**实验6**

**6.1 实验目的**

掌握二叉树的动态链表存储结构及表示。

掌握二叉树的三种遍历算法（递归和非递归两类）。

运用二叉树三种遍历的方法求解有关问题。

**6.2 实验要求**

按照C++面向对象方法编写二叉树类；二叉树的测试数据可用多种方式进行输入，如键盘输入、静态写入、文件读入等。

设计二叉树的二叉链表存储结构，编写算法实现下列问题的求解。

<1>打印出二叉树的三种遍历序列。

<2>设计算法按中序次序输出二叉树中各结点的值及其所对应的层次数。

<3>求二叉树的高度。

<4>求二叉树的结点数。

<5>求二叉树的叶子结点数。

<6>求二叉树的度为2的结点数。

<7>键盘输入一个元素x，求其父节点、兄弟结点、子结点的值，不存在时给出相应提示信息。对兄弟结点和孩子结点，存在时要明确指出是左兄弟、左孩子、右兄弟或右孩子。

<8>键盘输入一个元素x，求其在树中的层次，不存在时给出相应提示信息。

<9>将按顺序方式存储在数组中的二叉树转换为二叉链表形式。（数组中要扩展为完全二叉树）。

<10>交换二叉树中每个结点的左右孩子指针的值。（即：左子树变为右子树，右子树变为左子树）。

（下面为选做实验，有兴趣的同学完成）

<11>复制一棵二叉树T到T1。

<12>输出二叉树从每个叶子结点到根结点的路径（经历的结点）。

<13>对二叉链表表示的二叉树，按从上到下，从左到右打印结点值，即按层次打印。（提示：需要使用队列）

<14>对二叉链表表示的二叉树，求2个结点最近的共同祖先。

实验测试数据基本要求：

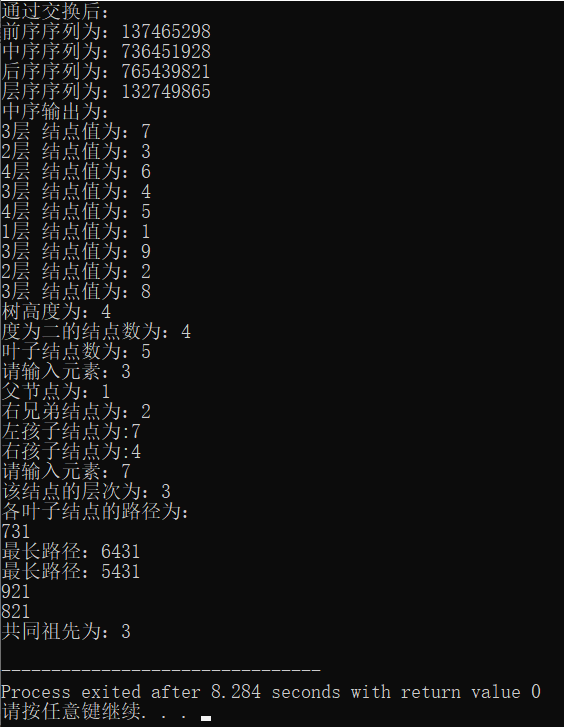
<15>求二叉树中一条最长的路径长度（边数），并输出路径上的个结点值。

实验测试数据基本要求：

**6.3 实验数据要求**

自我编写测试样例，要求每个功能函数的测试样例不少于两组

**6.4 运行结果截图及说明**



**6.5 附源代码**

**链式的：**

#include "OrderBinaryTree.h"

#include <iostream>

#include <queue>

//#include "BinaryTree.h"

using namespace std;

struct node{

node\* lchild;

node\* rchild;

int data;

};

class BinaryTree

{

public:

BinaryTree(int \*a);

BinaryTree(OrderBinaryTree obt);

BinaryTree(BinaryTree& bt);

~BinaryTree();

void create\_Tree(int \*a,node\* &ptr);

void create\_Tree2(int \*Tree,node \* &ptr,int i);

void create\_Tree3(node\* ptr1,node\* &ptr2);

void pre\_find(node\* ptr);

void mid\_find(node\* ptr);

void back\_find(node\* ptr);

void layer\_find();

void mid\_output(node\* ptr,int layer);

int set\_attribute(node \*ptr);

void get\_attribute();

void find\_node(node \*ptr,int x);

void get\_layer(node\* ptr,int x,int i,int &layer);

void swap\_child(node \*ptr);

void Print\_Path(node \*ptr,int i);

int find\_path(node \*ptr,int i,int \*Path,int x,int &k);

void closest\_path();

int max(int a,int b);

int leaf\_number;

int height;

int two\_node;

int Path[100]={0};

int Path2[100]={0};

int common\_path[100]={0};

node \*root;

int i;

int t;

protected:

};

BinaryTree::BinaryTree(int \*a)

{

root=NULL;

i=0;

height=0;

two\_node=0;

leaf\_number=0;

create\_Tree(a,root);

}

BinaryTree::BinaryTree(OrderBinaryTree obt){

root=NULL;

height=0;

two\_node=0;

leaf\_number=0;

create\_Tree2(obt.Tree,root,1);

}

BinaryTree::BinaryTree(BinaryTree &bt){

root=NULL;

height=0;

two\_node=0;

leaf\_number=0;

create\_Tree3(bt.root,this->root);

}

void BinaryTree::create\_Tree2(int \*Tree,node\* &ptr,int i){

if(Tree[i]!=0){

node\* new\_node=new node;

new\_node->data=Tree[i];

new\_node->lchild=NULL;

new\_node->rchild=NULL;

ptr=new\_node;

create\_Tree2(Tree,ptr->lchild,2\*i);

create\_Tree2(Tree,ptr->rchild,2\*i+1);

}

}

void BinaryTree::create\_Tree3(node\* ptr1,node \*&ptr2){

if(ptr1!=NULL){

node \*new\_node=new node;

new\_node->data=ptr1->data;

new\_node->lchild=NULL;

new\_node->rchild=NULL;

ptr2=new\_node;

create\_Tree3(ptr1->lchild,ptr2->lchild);

create\_Tree3(ptr1->rchild,ptr2->rchild);

}

}

BinaryTree::~BinaryTree()

{

}

void BinaryTree::pre\_find(node \*ptr){

if(ptr!=NULL){

cout<<ptr->data;

pre\_find(ptr->lchild);

pre\_find(ptr->rchild);

}

}

void BinaryTree::mid\_find(node \*ptr){

if(ptr!=NULL){

mid\_find(ptr->lchild);

cout<<ptr->data;

mid\_find(ptr->rchild);

}

}

void BinaryTree::back\_find(node \*ptr){

if(ptr!=NULL){

back\_find(ptr->lchild);

back\_find(ptr->rchild);

cout<<ptr->data;

}

}

void BinaryTree::layer\_find(){

queue<node\*> q;

q.push(root);

while(!q.empty()){

node \*ptr=q.front();

q.pop();

cout<<ptr->data;

if(ptr->lchild!=NULL){

q.push(ptr->lchild);

}

if(ptr->rchild!=NULL){

q.push(ptr->rchild);

}

}

}

void BinaryTree::mid\_output(node \*ptr,int layer){

if(ptr!=NULL){

mid\_output(ptr->lchild,layer+1);

cout<<layer<<"层 "<<"结点值为："<<ptr->data<<endl;

mid\_output(ptr->rchild,layer+1);

}

}

void BinaryTree::create\_Tree(int\* a,node\* &ptr){

if(a[i]!=0){

node\* new\_node=new node;

new\_node->data=a[i];

new\_node->lchild=NULL;

new\_node->rchild=NULL;

ptr=new\_node;

i++;

create\_Tree(a,ptr->lchild);

create\_Tree(a,ptr->rchild);

}

else{

i++;

}

}

int BinaryTree::max(int a,int b){

if(a>b){

return a;

}

else{

return b;

}

}

int BinaryTree::set\_attribute(node\* ptr){

if(ptr!=NULL){

if(ptr->lchild!=NULL&&ptr->rchild!=NULL){

two\_node++;

}

if(ptr->lchild==NULL&&ptr->rchild==NULL){

leaf\_number++;

}

return max(set\_attribute(ptr->lchild),set\_attribute(ptr->rchild))+1;

}

else{

return 0;

}

}

void BinaryTree::get\_attribute(){

height=set\_attribute(root);

cout<<"树高度为："<<height<<endl;

cout<<"度为二的结点数为："<<two\_node<<endl;

cout<<"叶子结点数为："<<leaf\_number<<endl;

}

void BinaryTree::find\_node(node \*ptr,int x){

if(ptr==root&&ptr->data==x){

cout<<"没有父结点！"<<endl;

cout<<"没有兄弟结点！"<<endl;

if(ptr->lchild!=NULL){

cout<<"左孩子结点为:"<<ptr->lchild->data<<endl;

}

else{

cout<<"没有左孩子结点"<<endl;

}

if(ptr->rchild!=NULL){

cout<<"右孩子结点为:"<<ptr->rchild->data<<endl;

}

else{

cout<<"没有右孩子结点"<<endl;

}

}

if(ptr->lchild!=NULL){

if(ptr->lchild->data==x){

cout<<"父节点为："<<ptr->data<<endl;

if(ptr->rchild!=NULL){

cout<<"右兄弟结点为："<<ptr->rchild->data<<endl;

}

else{

cout<<"没有兄弟结点!"<<endl;

}

if(ptr->lchild->lchild!=NULL){

cout<<"左孩子结点为:"<<ptr->lchild->lchild->data<<endl;

}

else{

cout<<"没有左孩子结点"<<endl;

}

if(ptr->lchild->rchild!=NULL){

cout<<"右孩子结点为:"<<ptr->lchild->rchild->data<<endl;

}

else{

cout<<"没有右孩子结点"<<endl;

}

}

else{

find\_node(ptr->lchild,x);

}

}

if(ptr->rchild!=NULL){

if(ptr->rchild->data==x){

cout<<"父节点为："<<ptr->data<<endl;

if(ptr->lchild!=NULL){

cout<<"左兄弟结点为："<<ptr->lchild->data<<endl;

}

else{

cout<<"没有兄弟结点!"<<endl;

}

if(ptr->rchild->lchild!=NULL){

cout<<"左孩子结点为:"<<ptr->lchild->lchild->data<<endl;

}

else{

cout<<"没有左孩子结点"<<endl;

}

if(ptr->rchild->rchild!=NULL){

cout<<"右孩子结点为:"<<ptr->rchild->rchild->data<<endl;

}

else{

cout<<"没有右孩子结点"<<endl;

}

}

else{

find\_node(ptr->rchild,x);

}

}

}

void BinaryTree::get\_layer(node \*ptr,int x,int i,int &layer){

if(ptr!=NULL){

i++;

if(ptr->data==x){

layer=i;

}

get\_layer(ptr->lchild,x,i,layer);

get\_layer(ptr->rchild,x,i,layer);

}

}

void BinaryTree::swap\_child(node \*ptr){

node\* lchild=ptr->lchild;

node\* rchild=ptr->rchild;

ptr->lchild=rchild;

ptr->rchild=lchild;

if(rchild!=NULL){

swap\_child(ptr->lchild);

}

if(lchild!=NULL){

swap\_child(ptr->rchild);

}

}

void BinaryTree::Print\_Path(node\* ptr,int i){

if(ptr!=NULL){

Path[i]=ptr->data;

Print\_Path(ptr->lchild,i+1);

Print\_Path(ptr->rchild,i+1);

if(ptr->lchild==NULL&&ptr->rchild==NULL){

if(i==height-1){

cout<<"最长路径：";

}

for(int j=i;j>=0;j--){

cout<<Path[j];

}

cout<<endl;

}

}

}

int BinaryTree::find\_path(node \*ptr,int i,int \*Path,int x,int &k){

if(ptr!=NULL){

Path[i]=ptr->data;

if(ptr->data==x){

k=i;

return 1;

}

if(find\_path(ptr->lchild,i+1,Path,x,k)){

return 1;

}

if(find\_path(ptr->rchild,i+1,Path,x,k)){

return 1;

}

}

return 0;

}

void BinaryTree::closest\_path(){

int i=0,j=0;

int \*Path1=new int[height];

int \*Path2=new int[height];

find\_path(root,0,Path1,7,i);

find\_path(root,0,Path2,5,j);

for(int a=i;a>=0;a--){

for(int b=j;b>=0;b--){

if(Path1[a]==Path2[b]){

cout<<"共同祖先为："<<Path1[a]<<endl;

return;

}

}

}

}

**顺序的**：

#include<iostream>

using namespace std;

class OrderBinaryTree

{

public:

OrderBinaryTree(int \*a);

void Tree\_create(int i,int \*a);

int Tree[100]={0};

int j;

protected:

};

OrderBinaryTree::OrderBinaryTree(int \*a)

{

j=0;

Tree\_create(1,a);

}

void OrderBinaryTree::Tree\_create(int i,int \*a){

if(a[j]!=0){

Tree[i]=a[j];

j++;

Tree\_create(2\*i,a);

Tree\_create(2\*i+1,a);

}

else{

j++;

}

}

**主函数：**

**#include <iostream>**

**#include "BinaryTree.h"**

**#include "OrderBinaryTree.h"**

**using namespace std;**

**int main(int argc, char\*\* argv) {**

**int a[100]={1,2,8,0,0,9,0,0,3,4,5,0,0,6,0,0,7};**

**OrderBinaryTree obt(a);**

**// OrderBinaryTree obt(a); 通过顺序存储**

**// BinaryTree bt(obt);**

**// BinaryTree bt(obt);**

**BinaryTree bt1(a);**

**BinaryTree bt(bt1); //复制构造函数**

**bt.swap\_child(bt.root);**

**cout<<"通过交换后："<<endl;**

**cout<<"前序序列为：";**

**bt.pre\_find(bt.root);**

**cout<<endl<<"中序序列为：";**

**bt.mid\_find(bt.root);**

**cout<<endl<<"后序序列为：";**

**bt.back\_find(bt.root);**

**cout<<endl<<"层序序列为：";**

**bt.layer\_find();**

**int x1,x2;**

**cout<<endl<<"中序输出为："<<endl;**

**bt.mid\_output(bt.root,1);**

**bt.get\_attribute();**

**cout<<"请输入元素：";**

**cin>>x1;**

**bt.find\_node(bt.root,x1);**

**int layer=0;**

**cout<<"请输入元素：";**

**cin>>x2;**

**bt.get\_layer(bt.root,x2,0,layer);**

**if(layer==0){**

**cout<<"结点不存在！"<<endl;**

**}**

**else{**

**cout<<"该结点的层次为："<<layer<<endl;**

**}**

**cout<<"各叶子结点的路径为："<<endl;**

**bt.Print\_Path(bt.root,0);**

**bt.closest\_path();**

**return 0;**

**}**

**6.6 调试过程中出现的bug总结**