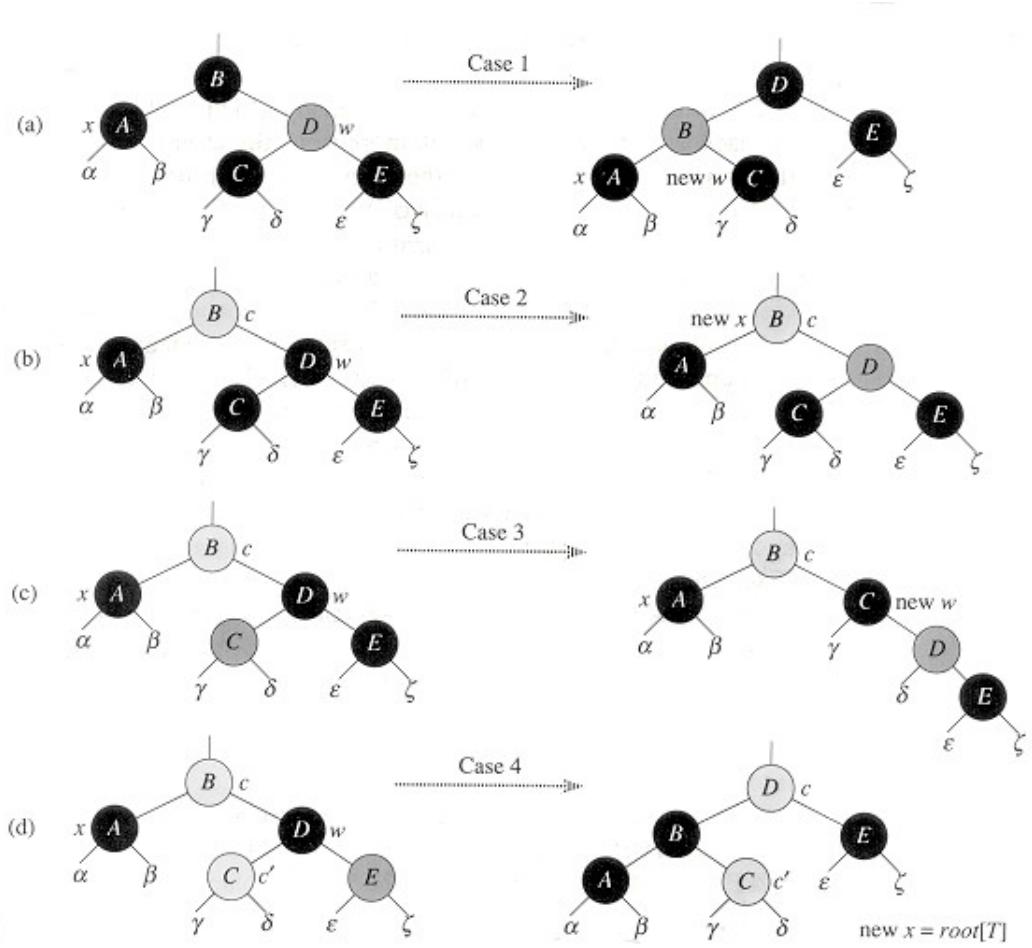

Figura 4 – Representação dos casos de remoção da árvore rubro negra

Implementação

A implementação teve como base a construção do código-fonte necessário para realizar as operações de: Inserção, Remoção, Busca, Checagem e Impressão, assim como suas funções auxiliares, baseado na leitura as funções de checagem de tamanho de chave, verificação de condições do arquivo também foram implementadas.

2.1 Estruturação do Projeto

2.1.1 struct.h

Contém a implementação de uma estrutura de árvore. Cada nós contém os seguinte dados:

1. Um apontador para o nodo
2. Uma chave
3. Um bit para cor
4. Um apontador para o pai
5. Apontadores para os filhos

```
1  #ifndef STRUCT_H_INCLUDED
2  #define STRUCT_H_INCLUDED
3
4  #include <iostream>
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  using namespace std;
9
10 enum type {BLACK, RED};
11
12 struct node {
13   enum type color;
14   string key;
15   struct node *left, *right, *parent;
16 };
17
18 struct node *temp, *nill, *root;
19
20 #endif // STRUCT_H_INCLUDED
```

2.1.2 functions.h

Contém todas funções do projeto, segue a lista das funções e suas respectivas funcionalidades.

Nome da Função	Funcionalidade
void Delete_Left_Rotate(struct node *x)	Função para fazer a rotação para esquerda na remoção do nó
void Delete_Right_Rotate(struct node *y)	Função para fazer a rotação para direita na remoção do nó
void RB_transplant(struct node *aux, struct node *auxchild)	Movimenta subárvores dentro da árvore de busca binária
node *tree_successor(struct node *p)	Busca o sucessor na sequência ordenada
RB_delete_fix(struct node *x, struct node *w)	Elimina um nó de uma árvore
RBDelete(struct node* z, struct node* y, struct node *x)	Remove um nó e verifica se o mesmo tem dois filhos
insert_Left_Rotate(node *&p)	Checa se o tio, o pai e o filho estão inclinados a esquerda, então realiza a rotação.
void insert_Right_Rotate(node *&p)	Checa se o tio, o pai e o filho estão inclinados a direita, então realiza a rotação.
insert_Double_Left_Right_Rotate(node * &p)	Realiza a rotação dupla da esquerda para direita
insert_Double_Right_Left_Rotate(node * &p)	Realiza a rotação dupla da direita para esquerda
insert_RB_Fixup(node * &p)	Restaura as propriedades da árvore
RBInsert(node * &tree, string key)	Insere um nó na árvore
node * search_node(node * tree, string key)	Faz a busca de um determinado nó
int is_black(node * p)	Checa se um nó é preto
int blackHeightNode(string key)	Verifica a altura preta da árvore
void create_Nill_and_Root()	Cria uma nova árvore com filhos NIL
void RBPrint(node* tree)	Imprime a árvore
void RBCheck(node * tree)	Imprime a árvore em pré-ordem
void insert_rb(string key)	Checa se o nó existe, se não existir insere o novo nó
void delete_rb(string key)	Checa se o nó já foi removido anteriormente

2.1.3 main.cpp

O arquivo main.cpp faz a verificação do arquivo de dicionário, se existe ou não e se pode ser lido. Também realiza a leitura do arquivo, faz a chamada de inserção ou remoção de acordo com a opção na linha do arquivo e contém também a função menu.

```
1  /**
2
3      Uma  rvore  de pesquisa bin ria    uma  rvore  rubro-negra se satifaz
       s
4      seguintes propriedades:
5      1. Todo n      rubro ou preto
6      2. A raiz    preta.
7      3. Toda folha (NIL)    preta.
8      4. Se um n      rubro, ent o ambos os seus filhos s o pretos.
9      5. Para cada n , todos os caminhos de um n  at  as folhas
      descendentes
10     cont m os mesmo n mero d e n s pretos.
11
12     @author Moroni Neres Vieira
13     @version 1.0 11/2018
14 */
15
16 //Bibliotecas de fun  es internas
17 #include <iostream>
18 #include <sstream>
19 #include <fstream>
20 #include <string>
21 #include <cstring>
22 #include <exception>
23 #include <stdexcept>
24
25 //Bibliotecas externas
26 #include "struct.h" //Defini  o da estrutura de uma  rvore  bin ria
27 #include "functions.h" //Fun  es para  rvore  rubro-negra
28
29 using namespace std;
30
31 int lerpalavras(string);
32 void menu();
33 void opcoes(int);
34
35 int main(int argc, char *argv[]) {
36
37     ifstream file;
38     string arquivo;
39     arquivo=argv[1];
40     file.open(arquivo);
```

```
41
42      //Cria a estrutura da  rvore
43      create_Nill_and_Root();
44
45      if(!file.is_open()){
46          cout << "Arquivo informado n o existe ou n o pode ser aberto!" <<
        endl;
47          return -1;
48
49      }else{
50          lerpalavras(argv[1]);
51      }
52
53      return 0;
54  }
55
56  int lerpalavras(string nomearq) {
57      ifstream dic;
58      string linha;
59
60      dic.open(nomearq);
61
62      while (!dic.eof()) {
63          getline(dic, linha);
64          if(!linha.empty()){
65
66              string::size_type pos1 = linha.find_first_of(' '); //Alias para
        string:size_type e a posi   o da palavra
67              string palavra = linha.substr(0, pos1);
68              string::size_type pos2 = linha.find_first_of(' ');
69              string num = linha.substr(pos1+1, pos2);
70
71              int opcao = stoi(num);
72
73                  if(palavra.length() > 20){
74                      cout << "Chave n o pode ser inserida por ter tamanho >
        20" << endl;
75
76                  }else{
77                      if (opcao == 1){
78                          cout << "Inserindo " << palavra << endl;
79                          insert_rb(palavra);
80                      }else{
81                      cout << "Removendo " << palavra << endl;
82                      delete_rb(palavra);
83                  }
84
```

```
85          }
86      }
87  }
88
89
90  menu();
91
92  dic.close();
93
94  return 0;
95  }
96
97  //Menu de op   es
98  void menu(){
99      int opcao;
100     string key;
101
102     do{
103
104         cout<<"\n\n\n";
105         cout << "Ap s a inclus o ou exclus o deseja fazer alguma
        opera   o: \n";
106
107         cout<<"(1) - Busca\n";
108         cout<<"(2) - Imprimir\n";
109         cout<<"(3) - Checar\n";
110         cout<<"(0) - Exit\n";
111         cout<<"Entre com uma op   o:" << endl;
112
113             cin >> opcao;
114
115             switch(opcao){
116                 case 1:
117                 {
118                     cout << "Entre com a chave para busca: " << endl;
119                     cin >> key;
120
121                     node * aux = search_node(root, key);
122                     if(aux != nill){
123                         cout<<"N   encontrado\n";
124                         cout<<"(";
125                         if(aux->parent!=nill)
126                             cout<<aux->parent->key<<", ";
127                         else
128                             cout<<"NILL, ";
129                         cout<<aux->key<<", ";
130                         if(aux->color==RED)
```

```
131                             cout<<" Vermelho, ";
132                         else
133                             cout<<" Preto, ";
134                         cout<<blackHeightNode(key)<<", ";
135                         if(aux->left != nill)
136                             cout<<aux->left->key<<", ";
137                         else
138                             cout<<"NILL, ";
139                         if(aux->right != nill)
140                             cout<<aux->right->key<<" )\n";
141                         else
142                             cout<<"NILL )\n";
143                     }
144                     else
145                         cout<<"N   n o encontrado!\n";
146                 }       break;
147 
148                 case 2:
149                 {
150                     cout << "Imprimindo a   rvore " << endl;
151                     RBPrint(root);
152                 }
153 
154                 case 3:
155                     cout << "Checando a   rvore " << endl;
156                     RBCheck(root);
157                 case 0:
158                     break;
159 
160                 default:
161                     cout << "Op   o inv lida, entre com uma op   o do
    menu v lida!";
162             }
163         }while(opcao!=0);
164 }
```

2.1.4 RBTree

O arquivo RBTree é o arquivo executável gerado após a compilação do código-fonte do projeto.

2.2 Roteiro para compilação do Projeto

O arquivo executável assim como o projeto foi desenvolvido em Sistema Operacional Linux Ubuntu 18.04 com a ferramenta Sublime, dessa forma esse ambiente precisa ser

replicado ou deve-se ter ambiente similar, sistema Linux, para executar o projeto.

O programa foi compilado utilizando o comando a seguir:

g++ main.cpp -o RBTree

2.3 Roteiro para execução do Projeto

1. Descompactar o arquivo compactado TrabalhoMoroni.zip
2. Abrir o terminal e ir até o diretório descompactado
3. Digitar o seguinte comando: **RBTree dicionario1.txt**
4. Escolher uma opção no menu após a execução.

2.4 Repositório do Projeto

Foi criado um repositório público no GitHub em caso de falha na abertura dos arquivos, o link está disponível em: https://github.com/moronivieira/projeto-estututura-dados/

2.5 Resultados esperados

Como resultado da execução do projeto espera-se que se obtenha a seguinte tela:

```
moroni@pcmoroni: ~/Documentos/Livros-Mestrado/Projeto-20
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
moroni@pcmoroni:~/Documentos/Livros-Mestrado/Projeto-2018-2-Estrutura_Dados/Proj$
Removendo teste


A palavra teste foi removida anteriormente ou nao foi inserida.

Inserindo abuso
Inserindo carro
Inserindo doce
Removendo gola


A palavra gola foi removida anteriormente ou nao foi inserida.

Inserindo gola
Inserindo palhaço
Inserindo taturana
Inserindo pacote
Inserindo bolha
Inserindo fussura
Inserindo batata
Inserindo estrela
Removendo taturana
Removendo a palavra taturana

Imprimindo em ordem:
abuso, batata, bolha, carro, doce, estrela, fussura, gola, pacote, palhaço,

Checando em pre-ordem:
(NILL, carro,  Preto, 2, batata, gola )
(carro, batata,  Preto, 1, abuso, bolha )
(batata, abuso,  Vermelho, 1, NILL, NILL )
(batata, bolha,  Vermelho, 1, NILL, NILL )
(carro, gola,  Vermelho, 2, estrela, palhaço )
(gola, estrela,  Preto, 1, doce, fussura )
(estrela, doce,  Vermelho, 1, NILL, NILL )
(estrela, fussura,  Vermelho, 1, NILL, NILL )
(gola, palhaço,  Preto, 2, pacote, NILL )
(palhaço, pacote,  Vermelho, 1, NILL, NILL )
Inserindo cataplana
Inserindo cerveja
Inserindo zebra
Inserindo lis
Inserindo almirante
Inserindo elefante
Inserindo espaço
Inserindo estrela
Chave estrela ja existe
Removendo cataplana
Removendo a palavra cataplana

Imprimindo em ordem:
abuso, almirante, batata, bolha, carro, cerveja, doce, elefante, espaço, estrela,
```

Figura 5 – Execução do Projeto

3 Considerações Finais

O Projeto foi desenvolvido com a motivação de apurar os conhecimentos adquiridos em sala de aula de forma prática, auxiliando no entendimento das propriedades e características das árvores rubro negras.