班级 计算机203 学号 202007020625 姓名 於俊涛 报告日期 11/25

**实验题目： 树与二叉树数据结构的关键算法实现及性能分析**

**一、实验目的**

**1. 能够运用高级程序设计技术实现树与二叉树数据结构及其关键算法；**

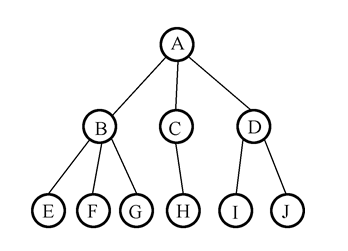
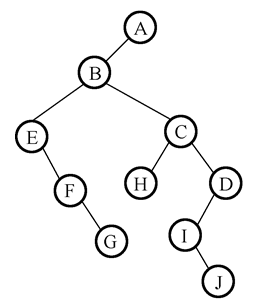
**2. 理解二叉树遍历的含义，能够阐明递归遍历和非递归遍历的工作原理；**

**3.** **能够运用集成开发环境编写和调试C语言程序；**

**二、实验原理**

1.树与二叉树数据结构的基本原理

逻辑结构：

主要性质：

1、二叉树的第i层上最多有2i-1个结点（i≥1）。

2、一棵深度为k的二叉树中，最多有2k-1个结点，最少有k个结点。

3、在一棵二叉树中，如果叶子结点数为n0，度为2的结点数为n2，则有: n0＝n2＋1。

4、完全二叉树的基本性质：

1> 具有n个结点的完全二叉树的深度为 【log2n】 （向下取整） +1。

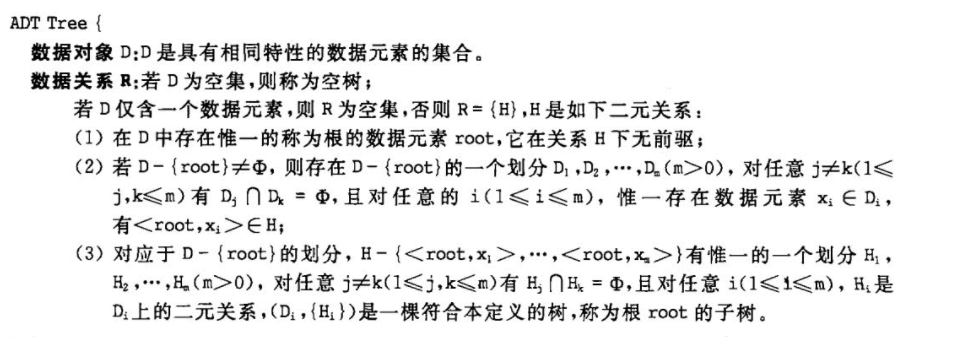
2> 对一棵具有n个结点的完全二叉树中从1开始按层序编号，则对于任意的序号为i（1≤i≤n）的结点（简称为结点i），有：

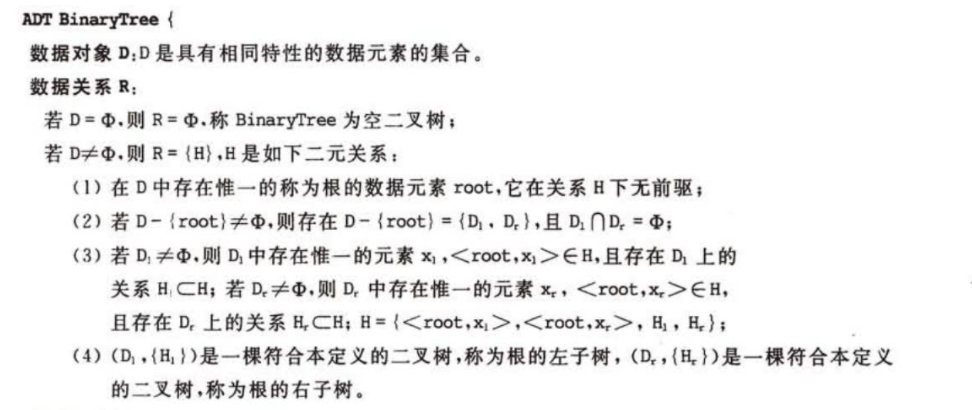
如果i＞1，则结点i的双亲结点的序号为 i/2；如果i＝1，则结点i是根结点，无双亲结点。

如果2i≤n，则结点i的左孩子的序号为2i；如果2i＞n，则结点i无左孩子。

如果2i＋1≤n，则结点i的右孩子的序号为2i＋1；如果2i＋1＞n，则结点 i无右孩子。

抽象数据类型定义：





2.关键算法设计原理及性能影响因素分析

遍历二叉树：先序遍历、中序遍历、后序遍历；

Eg:中序遍历：

Void InOrderTraverse(BiTree T){

If(T){

InOrderTraverse(T->lchild);//中序遍历左子树

Cout<<T->data;//访问根结点

InOrderTraverse(T->rchild);//中序遍历右子树}

}

3种遍历算法不同处仅在于访问根结点和遍历左、右子树的先后关系，在递归的角度来看3种算法遍历完全相同；无论是递归还是非递归遍历二叉树，每个节点只被访问1次，所以时间复杂度均为O(0);所需辅助空间为遍历过程中栈的最大容量，即树的深度，最坏情况为n，所以空间复杂度为O(0);

**三、实验方案设计**

1.存储方案设计

树的存储结构：

|  |  |  |
| --- | --- | --- |
| 储存结构 | 优势 | 不足 |
| 双亲表示法 | 求结点的双亲很方便，也容易求树的根； | 求结点的孩子不方便因为需要遍历整个结构； |
| 孩子表示法 | 求结点的孩子很方便； | 求结点的双亲不方便； |
| 孩子兄弟法 | 与二叉树的二叉链表表示完全一样，便于将一般的树结构转换为二叉树进行处理，利用二叉树的算法来实现对树的操作； |  |

双亲表示法与兄弟表示法优缺点相反，而孩子兄弟法是两者的结合，所以其在书的存储结构中最为常用；

二叉树的存储结构：

|  |  |  |
| --- | --- | --- |
| 储存结构 | 优势 | 不足 |
| 顺序存储 | 对完全二叉树较为方便； | 仅适用于完全二叉树，对存储空间有极大的浪费； |
| 链式存储 | 对于一般二叉树都适用； | 对二叉链表和三叉链表的使用需要分清楚； |

在不同的存储结构，实现二叉树的操作方法也不同，所以需要根据二叉树的形态和进行哪种操作来选用对应的存储结构；

2.算法的设计和实现

void InorderTraversal( BinTree BT ){//中序遍历

if(BT){

if(BT->Left)InorderTraversal(BT->Left);//遍历左孩子

printf(" %c",BT->Data);//访问根结点

if(BT->Right)InorderTraversal(BT->Right);//遍历右孩子

}

}

void PreorderTraversal( BinTree BT ){//先序遍历

if(BT){

printf(" %c",BT->Data); //访问根结点

if(BT->Left)PreorderTraversal(BT->Left); //遍历左孩子

if(BT->Right)PreorderTraversal(BT->Right); //遍历右孩子

}

}

void PostorderTraversal( BinTree BT ){//后序遍历

if(BT){

if(BT->Left)PostorderTraversal(BT->Left); //遍历左孩子

if(BT->Right)PostorderTraversal(BT->Right); //遍历右孩子

printf(" %c",BT->Data); //访问根结点

}

}

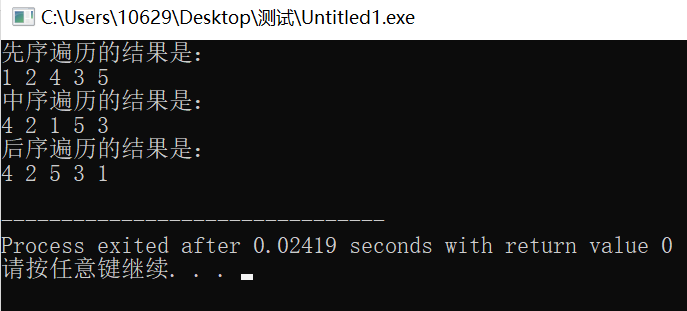
对于有n个结点的二叉树，时间和空间复杂度均为O(0);

3.测试方案

1

2 3

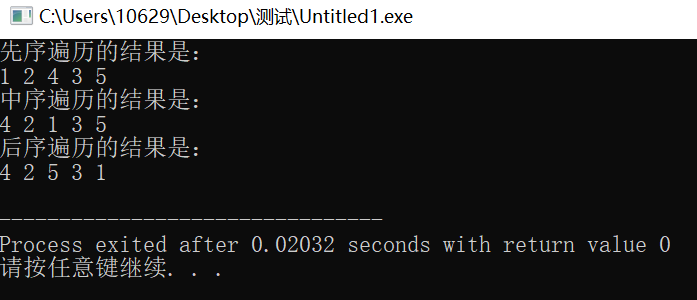
4 5



1

2 3

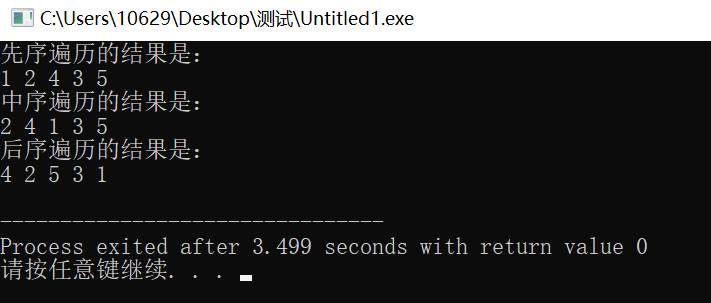
4 5



1

2 3

4 5



仅改变叶子结点的位置，只改变一种遍历算法的结果而对另两种遍历算法没有影响；