Rafaela Martins Vieira

Trabalho Prático
Gerenciador de Árvores AVL

Brasil

2018, v<1>

Rafaela Martins Vieira

Trabalho Prático
Gerenciador de Árvores AVL

Trabalho Prático de Gerenciador de Árvores AVL da disciplina de Estrutura de Dados

Instituto Federal de Minas Gerais – IFMG

Faculdade de Ciência da COmputação

Programa de Graduação

Brasil

2018, v<1>

Resumo

O Trabalho Prático de Gerenciador de Árvores AVL é um trabalho proposto pela disciplina de Estrutura de Dados do Curso de Ciência da Computação do Instituto Federal de Minas Gerais - IFMG Campus Formiga. O Gerenciado em suma cria uma árvore binária de busca balanceada. Os comandos e dados são executados no programa atráves de um arquivo de entrada txt que após lido as funções correspondentes são executadas.

Palavras-chaves: Estrutura-de-Dados. Computação. Árvore.

Lista de abreviaturas e siglas

ED	Estrutura de Dados
AVL	Árvore Binária de Busca Balanceada
ABB	Árvore Binária de Busca.
FB	Fator de Balanceamento

Sumário

Introdução

Este trabalho tem como finalidade práticar o uso de tipos abstratos de dados e estruturas do tipo Árvore. É pedido um programa para gerenciar o ciclo de vida de uma Árvore Binária de Busca de valores inteiros. Tal gerenciado deve garantir que seja possivel a criação de uma ABB, incluir elementos na ABB e mantê-la balanceada através das propriedades da AVL, excluir elementos da árvore binária de busca e manter a árvore balanceada, imprimir a árvore em detrimento da escolha de um dos percursos: pré-ordem, in-ordem e pós-ordem, buscar um dado elemento na árvore (usando a propriedade da árvore binária de busca) e destruir a árvore, liberando a memória utilizada. O gerenciador também deverá ler os dados de entrada a partir de um arquivo, cujo nome é passado como parâmetro na linha de comando.O arquivo de entrada é basicamente uma lista de comandos (um por linha) em formato texto. O último comando é a palavra FIM, que indica o final do arquivo. A elaboração do programa foi realizada, desenvolvido na lingugaem C de programação o algoritmo criado atende os requisitos e sana o problema dado pela disciplina. Sua saída é um arquivo txt com os resultados das funções equivalentes ao comando de entrada como pedido na especificação do trabalho. Visto estes problemas proposto pela disciplina de ED e que a criação do trabalho foi realizada com êxito, este relatório apresenta os passos de criação e funcionamento desse gerenciador.

Parte I

Preparação do relatório

0.1 Estrutura

A primeira estrutura se chama nodo, ela possui três variaveis em seu escorpo que são: "esq", "dir"e valor. A estrutura nodo foi renomeada para Node afim de auxiliar o programador. As variaveis "esq"e "dir"da estrutura são ponteiros do tipo Node, que tem como finalidade ser apontador pra outro Nó da árvore. O filho a esquerda é chamado na estrutura de "esq", já o filho a direita do nó é chamado de "dir". A variavel valor é do tipo inteiro e armazena o valor númerico daquele nó. Veja a estrutura

```
struct no {
    Node *esq;
    Node* dir;
    int valor
};
```

Sendo que:

- Node* esq = Aponta para o nó à esquerda.
- Node *dir = Aponta para o nó à direita.
- int valor = Armazena a informação inteira do nó.

Para inicialização da árvore tem-se outra estrutura que previamente se chama arvore mas é renomeada para *tree*. A Estrutura arvore em suma é um apontador pra raiz da arvore, desta forma o tipo do elemento da estrutura é do tipo apontador para um nó. Veja:

```
struct arvore {
    Node* root;
};
```

Sendo que:

- Node* root = É um apontador para o nó raiz da arvore, ou seja o apontador de inicio da ABB.

Foi definido no programa a seguinte contante: **#define MAX(a, b) ((a) > (b)? (a) : (b))**
Essa constante MAX representa um *if* condicional onde tem-se o comparativo de maior que entre as variaveis a e b, uma vez que a for maior que b retorna-se a variavel a se não retorna-se b.

0.2 Funções

tree* IniciaArvore() - Essa função tem como proposito criar a Arvore, basicamente alocando um espaço de memória do tipo tree e atribuindo valor nulo para o ponteiro da raiz.

Node* IniciaNodo(int valor) - A função IniciaNodo recebe o valor a ser inserido na arvore e aloca o tipo Node na memoria. Os ponteiros esq e dir recebem valor nulo, e o elemento valor da estrutura alocada recebe o valor passado por parametro para a função. Está função retorna um ponteiro para o novo nó criado.

int altura(Node *nodo) - Uma das funções mais usadas no código. A função altura recebe um nó inicial, este nó pode ser o root (raiz) da árvore ou apénas um nó da subarvore. A partir desse nó de forma recursiva todos os nós a partir desse nó inicial são vizitados, O retorno da função é a maior altura(direita ou a esquerda) acrescentado mais um.

int calculoFB(Node* nodo) - Essa função calcula o Fator de Balanceamento, de modo geral o FB é calculado da seguinte forma:

(Altura da subarvore a esquerda - Altura da subarvore a direita)

Para realizar tal calculo a função possui uma variavel chamada fb. A primeira condição da função é verificar se existe filho a esquerda se sim fb recebe a soma de fb mais a altura desse filho. Se existir filho a direita fb recebe a subtração de fb menos a altura correspondente a aquele nó. Percebe-se que o calculo realizado nessa variavel fb é o mesmo da formula citada anteriormente. O retorno da função é um inteiro com o resultado da fb.

Node* rotaciona_RR (Node* nodo) - Essa é a função de rotação simples a direita, uma das mais usadas durante o gerenciamento da arvore pois contribui para o balanceamento da arvore. Essa função recebe uma ponteiro para um nó, este nó será rotacionado para a direita.

Node* rotaciona_LL (Node *nodo) - Essa é a função de rotação simples a esquerda, também é muito usada durante o gerenciamento da arvore pois garante que ela esteja balanceada.Essa função recebe um ponteiro para um nó da arvore e este será rotacionado para a esquerda.

Node* rotationa_LR(Node* nodo) - Essa função de rotação dupla a direita os ponteiros do nó passado são trocados de modo em que é feito uma rotação simples a esquerda e logo após rotação simples a direita.

Node* rotaciona_RL(Node* nodo) - Essa função de rotação dupla a direita os ponteiros do nó passado são trocados de modo em que é feito uma rotação simples a direita e logo após rotação simples a esquerda.

Node* balanceamento(Node* nodo) - Responsável por chamar as principais funções que garantem a AVL. Inicialmente percorre a árvore recursivamente e a cada nivel da arvore chama a função que calcula o seu fator de balanceamento. De acordo com o resultado da FB é feito a verificação de rotação, caso seja maior ou igual a 2 ou menor ou igual a -2 verifica-se a subarvore se é negativo ou positivo o FB.

void balancear_tree (tree *Arvore) - Esta função tem como objetivo enviar a raiz da árvore para a função de balanceamento, após a execução da rotina de balanceamento ele verifica se o nó raiz de retorno da função é igual ao nó raiz enviado se não ele atualiza a raiz inicial da arvore.

void Insere(tree* Arvore, int valor) - A inserção verifica se o valor a ser inserido é maior ou menor que a raiz, se maior desce para a direita até achar nulo, se menor desce para a esquerda até achar nulo. Após achar o ultimo nó ele localiza qual dos dois filhos(esquerda ou direita) ele será do nó. Chama-se a função de criação do nó e após a criação do nó inserido, o ponteiro da raiz recebe o novo nó(filho a esquerda ou filho a direita). Após a inserção ao final da função chama-se a função de balanceamento da arvore, dessa forma a arvore é balanceada após a inserção.

Node *Buscar(tree *Arvore,int valor) - A busca é uma função que percorre a arvore a partir de um valor, ele verifica se o valor a ser buscado é maior ou menor que o nó raiz, se maior desce para a subarvore da direita se menor desce para a subarvore da esquerda.

Node* MaiorDireita(Node *no) - Percorre a subárvore do nó referenciado por parametro até o filho a direita chegar a nulo, se chegar verifica se existe filho a esquerda,o nó recebe o filho a esquerda e continua o percurso pela direita até que ambos filho a esquerda e filho a direita seja nulos, Desta forma retornando o maior elemento a direita.

Node* Excluir(Node *raiz, int valor) - Essa função recebe a raiz da arvore e o valor a ser excluido, é realizado uma busca por toda a árvore até encontrar o nó que possui esse valor, assim que é encontrado verifica-se se ele não possui filhos, se não possui sabe-se que ele é nó folha da arvore e de acordo com as propriedades da AVL é simplesmente exclui-lo. Caso ele tenha um filho a esquerda, o nó a ser excluido recebe o valor do seu filho a esquerda, ele será substituido pelo seu filho de modo a não existir mais na arvore. O mesmo é feito se ele tiver um filho a direita, o nó a ser excluido é substituido pelo seu filho a direita. Se o nó a ser excluido tiver dois filhos é realizada a seguinte ação, como decisão de projeto escolhi pegar o maior filho direito da subarvore esquerda a partir do nó a ser excluido, chama-se a função MaiorDireita e ela me retorna tal nó. Após encontrar o maior e encontrar o nó a ser excluido troca-se o valor desses dois de modo que, o nó a ser excluido recebe o valor inteiro do nó maior a direita. Assim o valor inteiro a ser excluido se tornará um nó folha, e o maior a direita da subarvore a esquerda o seu valor inteiro ficará na raiz onde estava o valor inteiro a ser excluido antes. Deste modo tendo essa organização na arvore, chama-se o Excluir novamente passando o nó á esquerda e o

valor a ser excluido(esse valor não muda pq n foi excluido ainda apenas mudou de lugar na arvore), assim que entrar na função excluir novamente o if condicional irá entrar no if em que o valor a ser excluido é um nó folha assim a exclusão é feita de forma simples. Após excluir o valor chama-se a função balancear_tree, para rebalancear a arvore.

FILE* listarPREORDER(Node* no,FILE* arq) - Esta função exibe os valores da arvore na ordem raiz, esquerda e direita de forma recursiva. É passado por parametro o nó do tipo ponteiro para Node e o ponteiro do arquivo de escrita. Escreve no arquivo o valor do nó atual, logo após recursivamente é chamada a mesma função passando o filho a esquerda do nó atual e por fim chama-se recursivamente a mesma função listarPOSORDEM porém passa-se agora pra função o filho a direita do nó.

FILE* listarINORDEM(Node* no, FILE* arq) - Esta função exibe os valores da arvore na ordem esquerda, raiz e direita de forma recursiva. É passado por parametro o nó do tipo ponteiro para Node e o ponteiro do arquivo de escrita. Recursivamente é chamada a mesma função passando o filho a esquerda do nó atual, escreve no arquivo o valor do nó atual e por fim chama-se recursivamente a mesma função listarPOSORDEM porém passa-se agora pra função o filho a direita do nó.

FILE* listarPOSORDEM(Node* no, FILE* arq) - Esta função exibe os valores da arvore na ordem esquerda, direita e raiz de forma recursiva. É passado por parametro o nó do tipo ponteiro para Node e o ponteiro do arquivo de escrita. Recursivamente é chamado a mesma função passando o filho a esquerda do atual, logo após, chama-se recursivamente a mesma função listarPOSORDEM passa-se agora pra função o filho a direita do nó. É escrito no arquivo o valor do nó e o retorno é o ponteiro do arquivo de escrita. void liberar(Node * no) - percorre a arvore dando free. void liberar_memoria(tree * no) - Envia a raiz da arvore por parametro e está função chama a função liberar.

Parte II

Resultados

0.3 Funcionamento do algoritmo

Com a implementação dessa TAD é possível garantir uma árvore AVL. Porém vale lembrar que é de encargo do programador chamar as funções de balanceamento sempre que necessário, uma vez que as funções que garantem a AVL são independentes das funções da Arvore ABB. De modo geral a estrutura do programa é realizada da seguinte forma: É realizado no main a criação da árvore chamando a função IniciaArvore. Abre-se o arquivo de leitura, e abre-se também o arquivo de escrita, esses dois só serão fechados após receber um final de arquivo de leitura. A inclusão de um novo nó se dá com a chamada da função insere, passando a raiz da arvore mais o valor. Essa função raiz percorre toda a árvore procurando qual a posição que esté nó será inserido, ao encontrar o espaço dá inserção, chama a função IniciaNode(), passando o valor inteiro, após a criação deste nó, ele é encaixado na árvore, no final da função chama-se Balaceamento_tree() passando a raiz da arvore e a partir dessa função chama-se balanceamento percorre a arvore calculando em cada nível o FB, e de acordo com o resultado do calculo é feito a rotação necessária. A exlusão é feita uma busca pela arvore se esse valor a ser excluido está a direita ou a esquerda. Ao encontrar esse valor verifica se ele é um nó folha, se ele possui um ou dois filhos, se for nó raiz realiza normalmente a exclusão, se tiver um filho(direita ou esquerda) esse filho se torna a raiz, se existir dois filhos procura o antecessor troxa os valores inteiros de forma em que o valor inteiro do antecessor fica no lugar da raiz que ia ser excluida e o valor inteiro da raiz(esse é o valor que queremos excluir) fica no lugar desse valor antecessor.Chama-se a função Excluir novamente de modo que agora o valor a ser excluido é um nó folha. A busca por um valor é feito com uma verificação sempre nó e dependendo se o valor buscado é maior ou menor que a raiz desce para a esquerda ou para a direita, ao identificar o valor retorna-se ele, caso não escreve no arquivo que o valor não existe. No final da leitura do arquivo a árvore é destruída.

Em sumo o algoritmo funciona dessa maneira, atingiu o abjetivo da disciplina, a implementação da arvore ABB com funções que garantem o balanceamento da mesma e a torne uma árvore AVL. O arquivo de saída retorna os resultados, a divisão do arquivo é feita por main, tadArvore.c e tadArvore.h. Os arquivos txt de teste estão junto ao arquivo compactado.

0.4 Como executar o programa

- Abra o terminal Linux
- Digite o seguinte comando **gcc *.c -o NomeDoExecutavel**
- Após compilar o arquivo e criar o executavél volte ao terminal

- Digite **.^ NomeDoExecutavel NomeArquivoEntrada.txt**
- Se der erro de não executável vá em propriedades do arquivo "NomeDoExecutavel"e marque a opção executável.
- Execute o ultimo comando novamente via terminal.

1 Conclusão

O trabalho enviado atende os requisitos pedidos na disciplina garantindo a criação e exclusão da àrvore, o balanceamento dos nós, a inserção e exclusão de um novo elemento, a busca e impressão da árvore. O algoritmo foi baseado no que consta no slide da disciplina ED.

Referências