《算法与数据结构》实验报告

实验名称：树型数据结构及其应用

系　　别：计算机科学与技术学院

专　　业：计算机类

班　　级：21 级计算机类 1 班

姓　　名：侯文辉

学　　号：2021463030114

实验日期：　2022 年 5 月 30 日

2. 实验名称：树型数据结构及其应用

2. 实验目的：

通过实验达到：

⑴ 理解和掌握树及二叉树的基本概念；

⑵ 理解和掌握二叉树的顺序存储结构、链式存储结构；

⑶ 理解和掌握采用二叉链式存储结构下二叉树的各种遍历操作的思想及其应用；

⑷ 加深对堆栈、队列的概念及其典型操作思想的理解；

⑸ 掌握典型二叉树操作算法的算法分析。

3. 对每个题目，包括：

设树结点的元素类型为 char，实现以下二叉树的各种基本操作的程序：

① 建立不少于 10 个结点的二叉树 T；

②用后序遍历方式输出树 T 的结点；

③用层次遍历方式输出树 T 的结点；

④输出树 T 的深度；

⑤输出树 T 的叶子结点和非叶子结点；

⑥主函数通过函数调用实现以上各项操作。

⑴ 所定义的数据结构：

队列和树的结构体：

```
typedef struct{
    char data;
    struct Node *pleft;
    struct Node *pright;
}Node;

//队列实现
typedef struct{
    Node **data;
    int rear;
    int head;
    int Size;
}Queue;
```

(2) 主要操作算法思想或算法步骤:

后序遍历：通过调整 1.输出 2.访问左节点 3.访问右节点这样的顺序，递归达到后序遍历的效果

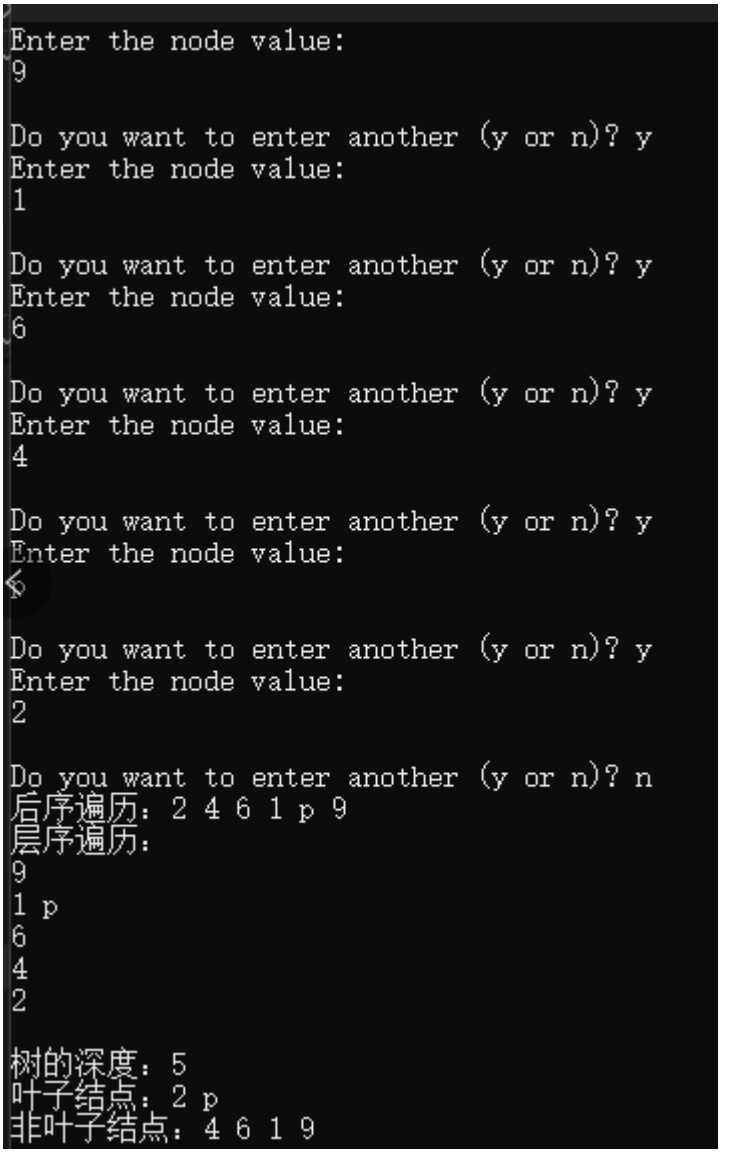
层次遍历：将使用队列这种先入先出的数据结构，达到顺序的层次遍历。首先会一层的节点加入队列，计数队列中节点个数，算出队列中含有的节点数，将该层其中的左右节点加入队列，之后他们本身出队列，这就完成了一层的遍历，循环到队列为空。

输出树 T 的深度：
利用递归，将全部节点 ，最长的路径就是他们的深度

输出树 T 的叶子结点和非叶子结点:
递归，判断左右节点是否为空，为空就是叶子结点，输出，不为空的都是非叶子节点。

(3)　主要操作算法的算法分析:
后序遍历：时间复杂度 O(n),空间复杂度 O(n)
层次遍历：时间复杂度 O(n),空间复杂度 O(n)
输出树 T 的深度：时间复杂度 O(n),空间复杂度 O(n)
输出树 T 的叶子结点和非叶子结点: 时间复杂度 O(n),空间复杂度 O(n)

(4)　程序运行过程及结果（抓屏粘贴）:



4. 问题与总结：

目前因为题目要求节点为 Char，无法实现负数的加入，有带加强，同时对节点的录入也有点问题，是按顺序加入的。

附件：代码

```
/************************ Dongguan-University of Technology
-ACE************************
 * @file Exp3.c
 * @author pansyhou 侯文辉 (1677195845lyb@gmail.com)
 * @brief
 * @version 0.1
 * @date 2022-04-01
 *
 * @copyright Copyright (c) 2022
 *
 ************************ Dongguan-University of Technology
-ACE************************/
#include "stdio.h"
#include "ctype.h"
#include "stdlib.h"
#include "stdbool.h"
typedef struct{
    char data;
    struct Node *pleft;
    struct Node *pright;
}Node;

//队列实现
typedef struct{
    Node **data;
    int rear;
    int head;
    int Size;
}Queue;

//队列创建
```

```c
Queue* QueueInit(int capacity)
{
    Queue *ret=(Queue *)malloc(sizeof(Queue));
    ret->data=(Node **)malloc(sizeof(Node *)*capacity);
    ret->Size=capacity;
    ret->rear=-1;
    ret->head=-1;
    return ret;
}

//node queue pop
Node* QueuePop(Queue *obj)
{
    Node *x=obj->data[obj->head];//暂存 头数据
    if(obj->rear==obj->head)//判断队列是否为空
    {
        obj->head=-1;
        obj->rear=-1;
        return x;
    }
    obj->head=(obj->head+1)%obj->Size;//头指向后一个
    return x;
}


//node queue push
void QueuePush(Queue *obj,Node *x)
{
    if(obj->head==-1)//当队列为空时
    {
        obj->head=0;
    }
    obj->rear=(obj->rear+1)%obj->Size;
    obj->data[obj->rear]=x;
}

//QueueISempty
bool QueueISempty(Queue *obj)
```

```
{
    return obj->head==-1;
}
//queue 数量
int QueueSize(Queue *obj)
{
    return (obj->rear-obj->head+obj->Size+1)%obj->Size;
}




Node* CreatNode(char value){
    Node *pnode = (Node *)malloc(sizeof(Node));
    pnode->data = value;
    pnode->pleft = pnode->pright = NULL;
    return pnode;
}

Node *addnode(char value,Node *pnode)
{
    if(pnode == NULL)
        return CreatNode(value);

    if(value == pnode->data)
    {
        return pnode;
    }

    if(value < pnode->data)
    {
        if(pnode->pleft == NULL)
        {
            pnode->pleft = CreatNode(value);
            return pnode->pleft;
        }
        else
        {
```

```
            return addnode(value, pnode->pleft);
        }
    }
    else
    {
        if(pnode->pright == NULL)
        {
            pnode->pright = CreatNode(value);
            return pnode->pright;
        }
        else
        {
            return addnode(value, pnode->pright);
        }
    }
}

//后序遍历
void postorder(Node *pnode)
{
    if(pnode == NULL)
        return;
    postorder(pnode->pleft);
    postorder(pnode->pright);
    printf("%c ",pnode->data);
}

//层序遍历
void levelorder(Node *pnode)
{
    Queue *q = QueueInit(100);
    QueuePush(q,pnode);
    while(!QueueISempty(q))
    {
        int num=QueueSize(q);
        for(int i=0;i<num;i++)
        {
            Node *p = QueuePop(q);
```

```
            printf("%c ",p->data);
            if(p->pleft != NULL)
                QueuePush(q,p->pleft);
            if(p->pright != NULL)
                QueuePush(q,p->pright);
        }
        printf("\n");
    }
}

int TreeDepth(Node *pnode)
{
    if(pnode == NULL)
        return 0;
    int left = TreeDepth(pnode->pleft);
    int right = TreeDepth(pnode->pright);
    return left>right?left+1:right+1;
}
//输出树 T 的叶子结点
void PrintLeaf(Node *pnode)
{
    if(pnode == NULL)
        return;
    if(pnode->pleft == NULL && pnode->pright == NULL)
        printf("%c ",pnode->data);
    PrintLeaf(pnode->pleft);
    PrintLeaf(pnode->pright);
}

//输出非叶子结点
void PrintNOTLeaf(Node *pnode){
    if(pnode == NULL)
        return;
    if(pnode->pleft == NULL && pnode->pright == NULL)
        return;
    PrintNOTLeaf(pnode->pleft);
    PrintNOTLeaf(pnode->pright);
    printf("%c ",pnode->data);
```

```
}


int main(void){
    char newvalue;
    Node *proot = NULL;
    char answer = 'n';
    do
    {
        printf("Enter the node value:\n");
        scanf(" %c", &newvalue);
        if(proot == NULL)
        {
            proot = CreatNode(newvalue);
        }
        else
        {
            addnode(newvalue, proot);
        }
        printf("\nDo you want to enter another (y or n)? ");
        scanf(" %c", &answer);
    } while(tolower(answer) == 'y');

    printf("后序遍历: ");
    postorder(proot);
    printf("\n");
    printf("层序遍历: \n");
    levelorder(proot);
    printf("\n");
    printf("树的深度: %d\n",TreeDepth(proot));
    printf("叶子结点: ");
    PrintLeaf(proot);
    printf("\n");
    printf("非叶子结点: ");
    PrintNOTLeaf(proot);
    printf("\n");

}
```