# 四川师范大学 实验报告

学期： 2024 至 2025 第一学期 实验成绩：

课程名字：《程序设计基础——数据结构(C语言版)》 专业： 信息与计算科学

班级： 2023 级 9 班 实验编号： 06

实验项目： 实验六 指导老师： 冯山

姓名： 刘智恒 学号： 2023060522

**一、实验题目**

二叉树的基本算法

**二、实验目的及要求**

1.掌握树和二叉树基本概念，以及其顺序与链式存储结构特点。

2.熟练掌握二叉树的创建、遍历等基础算法。

3.掌握二叉树的线索化原理，以及基于线索的二叉树遍历算法。

**三、实验内容：(类C算法的程序实现，任选其二)**

1.设计并实现二叉树的基本算法(如创建、输出和结点插入等)。(必做)

2.设计并实现二叉树的先序、后序、中序遍历算法。

3.设计并实现二叉树的深度计树算法。

4.设计并实现线索二叉树建立和遍历算法。

**四、实验准备**

1.计算机设备;

2.程序调试环境的准备，本实验采用**Microsoft Visual Studio**环境;

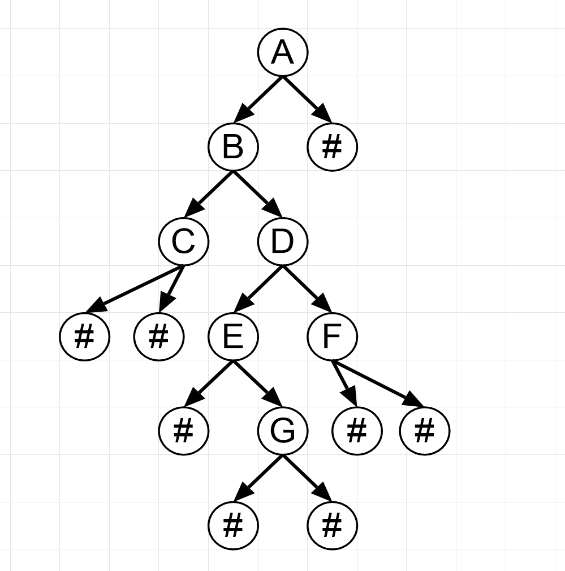
3.实验内容的算法分析与代码设计与分析准备；

4.实验源程序**Exp\_6(1)**准备。

**注：本实验中将实验内容1至3整合为Exp\_6(1)的内容，故以下的实验测试用例（1）（2）用于Exp\_6(1)的测试**

5.实验测试用例**（注‘#’表示该节点为空）**

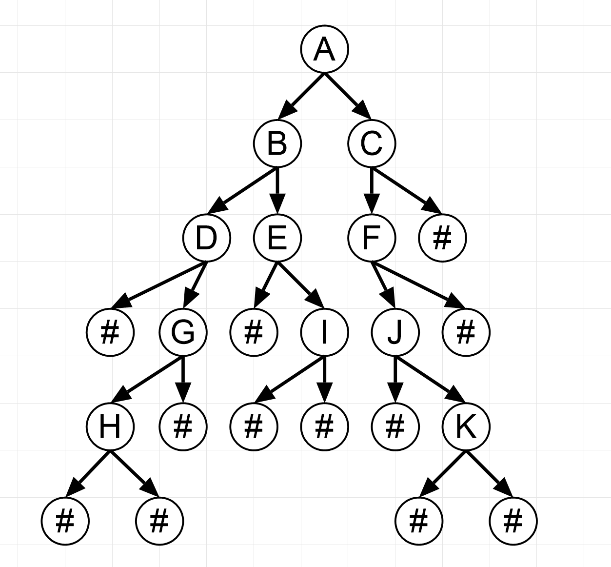
**（1）二叉树示例如下：**

****

**图4.1**

**（先序）输入序列：ABC##DE#G##F###；**

**（2）二叉树示例如下：**

****

**图4.2**

**（先序）输入序列：ABD#GH###E#I##CFJ#K####**

**五、实验内容**

**（一）问题分析**

（1）本实验中的实验内容1至3均与**递归**相关，即二叉树的结构定义是一个**递归定义**，其创建、先序遍历、中序遍历、后序遍历以及深度计算需要用到**递归算法**。二叉树的特点是**每个结点至多只有两棵子树（即二叉树中不存在度大于2的结点），并且二叉树的子树有左右之分，其次序不能颠倒**。在**Exp\_6(1)**中，针对二叉树的各项基本操作采用**链式**存储结构，即采用二叉树的**二叉链表**进行存储表示：

**typedef struct BiTNode{**

**TElemType data;**

**Struct BiTNode \*lchild, \*rchild;**

**}BiTNode, \*BiTree;**

（2）各项操作基本分析：

**[1]二叉树的创建**：采用**先序次序依次输入**二叉树中结点的值（即一个字符），以**‘#’**字符表示**当前结点为空（NULL）**，以此构造二叉链表表示的二叉树T。、

**[2]先序遍历二叉树：**按照**【根】->【左子树】->【右子树】**的顺序遍历二叉树T，对每个结点**调用函数Visit（访问）一次且仅一次**。

**[3]中序遍历二叉树：**按照**【左子树】->【根】->【右子树】**的顺序遍历二叉树T，对每个结点**调用函数Visit（访问）一次且仅一次**。

**[4]后序遍历二叉树：**按照**【左子树】->【右子树】->【根】**的顺序遍历二叉树T，对每个结点**调用函数Visit（访问）一次且仅一次**。

**[5]二叉树的深度计算：**同样采用**递归**的过程，先遍历整个左子树的深度，再遍历整个右子树的深度，最后进行大小比较得出二叉树的最终深度。

**（二）算法描述**

（1）**Status CreateBiTree(BiTree \*T)**;

//本算法用于创建二叉树：按先序次序输入二叉树中结点的值（即一个字符），以‘#’字符表示当前结点值为空。

（2）**Status Visit(TElemType e)**;

//本算法用于访问二叉树中的一个结点值：**将该函数（指针）作为其他函数参数**进行结点访问操作，即直接输出当前节点的值。

（3）**Status PreOrderTraverse(BiTNode \*T, Status(\*Visit)(TElemType e))**;

//本算法用于先序遍历二叉树：对二叉树T的每个结点调用函数Visit一次且仅一次，Visit是对结点操作的函数。

（4）**Status InOrderTraverse(BiTNode \*T, Status(\*Visit)(TElemType e))**;

//本算法用于中序遍历二叉树：对二叉树T的每个结点调用函数Visit一次且仅一次，Visit是对结点操作的函数。

（5）**Status PostOrderTraverse(BiTNode \*T, Status(\*Visit)(TElemType e))**;

//本算法用于后序遍历二叉树：对二叉树T的每个结点调用函数Visit一次且仅一次，Visit是对结点操作的函数。

1. **int CalculateDepth(BiTNode \*T)**;

//本算法用于计算二叉树的深度：如果节点T为NULL，则返回深度0；否则递归计算左子树的深度，再递归计算右子树的深度，最终返回左右子树深度的较大值加 1（当前节点）。

1. **void FreeBiTree(BiTree T)**;

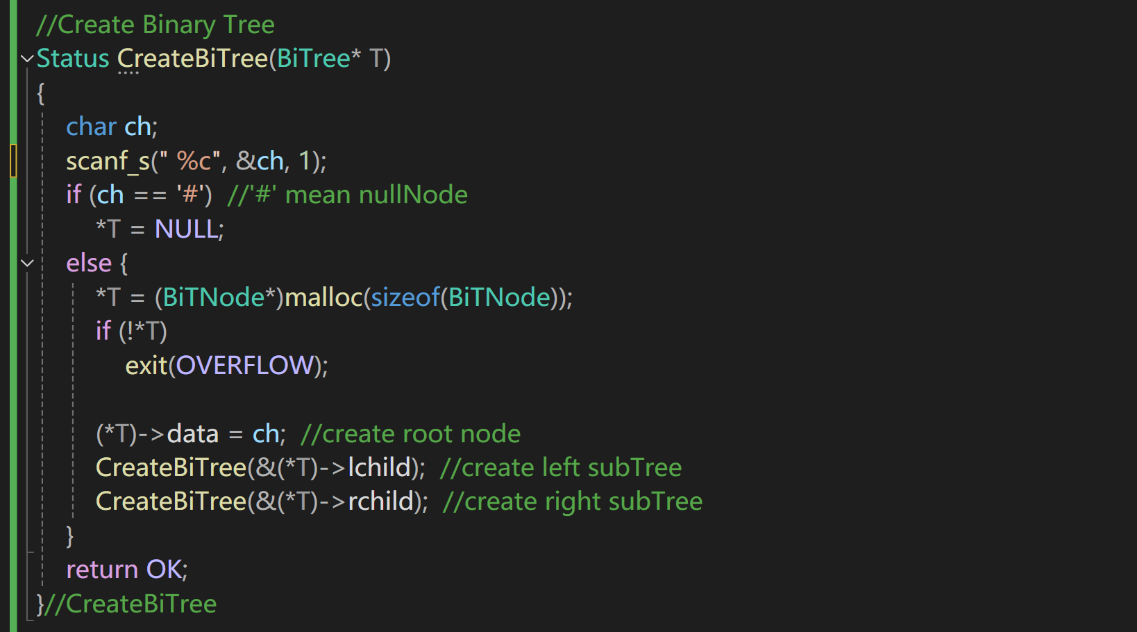
**//**本算法用于释放二叉树所占用的存储空间：如果节点T不为NULL，则递归释放左子树的内存，再递归释放右子树的内存，最终释放当前节点的内存。

**（三）程序代码**

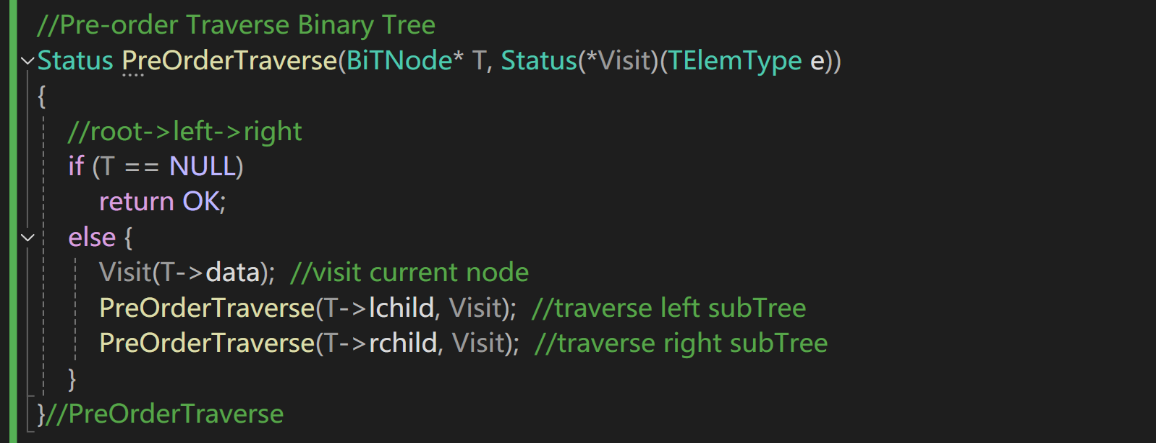
根据以上对**实验内容**的算法描述，求解的程序代码如下：

****

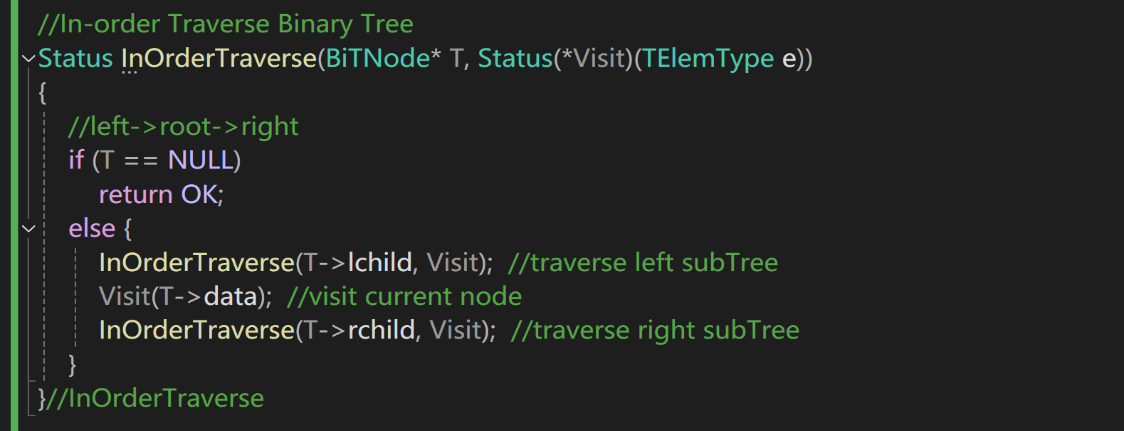
**注：此处只展示主要算法的程序代码图。**

****

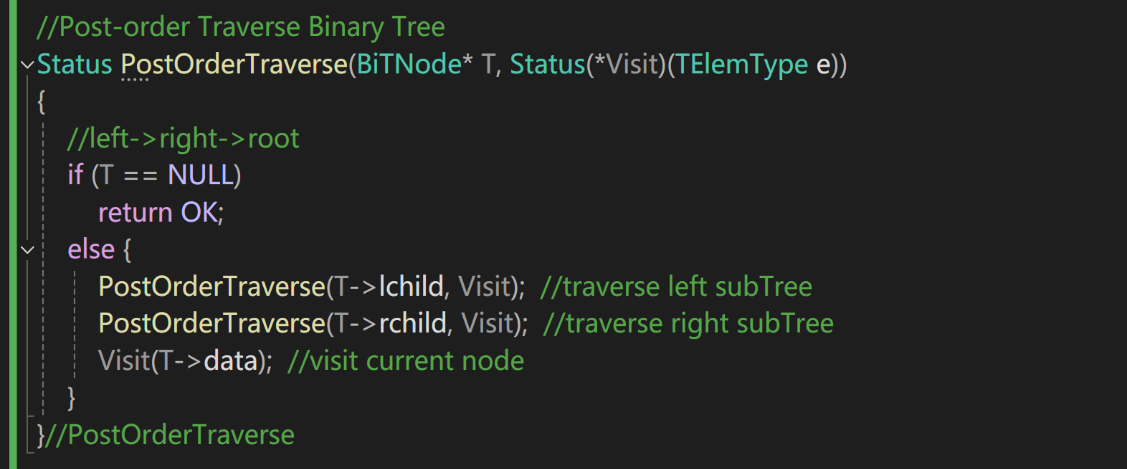
**图5.1创建二叉树**

****

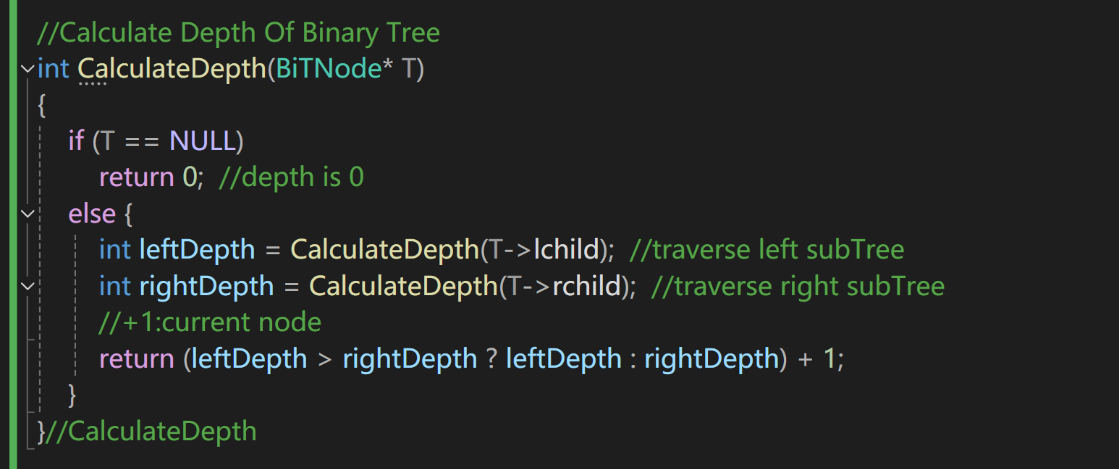
**图5.2先序遍历二叉树**

****

**图5.3中序遍历二叉树**

****

**图5.4后序遍历二叉树**

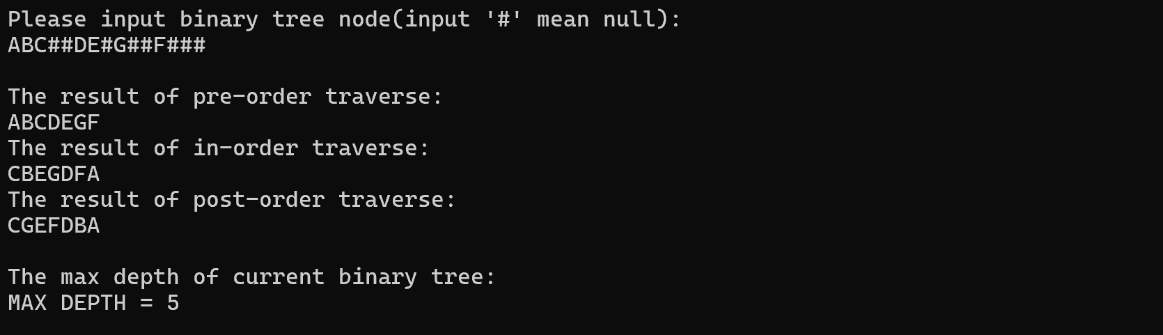
****

**图5.5计算二叉树深度**

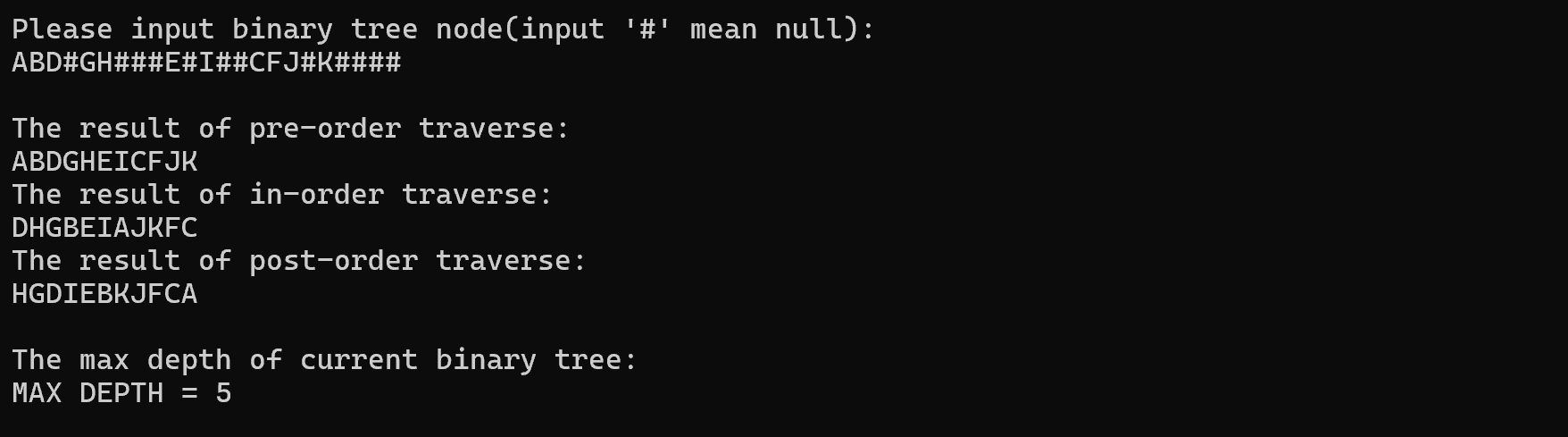
**六、实验结果**

**（一）结果呈现：**

1.根据以上**实验内容**的程序代码运行后进行多次测试，以下是根据**实验内容测试用例**进行测试后的结果示意图：

****

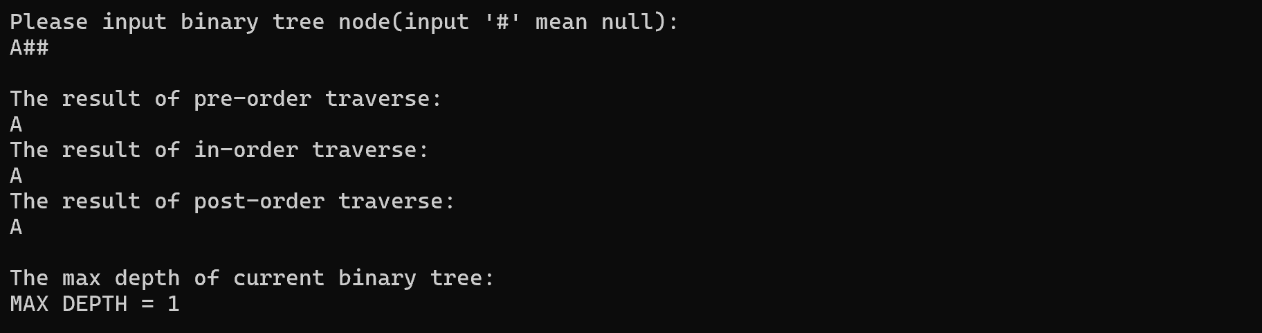
**图6.1实验测试用例（1）结果**

****

**图6.2实验测试用例（2）结果**

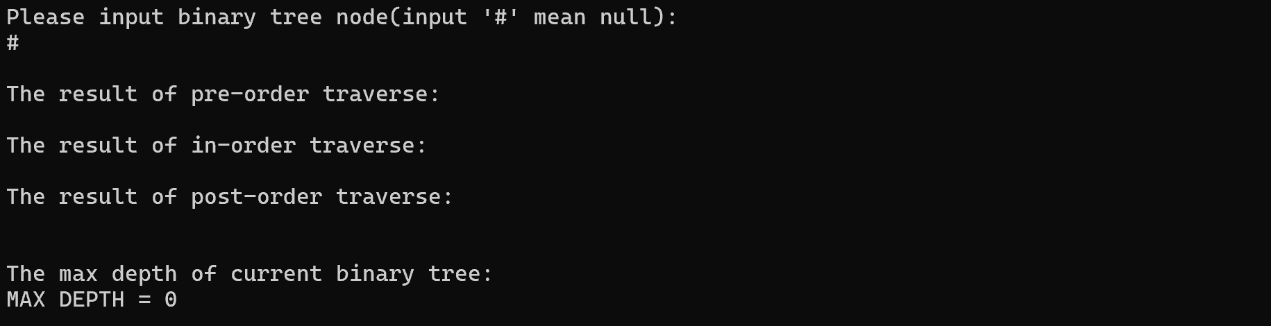
2.为检查**特殊情况**，此处增加**边界测试**，以下为边界测试结果呈现：

（1）**（先序）输入序列：A##**



**图6.3边界测试（1）**

（2）**（先序）输入序列：#**



**图6.4边界测试（2）**

**（二）结果分析：**

**1.基本分析**

**测试用例（1）：（先序）输入序列：ABC##DE#G##F###**

先序遍历结果：**A B C D E G F**

中序遍历结果：**C B E G D F A**

后序遍历结果：**C G E F D B A**

该二叉树的最大深度：**5**

**测试用例（2）：（先序）输入序列：ABD#GH###E#I##CFJ#K####**

先序遍历结果：**A B D G H E I C F J K**

中序遍历结果：**D H G B E I A J K F C**

后序遍历结果：**H G D I E B K J F C A**

该二叉树的最大深度：**5**

**综上，实验测试用例结果与预期结果及二叉树图像符合，边界测试结果也符合预期。**

**2.时间复杂度分析**

**【CreateBiTree】—— 创建二叉树**

该函数在遍历每个节点时都会被调用一次，因此总时间复杂度与节点数成正比，故时间复杂度为**O(n)**，其中**n**是二叉树的节点数量。

**【PreOrderTraverse】—— 先序遍历二叉树**

**每个节点会被访问一次，**故时间复杂度为**O(n)**，其中**n**是二叉树的节点数量。

**【InOrderTraverse】—— 中序遍历二叉树**

**每个节点会被访问一次，**故时间复杂度为**O(n)**，其中**n**是二叉树的节点数量。

**【PostOrderTraverse】—— 后序遍历二叉树**

**每个节点会被访问一次，**故时间复杂度为**O(n)**，其中**n**是二叉树的节点数量。

**【CalculateDepth】—— 计算二叉树深度**

**每个节点会被遍历一次来计算深度，**故时间复杂度为**O(n)**，其中**n**是二叉树的节点数量。

**3.空间复杂度分析**

**【创建二叉树】**

**递归调用栈时，**其空间复杂度**取决于递归的深度**（**如果是平衡树，空间复杂度为O(logn)），**在最坏情况下，空间复杂度为**O(h)**，其中**h**是树的高度。

**【前序遍历/中序遍历/后序遍历/计算深度】**

同样的，**递归调用栈的空间复杂度取决于树的高度**，故空间复杂度为**O(h)**，其中**h**是树的高度。

**七、实验总结**

1.在对二叉树的某个结点进行访问时，可以进行适当的操作简化：不必**利用函数（指针）来作为函数的参数**进行结点访问，可以**直接使用printf("%c", e);**，并在必要时用**返回值**来判断是否成功执行。

2.在进行**递归操作**时，需要注意一些问题：在递归函数中**必须明确递归的终止条件，否则会导致无限递归**。例如，在处理二叉树时，应该在节点为空时**（T == NULL）**返回。

3.递归调用时，**每次调用都会在栈上分配空间**。当递归深度过大时，可能会**导致栈溢出**。因此需要考虑输入数据的规模，**避免递归过深**。由此在某些情况下，可以考虑使用**循环**或**迭代**的方式来替代递归。