## 实验课（九）——树的创建,查找,删除操作(雷镇豪)

一、实验目的

1. 了解树的概念
2. 掌握树的三种遍历算法
3. 掌握普通二叉树的构建
4. 掌握对二叉查找树的插入、查找、删除

二丶实验内容（代码）

1. BinaryTree.h：

#include<iostream>

#include <stack>

using namespace std;

struct TreeNode{

TreeNode \*left;

TreeNode \*right;

string value;

TreeNode(){

left=right=NULL;

value="";

}

TreeNode(char c){

left=right=NULL;

value=c;

}

};

class BinaryTree

{

public:

BinaryTree();

virtual ~BinaryTree();

void BuildTree\_Postorder(TreeNode\* &aroot, string &preorder, int &index);//创建树时必须在aroot形参前加上&

void Print(TreeNode\* aroot);//.h文件中的aroot只是一个名称

int GetTreeHeight(TreeNode\* aroot);

TreeNode \*tree\_root;//树节点结构体类型的指针(类成员的成员变量)

void Search(TreeNode\* aroot,string str);

void PrintPath();

stack<string> stack1;//存路径

stack<string> stack2;

TreeNode\* FatherNode(TreeNode\* Root, string del\_str);//找到del\_str对应节点的父亲节点

void find\_and\_del(TreeNode\* &aroot, string del\_str, TreeNode\* &Root);//找到待删除位置节点

void del(TreeNode\* &aroot, string del\_str, TreeNode\* &Root);

TreeNode\* FindMax(TreeNode\*);

};

1. BinaryTree.cpp：

#include "BinaryTree.h"

BinaryTree::BinaryTree()

{

//ctor

tree\_root=NULL;

}

BinaryTree::~BinaryTree()

{

//dtor

}

void BinaryTree::Print(TreeNode \*aroot){

if(aroot!=NULL){

Print(aroot->left);

Print(aroot->right);

cout<<aroot->value<<" ";

}

}

int BinaryTree::GetTreeHeight(TreeNode\* aroot){

if(aroot==NULL)

return 0;

else {

int l=GetTreeHeight(aroot->left)+1;

int r=GetTreeHeight(aroot->right)+1;

int lr=l>r?l:r;

return lr;

}

}

void BinaryTree::BuildTree\_Postorder(TreeNode\* &aroot, string &preorder, int &index){

if(index>=0){

if(preorder[index] == '#'){

aroot = NULL;

}

else{

aroot = new TreeNode;

aroot->value = preorder[index];

BuildTree\_Postorder(aroot->right,preorder,--index);

BuildTree\_Postorder(aroot->left,preorder,--index);

}

}

return;

}

void BinaryTree::Search(TreeNode\* aroot, string str)

{

if(aroot != NULL){

if(aroot->value == str){

stack1.push(aroot->value);

return;

}

else{

if(str < aroot->value){

stack1.push(aroot->value);

Search(aroot->left,str);

}

else{

stack1.push(aroot->value);

Search(aroot->right,str);

}

}

}

}

void BinaryTree::PrintPath()

{

cout << "根节点到c的路径为:";

string item;

while(!stack1.empty()){

item = stack1.top();

stack1.pop();

stack2.push(item);

}

while(!stack2.empty()){

item = stack2.top();

stack2.pop();

cout << item;

if(item != "c") cout << " -> ";

}

cout << endl;

}

TreeNode\* BinaryTree::FatherNode(TreeNode\* Root, string del\_str)//Root是最初的根节点

{

if(Root->value < del\_str){

if(Root->right->value == del\_str){//str是Root的右孩子

return Root;

}

else return FatherNode(Root->right,del\_str);///debug 返回值是TreeNode\*类型

}

else if(Root->value > del\_str){///debug

if(Root->left->value == del\_str){//str是Root的左孩子

return Root;

}

else return FatherNode(Root->left,del\_str);

}

}

void BinaryTree::find\_and\_del(TreeNode\* &aroot, string del\_str, TreeNode\* &Root)

{

if(aroot != NULL){

if(aroot->value == del\_str){

del(aroot,del\_str,Root);

}

else{

if(del\_str < aroot->value){

find\_and\_del(aroot->left,del\_str,Root);

}

else{

find\_and\_del(aroot->right,del\_str,Root);

}

}

}

}

TreeNode\* BinaryTree::FindMax(TreeNode\* aroot)//寻找从当前节点开始小于它的最大节点

{

if(aroot->right==NULL) return aroot;

else return FindMax(aroot->right);

}

void BinaryTree::del(TreeNode\* &aroot, string del\_str, TreeNode\* &Root)///传入的aroot是待删除位置

{ ///Root是最初的根节点

if(aroot->left==NULL && aroot->right==NULL){//左右均为NULL

aroot = NULL;

}

else if(aroot->left==NULL||aroot->right==NULL){

TreeNode\* Father = FatherNode(Root,del\_str);//找到待删除节点的父亲节点

if(aroot->left==NULL){//左子树为空，右子树不为空

if(Father->left->value == del\_str){//待删除位置是父亲节点的左孩子

Father->left = aroot->right;//把父亲的左指针指向左孩子的右孩子

}

else Father->right = aroot->right;

}

else if(aroot->right==NULL){//右子树为空，左子树不为空

if(Father->left->value == del\_str){

Father->left = aroot->left;

}

else Father->right = aroot->left;

}

}

else{//左右均有子节点

TreeNode\* LeftMax = FindMax(aroot->left);//找到待删除节点左边最大节点

TreeNode\* Father = FatherNode(Root,LeftMax->value);//找到左边最大节点的父亲

aroot->value = LeftMax->value;

Father->right = LeftMax->left;

}

}

3、main.cpp:

#include <iostream>

#include"BinaryTree.h"

using namespace std;

int main()

{

BinaryTree tree1;

string preorder="###ca##ji####spom";

int index = preorder.length()-1;

tree1.BuildTree\_Postorder(tree1.tree\_root,preorder,index);

cout << "后序遍历所构建的树:";

tree1.Print(tree1.tree\_root);

cout << endl;

tree1.Search(tree1.tree\_root,"c");

tree1.PrintPath();

string del\_str;

cout << "请输入待删除节点:";

cin >> del\_str;//待删除节点

cout << "删除节点后继续后序遍历该树:";

TreeNode\* Root = tree1.tree\_root;//Root保存最初的根节点,用于找父亲的函数

tree1.find\_and\_del(tree1.tree\_root,del\_str,Root);//第一个参数必须引用传递

tree1.Print(tree1.tree\_root);

return 0;

}

1. 实验结果

