HW3 树和二叉树 实验报告

姓名：徐恒 学号：2050035 日期：2022 年 5 月 26 日

1.涉及数据结构和相关背景

实验主要涉及到对树和二叉树的基本操作。树形结构是一类非常重要的非线性数据结构，其中以树和二叉树最为常用，直观来看，树是以分支关系定义的层次结构。二叉树的特点是每个节点最多只有两棵子树，且有左右之分，次序不能颠倒。

2. 实验内容

2.1 二叉树的创建与遍历

2.1.1 问题描述

本题练习二叉树的基本操作，包括先序建立二叉树，先序遍历、中序遍历、后序遍历、层次遍历二叉树，输出二叉树的树型图。

2.1.2 基本要求

输入一行先序序列，内部结点用一个字符表示，空结点用#表示。分别输出先序、中序、后序、层次遍历序列以及二叉树的树型图；其中树形图要求逆时针旋转 90 度，且每一层之间输出 5 个空格。

2.1.3 数据结构设计

通过输入的先序序列构建二叉树，并依次调用对应函数对其作先序、中序、后序、层次遍历，最终打印旋转九十度的树形图。

2.1.4 功能说明（函数、类）

```
typedef struct BiTNode{
    char data;
    struct BiTNode *lchild,*rchild;
}BiTNode, *BiTree;      //定义结构类型 BiTNode，作为树的结点
typedef struct{
    BiTree *base;
    int front,rear;}SqQueue;   //定义结构类型 SqQueue，作为队列
int CreateBiTree(BiTree &T)     //  创建树 T
void PreOrderTraverse(BiTree T)     //先序遍历树 T 并输出
void InOrderTraverse(BiTree T)      //中序遍历树 T 并输出
void PostOrderTraverse(BiTree T)    //后序遍历树 T 并输出
```

void LevelOrderTraverse(BiTree T) //层次遍历树 T 并输出
void PrintBiTree(BiTree T,int n)　//打印 T 的树形图

2.1.5 调试分析（遇到的问题和解决方法）

2.1.6 总结和体会

熟悉二叉树的创建、遍历、打印等操作。

2.2 二叉树的非递归遍历

2.2.1 问题描述

如果已知中序遍历的栈的操作序列，就可唯一地确定一棵二叉树。请编程输出该二叉树的后序遍历序列。

2.2.2 基本要求

输入时第一行一个整数 n，表示二叉树的结点个数。接下来 2n 行，每行描述一个栈操作，格式为：push X (X 用一个字符表示)表示将结点 X 压入栈中，pop 表示从栈中弹出一个结点。

输出时为一行后序遍历序列。

2.2.3 数据结构设计

先根据给定树的中序遍历的入栈出栈操作，建立二叉树 T，再对其后序遍历输出。注意到中序遍历入栈时，若上一次执行操作仍为 push，则此次入栈结点 p 即为上次入栈元素的左子树；若上一次执行操作为 pop，则此次入栈结点 p 即为上次入栈元素的右子树；特别地，第一次入栈结点为根节点，记为 T。故需建立一个整型指标 flag，用以判断上次操作；以及结点 q，用以记录上次操作的结点。依次，即可建立符合题干的二叉树。

2.2.4 功能说明（函数、类）

```
typedef struct BiTNode{
    char data;
    struct BiTNode *lchild,*rchild;
}BiTNode, *BiTree;  //定义结构类型 BiTNode，作为树的结点
typedef struct{
    BiTree *base;
    BiTree *top;
    int stacksize;
}SqStack;  定义结构类型 SqStack，作为栈
int InitStack(SqStack &S)    //对栈 S 进行初始化
int Push(SqStack &S,BiTree e) //将元素 e 入栈
int Pop(SqStack &S,BiTree &e) //对栈 S 作出栈处理，并将出栈元素返回值给 e
void PostOrderTraverse(BiTree T)  //输出树 T 的后序遍历序列
```

2.2.5 调试分析（遇到的问题和解决方法）

For 循环时结束指标定义出错

2.2.6 总结和体会

非递归实现后序遍历较为复杂，不如使用递归实现。

2.3 二叉树的同构

2.3.1 问题描述

给定两棵树 T1 和 T2。如果 T1 可以通过若干次左右孩子互换变成 T2，则我们称两棵树是“同构”的。

现给定两棵树，请你判断它们是否是同构的。并计算每棵树的深度。

2.3.2 基本要求

输入一个非负整数 N1，表示第 1 棵树的结点数；随后输入 N 行，依次对应二叉树的 N 个结点（假设结点从 0 到 N-1 编号），每行有三项，分别是 1 个英文大写字母、其左孩子结点的编号、右孩子结点的编号。如果孩子结点为空，则在相应位置上给出“-”。给出的数据间用一个空格分隔。

接着输入一行是一个非负整数 N2，表示第 2 棵树的结点数；随后输入 N 行同上描述一样，依次对应二叉树的 N 个结点。

输出时，如果两棵树是同构的，输出“Yes”，否则输出“No”。再输出两行分别是两棵树的深度。

2.3.3 数据结构设计

1.建立两颗二叉树 2.判断这两棵树是否同构

如果两个二叉树为空，则是同构。若干其中一个是空树，一个不是空树，则非同构树根的值不一样。非同构左孩子为空，则判断右孩子是否是同构，左孩子非空，且左孩子相同，则判断两个树的左子树与左子树是否同构，右子树与右子树是否同构，否则判断 T1 左子树和 T2 右子树是否同构，T1 右子树和 T2 左子树是否同构。

2.3.4 功能说明（函数、类）

```
struct Tri{
    char data;
    int left,right;
}T1[10101],T2[10101];   //定义三元组 Tri，存放树的结点的数据与左右孩子结点位置，T1，T2 两个数组分别存放两棵树的数据
int judge(int m,int n)      //判断以 T1[m]为根节点的树与以 T2[n]为根节点的树是否同构
int CreateTree(Tri *T)     //构建存放树 T 的三元组
int coutlevel1(int m)      //对以 T1[m]为根节点的树遍历求深度
int coutlevel2(int n)      //对以 T2[n]为根节点的树遍历求深度
```

2.3.5 调试分析（遇到的问题和解决方法）

调试中并未出现错误。

2.3.6 总结和体会

由于输入数据并非一般的树的构建形式，本题的难点在于如何利用给出的数据进行同

构的判断。

2.4 求树的后序

2.4.1 问题描述

给出二叉树的前序遍历和中序遍历，求树的后序遍历。

2.4.2 基本要求

输入若干行，每一行有两个字符串，中间用空格隔开同行的两个字符串从左到右分别表示树的前序遍历和中序遍历，由单个字符组成，每个字符表示一个节点

每一行输入对应一行输出。

若给出的前序遍历和中序遍历对应存在一棵二叉树，则输出其后序遍历，否则输出Error。

2.4.3 数据结构设计

对于给出的先序序列和中序序列，显然先序序列首元素为根，找到其在中序序列中的位置，左边为左子树，右边为右子树，然后依次对左右子树进行递归后序遍历，最终输出根节点。

2.4.4 功能说明（函数、类）

int flag=1;　//定义全局变量 flag,作为判断是否终止递归函数 Traverse 的指标
int Traverse(char *p,char *q,int length)　//p、q 分别存放树的先序遍历序列和中序遍历序列，length 为序列长度，该函数会给出其后序遍历序列

2.4.5 调试分析（遇到的问题和解决方法）

2.4.6 总结和体会