# 《数据结构》实验报告四/五

|  |  |  |
| --- | --- | --- |
| 题目： 二叉树的遍历 | | 班级： 通信1601 |
| 学号： 29 |  | 姓名： 郑祖闯 |
| 日期： |  | 程序名： |

### 一、上机实验的问题和要求：

要求采用二叉链表作为存储结构，完成二叉树的建立，前序、中序和后序遍历的操作，求所有叶子及结点总数的操作等。具体实现要求：

1. 基于先序遍历的构造算法：输入是二叉树的先序序列，但必须在其中加入虚结点以示空指针的位置。假设虚结点输入时用空格字符表示。
2. 分别利用前序遍历、中序遍历、后序遍历所建二叉树。
3. 求二叉树结点总数，观察输出结果。
4. 求二叉树叶子总数，观察输出结果。
5. 交换各结点的左右子树，用广义表表示法显示新的二叉树。

6. 编写在后序线索化树上的遍历算法。

★阅读理解建立Huffman树的算法，对该算法进行运行及调试。具体实现要求：

1. 调试并运行Huffman算法
2. 求Huffman树的带权路径长度WPL

### 二、程序设计的基本思想，原理和算法描述：

（包括程序的结构，数据结构，输入/输出设计，符号名说明等）

二叉树用二叉链表来储存

Typedef struct BiTNode

{

TElem Type data； //数据域

Struct BiTNode \*lchild，\*rchild；//指向左右孩子的指针

}

所以操作数据用字符型 则有

Typedef char TElemType； //定义TElemType为char型

以下是七个二叉树基本操作的函数原型说明：

Status CreateBiTree(BiTree &T); //先序创建二叉树

Status PreOrderTraverse(BiTree T); //先序遍历

Status InOrderTraverse(BiTree T); //中序遍历

Status PostOrderTraverse(BiTree T); //后序遍历

int CountNode (BiTree T); //统计结点总数

int CountLeaf (BiTree T); //统计叶子总数

Status Exchange(BiTree &T) //交换左右子树

先序创建二叉树时输入顺序为

"-+a\t\t\*b\t\t-c\t\td\t\t/e\t\tf\t\t" \t是空格键，表示虚结点

### 三、源程序及注释：

#include<stdio.h>

#include<malloc.h>

typedef char TElemType; //定义TElemType为 char 型 typedef

enum {ERROR,OK} Status; //状态变量OK ERROR

typedef struct BiTNode

{

TElemType data; //数据域

struct BiTNode \*lchild,\*rchild;//指向左右孩子的指针

lchild,rchild

}BiTNode,\*BiTree;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 函数原型说明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status CreateBiTree(BiTree &T); //先序创建二叉树

Status PreOrderTraverse(BiTree T); //先序遍历

Status InOrderTraverse(BiTree T); //中序遍历

Status PostOrderTraverse(BiTree T); //后序遍历

int CountNode (BiTree T); //统计结点总数

int CountLeaf (BiTree T); //统计叶子总数

Status Exchange(BiTree &T) //交换左右子树

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 主函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main (void)

{

BiTree T;

printf("先序遍历创建二叉树：输入字符型数据，空格表示虚结点\n");

CreateBiTree(T); //先序创建二叉树

printf("先序遍历结果:\n");

PreOrderTraverse(T); printf("\n"); //先序遍历

printf("中序遍历结果:\n");

InOrderTraverse(T); printf("\n"); //中序遍历

printf("后序遍历结果:\n");

PostOrderTraverse(T); printf("\n"); //后序遍历

printf("结点总数:%d\n",CountNode(T)); //节点总数

printf("叶子总数:%d\n",CountLeaf(T)); //节点总数

Exchange(T); //交换左右子树

printf("交换左右子数后中序遍历结果:\n");

InOrderTraverse(T); printf("\n"); //交换后中序遍历

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 先序创建二叉树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status CreateBiTree(BiTree &T)

{

char ch;

scanf("%c",&ch);

if(ch==' ') T=NULL;

else

{

T=(BiTNode \*)malloc(sizeof(BiTNode));

if(!T) return ERROR;

T->data=ch;

CreateBiTree(T->lchild);

CreateBiTree(T->rchild);

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 先序遍历访问二叉树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status PreOrderTraverse(BiTree T) //先序遍历

{

if(T) //非空二叉树

{

printf("%c",T->data); //访问Data

PreOrderTraverse(T->lchild); //递归遍历左子树 PreOrderTraverse(T->rchild); //递归遍历右子树

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 中序遍历访问二叉树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status InOrderTraverse(BiTree T) //中序遍历

{

if(T) //非空二叉树

{

InOrderTraverse(T->lchild); //递归遍历左子树

printf("%c",T->data);

InOrderTraverse(T->rchild); //递归遍历右子树

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 后序遍历访问二叉树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status PostOrderTraverse(BiTree T) //后序遍历

{

if(T) //非空二叉树

{

PostOrderTraverse(T->lchild); //递归遍历左子树 PostOrderTraverse(T->rchild); //递归遍历右子树 printf("%c",T->data);

}

return OK;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 统计结点总数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int CountNode (BiTree T)

{

if(T)

return 1+CountNode (T->lchild)+CountNode (T->rchild);

else return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 统计叶子总数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ int CountLeaf (BiTree T)

{

if (T)

{

if (!T->lchild && !T->rchild) return 1;

else

return CountLeaf(T->lchild) + CountLeaf(T->rchild);

}

else return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5 交换左右子树

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ Status Exchange(BiTree &T)

{

if(T)

{

BiTree temp=T->lchild;

T->lchild=T->rchild;

T->rchild=temp;

if(T->lchild) Exchange(T->lchild);

if(T->rchild) Exchange(T->rchild);

}

return OK;

}

### 四、运行输出结果：

### 说明: C:\Users\Jun\Desktop\数据结构\4-5.JPG

### 五、调试和运行程序过程中产生的问题及采取的措施：

本次实验的内容相对比较容易，实现过程也不复杂，所以没有什么问题

### 六、对算法的程序的讨论、分析，改进设想，其它经验教训：

算法都是由递归实现的，很容易理。可以改成由非递归实现，这样就比较复杂了，需要注意很多细节，能够体现出对具体过程的理解

### 七、对实验方式、组织、设备、题目的意见和建议：

本次的二叉树实验比较简单，应该选择比较有实际应用的编程题，带着兴趣编程才是最重要的。