《数据结构》上机报告

2018 年 11 月 10 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分：

实验题目	二叉树
问题描述	对二叉树进行先中后遍历分别得到一个**线性序列**，在该序列中每个结点都只有一个直接前驱和直接后继（除第一个和最后一个结点）。用二叉链表表示二叉树，有 n+1 个指针域为空，若用其存储该结点的前驱或后继结点，则称该指针域为线索。**线索二叉树**就是对二叉树进行了某种线索化的二叉树。
基本要求	1. 练习线索二叉树的基本操作，包括先序线索化，先序遍历线索二叉树、输出二叉树。 2. 练习中序遍历线索二叉树的基本操作，包括中序线索化，中序遍历线索二叉树、查找某元素的中序遍历的后继结点，前驱结点。
	已完成基本内容（序号）： 1，2
选做要求	
	已完成选做内容（序号）

数据结构设计	`typedef enum { Link, Thread }PointerTag;`
	`typedef char TElemType;`
	`typedef struct Node`
	`{`
	`TElemType data;`
	`Node *rchild, *lchild;`
	`PointerTag Ltag = Link, Rtag = Link;`
	`}BiThrNode, *BiThrTree;`
	本次实验的数据结构是二叉树，不过在这个结构中又添加了两个新成员——左右标志 Ltag 与 Rtag，因为要线索化二叉树，所以用它们来记录是否有左右孩子及是否需要线索化（Link 表示有孩子结点，不需要线索化；Thread 表示没有孩子结点，需要线索化），左右孩子指针*lchild,*rchild 则是线索化后指向前驱和后继，这样就很好地将树形存储结构转化成了线性存储结构，可以按照线性结构访问的方式来访问二叉树的数据。

功能(函数)说明	`/*********************************************************************`
	`函数功能：创建树`
	`说明：利用递归的方法创建树，遇到‘#’时将该结点置空；否则就新建一个节点来存储该数据，再递归对该结点的左右孩子结点进行判断赋值。`
	`*********************************************************************/`
	`void CreatBiTree(BiThrTree &T)`
	`{`
	`TElemType c;`
	`cin >> c;`

```
    if (c == '#')
        T = NULL;
    else
    {
        T = new Node;
        if (!T)
            exit(-1);
        T->data = c;
        CreatBiTree(T->lchild);
        CreatBiTree(T->rchild);
    }
}
/*************************************************************************
函数功能：先序递归线索化二叉树
函数说明：先检查根结点是否有左孩子，若没有左孩子，就让它的左标志等于Thread，表示可线索化，然后让该结点的左孩子指向前驱结点；再检查是否有右孩子，若无右孩子，就让它的右标志为Thread，表示可线索化；若果前驱结点不为空，就让前驱结点的右标志为Thread，表示可线索化，然后让前驱结点的右孩子指向该结点后继结点；最后将该结点赋值给前驱结点pre;接下来，如果该结点的左标志为Link即有左孩子，就继续递归，直到找到结点没有左孩子，再对当前结点的右标志检查，如果该结点的右标志为Link,即有右孩子，也让它递归直到找到结点既没有左孩子也没有右孩子(即叶子结点)，以此类推，逐层退出递归。
*************************************************************************/
void PreOrderThreading(BiThrTree &T)
{
    if (!T->lchild)
    {
        T->Ltag = Thread;
        T->lchild = pre;
    }
    if (!T->rchild)
        T->Rtag = Thread;
    if (pre&&pre->Rtag == Thread)
        pre->rchild = T;
    pre = T;
    if (T->Ltag == Link)PreOrderThreading(T->lchild);
    if (T->Rtag == Link)PreOrderThreading(T->rchild);
}
/*************************************************************************
函数功能：去线索化
说明：去线索化就是将已经建立的线索二叉树上面的线性序列拆分，回到二叉树形结构。在先序建立二叉树的基础上，只要每一次递归都对当前结点的左右标志进行判断。如果是Thread，即已经进行了线索化，那就让该孩子结点置空。那么递归完成时，线索二叉树就变成了最初创建的二叉树。
*************************************************************************/
```

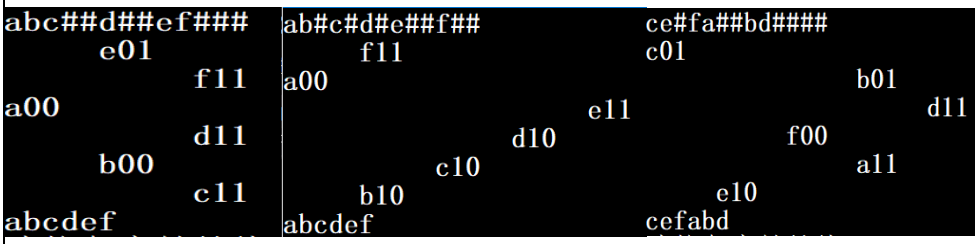
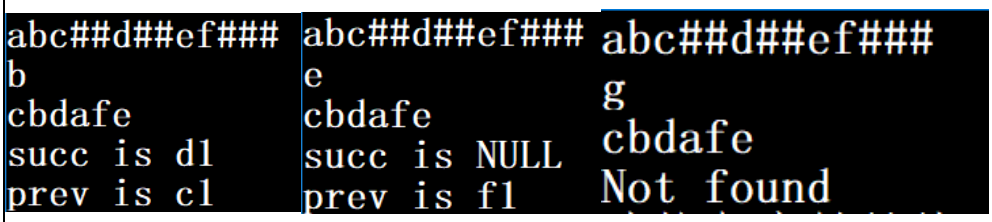
```
void RemoveThreading(BiThrTree &T)
{

    if (T->Ltag == Thread)
        T->lchild = NULL;
    if (T->Rtag == Thread)
        T->rchild = NULL;
    if (T->Ltag == Link)RemoveThreading(T->lchild);
    if (T->Rtag == Link)RemoveThreading(T->rchild);
}
/*****************************************************************************
函数功能：输出二叉树树形
说明：由题目给出的树形分析得知，每行的元素正好是中序遍历得到元素的逆序排列由此可以将中序遍历的递归方法中的左右孩子访问次序对换，那么输出的数据次序就是树形输出次序，空格个数是该结点的深度，所以在递归中应该加一个Depth来计算每个结点的深度，实际上递归的层次数就是该结点所在的深度，这样每次的访问不管是进入递归还是退出递归都会对应一个相应的Depth，那样就很容易记录每个结点的深度，从而输出树形。在此因为输出的是线索二叉树的树形，所以输出之后要将结点的左右标志置为Link，即0，初始化，便于下一次线索化。
*****************************************************************************/
void ShapeBiThrTree(BiThrTree &T, int Depth)
{
    if (T)
    {
        ShapeBiThrTree(T->rchild, Depth + 1);
        for (int i = 1; i < Depth; i++)
            cout << "    ";
        cout << T->data << T->Ltag << T->Rtag;
        T->Ltag = Link;
        T->Rtag = Link;
        cout << endl;
        ShapeBiThrTree(T->lchild, Depth + 1);
    }
}
/*****************************************************************************
函数功能：遍历先序线索二叉树
说明：新建一个节点，将根节点赋给它，比如p=T；然后利用循环，当右孩子不空的时候就一直遍历，如果当前结点有左孩子，就指向左孩子；否则指向右孩子，然后输出孩子结点的数据。直到右孩子为空，退出循环。
*****************************************************************************/
void PreOrderTraverse_Thrt(BiThrTree &T)
{
    BiThrTree  p = T;
    cout << p->data;
    while (p->rchild)
```

```
    {
        if (p->Ltag == Link)
            p = p->lchild;
        else
            p = p->rchild;
        cout << p->data;
    }
}
/*************************************************************************
函数功能：中序递归线索化二叉树
函数说明：中序线索化二叉树，首先从根结点的左孩子结点开始寻找，找到第一个左孩子为空的结点，如果该结点的左孩子为空，就让左标志等于Thread，表示可线索化，然后将左孩子结点指向前驱结点；如果右孩子为空，就让右标志为Thread，表示可线索化；如果前驱结点不为空并且前驱结点的右标志为Thread，则让它的右孩子指向当前结点；再把前驱结点赋值为当前结点；然后再考察该结点的右孩子，每次进入递归时都是优先考察进入时结点左孩子，一直递归直到结束退出递归。
*************************************************************************/
void InOrderThreading(BiThrTree &T)
{
    if (T)
    {
        InOrderThreading(T->lchild);
        if (!T->lchild)
        {
            T->Ltag = Thread;
            T->lchild = pre;
        }
        if (!T->rchild)
            T->Rtag = Thread;
        if (pre&&pre->Rtag == Thread)
            pre->rchild = T;
        pre = T;
        InOrderThreading(T->rchild);
    }
}
/*************************************************************************
函数功能：遍历中序线索二叉树
函数说明：定义一个p，让其指向树的根节点T;
然后对p进行操作，只要p不空就进行循环，
    如果结点的左标志为Link,即有左孩子就进行循环，指向左孩子，
    直到结点没有左孩子，退出循环；
    然后输出结点的数据；
    接着如果刚才结点右标志为Thread，并且右孩子不空进行循环，指向该结点的右孩，输出右孩子结点的数据；
    指向刚才结点的右孩子；
```

```
p为空时退出循环，也就完成了遍历.
*************************************************************************/
void InOrderTraverse_Thrt(BiThrTree &T)
{
    BiThrTree p = T;
    while (p)
    {
        while (p->Ltag == Link)
            p = p->lchild;
        cout << p->data;
        while (p->Rtag == Thread && p->rchild)
        {
            p = p->rchild;
            cout << p->data;
        }
        p = p->rchild;
    }
}
/*************************************************************************
函数功能：寻找目标结点
返回值：目标结点
说明：寻找目标结点实际是在遍历的基础上进行的，从根节点找到第一个没有左孩子的结点，如果数据等于要寻找的目标数据，则返回该结点指针；接着继续循环，每次循环都指向结点的右孩子，如果该结点的数据等于目标数据，就返回该结点指针；退出该循环之后，
继续指向结点右孩子，直到结点右孩子为空，退出总循环。
此外，设置一个标志flag，若找到flag=1；否则，flag=0。
*************************************************************************/
BiThrNode* SearchNode(BiThrTree &T, TElemType ch, int&flag)
{
    BiThrTree p = T;
    while (p)
    {
        while (p->Ltag == Link)
        {
            p = p->lchild;
        }
        if (p->data == ch)
        {
            flag = 1;
            return p;
        }
        while (p->Rtag == Thread && p->rchild)
        {
```

```
            p = p->rchild;
            if (p->data == ch)
            {
                flag = 1;
                return p;
            }
        }
        p = p->rchild;
    }
    flag = 0;
}
/*****************************************************************************
函数功能：寻找目标结点前驱结点
返回值：前驱结点指针
说明：如果目标结点左孩子已经线索化，那么就直接返回目标节点的左孩子指针即可；
否则，建立pre指针指向目标结点的左孩子，当pre结点的右标志为Link，即本来就有右
孩子，就继续循环让它再指向右孩子，直到pre右孩子是线索化的(未线索化之前为空)，
即pre右标志为Thread，退出循环，返回该pre指针即可；
*****************************************************************************/
BiThrNode* InOrderPre(BiThrTree &p)
{
    BiThrTree pre;
    if (p->Ltag == Thread)
        return p->lchild;
    else
    {
        pre = p->lchild;
        while (pre->Rtag == Link)
            pre = pre->rchild;
        return pre;
    }
}
/*****************************************************************************
函数功能：寻找目标结点后继结点
返回值：后继结点指针
说明：如果目标结点右孩子已经线索化，那么就直接返回目标节点的右孩子指针即可；
否则，就建立succ指针，让其指向目标结点的右孩子，当succ结点的左标志为Link的时候
就循环，每次循环succ都指向它的左孩子，直到succ的左孩子是线索化的，(未线索化之
前为空)即succ左标志为Thread，退出循环，返回该succ指针即可.
*****************************************************************************/
BiThrNode* InOrderSucc(BiThrTree &p)
{
    BiThrTree succ;
    if (p->Rtag == Thread)
```

```
        return p->rchild;
    else
    {
        succ = p->rchild;
        while (succ->Ltag == Link)
            succ = succ->lchild;
        return succ;
    }
}
```

开发环境	Win10，vs2017,C++高级程序语言设计
调试分析	1.  2. 
心得体会	本次实验主要是以**二叉树的线索化、遍历线索二叉树**为主。在线索化二叉树过程中，始终要遵循左孩子为空，那它指向前驱结点，右孩子为空，那它始终指向后继结点。在此基础上再进行二叉树的线索化。**二叉树线索化有先序线索化、中序线索化、后序线索化。** 先序线索化二叉树的过程中，首先要搞懂先序的顺序，然后根据这个顺序，再按照规则进行线性链接，前驱后继也就十分明确了。但是在输出线索二叉树形的时候（还必须输出左右标志），就遇到了一个比较棘手的问题，因为在原来的二叉树（没被线索化）中，可以利用中序递归的方法，交换左右孩子结点的顺序，再计算深度输出即可，一旦线索化，就没法使用那种方法输出标志了，所以我想的办法是**将二叉树先线索化**，这样就先**得到了左右标志**，之后在**保留左右标志的基础上去线索化**，恢复到原来二叉树，再利用以前的方法就能准确输出树形及左右标志。 中序线索化的过程和先序线索化基本一致，主要就是两者的顺序不一致，只需改变顺序即可。这个问题中主要的是寻找目标结点的前驱结点和后继结点，在此我利用先寻找目标结点，再寻找它的前驱和后继的方法来解决这个问题。在寻找目标的时候很简单，实际还是遍历线索二叉树；而在寻找它的***前驱/后继***的时候，最关键的一点就是看它的***左/右***孩子结点是否线索化（即***左/右***标志是否为 Thread），因为***左/右***孩子线索化就意味着它线索化之前为空，那线索化之后它必定连接着该结点的***前驱/后继***；相反若是没有线索化，就看它***左/右***孩子的***右/左***子树最***右/左***边的孩子是否线索化，若是线索化，那么它肯定指向一个结点，而这个结点就是该***右/左***孩

	子的***后继/前驱***，也就是目标结点，故此***右/左***孩子为目标结点的***前驱/后继***。从前驱和后继的寻找中可以看出，寻找遵循的原则依旧是中序线索化、中序遍历的原则，和它们的遍历的顺序息息相关。 同时，将二叉树线性化，也将树的存储更加直观，也可以以访问线性结构的方式去访问树形结构，相比递归访问，这样也将提高访问的效率。