哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验名称：树型结构与应用

实验项目：

1、 利用前序序列和中序序列构造二叉树

2、 完成二叉树的遍历操作（7 种）

3、 对任意一篇英文文章，利用 Huffman 树进行编码、译码、压缩存储。

班级：
学号：
姓名：

一、实验目的

熟悉二叉树的建立与遍历。

二、实验要求及实验环境

实验要求：实现递归、非递归的遍历，前序、中序和后序都要求实现。

实验环境：Windows, Code::Blocks。

三、设计思想（本程序中的用到的所有数据类型的定义，核心算法的流程图等）

数据定义如下：

```
template <typename T>
class Node {
public:
    Node * next;
    T data;
    Node(T _dat,Node * p = NULL)
    Node(){}
};
template <typename T> class Stack {
public:
    Node<T>* head;
    //初始化一个栈
    Stack()
    //析构自动释放内存
    ~Stack()
    //判空
```

```
    bool empty()
    //弹栈
    T pop()
    //栈顶
    T top()
    //入栈
    Stack<T>* push(T data)
};

template <typename T>class Queue {
    Node<T> * frontptr;
    Node<T> * rearptr;
public:
    Queue()
    ~Queue()
    bool empty()
    T front()
    Queue * enqueue(T data)
    T dequeue()
    void showqueue()
};
```

```
template <typename T>
class TreeNode {
public:
	TreeNode * left;
	TreeNode * right;
	T data;
	TreeNode(T	_dat,TreeNode*	_lc=NULL,TreeNode	*
_rc=NULL)
};
namespace tree {
#define PROORDER 0
#define INORDER 1
#define POSTORDER 2
}
```

非递归先序遍历二叉树算法如图 3-1

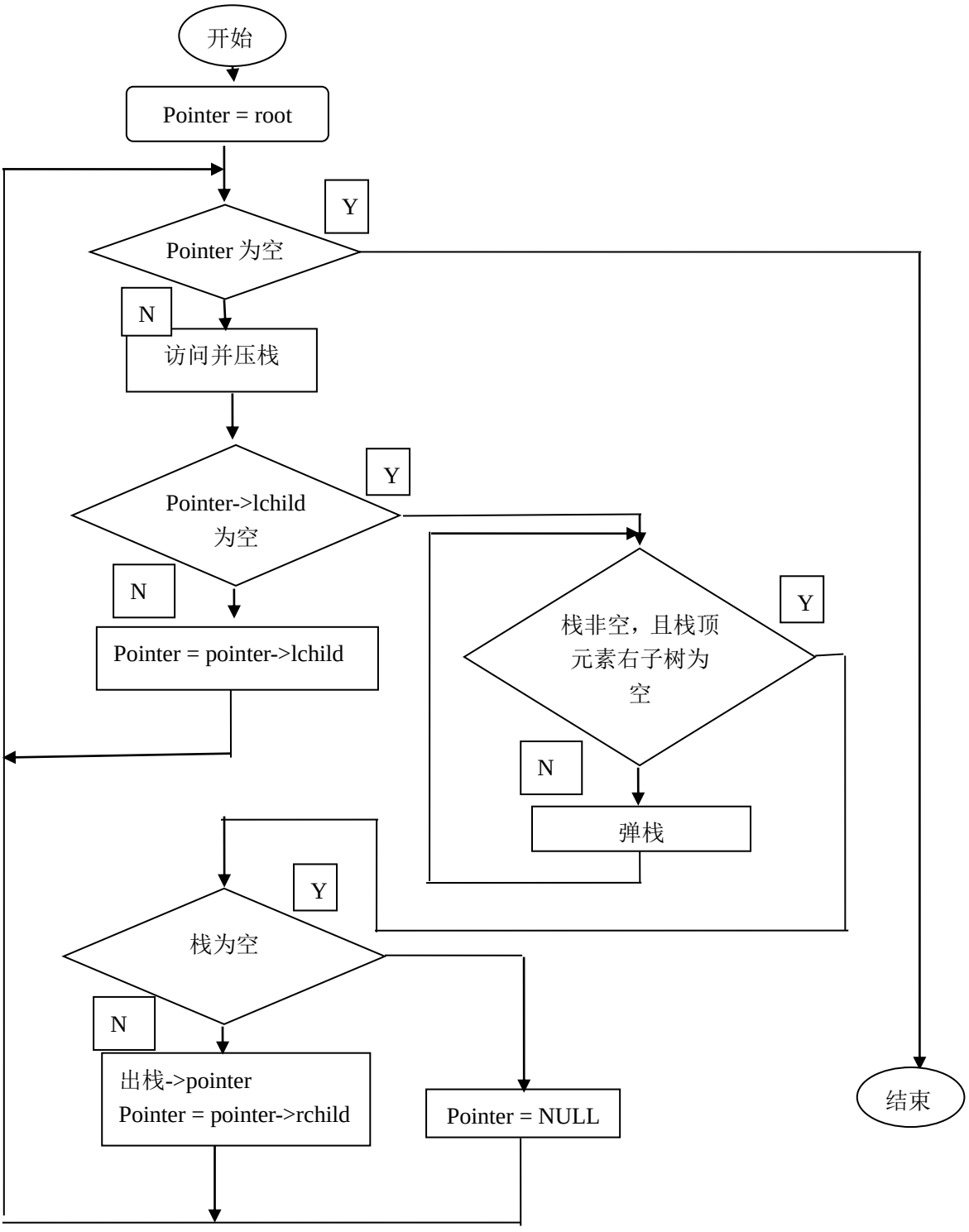

图 3-1 非递归先序遍历二叉树算法

四、测试结果如图 4-1

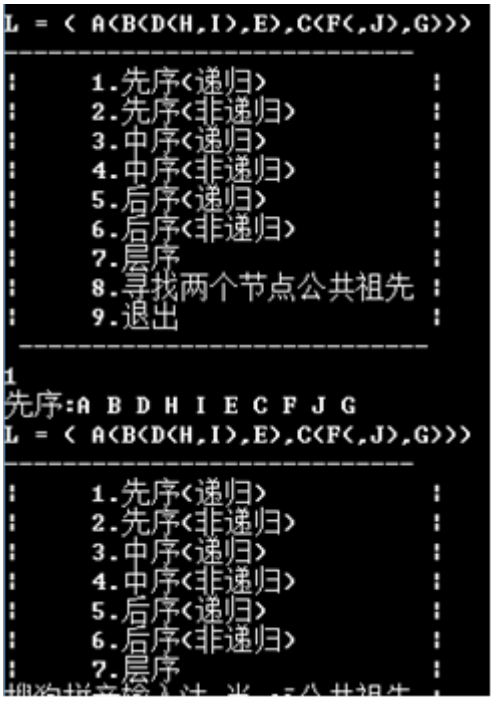

图 4-1 二叉树遍历结果图示

五、系统不足与经验体会

本次实验深刻理解了面向对象设计思想，在其中引入了一些函数式方法，使得程序更加灵活。掌握了一种空间、时间占用 o(1)的非递归遍历树方法。体会了算法的魅力。

六、附录：源代码（带注释）

```
#define MAXINPUT 200
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <string>
#include <math.h>
#include <sstream>
#include "mystl.h"
#define ESP 1e-6
using namespace::std;
using namespace::std;
template <typename T>
```

```
class Node {//线性结构节点类，栈，队列，链表
public:
	Node * next;
	T data;
	Node(T _dat,Node * p = NULL){
		next = p;
		data = _dat;
	}
	Node(){}
};
//栈模板类
template <typename T> class Stack {
public:
	Node<T>* head;
	//初始化一个栈
	Stack(){
		this->head=NULL;
	}
	//析构自动释放内存
	~Stack(){
		Node<T>* p;
		while (this->head != NULL) {
			p=head;
			head=head->next;
			delete p;
		}
	}
	//判空
	bool empty(){
		return head == NULL;
	}
	//弹栈
	T pop(){
		if (empty()) {
			string err("EMPTY STACK!");
			throw err;
		}
		Node<T>* p = head;
		head = head->next;
		T _data = p->data;
		delete p;
		return _data;
	}
	//栈顶
```

```
    T top(){
        if (empty()) {
            string err("EMPTY STACK!");
            throw err;
        }
        return head->data;
    }
    //入栈
    Stack<T>* push(T data){
        Node<T>* p = new Node<T>(data,head);
        if (p==NULL) {
            string err("OUT OF MEMERAY!");
            throw err;
        }
        head = p;
        return this;
    }
    //非栈功能函数，调试使用
    void show(){
        cout<<"\n##########\n";
        Node<T>* p=head;
        int i=1;
        while (p!=NULL) {
            cout<<i<<"|   "<<p->data<<endl;
        }
    }
};
//队列模板类
template <typename T>class Queue {
    Node<T> * frontptr;
    Node<T> * rearptr;

public:
    Queue(){//初始化
        frontptr=NULL;
        rearptr=NULL;
    }
    ~Queue(){//析构一个队列，释放内存
        Node<T> *p;
        while (frontptr!=rearptr) {
            p=frontptr;
            frontptr=frontptr->next;
            delete p;
        }
```

```
        delete frontptr;
}//判空
bool empty(){
        return frontptr==NULL;
}
//获得最前元素
T front(){
        if (empty()) {
                string err("EMPTY QUEUE!");
                throw err;
        }
        return frontptr->data;
}
//入队
Queue * enqueue(T data){
        if (empty()) {
                rearptr=frontptr=new Node<T>(data);
                return this;
        }
        rearptr->next=new Node<T>(data);
        rearptr=rearptr->next;
        return this;
}
//出队
T dequeue(){
        if (empty()) {
                string err("EMPTY QUEUE!");
                throw err;
        }
        T data=frontptr->data;
        if (frontptr==rearptr) {
                frontptr=rearptr=NULL;
        }else{
                frontptr=frontptr->next;
        }
        return data;
}

void showqueue(){
        if (empty()) {
                string err("EMPTY QUEUE!");
                throw err;
        }
        cout<<"\n##########\nQUEUE FRONT\n";
```

```
            int i(0);
            Node<T>* p=frontptr;
            for (; p!=rearptr; i++,p=p->next) {
                cout<<i<<" : "<<p->data<<endl;
            }
            cout<<i<<" : "<<p->data<<endl;
        }
};

//模板树节点
template <typename T>
class TreeNode {
public:
        TreeNode * left;
        TreeNode * right;
        T data;
        TreeNode(T _dat,TreeNode* _lc=NULL,TreeNode * _rc=NULL){
            data=_dat;
            left=_lc;
            right=_rc;
        }
};
//遍历方案模板宏
namespace tree {
#define PROORDER 0
#define INORDER 1
#define POSTORDER 2

}
//树
template <typename T>
class Tree {
        TreeNode<T>* root;//根

        //广义表输出
        void showtreewithtablenode(function<void (T)>show,TreeNode<T>* localroot){
            if (localroot==NULL) {
                return;
            }else{
                show(localroot->data);
                if (localroot->left==NULL&&localroot->right==NULL) {
                    return;
                }
                cout<<'(';
```

```
            if (localroot->left) {
                showtreewithtablenode(show, localroot->left);
            }

            cout<<',';
            if (localroot->right) {
                showtreewithtablenode(show, localroot->right);

            }
            cout<<')';
        }
}
//删除树
  void deleteTree(TreeNode<T>* _localroot){
        if (_localroot==NULL) {
            return;
        }else{
            deleteTree(_localroot->left);
            deleteTree(_localroot->right);
            delete _localroot;
        }
}
//字符串建立树
TreeNode<T> * creatTreeWithString(T * &str,T nullmark){
        T _ch=str[0];
        str++;
        if (_ch==nullmark) {
            return NULL;
        }else{
            TreeNode<T> *    _tree = new TreeNode<T>(_ch);
            _tree->left=creatTreeWithString(str,nullmark);
            _tree->right=creatTreeWithString(str,nullmark);
            return _tree;
        }
}
//节点遍历
template<char ORDER=PROORDER>
void mapnode(void(func)(TreeNode<T>* thisnode),TreeNode<T>* localroot){
        if (localroot==NULL) {
            return;
        }else{
            if(ORDER==PROORDER)func(localroot);
            mapnode(func, localroot->left);
            if(ORDER==INORDER)func(localroot);
```

```
            mapnode(func, localroot->right);
            if (ORDER==POSTORDER)func(localroot);
        }
    }
    template<char ORDER=PROORDER>
    void mapnode(void(func)(TreeNode<T>* thisnode,string,void *),TreeNode<T>* localroot,string path,void * ptr=NULL){
        if (localroot==NULL) {
            return;
        }else{
            if(ORDER==PROORDER)func(localroot,path,ptr);
            mapnode<ORDER>(func, localroot->left,path+"L",ptr);
            if(ORDER==INORDER)func(localroot,path,ptr);
            mapnode<ORDER>(func, localroot->right,path+"R",ptr);
            if (ORDER==POSTORDER)func(localroot,path,ptr);
        }
    }
    // 支持 lambda 表达式和 operater()的闭包调用方式。
    template<char ORDER=PROORDER>
    void mapnode(function<void(TreeNode<T>*,string)>func,TreeNode<T>* localroot,string path){
        if (localroot==NULL) {
            return;
        }else{
            if(ORDER==PROORDER)func(localroot,path);
            mapnode<ORDER>(func, localroot->left,path+"L");
            if(ORDER==INORDER)func(localroot,path);
            mapnode<ORDER>(func, localroot->right,path+"R");
            if (ORDER==POSTORDER)func(localroot,path);
        }
    }
    //深度
    int subtreedeep(TreeNode<T>* localroot ,int deep=0){
        if (localroot==NULL) {
            return deep;
        }else{
            int ld=subtreedeep(localroot->left,deep+1);
            int lr=subtreedeep(localroot->right,deep+1);
            return ld>lr ? ld:lr ;
        }
    }
    //前中序建树
    TreeNode<T>* creatTreeWithFRLandMRL(T * frl,T* mrl,int len){
        if (len<=0) {
```

```
            return NULL;
        }
        TreeNode<T> * _tree = new TreeNode<T>(*frl);
        int index = nodestringfind(mrl, *frl, len);
        _tree->left = creatTreeWithFRLandMRL(frl+1, mrl, index);
        _tree->right = creatTreeWithFRLandMRL(frl+index+1, mrl+index+1, len-index-1);
        return _tree;
    }
    //路径找节点
    int nodestringfind(T * str,T d,int maxlen){
        for (int i=0; i<maxlen; i++) {
            if (d==str[i]) {
                return i;
            }
        }
        return -1;
    }
public:
    //构造
    Tree(){
        root = NULL;
    }
    //字符串建树
    Tree * refreshTreeFormstring(char * str,T nullmark){
        int cont=0;
        for (int i=0; str[i]!='\0'; i++) {
            if (str[i]==nullmark) {
                cont++;
            }else{
                cont--;
            }
        }
        if (cont!=1) {
            string err("cant match");
            throw err;
        }
        deleteTree();                       //clear old tree
        root = creatTreeWithString(str, nullmark);
        return this;
    }
    //前中序建树
    Tree * refreshTreeFormFRLandMRL(T * frl,T* mrl,int len){
        deleteTree();
        root = creatTreeWithFRLandMRL(frl, mrl,len);
```

```
        return this;
    }
    //遍历三种方式
    template<char ORDER=PROORDER>
    void map(void(func)(TreeNode<T>* thisnode)){
        mapnode<ORDER>(func, root);
    }
    template<char ORDER=PROORDER>
    void map(void(func)(TreeNode<T>* thisnode,string ,void *),void * ptr=NULL){
        mapnode<ORDER>(func, root,"",ptr);
    }
    //  支持 lambda 表达式和 operater()的闭包调用方式。
    template<char ORDER=PROORDER>
    void                    map(function<void(TreeNode<T>*,string)>foo=[=](TreeNode<T>*
node,string){cout<<node->data;}){
        mapnode<ORDER>(foo, root,"");
    }
    //深度
    unsigned long static nodedeepformpath(string path){
        return path.length();
    }
    //层序
    void levelorder(){
        Queue<TreeNode<T> *> tq;
        if (root==NULL) {
            return;
        }
        tq.enqueue(root);
        while (!tq.empty()) {
            TreeNode<T>* p= tq.dequeue();
            cout<<p->data;
            if (p->left!=NULL) {
                tq.enqueue(p->left);
            }
            if (p->right!=NULL) {
                tq.enqueue(p->right);
            }
        }
    }
    //获得路径
    string    getnodepath(T nodedata){
        string    _path;
        auto getnodepathcallbackfunc = [&](TreeNode<T> * thisnode,string path) {
            if (nodedata==thisnode->data) {
```

```
                _path=path;
            }
        };
        map(getnodepathcallbackfunc);
        return _path;
}
//路径得到节点
T getnodebypath(string path){
        TreeNode<T>*ptr=root;
        while (path!="") {
            if (ptr==NULL) {
                string err("□中无此节点！");
                throw err;
            }
            switch (path[0]) {
                case 'L':
                    ptr=ptr->left;
                    break;
                case 'R':
                    ptr=ptr->right;
                    break;
                default:
                    string err("路径不合法！");
                    throw err;
                    break;
            }
            path.erase(0,1);
        }
        return ptr->data;
}
//树深度
int treedeep(int deep=0){
        return subtreedeep(root);
}
//前中序非递归
Tree * refreshTreeFormFRLandMRLnore(T * frl,T * mrl,int len){
        deleteTree();
        Stack<TreeNode<T>*> stk;
        for (int i=0; i<len; i++) {
            if (stk.empty()) {
                TreeNode<T> * newnodeptr = new TreeNode<T>(frl[i]);
                if (root==NULL) {
                    root=newnodeptr;
                }
```

```
                stk.push(newnodeptr);
                continue;
            }else{

                int sindex=nodestringfind(mrl, stk.top()->data, len);
                int cindex=nodestringfind(mrl, frl[i], len);
                if (sindex==-1||cindex==-1) {
                    string err("match failed!");
                    throw err;
                }
                if(sindex>cindex){
                    stk.top()->left=new TreeNode<T>(frl[i]);
                    stk.push(stk.top()->left);
                }else{
                    TreeNode<T>* faptr=stk.top();
                    while (sindex<cindex) {
                        faptr=stk.pop();
                        sindex=stk.empty() ? len : nodestringfind(mrl, stk.top()->data,
len);//bugpoint 1 26.1.03 过根 break;
                        cindex=nodestringfind(mrl, frl[i], len);
                    }
                    faptr->right=new TreeNode<T>(frl[i]);
                    stk.push(faptr->right);
                }
            }
        }
return this;
    }
    //广义表
    void showtreewithtable(function<void (T)>show=[=](T t){cout<<t;} ){

        cout<<'(';
        showtreewithtablenode(show, root);
        cout<<')';
    }
    //前序迭代
    void showtreeprenore(){
        TreeNode<T>* p=root;
        while (p!=NULL) {

            if (p->left==NULL) {
                cout<<p->data;
                p=p->right;
            }else{
```

```
                TreeNode<T> * pp=p->left;
                if (pp->right==NULL) {
                    cout<<p->data;
                    pp->right=p;
                    p=p->left;
                    continue;
                }
                while (pp->right!=NULL && pp->right!=p) {
                    pp=pp->right;
                }
                if (pp->right==NULL) {
                    cout<<p->data;
                    pp->right=p;
                    p=p->left;
                }else{
                    pp->right=NULL;
                    //cout<<p->data;
                    p=p->right;
                }
            }
        }
}
//中序迭代
void showtreeinnore(){
    TreeNode<T>* p=root;
    while (p!=NULL) {

        if (p->left==NULL) {
            cout<<p->data;
            p=p->right;
        }else{
            TreeNode<T> * pp=p->left;
            if (pp->right==NULL) {
                pp->right=p;
                p=p->left;
                continue;
            }
            while (pp->right!=NULL && pp->right!=p) {
                pp=pp->right;
            }
            if (pp->right==NULL) {
                pp->right=p;
                p=p->left;
            }else{
```

```
                    pp->right=NULL;
                    cout<<p->data;
                    p=p->right;
                }
            }
        }
}
//后续迭代
void showtreelastnore(){
    T * td=new T;
    TreeNode<T>* newroot=new TreeNode<T>(*td);
    newroot->left=root;
    TreeNode<T>* p=newroot;
    while (p!=NULL) {

        if (p->left==NULL) {
            p=p->right;
        }else{
            TreeNode<T> * pp=p->left;
            if (pp->right==NULL) {
                pp->right=p;
                p=p->left;
                continue;
            }
            while (pp->right!=NULL && pp->right!=p) {
                pp=pp->right;
            }
            if (pp->right==NULL) {
                pp->right=p;
                p=p->left;
            }else{
                T tempdata=p->data;
                do{
                    pp=p->left;
                    while (pp->right->data!=tempdata) {
                        pp=pp->right;
                    }
                    if (pp->right->data==p->data) {
                        pp->right=NULL;
                    }
                    cout<<pp->data;
                    tempdata=pp->data;
                }while(pp!=p->left);
                p=p->right;
```

```
                    }
                }
            }
            delete newroot;
            delete td;
        }
        //析构
        ~Tree(){
            deleteTree();
        }
        //删除
        void deleteTree(){
            deleteTree(root);
        }

};

//主函数
int main(int argc, const char * argv[])
{
    Tree<char> tree ;
    while (1) {
        try{

        string select;
        cout<<"\n  选择一种树的构造方式：1,满前序串,2,前序中序串,3,前序中序非递归,4,使用预设树,5,取镜像树,6,查找共同父节点,7,输出,0 退出"<<endl;
        cin>>select;
        switch (select[0]) {//进入菜单
            case '1':
                cout<<"输入串，#为空，可重复节点\n";
                char str[50];
                scanf("%s",str);
                tree.refreshTreeFormstring(str,'#');
                break;
            case '2':
                char sp[50];
                char sm[50];
                cout<<"输入前序串\n";
                scanf("%s",sp);
                cout<<"输入中序串\n";
                scanf("%s",sm);
                tree.refreshTreeFormFRLandMRL(sp, sm, (int)strlen(sp));
                break;
```

```
                case '3':
                    char sp2[50];
                    char sm2[50];
                    cout<<"输入前序串\n";
                    scanf("%s",sp2);
                    cout<<"输入中序串\n";
                    scanf("%s",sm2);
                    tree.refreshTreeFormFRLandMRLnore(sp2, sm2,(int ) strlen(sp2));
                    break;
                case '4':
                    tree.refreshTreeFormFRLandMRLnore("ABDHIECFJG",    "HDIBEAFJCG",
10);
                    break;
                case '5':

                    tree.map([=](TreeNode<char>* node,string){
                        TreeNode<char> *temp=node->left;
                        node->left=node->right;
                        node->right=temp;
                    });
                    break;
                case '0':
                    return 0;
                case '6':
                {
                    cout<<"\n>>>>>>>>>>>>>>>找最近共同祖先\n";
                    char a,b;
                    do{
                        cout<<"\n>>>>>>>>>>>>>>>输入两个节点，空格分隔\n";
                    }while (scanf(" %c %c",&a,&b)!=2);
                    string path1=tree.getnodepath(a);
                    string path2=tree.getnodepath(b);
                    int i=0;
                    for (; path1[i]==path2[i]; i++);
                    string bothpath=path1.erase(i);
                    cout<<"公共父节点>"<<tree.getnodebypath(bothpath);
                    break;
                }
                case '7':
                    cout<<"\n>>>>>>>>>>>>>>>树的广义表\n";
                    tree.showtreewithtable();

                    cout<<"\n>>>>>>>>>>>>>>>前序遍历\n";
                    tree.map<PROORDER>();
```

```
                cout<<"\n>>>>>>>>>>>>>>>>前序遍历非递归，时 o(n)空 o(1)\n";
                tree.showtreeprenore();
                cout<<"\n>>>>>>>>>>>>>>>>中序遍历\n";
                tree.map<INORDER>();
                cout<<"\n>>>>>>>>>>>>>>>>中序遍历非递归，时 o(n)空 o(1)\n";
                tree.showtreeinnore();

                cout<<"\n>>>>>>>>>>>>>>>>后序遍历\n";
                tree.map<POSTORDER>();
                cout<<"\n>>>>>>>>>>>>>>>>后序遍历非递归，时 o(n)空 o(1)\n";
                tree.showtreelastnore();
                cout<<"\n>>>>>>>>>>>>>>>>层序遍历\n";
                tree.levelorder();
                break;
            default:

                cout<<"输入错误！";
                continue;

        }
        }
        catch(...){
            cout<<"输入有误，请重新输入！树已被置为预设树。";
            tree.refreshTreeFormFRLandMRLnore("ABDHIECFJG", "HDIBEAFJCG", 10);

        }

    }

    return 0;
}
```