ALGORITMO E ESTRUTURA DE DADOS II

Árvores AVL – Lab 3

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 árvores AVL em linguagem C++. Uma árvore binária é denominada AVL quando a diferença de altura entre as sub-árvores esquerda e direita de um nó qualquer não é superior a 1. O fato de a árvore estar balanceada resulta em ganho de desempenho para operações realizadas sobre árvores binárias, como inserção, remoção e busca. Se a árvore binária respeita as imposições da árvore AVL, todas estas operações gastam, no pior caso, um tempo O(logn), que neste caso representa a altura da árvore.

Arquitetura de Projeto

A árvore AVL 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 inciando criando-se a interface BinaryTree. Nela foram listadas todas as operações que se espera que uma árvore binária realize, tais como, inserção, busca e remoção. Como nosso objetivo aqui é a implementação de uma árvore binária do tipo AVL, construiu-se a classe AVLTree que implementa as funcionalidades da interface BinaryTree. Assim, futuramente caso se deseje implementar árvore binárias de outros tipos, como Árvores Rubro Negras 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 AVL está implementada na classe AVLApp 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.

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.

Operações Implementadas

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

Inserção

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

```
AVLTree 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 AVL 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:

```
AVLTree 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...

2 3 5
2 1

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 do fator de balanceamento. 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[]){
    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:

```
(2,(1,(),()),(3,(),(5,(),())))
-------------------------------------
Node:2(1)
LEFT:1
RIGHT:3
-------------------------------------
Node:1(0)
LEFT: NDA
RIGHT: NDA
-------------------------------------
Node:3(1)
LEFT: NDA
RIGHT:5
-------------------------------------
Node:5(0)
LEFT: NDA
RIGHT: NDA
```

O Aplicativo AVLApp

O aplicativo AVLApp utiliza a biblioteca AVLTree, 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

implementar 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: ./avlapp arquivo_de_entrada arquivo_de_saida"<<endl<< "\t
           Por exemplo: ./avlapp input.txt output.txt"<<endl;
   }else{
            AVLApp *app = new AVLapp()
            app->process(argv[1], argv[2]);
            delete app;
  }

   return 0;
}
```

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 avlapp dentro do diretório src. Para executa-lo utilize o comando:

./avlapp

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

.../src$ make clear

Processando Arquivos de Entrada

O programa avlapp 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 aytavés do seguinte comando:

```
./avlapp entrada.txt saida.txt
Arquivo de Entrada:input.txt
Arquivo de Saida:saida.txt
Processando...
Comando: i 2
Inserindo 2 ...
Comando: i 3
Inserindo 3 ...
Comando: i 4
Inserindo 4 ...
Comando: i 5
Inserindo 5 ...
Comando: b 4
Buscando no 4: 3 4 ...
Comando: b 9
Buscando no 9: 3 4 5  ...
Processamento finalizado...
```

Arquivo “entrada.txt”:

```
i 2
i 3
i 4
i 5
b 4
b 9
```

Arquivo “saida.txt”:

```
i 2
(2,(),())
i 3
(2,(),(3,(),()))
i 4
(3,(2,(),()),(4,(),()))
i 5
(3,(2,(),()),(4,(),(5,(),())))
b 4
3 4
b 9
3 4 5
```

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 AVL 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. Uma possível inclusão seria uma implementação de árvores rubro negras. O aplicativo foi testado exaustivamente e em nenhum dos casos pode se relatar qualquer erro de lógica ou vazamento/leak de memória.