山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：二叉树操作		
实验学时：2	实验日期： 11.23	

实验目的：
二叉树基础、遍历

软件开发工具：
Vscode

1. 实验内容
（1）创建二叉树类。二叉树的存储结构使用链表。提供操作：前序遍历、中序遍历、后序遍历、层次遍历、计算二叉树结点数目、计算二叉树高度，其中前序遍历要求以递归方式实现，中序遍历、后序遍历要求以非递归方式实现。
（2）接收二叉树前序序列和中序序列(各元素各不相同)，输出该二叉树的后序序列。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）
（1）
前序遍历：递归实现，先输出根元素，再递归左右子树
中序遍历：非递归，定义一个栈，不断将根弹入栈，访问左子树，左子树到底之后，再弹出栈，访问根元素，再遍历右子树
后序遍历：同样先将根入栈，访问左子树，到底后将跟弹出栈，一个指针 pre 标记弹出的元素，然后访问右子树，若右子树遍历完，即此时元素等于 pre 即为根元素，访问根元素
层序遍历：定义一个队列，从根节点开始，将左右节点放入队列，再不断从队首弹出元素并访问，再放入左右节点
节点个数：递归，求该节点左右子树节点个数和加一，不断递归
高度：递归，找到左右子树高度较大的一个，h++为根的高度，求左右子树高度即递归

（2）
从前序遍历数组中找根，再从中序遍历中找到该根在的位置，往左往右找到它的左右子树，进行建树，不断递归。建树完成后用递归实现后序遍历

3. 测试结果（测试输入，测试输出）
（1）

```
10
2 -1
4 3
6 -1
5 8
9 7
-1 -1
-1 -1
-1 -1
10 -1
-1 -1
1 2 4 5 9 10 7 8 3 6
10 9 5 7 4 8 2 6 3 1
10 9 7 5 8 4 6 3 2 1
1 2 4 3 5 8 6 9 7 10
10 9 2 6 4 1 1 1 2 1
6 5 2 4 3 1 1 1 2 1
```

（2）

```
5
1 2 4 5 3
4 2 5 1 3
4 5 2 3 1
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

第一遍写的代码不知哪里有问题，只能过样例

```
#include <iostream>
using namespace std;

template <class T>
struct binarytreenode{
    T element;
    binarytreenode<T>* leftchild,
                     * rightchild;
    binarytreenode(){
        element=0;
        leftchild=rightchild=NULL;
    }
    binarytreenode(const T& thelement){
        element=thelement;
        leftchild=NULL;
        rightchild=NULL;
    }
    binarytreenode(const T& thelement, binarytreenode<T>* theleftchild,
binarytreenode<T>* therightchild){
        element=thelement;
        leftchild=theleftchild;
        rightchild=therightchild;
```

```
    }
};

template<class T>
class mystack{   //数组描述
public:
    mystack(int n=10);
    bool empty();
    int size();
    T top();
    void pop();
    void push(T& thelement);
    ~mystack();
private:
    T* element;
    int stacktop;
    int stacksize;
};

template<class T>
mystack<T>::mystack(int n){
    stacksize=n;
    element=new T [n];
    stacktop=-1;
}

template<class T>
bool mystack<T>::empty(){
    return stacktop==-1;
}

template <class T>
int mystack<T>::size(){
    return stacktop+1;
}

template <class T>
T mystack<T>::top(){
    if(stacktop!=-1){
        return element[stacktop];
    }
    else{
        return 0;
    }
}
```

```
template <class T>
void mystack<T>::pop(){
    if(stacktop!=-1){
        element[stacktop--]=0;
    }
}

template <class T>
void mystack<T>::push(T& thelement){
    if(stacktop+1==stacksize){                    //若空间不够,重新进行动态分配
        stacksize*=2;
        T* temp = new T [stacksize];
        for(int i=0;i<stacksize;i++){
            temp[i]=element[i];
        }
        element=temp;
    }
    element[++stacktop]=thelement;
}

template<class T>
mystack<T>::~mystack(){
    stacksize=0;
    stacktop=-1;
    T* p=element;
    delete [] p;
    element=NULL;
}

template<class T>
class myqueue{
public:
    myqueue(int n=100);
    bool empty();
    void push(T& thelement);
    void pop();
    T front();
    T back();
    int size();
private:
    int queuefront;
    int queueback;
    int queuelength;
    int queuesize;
    T* element;
};
```

```
template<class T>
myqueue<T>::myqueue(int n){
    element=new T [n];
    queuefront=n-1;
    queueback=n-1;
    queuelength=n;
    queuesize=0;
}

template <class T>
bool myqueue<T>::empty(){
    return queuefront == queueback;
}

template <class T>
void myqueue<T>::push(T& thelement){
    if(queuefront==(queueback+1)%queuelength){                    //若空间不够，
重新进行动态分配
        queuelength*=2;
        T* temp = new T [queuelength];
        for(int i=0;i<queuelength;i++){
            temp[i]=element[i];
        }
        element=temp;
    }
    queueback = (queueback+1)%queuelength;
    element[queueback]=thelement;
    queuesize++;
}

template<class T>
void myqueue<T>::pop(){
    queuefront=(queuefront+1)%queuelength;
    element[queuefront]=0;
    queuesize--;
}

template<class T>
T myqueue<T>::front(){
    return element[(queuefront+1)%queuelength];
}

template<class T>
T myqueue<T>::back(){
    return element[queueback];
```

```
}

template<class T>
int myqueue<T>::size(){
    return queuesize;
}

template <class T>
class binarytree{
public:
    binarytree(){ root = NULL; treesize=0;}
    void init();
    void visit(binarytreenode<T>* x){
        cout<<x->element<<' ';
    }
    bool empty(){ return treesize==0;}
    void preorder(binarytreenode<T>* t);
    void inorder(binarytreenode<T>* t);
    void postorder(binarytreenode<T>* t);
    void levelorder(binarytreenode<T>* t);
    void size();
    int size(binarytreenode<T>* t);
    void height();
    int height(binarytreenode<T>* t);
    binarytreenode<T>* root;
private:
    int treesize;
};

template<class T>
void binarytree<T>::init(){
    int n;
    cin>>n;
    treesize = n;
    myqueue<binarytreenode<T>*> Q;
    root = new binarytreenode<T> (1);
    binarytreenode<T>* t = root;
    T l[n+1]={0};
    T r[n+1]={0};
    for(int i=1;i<=n;i++) cin>>l[i]>>r[i];
    for(int i=1;i<=n;i++){
        int num = t->element;
        if(l[num]!=-1){
            t->leftchild = new binarytreenode<T> (l[num]);
            Q.push(t->leftchild);
        }
```

```
        if(r[num]!=-1){
            t->rightchild = new binarytreenode<T> (r[num]);
            Q.push(t->rightchild);
        }
        if(!Q.empty()){
            t=Q.front();
            Q.pop();
        }
    }
}

template<class T>
void binarytree<T>::preorder(binarytreenode<T>* t){
    if(t!=NULL){
        visit(t);
        preorder(t->leftchild);
        preorder(t->rightchild);
    }
}

template<class T>
void binarytree<T>::inorder(binarytreenode<T>* t){
    mystack<binarytreenode<T>*> Q;
    while(t!=NULL || !Q.empty()){
        if(t!=NULL){
            Q.push(t);
            t=t->leftchild;
        }
        else{
            if(Q.empty()) return ;
            t=Q.top();
            visit(t);
            Q.pop();
            t=t->rightchild;
        }
    }
}

template<class T>
void binarytree<T>::postorder(binarytreenode<T>* t){
    mystack<binarytreenode<T>*> Q;
    binarytreenode<T>* pre=NULL;
    while(t!=NULL || !Q.empty()){
        if(t!=NULL){
            Q.push(t);
            t=t->leftchild;
```

```
        }
        else{
             t=Q.top();
            if(t->rightchild==NULL || t->rightchild==pre){//当前节点是一个根节点
                visit(t);
                Q.pop();
                pre=t;//修改访问过的节点
                t=NULL;
            }
            else//右子树还未被遍历
                t=t->rightchild;
        }
    }
}

template<class T>
void binarytree<T>::levelorder(binarytreenode<T>* t){
    myqueue<binarytreenode<T>*> Q;
    while(t!=NULL){
        visit(t);
        if(t->leftchild!=NULL) Q.push(t->leftchild);
        if(t->rightchild!=NULL) Q.push(t->rightchild);
        if(!Q.empty()){
            t=Q.front();
            Q.pop();
        }
        else{
            t=NULL;
        }
    }
}

template <class T>
void binarytree<T>::size(){
    binarytreenode<T>* t = root;
    myqueue<binarytreenode<T>*> Q;
    int s[100001]={0};
    while(t!=NULL){
        s[t->element] = size(t);
        if(t->leftchild!=NULL) Q.push(t->leftchild);
        if(t->rightchild!=NULL) Q.push(t->rightchild);
        if(!Q.empty()){
            t=Q.front();
            Q.pop();
        }
        else{
```

```
            t=NULL;
        }
    }
    for(int i=1;i<=treesize;i++) cout<<s[i]<<' ';
}

template<class T>
int binarytree<T>::size(binarytreenode<T>* t){
    if(t!=NULL){
        return 1+size(t->leftchild)+size(t->rightchild);
    }
    else{
        return 0;
    }
}

template <class T>
void binarytree<T>::height(){
    binarytreenode<T>* t = root;
    myqueue<binarytreenode<T>*> Q;
    int h[100001]={0};
    while(t!=NULL){
        h[t->element] = height(t);
        if(t->leftchild!=NULL) Q.push(t->leftchild);
        if(t->rightchild!=NULL) Q.push(t->rightchild);
        if(!Q.empty()){
            t=Q.front();
            Q.pop();
        }
        else{
            t=NULL;
        }
    }
    for(int i=1;i<=treesize;i++) cout<<h[i]<<' ';
}

template<class T>
int binarytree<T>::height(binarytreenode<T>* t){
    if(t==NULL){
        return 0;
    }
    int hl=height(t->leftchild);
    int hr=height(t->rightchild);
    if(hl>hr) return ++hl;
    else return ++hr;
}
```

```
int main(){
    binarytree<int> tree;
    tree.init();
    tree.preorder(tree.root);
    cout<<endl;
    tree.inorder(tree.root);
    cout<<endl;
    tree.postorder(tree.root);
    cout<<endl;
    tree.levelorder(tree.root);
    cout<<endl;
    tree.size();
    cout<<endl;
    tree.height();
}
```

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

（1）省略栈和队列的代码

```
#include<iostream>
using namespace std;
//链表节点的结构定义
template <class T>
struct chainNode
{
   T element;
   chainNode<T> *next;
   chainNode() {}
   chainNode(const T& element)
      {this->element = element;}
   chainNode(const T& element, chainNode<T>* next)
      {this->element = element;
       this->next = next;}
};
//链表二叉树
template<class T>
class binaryTree{
public:
    binaryTree(){root=NULL;}//构造函数
    binaryTree(binaryTreeNode<T>* r){root=r;}
    ~binaryTree(){};

    void preOrder(binaryTreeNode<T>*t);//前
    void inOrder();//中
```

```
    void postOrder();//后
    void levelOrder();//层次

    int nodeSize(binaryTreeNode<T>*t);//节点个数
    int height(binaryTreeNode<T>*t);//计算高度
    binaryTreeNode<T> *root;//指向根的指针
};

//前序遍历，根左右，递归
template<class T>
void binaryTree<T>::preOrder(binaryTreeNode<T>*t){
    if(t != NULL){
        cout<<t->element<<" ";//访问根
        preOrder(t->leftChild);//访问左子树
        preOrder(t->rightChild);//访问右子树
    }
}
//中序遍历，左根右，非递归
template<class T>
void binaryTree<T>::inOrder(){
    binaryTreeNode<T> *t = root;
    linkedStack<binaryTreeNode<T>*> instack;
    while (t != NULL || !instack.empty())
    {
        if(t!=NULL){//遍历左子树
            instack.push(t);
            t=t->leftChild;
        }
        else{
            if(instack.empty())
                return ;
            t=instack.top();//弹出并访问栈顶元素
            cout<<t->element<<" ";
            instack.pop();
            t=t->rightChild;//访问右子树
        }
    }
    cout << endl;
}
//后序遍历，左右根，非递归
template<class T>
void binaryTree<T>::postOrder() {
    linkedStack<binaryTreeNode<T>*> poststack;
    binaryTreeNode<T>* t=root;//当前访问节点
    binaryTreeNode<T>* pre=NULL;//上次访问节点
    while(t!=NULL || !poststack.empty()){
```

```
        if(t!=NULL){
            poststack.push(t);
            t=t->leftChild;//遍历左子树节点并入栈
        }
        else{
            t=poststack.top();
            if(t->rightChild==NULL || t->rightChild==pre){//当前节点是一个根节点
                cout<<t->element<<" ";
                poststack.pop();
                pre=t;//修改访问过的节点
                t=NULL;
            }
            else//右子树还未被遍历
                t=t->rightChild;
        }
    }
    cout<<endl;
}
//层次遍历
template <class T>
void binaryTree<T>::levelOrder(){
    linkedQueue<binaryTreeNode<T>*>q;
    binaryTreeNode<T>*t=root;
    while(t!=NULL){
        cout << t->element << " ";
        //将左右孩子放入队列
        if(t->leftChild)
            q.push(t->leftChild);
        if(t->rightChild)
            q.push(t->rightChild);
        if(q.empty()){
            cout<<endl;
            return;
        }
        t=q.front();//访问下一个节点
        q.pop();
    }
    cout<<endl;
    return ;
}

//节点数目，递归
template<class T>
int binaryTree<T>::nodeSize(binaryTreeNode<T>*t){
    int n=0;
    if(t!=NULL){
```

```
        n=nodeSize(t->leftChild)+nodeSize(t->rightChild)+1;//左+右+根
    }
    return n;
}
//计算高度
template<class T>
int binaryTree<T>::height(binaryTreeNode<T>*t){
    if(t==NULL)
        return 0;
    int lh=height(t->leftChild);//左子树高度
    int rh=height(t->rightChild);//右子树高度
    if(lh>rh)//高度为大的+根
        return ++lh;
    else
        return ++rh;
}

int main(){
    int n;
    cin>>n;
    binaryTreeNode<int>** tree = new binaryTreeNode<int>*[n+1];
    for (int i = 1; i <= n; i++)
        tree[i] = new binaryTreeNode<int>(i);

    for(int i=1;i<=n;i++){//构建树
        int x,y;
        cin>>x>>y;
        if(x!=-1)
            tree[i]->leftChild=tree[x];
        else
            tree[i]->leftChild=NULL;
        if(y!=-1)
            tree[i]->rightChild=tree[y];
        else
            tree[i]->rightChild=NULL;

    }
    binaryTree<int>res(tree[1]);
    int *a=new int[n+1];
    int *b=new int[n+1];
    res.preOrder(res.root);
    cout<<endl;
    res.inOrder();
    res.postOrder();
    res.levelOrder();
    for(int i=1;i<=n;i++){//记录节点个数和高度
```

```
        a[i]=res.nodeSize(tree[i]);
        b[i]=res.height(tree[i]);
    }
    for(int i=1;i<=n;i++)
        cout<<a[i]<<" ";
    cout<<endl;
    for(int i=1;i<=n;i++)
        cout<<b[i]<<" ";

    delete[] a;
    delete[] b;
    for (int i = 1; i <= n; i++)
        delete tree[i];
    delete[] tree;
    return 0;
}
```

(2)

```
#include <iostream>
using namespace std;

template <class T>
struct binarytreenode{
    T element;
    binarytreenode<T>* leftchild,
                     * rightchild;
    binarytreenode(){
        element=0;
        leftchild=rightchild=NULL;
    }
    binarytreenode(const T& thelement){
        element=thelement;
        leftchild=NULL;
        rightchild=NULL;
    }
    binarytreenode(const T& thelement, binarytreenode<T>* theleftchild,
binarytreenode<T>* therightchild){
        element=thelement;
        leftchild=theleftchild;
        rightchild=therightchild;
    }
};

template <class T>
class binarytree{
public:
```

```
    binarytree(){ root = NULL; treesize=0;}
    void visit(binarytreenode<T>* x){
        cout<<x->element<<' ';
    }
    int tsize(){ return treesize;}
    bool empty(){ return treesize==0;}
    void postorder(binarytreenode<T>* &t);
    binarytreenode<T>* root;
private:
    int treesize;
};

template<class T>
void binarytree<T>::postorder(binarytreenode<T>* &t){  //后序遍历
    if(t!=NULL){
        postorder(t->leftchild);
        postorder(t->rightchild);
        visit(t);
    }
}


int n;
int pre[100000]={0};
int in[100000]={0};
int root_i=1;

template<class T>
void createtree(int left,int right,binarytreenode<T>* &theroot){
//从前序遍历数组中找根,再从中序遍历中找到该根在的位置,往左往右找到它的左右子树,进行创建, 不断递归
    if(left>right) return ;
    for(int i=left;i<=right;i++){
        if(pre[root_i]==in[i]){
            if(theroot==NULL){
                theroot = new binarytreenode<T> (pre[root_i]);
                root_i++;
                createtree(left,i-1,theroot->leftchild);
                createtree(i+1,right,theroot->rightchild);
            }
        }
    }
}


int main(){
```

```
    cin>>n;
    for(int i=1;i<=n;i++) cin>>pre[i];
    for(int i=1;i<=n;i++) cin>>in[i];
    binarytree<int> tree;
    createtree(1,n,tree.root);
    tree.postorder(tree.root);
}
```