

**课 程 实 验 报 告**

2024 -2025 学年 第 一 学期

课程名称： 数据组织与管理

专业： 计算机科学与技术

班级： 4231091103

姓名： 陈明浩

学号： 4231091103

信息工程学院

2024 年 11 月

# **实验三 树和二叉树**

## 实验目的

1. 应用链式存储结构（二叉链表）构建二叉树
2. 实现二叉树遍历（先序、中序、后序及层次遍历）算法
3. 熟练完成递归和非递归程序的转换

## 实验内容

参照课本上的抽象数据类型二叉树的定义，采用链式存储结构表示二叉树及实现二叉树的如下操作：

1. 创建（CreateBiTree）：基于链式存储结构构建二叉树多项式加法
2. 实现先序遍历二叉树(递归和非递归算法)
3. 实现中序遍历二叉树(递归和非递归算法)
4. 实现后序遍历二叉树(递归算法)
5. 实现层次遍历二叉树
6. 求二叉树中叶子结点的个数
7. 求二叉树的深度

## 实验过程

## 3.1 分析问题

## ADT Tree {

## 数据对象:

## Node = {value, children}

## Tree = {root: Node}

## 

## 数据关系:

## R = {<parent, child> | parent.children contains child}

## 基本操作:

## CreateTree(Tree &root): 创建一棵空树

## preOrder(Tree root) : 前序遍历

inOrder(Tree root) : 中序遍历

postOrder(Tree root) : 后序遍历

preOrderNoRecursion(Tree root) : 非递归方式实现前序遍历

inOrderNoRecursion(Tree root) : 非递归方式实现中序遍历

levelOrder(Tree root) : 层序遍历

leafNode(Tree root) : 求叶子节点的个数

depth(Tree root) : 求树的深度

## }

## 3.2 详细设计

### 数据结构实现

struct TreeNode{

TreeNode \*lChild;

TreeNode \*rChild;

char data;

};

typedef TreeNode \*root;

### 主要算法

void createTree(Tree &root){

char ch; cin >> ch;

if(ch == '#'){ //录入到#则返回

root = nullptr;

return ;

}

root = new TreeNode();

root->data = ch;

createTree(root->lChild);

createTree(root->rChild);

}

void preOrder(Tree root){

if(root == nullptr) return ;

cout << root->data;

preOrder(root->lChild);

preOrder(root->rChild);

}

void inOrder(Tree root){

if(root == nullptr) return ;

inOrder(root->lChild);

cout << root->data;

inOrder(root->rChild);

}

void postOrder(Tree root){

if(root == nullptr) return ;

postOrder(root->lChild);

postOrder(root->rChild);

cout << root->data;

}

void preOrderNoRecursion(Tree root){

if(root == nullptr) return ;

stack<Tree> stk; //初始化存放节点指针的栈

while(root != nullptr || !stk.empty()){

if(root != nullptr){

cout << root->data; //输出根节点

stk.push(root);

root = root->lChild; //向左子节点移动

}

else{

root = stk.top(); stk.pop();

//向右子节点移动

root = root->rChild;

}

}

}

//同理这个只是语序改变先向左子节点移动之后再输出

void inOrderNoRecursion(Tree root){

if(root == nullptr) return ;

Stack stk;

while(root != nullptr || !isEmptyStack(stk)){

if(root != nullptr){

pushStack(stk,root);

root = root->lChild;

}

else{

//移动到根节点

root = peekStack(stk);

popStack(stk);

cout << root->data;

root = root -> rChild;

}

}

}

void levelOrder(Tree root){

if(root == nullptr) return ;

Queue que; initQueue(que);

pushQueue(que,root);

while(!isEmptyQueue(que)){

Tree temp = getHead(que); popQueue(que);

cout << temp->data;

if(temp->lChild != nullptr) pushQueue(que,temp->lChild);

if(temp->rChild != nullptr) pushQueue(que,temp->rChild);

}

}

int leafNode(Tree root){

if(root->lChild == nullptr && root->rChild == nullptr) return 1;

int n = leafNode(root->lChild);

int m = leafNode(root->rChild);

return n + m;

}

int depth(Tree root){

int m = depth(root->lChild);

int n = depth(root->rChild);

//这里将问题规模缩小，缩小成节点的左子树和右子树的最大深度加1

return (m > n ? m : n) + 1;

}

## 3.3 实验结果分析

使用递归来实现的三种遍历方式，由于每个节点都会被遍历到，所以时间复杂度为O(n),最差的情况而言，树变成一个链表，这里空间复杂度也为O(n),使用队列来实现的层序遍历每个节点都会被遍历所以时间复杂度也为O(n),对于完全二叉树来说，最最差的时间复杂度为O(n)，因为最大一层为(n/2)个节点

## 心得体会

通过树和二叉树，熟悉了递归调用的本质以及使用场景(可以分成相同更小问题时)，利用递归调用可以将复杂问题变得简单但是空间复杂度会变大，可能会造成系统的堆栈溢出，并且通过实现非递归调用熟悉了栈和队列的使用场景和用法。

## 评语和成绩

评语：

实验项目的背景、目的以及预期的功能需求阐述清晰。选择了适合的数据结构和算法，系统模块划分合理。算法描述准确，注释详细，图表准确，讲解较清楚。代码整洁规范、可读性强，且正确实现了设计文档中的功能需求。通过有效的注释和测试用例，展示了良好的编程习惯和测试覆盖率。

成绩：优