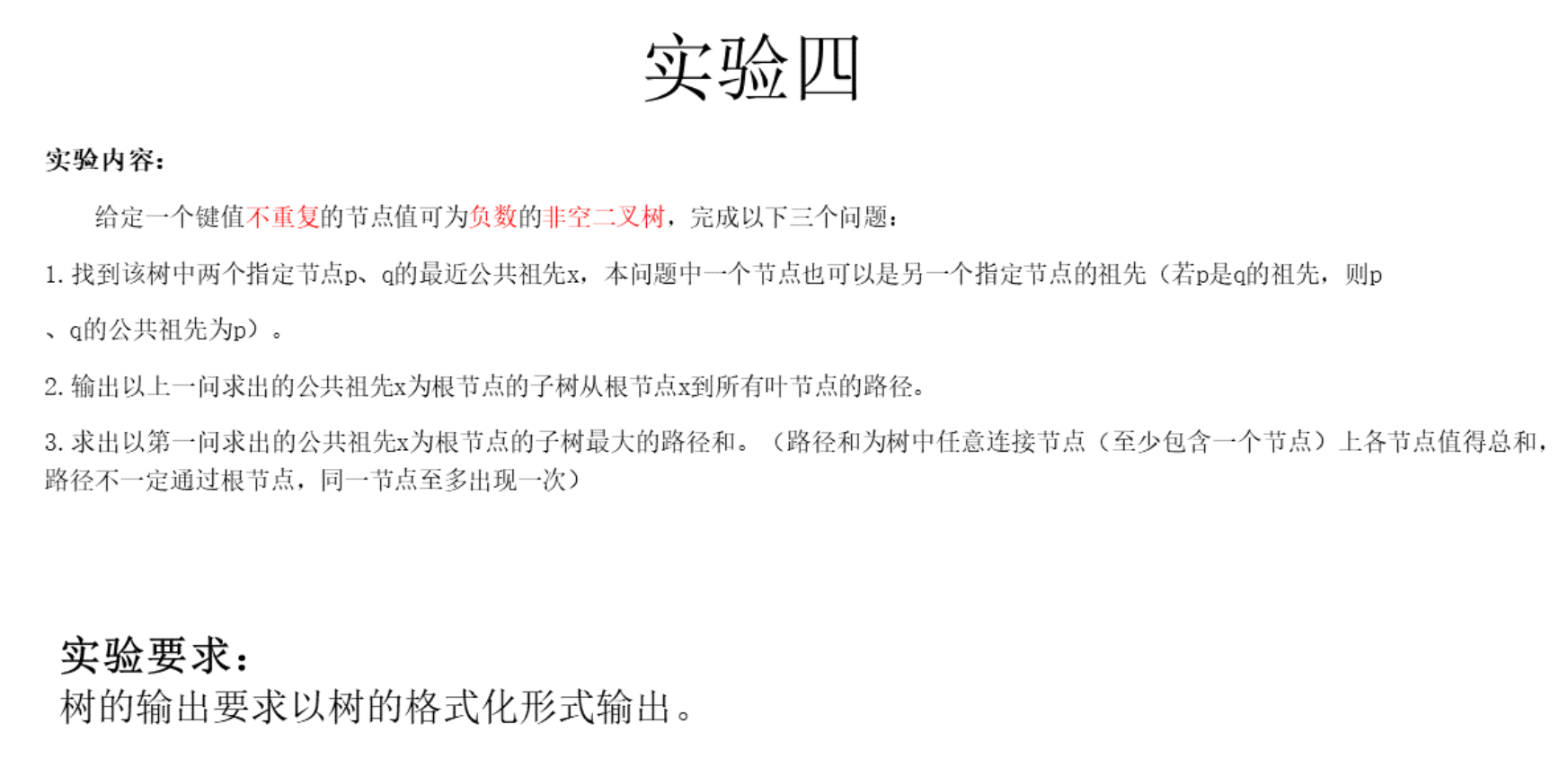
数据结构实验四

班级：20信安法 学号：2013484 姓名：张世伟

1. 实验内容



1. 设计思路
2. 输入

层序输入，使用队列存放待插入的根节点，使用bool变量t来判断，插入的是左子树还是右子树

1. 问题1
   1. findPosition()函数

根据输入的p和q找到二叉树中对应的节点P和Q

* 1. findPublicFather()函数

判断节点P和Q的高低，高节点每向上寻找一个父节点，低节点向上寻找到与高节点同一层，直至找到公共祖先

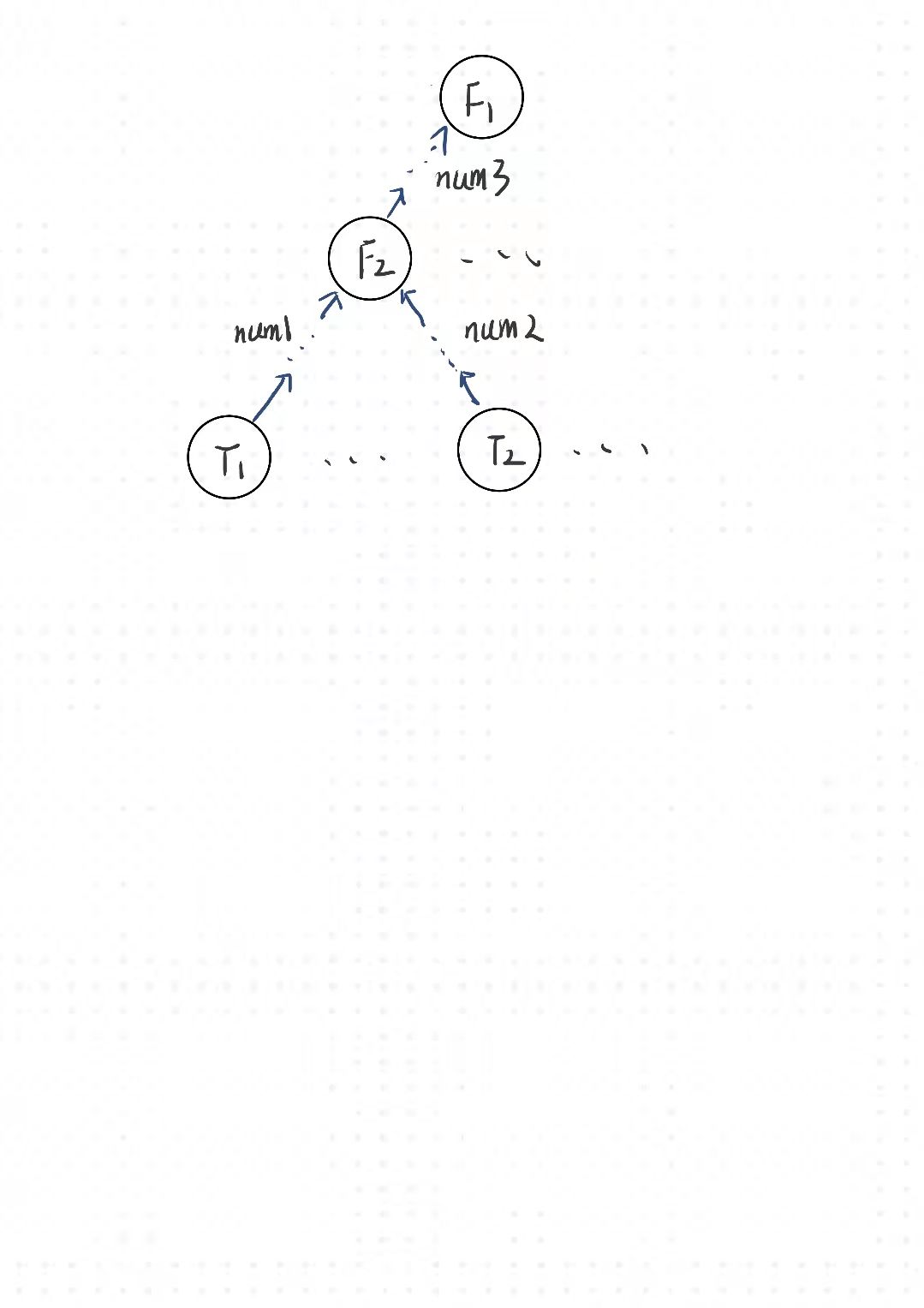
1. 问题2
   1. findChildren()函数

根据先序遍历的思路，寻找传入节点的所有叶节点，并将所有叶节点push进入所传队列Queue中

* 1. printPaths()函数

根据findChildren()函数得到的队列Queue，遍历所有叶节点，向上寻址并输出

1. 问题3



根据findChildren()函数得到的存有公共祖先publicFather的所有叶节点，将所有叶节点配对得到叶节点对（T1，T2），使用 findPublicFather()函数，寻找T1和T2的第一个公共祖先publicFather2（F2）。

分别从T1和T2向上遍历，直到publicFather2，计算路径和分别计为num1和num2，再从F2向上遍历直到publicFather，计算路径和，记为num3。将num1、num2、num3求和，即可得到符合问题3要求的（T1，T2）的路径和m。

Queue每次pop都复制一次得到队列tempQueue，Queue与tempQueue错位pop即可实现叶节点对的遍历。

找到最大的路径和

1. 核心代码
2. 输入

void Insert(node\* T)

{

queue<node\*>Q;

Q.push(T);

bool t = true;

string n;

cin >> n;

while (n != "]")

{

if (n == "null")//输入为空

{

if (t)//左子树为空

t = false;

else//右子树为空

{

Q.pop();

t = true;

}

}

else if (n != "]")//输入不为null

{

node\* temp = new node(n);

if (t)//插入的位置是左子树

{

Q.front()->lcild = temp;

temp->father = Q.front();

temp->height = Q.front()->height + 1;

Q.push(temp);

t = false;

}

else//插入的位置为右子树

{

Q.front()->rcild = temp;

temp->father = Q.front();

temp->height = Q.front()->height + 1;

Q.push(temp);

Q.pop();//去下一个节点

t = true;//插入位置换回左子树

}

}

cin >> n;

}

}

1. 问题1
2. findPosition()

node\* findPosition(node\* T, string x)//寻找data = x 的节点的地址

{

node\* temp = new node();

if (!T)

temp = nullptr;

else

{

if (T->data == x)

{

temp = T;

}

else

{

temp = findPosition(T->lcild, x);

if (temp == nullptr)

{

temp = findPosition(T->rcild, x);

}

}

}

return temp;

}

1. findPublicFather()

node\* findPublicFather(node\* P, node\* Q)//寻找公共祖先

{

node\* high, \* low;

if (P->height <= Q->height)

high = P, low = Q;

else

high = Q, low = P;

while (high)

{

//cout << P->height << endl;

node\* Temp = low;

while (Temp->height >= high->height)

{

//cout <<"Q->height:"<< Temp->height << endl;

if (Temp->data == high->data)

{

return Temp;

}

else

Temp = Temp->father;

}

high = high->father;

}

}

1. 问题2
2. findChildren()

void findChildren(node\* T, queue<node\*>& Queue)//将T父节点的所有叶节点存在一个队列里面

{

if (T)

{

if (!T->lcild && !T->rcild)

{

Queue.push(T);

}

else

{

if (T->lcild)

{

findChildren(T->lcild, Queue);

}

if (T->rcild)

{

findChildren(T->rcild, Queue);

}

}

}

}

1. printPaths()

void printPaths(node\* T, node\* L)//输出各个路径

{

stack<node\*>Stack;

while (L != T)

{

Stack.push(L);

L = L->father;

}

Stack.push(T);

while (!Stack.empty())

{

visit(Stack.top());

Stack.pop();

}

cout << endl;

}

1. 问题3
2. pathLength()

int pathLength(node\* T1, node\* T2, node\* publicFather1)

{

node\* publicFather2 = findPublicFather(T1, T2);

int num1 = 0, num2 = 0, num3 = 0;

//cout << "father = " << publicFather2->data << endl;

while (T1 != publicFather2)

{

num1 += stoi(T1->data);

T1 = T1->father;

//cout <<"num1 = "<< num1 << endl;

}

while (T2 != publicFather2)

{

num2 += stoi(T2->data);

T2 = T2->father;

//cout << "num2 = " << num2 << endl;

}

while (publicFather2 != publicFather1)

{

num3 += stoi(publicFather2->data);

publicFather2 = publicFather2->father;

//cout << "num3 = " << num3 << endl;

}

num3 += stoi(publicFather1->data);

return num1 + num2 + num3;

}

1. findMax()

void findMax(queue<node\*>Queue, node\* publicFather)

{

int max = 0;

while (Queue.size() > 1)

{

queue<node\*>tempQueue = copyQueue(Queue);

while (!tempQueue.empty())

{

int m = pathLength(Queue.front(), tempQueue.front(), publicFather);

if (m > max)

max = m;

tempQueue.pop();

}

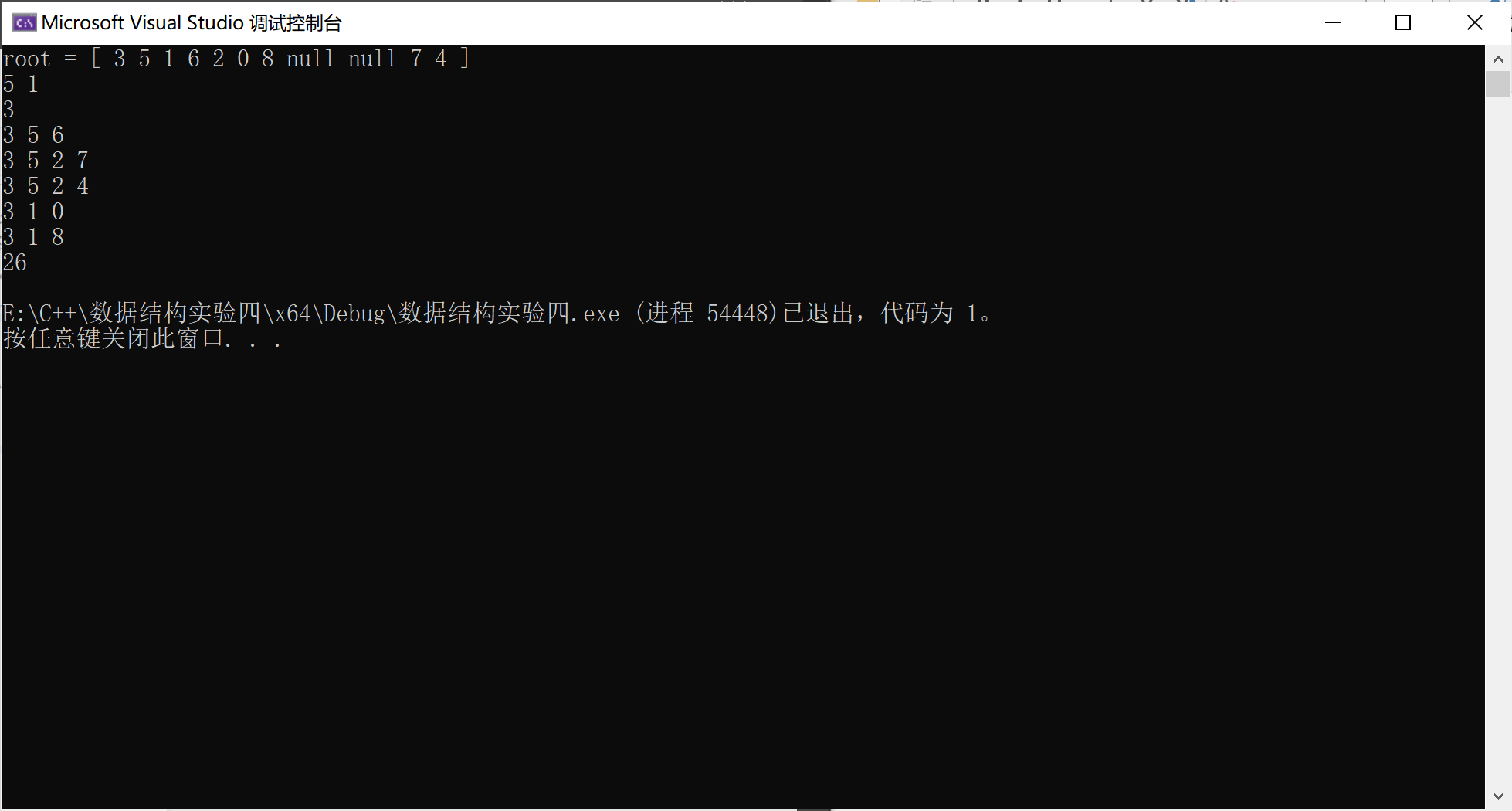
Queue.pop();

}

cout << max << endl;

}

1. 实验结果





1. 复杂度分析
2. 输入
3. 问题1
4. findPosition()
5. findPublicFather()
6. 问题
7. findChildren()
8. printPaths()
9. 问题3
10. pathLength()
11. findMax()