计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：\_杨佳森\_学号：2112080106 专业：\_数据科学与大数据技术年级：\_2021\_\_

课程： 数据结构 主讲教师：\_\_刘成\_ 辅导教师：\_\_无\_

实验时间：\_2023\_\_年 4月 25日 下\_午17\_\_时至\_19\_时，实验地点\_计算机大楼213\_

实验题目： 二叉树的应用

实验目的： 通过该实验，使学生理解二叉树的链式存储，掌握二叉树的几种遍历算法，并通过该实验使学生理解递归的含义，掌握C 语言编写递归函数的方法和注意事项。

实验环境（硬件和软件） PC和vs code

实验内容：

实现教材中算法6.4描述的二叉树创建算法，在此基础上实现二叉树的先序、后序递归遍历算法、两种非递归中序遍历、层序遍历、求二叉树的深度。

实验步骤：

1、先序遍历创建二叉树

//1、先序遍历创建二叉树

void CreateBiTree(BiTree& T) {

    char ch;

    cin >> ch;

    if (ch == '#')

        T = NULL;

    else {

        T = new BiNode;/\*创建一个新节点\*/

        T->ch = ch;

        CreateBiTree(T->lchild);

        CreateBiTree(T->rchild);

    }

}

2、先序遍历二叉树

//2、先序遍历二叉树

void PreOrderTraverse(BiTree& T) {

    if (T) {

        cout << T->ch <<" ";

        PreOrderTraverse(T->lchild);

        PreOrderTraverse(T->rchild);

    }

}

3、中序遍历二叉树1

//3、中序遍历二叉树(标记法)

void InOrderTraverse(BiTree& T) {

    stack<pair<BiNode\*, bool>> st;

    st.push(make\_pair(T, false));

    while (!st.empty()) {

        pair<BiNode\*, bool> curr = st.top();

        st.pop();

        if (curr.first == NULL) {

            continue;

        }

        if (curr.second) {

            cout << curr.first->ch << " ";

        } else {

            // 栈中元素按照右、根、左的顺序压入

            st.push(make\_pair(curr.first->rchild, false));

            st.push(make\_pair(curr.first, true));

            st.push(make\_pair(curr.first->lchild, false));

        }

    }

4、中序遍历二叉树2

//4、中序遍历二叉树(栈模拟)

void InOrderTraverse2(BiTree& T) {

    stack<BiTree>stk;

    auto cur = T;

    while (cur != NULL || !stk.empty()) {

        if (cur != NULL) {

            stk.push(cur);

            cur = cur->lchild;

        }

        else {

            cur = stk.top();

            cout << cur->ch << " ";

            stk.pop();

            cur = cur->rchild;

        }

    }

}

5、后序遍历二叉树

//5、后序遍历二叉树

void PostOrderTraverse(BiTree& T) {

    if (T) {

        PostOrderTraverse(T->lchild);

        PostOrderTraverse(T->rchild);

        cout << T->ch <<" ";

    }

}

6、层序遍历二叉树

//6、层序遍历二叉树

void levelOrder(BiTree& T) {

    vector <vector<char>>v;

    if (!T)

        cout << "二叉树为空" << endl;

    queue<BiTree>q;

    q.push(T);

    while (!q.empty()) {

        int len = q.size();

        v.push\_back(vector<char>());

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

            auto node = q.front();

            q.pop();

            v.back().push\_back(node->ch);

            if (node->lchild) q.push(node->lchild);

            if (node->rchild) q.push(node->rchild);

        }

    }

    for (int i = 0; i < v.size(); i++) {

        for (int j = 0; j < v[i].size(); j++)

            cout << v[i][j] << " ";

    }

}

7、求树的深度

//7、求树的深度

int Depth(BiTree& T) {

    if (T ==NULL)

        return 0;

    else {

        int i = Depth(T->lchild);

        int j = Depth(T->rchild);

        if (i >= j) return i + 1;

        else

            return j + 1;

    }

}

实验数据记录：

创建

输入 ：ABC$$DE$G$$F$$$ ($表示空格)

先序 屏幕输出 A B C D E G F

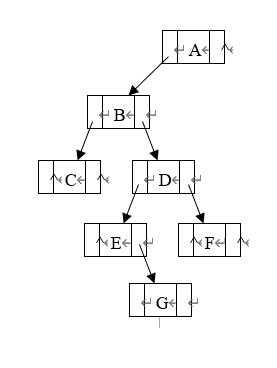
后序 屏幕输出 C G E F D B A

中序 屏幕输出 C B E G D F A

(两种中序非递归还需看源代码)

层序 屏幕输出 A B C D E F G

深度 屏幕显示 深度为5



问题讨论：

尝试在顺序存储结构上设计这些遍历算法，并在时空效率上与链式存储进行分析对比：链表法时间复杂度较高，空间复杂度较低；数组法时间复杂度较低，空间复杂度较高。因为数组法一开始就定义好树的大小，如果有空节点就浪费了空间，而链表法不会创建空结点，因此数组法的空间复杂度较高。链表法对指针的操作较繁琐，所需时间长，因此链表法的时间复杂度较低。

**源代码：**

#include<iostream>

#include<string>

#include<algorithm>

#include<vector>

#include<stack>

#include<queue>

using namespace std;

//定义二叉树的结构

typedef struct Node {

    char ch;

    struct Node \* lchild, \* rchild;

}\*BiTree, BiNode;

//1、先序遍历创建二叉树

void CreateBiTree(BiTree& T) {

    char ch;

    cin >> ch;

    if (ch == '#')

        T = NULL;

    else {

        T = new BiNode;/\*创建一个新节点\*/

        T->ch = ch;

        CreateBiTree(T->lchild);

        CreateBiTree(T->rchild);

    }

}

//2、先序遍历二叉树

void PreOrderTraverse(BiTree& T) {

    if (T) {

        cout << T->ch <<" ";

        PreOrderTraverse(T->lchild);

        PreOrderTraverse(T->rchild);

    }

}

//3、中序遍历二叉树(标记法)

void InOrderTraverse(BiTree& T) {

    stack<pair<BiNode\*, bool>> st;

    st.push(make\_pair(T, false));

    while (!st.empty()) {

        pair<BiNode\*, bool> curr = st.top();

        st.pop();

        if (curr.first == NULL) {

            continue;

        }

        if (curr.second) {

            cout << curr.first->ch << " ";

        } else {

            // 栈中元素按照右、根、左的顺序压入

            st.push(make\_pair(curr.first->rchild, false));

            st.push(make\_pair(curr.first, true));

            st.push(make\_pair(curr.first->lchild, false));

        }

    }

}

//4、中序遍历二叉树(栈模拟)

void InOrderTraverse2(BiTree& T) {

    stack<BiTree>stk;

    auto cur = T;

    while (cur != NULL || !stk.empty()) {

        if (cur != NULL) {

            stk.push(cur);

            cur = cur->lchild;

        }

        else {

            cur = stk.top();

            cout << cur->ch << " ";

            stk.pop();

            cur = cur->rchild;

        }

    }

}

//5、后序遍历二叉树

void PostOrderTraverse(BiTree& T) {

    if (T) {

        PostOrderTraverse(T->lchild);

        PostOrderTraverse(T->rchild);

        cout << T->ch <<" ";

    }

}

//6、层序遍历二叉树

void levelOrder(BiTree& T) {

    vector <vector<char>>v;

    if (!T)

        cout << "二叉树为空" << endl;

    queue<BiTree>q;

    q.push(T);

    while (!q.empty()) {

        int len = q.size();

        v.push\_back(vector<char>());

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

            auto node = q.front();

            q.pop();

            v.back().push\_back(node->ch);

            if (node->lchild) q.push(node->lchild);

            if (node->rchild) q.push(node->rchild);

        }

    }

    for (int i = 0; i < v.size(); i++) {

        for (int j = 0; j < v[i].size(); j++)

            cout << v[i][j] << " ";

    }

}

//7、求树的深度

int Depth(BiTree& T) {

    if (T ==NULL)

        return 0;

    else {

        int i = Depth(T->lchild);

        int j = Depth(T->rchild);

        if (i >= j) return i + 1;

        else

            return j + 1;

    }

}

void show() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*1.先序创建二叉树\*\*\*" << endl;

    cout << "\*\*\*2.先序遍历二叉树\*\*\*" << endl;

    cout << "\*\*\*3.中序遍历二叉树1\*\*" << endl;

    cout << "\*\*\*4.中序遍历二叉树2\*\*" << endl;

    cout << "\*\*\*5.后序遍历二叉树\*\*\*" << endl;

    cout << "\*\*\*6.层序遍历二叉树\*\*\*" << endl;

    cout << "\*\*\*7.求二叉树的深度\*\*\*" << endl;

    cout << "\*\*\*8.退出\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main() {

    int choice;

    BiTree T;

    show();

    while (1) {

        cout << "请输入你的选择：";

        cin >> choice;

        switch (choice) {

        case 1:

            cout << "先序遍历输入(#为空格):";

            CreateBiTree(T);

            break;

        case 2:

            cout << "先序遍历输出:";

            PreOrderTraverse(T);

            cout << endl;

            break;

        case 3:

            cout << "中序遍历输出1:";

            InOrderTraverse(T);

            cout << endl;

            break;

        case 4:

            cout << "中序遍历输出2:";

            InOrderTraverse2(T);

            cout << endl;

            break;

        case 5:

            cout << "后序遍历输出:";

            PostOrderTraverse(T);

            cout << endl;

            break;

        case 6:

            cout << "层序遍历输出:";

            levelOrder(T);

            cout << endl;

            break;

        case 7:

            cout << "二叉树深度为:" << Depth(T) << endl;

            break;

        case 8:

            exit(0);

            break;

        default:

            break;

        }

    }

    return 0;

}