中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/11/18	完成日期	2021/11/21

1. 实验题目

1）Play with Tree

2）Width of Binary Trees

2.实验目的

1）完成函数 void query(const Node *root, int &size, int &height)；

给出根节点 root，求该二叉树的顶点数 size 和树高 height

2）完成函数 int width(const treeNode * root);

给出根节点 root，求该二叉树的最大宽度 width

3.程序设计

1）Play with Tree

设计思路

求该二叉树的顶点数 size 和树高 height，可将问题转化为根节点+左子树+右子树的问题，如：

size（根节点） = 1 + size（左子树）+ size（右子树）

height(根节点) = 1 + max{ height(左子树) ,height(右子树)}

于是可以采用递归的方法分别构造求 size 和求 height 的函数：

```
int count0(const Node *root){//顶点数=1(当前顶点)+左子树顶点数+右子树顶点数
    if(root == NULL) return 0;
    return 1 + count0(root->lc)+count0(root->rc);
}
int count1(const Node *root){//树高 = 1+max(左子树高，右子树高)
    if(root == NULL) return 0;
    int l= count1(root->lc);
    int r = count1(root->rc) ;
    return 1+(l>r?l:r);
}
```

同时，考虑到二叉树是完全无向图，顶点数=边数+1，可将问题转化为：

顶点数 size = 总边数 + 1 ，

总边数 = 顶点出度和 = 左子树出度和 + 右子树出度和；

树高 height = 最大边长+1，

最大边长 = max{左子树最大边长，右子树最大}；

于是也可以采用递归的方法分别构造求总边数和求最大边长的函数：

```
int count0(const Node *root){//size = count0(root)+1;
//顶点和=出度和(边数)+1 ; 边数和 = 当前顶点出度 + 左子树出度+右子树出度
    if(root == NULL) return 0;
    int h0 = 0 ;
    if(root->lc!= NULL) h0++;
    if(root->rc!= NULL) h0++;
    return h0 + count0(root->lc)+count0(root->rc);
}

int count1(const Node *root){ //最大边长 = max{左子树最大边长，右子树最大}
    if(root == NULL || (root->rc==NULL&&root->lc ==NULL) ) return 0;//顶点无出度时
    int l = count1(root->lc);
    int r = count1(root->rc);
    int max = l>r ? l : r ;
    return 1+max;
}
```

代码

```
#include "play.h"
#include<iostream>
using namespace std;

int count0(const Node *root){//顶点数=1(当前顶点)+左子树顶点数+右子树顶点数
    if(root == NULL) return 0;
    return 1 + count0(root->lc)+count0(root->rc);
}
int count1(const Node *root){//树高 = 1+max(左子树高，右子树高)
    if(root == NULL) return 0;
    int l= count1(root->lc);
    int r = count1(root->rc) ;
    return 1+(l>r?l:r);
}
void query(const Node *root, int &size, int &height){
    size = count0(root);//顶点和
    height = count1(root) ;//树高
}
```

或

```
#include "play.h"
#include<iostream>
int count0(const Node *root){//size = count0(root)+1;
//顶点和=出度和(边数)+1 ; 边数和 = 当前顶点出度 + 左子树出度+右子树出度
    if(root == NULL) return 0;
    int h0 = 0 ;
    if(root->lc!= NULL) h0++;
    if(root->rc!= NULL) h0++;
    return h0 + count0(root->lc)+count0(root->rc);
}
int count1(const Node *root){ //最大边长 = max{左子树最大边长，右子树最大}
    if(root == NULL || (root->rc==NULL&&root->lc ==NULL) ) return 0;//顶点无出度时
    int l = count1(root->lc);
    int r = count1(root->rc);
    int max = l>r ? l : r ;
    return 1+max;
}
void query(const Node *root, int &size, int &height){
    size = count0(root)+1;  //顶点和=边数和+1
    height = count1(root)+1 ;  //树高  = 最大边长+1
}
```

2）Width of Binary Trees

设计思路

求树的最大宽度，基本思路是求出每层树的宽度，再求出宽度的最大值。

于是我设计函数 width_w，携带参数 k 和 max 还有数组 w[],其中 k 表示遍历到的树的第 k 层（0,1,…,height-1）,w[k]表示第 k 层的宽度（当这一层遍历完时），所以 w[]的容量应为树高 height，这里使用 vector，可以避免求树高：

if(w.size()== k) w.push_back(0); //使用 vector 可以不用求树高

每次遍历了第 k 层的 1 个节点，都令 w[k]+=1;再令 max 取 max 和 w[k]中较大者。再调用递归遍历该节点的下一层的两个儿子节点。

这样，当所有节点都遍历完时，便求得了树的最大宽度 max，第 k 层树宽 w[k]，树的最大高度 w.size().其中 max 便是我们的目标结果。

代码

```
#include"width.h"
#include<vector>
void width_w(const treeNode* root ,vector<int>& w ,int k,int& max){
    if(root == NULL ) return ;
    if(w.size()== k) w.push_back(0);//使用 vector 可以不用求树高
    w[k]+=1;
    max = max > w[k] ? max : w[k] ;
    width_w(root->left,w,k+1,max);
    width_w(root->right,w,k+1,max);
}
int width(const treeNode*root){
     vector<int>w;
     int max=0;
     width_w(root,w,0,max);
     return max;
}
```

4.程序运行与测试

本次实验不用写程序运行与测试。

5.实验总结与心得

本次实验求树的顶点数，树高，最大树宽，都是采用递归的方法，可见递归在解决有关二叉树的问题中有着相当重要的地位。

在求解 2）Width of Binary Trees，开始时我写成了求叶子数的函数：

```
#include"width.h"
#include<iostream>
int width(const treeNode* root){
    if(!root) return 0;
    if(root->left==NULL&& root->right==NULL) return 1;
    return width(root->left)+width(root->right);
}
```

发现竟然可以通过测试样例，于是我逐个检查测试样例，竟然每个测试样例的最大树宽都等于叶子数，由此可见测试样例还是不够充分，有时候能过评测系统不一定代表你的代码就是对的，可能充满各种凑巧。

然后再写后来的代码，开始时不想再加个求树高的函数，由于测试样例少，我就把数组 w[]容量设置为 20，正常应该为树高。所以这仍是不完美的代码，于是我想用动态内存分配空间，发现不怎么会用，这时想到可以用 vector，当访问到第 k 层时，如果 k == w.size()（说明是新的一层），就可以 w.push_back(0), 这就解决了没求树高的问题。

附录、提交文件清单