数据结构

第三次上机作业实验文档

学号2017211123

班级 2017211301

序号 17

姓名 褚逸豪

## **问题描述**

设二叉树结点值为大写字母，输入二叉树的前序遍历和中序遍历序列，生成此二叉树，输出该二叉树的后序遍历和按层次遍历序列。输入某结点值，在二叉树中查找该结点，若该结点存在，则输出从根到该结点的路径，否则给出不存在信息。

## 算法思路

递归地处理前序遍历和中序遍历序列，根据前序序列的第一个元素可知根节点编号，进而可以在中序遍历序列中确定左右子树的大小，进而将剩余的前序遍历序列和中序遍历序列分成两部分，向下递归，同时建树，即可得出相应的二叉树

## 算法描述

1. 函数parse（参数1 前序序列， 参数2 中序序列）： 节点
   1. 如果任意一个序列为空，则返回空节点
   2. 新建节点t，值为前序序列的第一个元素
   3. 找到前序序列第一个元素在中序序列中的位置，这个位置将中序序列分割为两个序列，左子树中序序列，右子树中序序列
   4. 根据左子树中序序列和右子树中序序列我们可以确定左子树和右子树的大小，进而可以将前序序列除去第一个元素的剩余部分分割为左子树前序序列和右子树前序序列
   5. 令节点t的左儿子为parse（左子树前序序列，左子树中序序列）的结果
   6. 令节点t的右儿子为parse（右子树前序序列，右子树中序序列）的结果
   7. 返回节点t
2. 后序遍历与按层次遍历的描述（略）

## 源程序及驱动程序

编译命令：g++ <文件名.cpp>

#include <cstdio>

#include <cstring>

#include <iostream>

#include <string>

#include <cassert>

#include <queue>

using namespace std;

namespace bintree\_questions {

const int MXN = 1007;

struct node { // 节点类型

node \*c[2], \*p;

int id;

} buffer[MXN], \*cur, \*root;

queue<node\*> q; // 层次遍历（bfs）所用的队列

void init() { // 初始化

memset(buffer, 0, sizeof(buffer));

cur = buffer;

root = 0;

}

node \*newnode(int id, node \*p) { // 结点分配函数

cur->id = id;

cur->p = p;

cur->c[0] = cur->c[1] = 0;

return cur++;

}

node \*parsesequence(node \*p, int seqa[], int seqb[], int la, int ra, int lb, int rb) { // 解析序列并建立树，返回值为被解析的序列所对应的树的根节点

if (la > ra || lb > rb) return 0;

node \*tmp = newnode(seqa[la], p);

int pp;

for (pp = lb; seqb[pp] != seqa[la] && pp <= rb; ++pp);

assert(pp <= rb);

tmp->c[0] = parsesequence(tmp, seqa, seqb, la + 1, la + pp - lb, lb, pp - 1);

tmp->c[1] = parsesequence(tmp, seqa, seqb, la + pp - lb + 1, ra, pp + 1, rb);

return tmp;

}

void analyze(int n, int a[], int b[]) { // 套了个壳

init();

root = parsesequence(0, a, b, 0, n - 1, 0, n - 1);

}

void houxu(node \*x) { // 后序遍历

if (!x) return;

houxu(x->c[0]);

houxu(x->c[1]);

printf("%d ", x->id);

}

void cengci(node \*x) { // 层次遍历

q.push(x);

while (!q.empty()) {

printf("%d ", q.front()->id);

if (q.front()->c[0]) q.push(q.front()->c[0]);

if (q.front()->c[1]) q.push(q.front()->c[1]);

q.pop();

}

}

int seq[MXN], mxdep;

bool dfss(node \*x, int id, int dep) { // dfs待搜索的点

if (x->id == id) {

mxdep = dep;

seq[dep] = id;

return true;

}

if (x->c[0] && dfss(x->c[0], id, dep + 1) || x->c[1] && dfss(x->c[1], id, dep + 1)) {

seq[dep] = x->id;

return true;

} else {

return false;

}

}

void search(int id) { // 套了个壳

if (dfss(root, id, 0)) {

printf("Path to %d found: ", id);

for (int i = 0; i <= mxdep; ++i) {

printf("%d ", seq[i]);

}

} else {

printf("No path to %d is found", id);

}

}

};

int n;

int a[1007], b[1007];

int main() {

scanf("%d", &n);

for (int i = 0; i < n; ++i)

scanf("%d", a + i);

for (int i = 0; i < n; ++i)

scanf("%d", b + i);

bintree\_questions::analyze(n, a, b);

puts("Hou Xu");

bintree\_questions::houxu(bintree\_questions::root);

putchar('\n');

puts("Ceng Ci Bian Li");

bintree\_questions::cengci(bintree\_questions::root);

putchar('\n');

int x;

while (~scanf("%d", &x)) {

bintree\_questions::search(x);

putchar('\n');

}

}

## 测试数据

### Input

6

1 2 4 3 5 6

4 2 1 5 3 6

1 2 3 4 5 6 7

### Output

Hou Xu

4 2 5 6 3 1

Ceng Ci Bian Li

1 2 3 4 5 6

Path to 1 found: 1

Path to 2 found: 1 2

Path to 3 found: 1 3

Path to 4 found: 1 2 4

Path to 5 found: 1 3 5

Path to 6 found: 1 3 6

No path to 7 is found

## Online Test

<https://ideone.com/4nufH6>

## 结果分析

实际输出与预计输出相符。从遍历序列建树的复杂度为线性，每次查找元素的复杂度也为线性。

## 结论

给定二叉树的先序遍历和中序遍历，可以唯一的确定这棵二叉树

## 心得体会

递归的分解问题将使问题简化