ALGORITMO E ESTRUTURA DE DADOS II

Árvores Rubro-Negras – Lab 4

Aluno: Gabriel Nobrega de Lima
E-mail: gabriel.nobrega.lima@gmail.com
Professor: David Correa Martins Jr

Índice

Introdução

Este relatório descreve a implementação de árvore Rubro-Negra requisitada na Atividade 4 da disciplina de Algoritmo e Estrutura de Dados 2. Árvores Rubro Negras são construídas com o objetivo de apresentar árvores balanceadas independentemente da ordem em que as chaves são inseridas. Para que uma árvore seja Rubro-Negra são necessários o cumprimento das regras 1 a 5:

1. Nós são vermelhos ou pretos.
2. A raiz é preta.
3. Todas as folhas são pretas.
4. Os filhos de um nó vermelho são pretos.
5. Todos os caminhos de algum nó para suas folhas contêm o mesmo número de nós pretos.

O projeto em questão implementa árvores Rubro-Negras como uma biblioteca em linguagem C++. Esta biblioteca é utilizada por uma aplicativo para a construção de árvores descritas em um arquivo de texto. A árvore é descrita através de inserções sucessivas de nós, em uma sintaxe particular. Cada inserção de nó gera um novo estado na árvore que é impresso em um arquivo de sáida definido pelo usuário. Além das operações de inserção, o arquivo de entrada pode conter operações de busca, sendo os resultados também relatados no arquivo de sáida.

Arquitetura de Projeto

A árvore Rubro-Negra foi implementada e encapsulada como uma biblioteca e pode ser consultada dentro do diretório ../src/lib. Visando as boas práticas de programação o projeto foi iniciando criando-se a interface BinaryTree. Nela foram listadas todas as operações que se espera que uma árvore binária realizasse, tais como, inserção, busca e remoção. Como nosso objetivo aqui é a implementação de uma árvore binária do tipo Rubro-Negra, construiu-se a classe RedBlackTree que implementa as funcionalidades da interface BinaryTree. Assim, futuramente caso se deseje implementar árvore binárias de outros tipos, poderá se manter uma interface similar, fazendo com as aplicações que as utilizem possam intercambiar de uma árvore a outra sem práticamente nenhuma modificação de código.

A aplicação que utiliza a biblioteca de árvore Rubro-Negra está implementada na classe RedBlackApp no diretório ../src. A classe utiliza a interface BinaryTree para realizar as operações na árvore. A função main, disposta em main.cpp controla a classe da aplicação definindo os arquivos de entrada e saída para processamento.

Na realidade arquitetura supracitada foi projetada no projeto de árvores AVL. No relatório da mesma já havia sido previsto a implementação de outros tipos de árvores binárias. O trecho abaixo mostra como a arquitetura foi reaproveitada do projeto de árvores AVL para árvores Rubro-Negras:

“Para exemplificar a utilidade da arquitetura orientada à interface BinaryTree, suponha que a próxima atividade seja a implementação de árvores Rubro-Negras e que a aplicação segue exatamente os mesmos requisitos desta. Será necessário implementar a árvore Rubro-Negra em uma classe qualquer(por exemplo RedBlackTree) que implemente a interface BinaryTree. A aplicação poderia ser alterada para utilizar árvores Rubro-Negras modificando apenas a linha BinaryTree tree = new AVLTree() para BinaryTree tree = new RedBlackTree(). A utilização de padrões de projeto e boas práticas de programação permitem grande economia de tempo na manutenção de código. Como um dos requisitos do trabalho é a programação de qualidade tal arquitetura foi adotada.”

O fragmento acima foi retirado do relatório de árvores AVL e mostra que uma boa implementação de código facilita a manutenção do software. Como previsto pode se re-aproveitar toda a aplicação, modificando apenas o tipo de instância da biblioteca, de AVLTree para RedBlackTree.

Documentação do Projeto

A biblioteca foi inteiramente documentada com doxygen. Esta documentação pode ser acessada no diretório .../doc na raiz do projeto. A Figura 1, mostra a página de documentação da classe RedBlackTree.

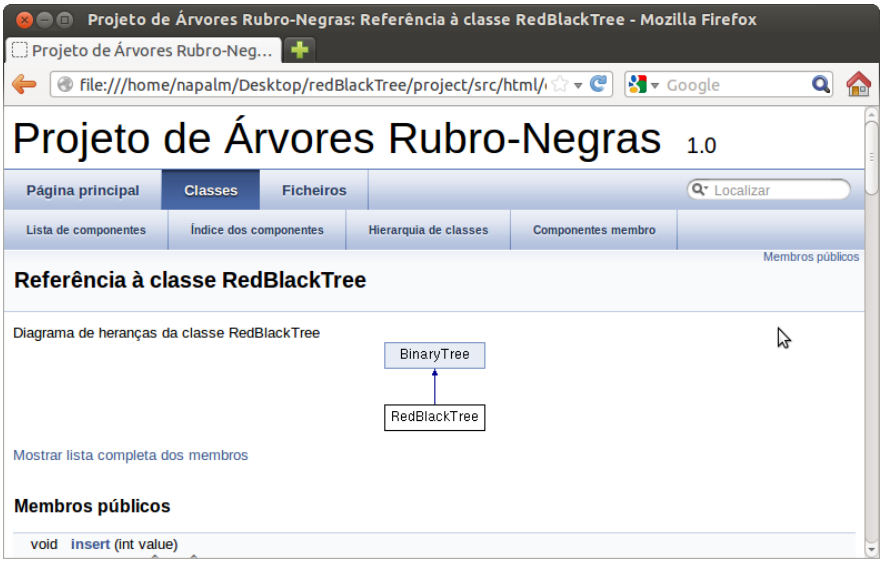

Figura 1 - Página de documentação da classe RedBlackTree.

Além da documentação da biblioteca, foram inseridos comentários de desenvolvimento nas partes mais complexas do código fonte.

Operações Implementadas

Nesta seção serão abordados os modos de como utilizar a biblioteca para as operações de visualização, inserção e busca em Rubro-Negras.

Inserção

A inserção de nós na árvore Rubro-Negra pode ser realizada simplesmente chamando o método insert. O fragmento de código abaixo exemplifica a inserção dos nós 1,2 e 3 sucessivamente:

```
RedBlackTree tree;

tree.insert(1);
tree.insert(2);
tree.insert(3);
```

O balanceamento da árvore é realizado automáticamente, sempre que uma inserção faça com que árvore fique desbalanceada.

Busca

A busca por nós na árvore Rubro-Negra pode ser realizada através de dois métodos, find() e traceFind(). O método find() retorna verdadeiro se o nó buscado está presente na árvore, caso contrário falso. O método traceFind() retorna uma string contendo os nós percorridos a partir da raiz da árvore até encontrar o nó procurado. Caso o nó procurado esteja presente na árvore, o último nó retornado na string deverá ser ele, do contrário o nó não está contido na árvore. O fragmento de código abaixo mostra um exemplo de como utilizar ambos os métodos:

```
RedBlackTree tree;

void search(int key){
    if(tree.find(key))
        cout<<"No "<<key<<" encontrado..."<<endl;else
        cout<<"No "<<key<<" nao encontrado..."<<endl;
}
int main(int argc, char *argv[]){
    tree.insert(1);
    tree.insert(2);
    tree.insert(3);
    tree.insert(5);
    search(1);
    search(2);
    search(3);
    search(50);
    cout<<tree.traceFind(5)<<endl;
    cout<<tree.traceFind(-1)<<endl;

return 0;
}
```

Resultado no terminal:

```
No 1 encontrado...
No 2 encontrado...
No 3 encontrado...
No 50 nao encontrado...
2P,3P,5V
2P,1P
```

Perceba que o método traceFind() permitiu visualizar todos os nós da árvore, estes já encontram-se perfeitamente balanceados de acordo com os requisitos da ávore Rubro-Negra.

Visualização

A visualização da árvore pode ser realizada de duas maneiras, através do método viewTree() ou do método preOrderedState(). O método viewTree() mostra a configuração de cada nó na árvore, isto é, os filhos direito e esquerdo além da cor e nó pai. O método preOrderedState imprime toda a árvore em pré-ordem, seguindo a sintaxe proposta no enunciado da atividade. O Fragmento de código abaixo exemplifica como utilizar ambos os métodos:

```
int main(int argc, char *argv[]){
    RedBlackTree tree;
    tree.insert(1);
    tree.insert(2);
    tree.insert(3);
    tree.insert(5);
    cout<<tree.preOrderedState()<<endl;
    cout<<tree.viewTree()<<endl;

    return 0;
}
```

Resultado no terminal:

```
(2P,(1P,(),()),(3P,(),(5V,(),())))
-------------------------------------
Node:2(P)
FATHER:NA

LEFT:1
RIGHT:3
-------------------------------------
Node:1(P)
FATHER:2

LEFT: NA
RIGHT: NA
-------------------------------------
Node:3(P)
FATHER:2

LEFT: NA
RIGHT:5
-------------------------------------
```

Node:5(V)
FATHER:3

LEFT: NA
RIGHT: NA

Pode ser interessante para o corretor utilizar o método viewTree. Ele permite visualizar árvores com grande número de nós de uma maneira mais clara. O método de pré-ordem consegue descrever árvores de maneira mais compacta, porém quando o número de nós em testes ficam elevados a utilização excessiva de parenteses pode confundir.

O Aplicativo RedBlackApp

O aplicativo RedBlackApp utiliza a biblioteca RedBlackTree, seu objetivo é processar os comandos de inserção e busca de um arquivo de entrada e mostrar os resultados de cada processamento em um arquivo de saída. O arquivo fonte main.cpp contido em ../src utiliza esta classe para executar a aplicação, passando os arquivos de entrada e saída atribuídos pelo usuário através da linha de comando. O programa pode ser visto no fragmento de código abaixo:

```
int main(int argc, char *argv[]){

   if(argc<=2){
      cout<<"Sintaxe incorreta, utilize: ./redblackapp arquivo_de_entrada arquivo_de_saida"<<endl<< "\t
Por exemplo: ./redblackapp input.txt output.txt"<<endl;
   }else{
      RedBlackApp *app = new RedBlackApp();
      app->process(argv[1], argv[2]);
      delete app;
   }

   return 0;

}
```

O arquivo de entrada suporta apenas um comando por linha. Pode se realizar operação de inserção, fazendo “i nro_nó”, ou operação de busca com “b nro_nó”. A seção “Processando Arquivos de Entrada” ilustra um exemplo prático da sintaxe do arquivo de entrada.

Compilando e Executando

O projeto foi desenvolvido em linguagem C++, sendo assim será necessário a utilização do compilador g++. Tente executar o g++ pelo terminal, caso este não seja localizado será necessário instala-lo. Nas distribuições derivadas do Debian (por ex:Ubuntu), a instalação pode ser realizada através do comando:

sudo apt-get install g++

Com o compilador instalado o projeto pode ser compilado com o comando make dentro do diretório src do projeto.

.../src$ make

O comando irá gerar o binário redblackapp dentro do diretório src. Para executa-lo utilize o comando:

./redblackapp

Para eliminar todos os arquivos binários gerados no processo de compilação execute o comando:

.../src$ make clear

Processando Arquivos de Entrada

O programa redblackapp permite o usuário passe apenas dois parâmetros, o arquivo de entrada e o arquivo de saída. Ambos os parâmetros são obrigatórios para a execução do software.Por exemplo, caso se deseje processar as operações do arquivo “entrada.txt” e obter os resultados no arquivo “saida.txt”, execute o programa através do seguinte comando:´

Conteúdo do arquivo “entrada.txt”:

```
i 10
i 6
i 4
i 5
i 0
i 3
i 9
i 2
i 1
i 8
i 7
b 7
b 4
b 60
```

Executando o software para realizar as operações descritas no arquivo “entrada.txt” e gravar os estados no arquivo “saida.txt”:

```
./redblackapp entrada.txt saida.txt
Arquivo de Entrada:entrada.txt
Arquivo de Saida:saida.txt
Processando...
Comando: i 10
```

Inserindo 10 ...
Comando: i 6
Inserindo 6 ...
Comando: i 4
Inserindo 4 ...
Comando: i 5
Inserindo 5 ...
Comando: i 0
Inserindo 0 ...
Comando: i 3
Inserindo 3 ...
Comando: i 9
Inserindo 9 ...
Comando: i 2
Inserindo 2 ...
Comando: i 1
Inserindo 1 ...
Comando: i 8
Inserindo 8 ...
Comando: i 7
Inserindo 7 ...
Comando: b 7
Buscando no 7: 4P,6P,9V,8P,7V ...
Comando: b 4
Buscando no 4: 4P ...
Comando: b 60
Buscando no 60: 4P,6P,9V,10P ...
Processamento finalizado...

Após o processamento cada uma das operações, os estados assumidos pela árvore podem ser visualizados no arquivo “saida.txt”, como mostrado abaixo:

i 10
(10P,(),())
i 6
(10P,(6V,(),()),())
i 4
(6P,(4V,(),()),(10V,(),()))
i 5
(6P,(4P,(),(5V,(),())),(10P,(),()))
i 0
(6P,(4P,(0V,(),()),(5V,(),())),(10P,(),()))
i 3
(6P,(4V,(0P,(),(3V,(),())),(5P,(),())),(10P,(),()))
i 9
(6P,(4V,(0P,(),(3V,(),())),(5P,(),())),(10P,(9V,(),()),()))
i 2
(6P,(4V,(2P,(0V,(),()),(3V,(),())),(5P,(),())),(10P,(9V,(),()),()))
i 1
(4P,(2V,(0P,(),(1V,(),())),(3P,(),())),(6V,(5P,(),()),(10P,(9V,(),()),())))
i 8
(4P,(2V,(0P,(),(1V,(),())),(3P,(),())),(6V,(5P,(),()),(9P,(8V,(),()),(10V,(),()))))
i 7
(4P,(2P,(0P,(),(1V,(),())),(3P,(),())),(6P,(5P,(),()),(9V,(8P,(7V,(),()),()),(10P,(),()))))
b 7
4P,6P,9V,8P,7V
b 4
4P
b 60
4P,6P,9V,10P

Testes

Foram realizados testes exaustivos e não pode ser verificado qualquer erro de implementação. Em todos os testes foi utilizado o Valgrind com a opção de detecção de vazamento e leaks de memória. Todos os relatórios obtidos com o Valgrind foram semelhantes a este:

```
==5099== All heap blocks were freed -- no leaks are possible
==5099==
==5099== For counts of detected and suppressed errors, rerun with: -v
==5099== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Conclusão

Nesta prática foi implementada uma biblioteca para a manipulação de árvores Rubro-Negras permitindo as operações de busca, inserção e visualização. A biblioteca foi implementada de maneira que se possa amplia-la, sem que o aplicativo sofra alterações. Ela foi também devidamente comentada com apoio do doxygen, além dos comentários de desenvolvimento que podem ser vistos durante todo o código. O aplicativo foi testado exaustivamente e em nenhum dos casos pode se relatar qualquer erro de lógica ou vazamento/leak de memória.