《数据结构》上机报告

2018 年 11 月 11 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">线索二叉树</td></tr>
<tr><td>问题描述</td><td colspan="2">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。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1) 本题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. (p2) 本题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。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1，2</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
<tr><td>数据结构设计</td><td colspan="2">

```
typedef enum {
    Link, Thread
} PointerTag;

typedef struct BiThrNode {
    ElemType data;
    BiThrNode *lchild, *rchild;
    PointerTag ltag, rtag;
}BiThrNode, *BiThrTree;
```

</td></tr>
</table>

功能(函数)说明

1、 先序创建线索二叉树

先序创建线索二叉树时，每输入一个数据，就申请一个空间储存数据。并且把结点的左线索设置为Link，右线索设置为Link。之后线索化时在进行重置。由于是先序创建二叉树，先对左孩子进行递归，再对右儿子进行递归。

```
Status CreateBiThrTree(BiThrTree &PreT) {
    ElemType ch;
    cin >> ch;
    if (ch == '#') {
        PreT = NULL;
    }
    else {
        if (!(PreT = (BiThrNode*)malloc(sizeof(BiThrNode))))
            exit(OVERFLOWED);
        PreT->data = ch;
        PreT->ltag = Link;
        PreT->rtag = Link;
        CreateBiThrTree(PreT->lchild);
        CreateBiThrTree(PreT->rchild);
    }
    return OK;
}
```

2、 先序线索化

先设置一个全局变量储存前一位结点。对二叉树进行先序线索化。如果结点的左节点不存在，将该结点的左线索置为线索，将左子节点连接到上一个节点上。再对结点的前一个节点的右节点进行线索化，如果右节点不存在，将该结点的右线索置为线索，将右节点连接到该结点上。在之后，将前一个结点置为当前结点。接下来进行判断，如果该结点左线索是Link，对左节点进行线索化。对所有的左子树完成线索化后，再对右子树进行线索化。

```
BiThrTree pre;
void PreThreading(BiThrTree &p) {
    if (p) {
        if (!p->lchild) {
            p->LTag = Thread;
            p->lchild = pre;
        }
        if (!pre->rchild) {
            pre->RTag = Thread;
            pre->rchild = p;
        }
        pre = p;
        if (p->LTag == Link)
            PreThreading(p->lchild);
        if (p->RTag == Link)
```

```
            PreThreading(p->rchild);
    }
}
```

3、 设置线索化头结点

设置一个二叉树的头结点。对一棵二叉树加线索时，必须首先申请一个头结点，建立头结点与二叉树的根结点的指向关系，对二叉树线索化后，还需建立最后一个结点与头结点之间的线索。先申请一个结点，将右线索设置为Thread，右子节点连接到原来的二叉树上，左线索设置为Link。接下来对新树进行线索化。

```
Status PreOrderThreading(BiThrTree &Thrt, BiThrTree T) {

    if (!(Thrt = (BiThrTree)malloc(sizeof(BiThrNode))))
        return ERROR;

    Thrt->RTag = Thread;
    Thrt->rchild = Thrt;
    Thrt->LTag = Link;
    if (!T) {
        Thrt->lchild = Thrt;
    }
    else {
        Thrt->lchild = T;
        pre = Thrt;
        PreThreading(T);
        pre->rchild = Thrt;
        pre->RTag = Thread;
        Thrt->rchild = pre;
    }
    return OK;
}
```

4、 先序线索化遍历

由于线索二叉树设置为头结点，所以先定义一个树节点的变量，将该变量定为头结点的左子节点。当该结点没指向根节点时，就继续进行遍历。由于是先序遍历，所以先将根节点输出。如果左子节点为Link，说明该结点有左子节点，那么直接将该结点赋给左子节点，否则，将该结点指向右子节点。因为此时右子节点必然指向当前节点的后继。直到遍历到整个树的根节点。

```
Status PreOrderTraverse_Thr(BiThrTree T) {

    BiThrTree p;
    p = T->lchild;
    while (p != T) {
        cout << p->data;
        if (p->LTag == Link)
            p = p->lchild;
```

```
        else
            p = p->rchild;
    }
    cout << endl;
    return OK;
}
```

5、 线索二叉树的树型输出

线索二叉树的树型输出和原先二叉树有所不同。因为要输出左线索右线索，并且对下一个结点的获取是通过线索进行获取。所以先进行判断，如果当前结点的右线索是 Link，说明该结点的右子节点存在，那么进入下一层递归。使用一个整型变量储存层值。直到该子节点的右子节点不存在，那么就输出 level 个五个空格，输出数据。所有的右子树完成之后，左子树再进行树的打印。

```
void TreePrint(BiThrTree T, int level)
{
    int i;
    if (!T) {
        return;
    }
    if (T->RTag == Link) {
        TreePrint(T->rchild, level + 1);
    }
    for (i = 0; i<level; i++) {
        cout << "     ";
    }
    cout << T->data << T->LTag << T->RTag << endl;
    if (T->LTag == Link)
        TreePrint(T->lchild, level + 1);
}
```

6、 中序线索化

先设置一个全局变量储存前一位结点。对二叉树进行中序线索化。由于是中序线索化，首先对左子节点进行线索化，一直到左子树的最后一个结点。如果当前节点的左子节点不存在，将左子节点的线索设置为 Thread，左子节点设置为前一个结点。再次，之前的结点的右节点如果不存在，将右线索设置为 Thread，将右子节点设置为当前结点。最后再对当前节点的右子树进行线索化。

```
void InThreading(BiThrTree p, BiThrTree &pre) {
    if (p) {
        InThreading(p->lchild, pre);
        if (!p->lchild) {
            p->ltag = Thread;
            p->lchild = pre;
        }
        if (!pre->rchild) {
```

```
            pre->rtag = Thread;
            pre->rchild = p;
        }
        pre = p;
        InThreading(p->rchild, pre);
    }
}
```

7、 中序线索化设置头结点

设置一个二叉树的头结点。对一棵二叉树加线索时，必须首先申请一个头结点，建立头结点与二叉树的根结点的指向关系，对二叉树线索化后，还需建立最后一个结点与头结点之间的线索。先申请一个结点，将右线索设置为Thread，右子节点连接到原来的二叉树上，左线索设置为Link。接下来对新树进行线索化。

```
Status InOrderThreading(BiThrTree &T, BiThrTree PreT) {
    BiThrTree pre;
    if (!(T = (BiThrTree)malloc(sizeof(BiThrNode))))
        exit(OVERFLOWED);
    T->ltag = Link;
    T->rtag = Thread;
    T->rchild = T;
    if (!PreT)
        T->lchild = T;
    else {
        T->lchild = PreT;
        pre = T;
        InThreading(PreT, pre);
        pre->rchild = T;
        pre->rtag = Thread;
        T->rchild = pre;
    }
    return OK;
}
```

8、 中序线索化遍历

由于线索二叉树设置为头结点，所以先定义一个树节点的变量，将该变量定为头结点的左子节点。当该结点没指向根节点时，就继续进行遍历。由于是中序遍历，所以先将左子节点输出。如果右线索为Thread，说明该结点有左子节点，那么直接将该结点赋给右子节点。因为此时右子节点必然指向当前节点的后继。直到遍历到整个树的根节点。

```
Status InOrderTraverse_Thr(BiThrTree T) {
    BiThrTree p = T->lchild;
    while (p != T) {
        while (p->ltag == Link)
            p = p->lchild;
```

```
        cout << p->data;
        while (p->rtag == Thread && p->rchild != T) {
            p = p->rchild;
            cout << p->data;
        }
        p = p->rchild;
    }
    return OK;
}
```

9、 寻找结点的前驱和后继

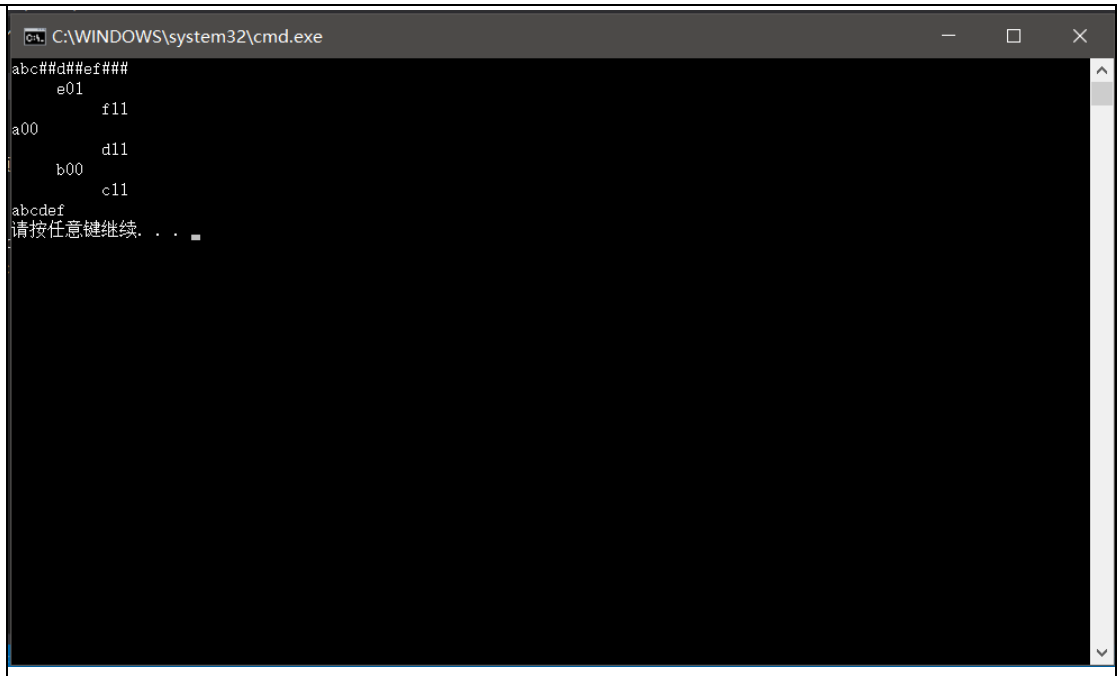
先设置三个 bool 值，判断是否找到要寻找的结点；判断前驱是否存在；判断后继是否存在。中序遍历结点，如果找到该结点，就将 is_found 的值定为 true。接着找到该结点的前驱和后继。再设置两个结点树节点 prev，succ。在遍历的时候每一次都将 prev 设置为当前结点的前驱结点。找到目标节点后，首先将遍历的结果输出。在进行前驱和后继结点的输出，如果某一个结点不存在，输出 NULL。

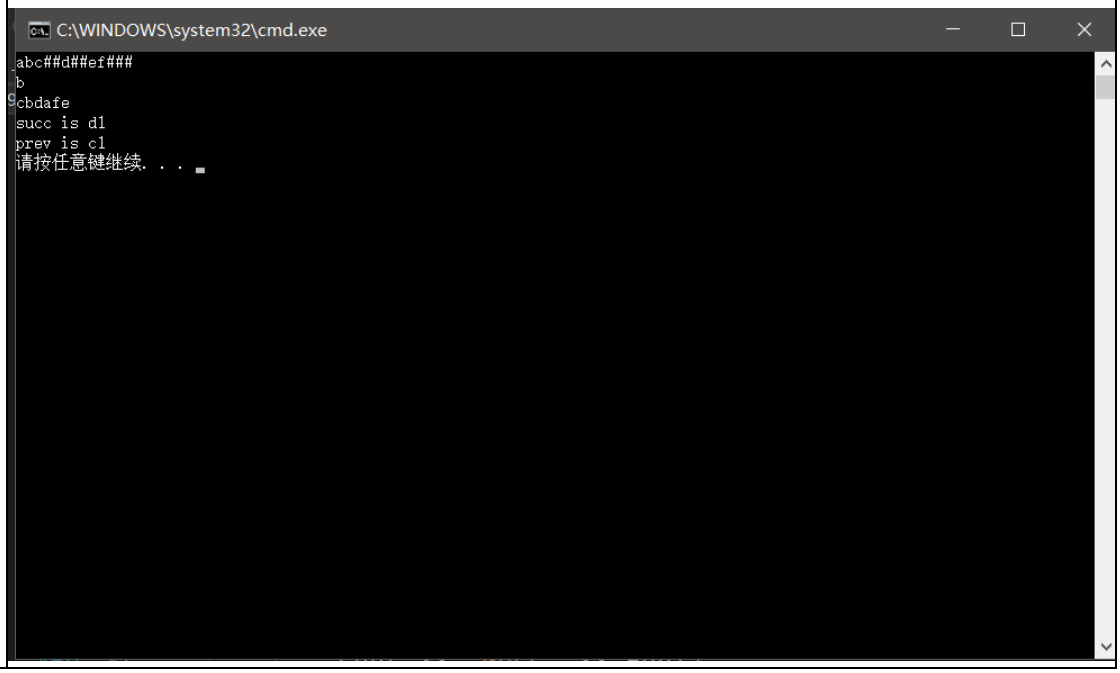
```
Status InOrderFind(BiThrTree T,  char target) {
BiThrTree p = T->lchild;
BiThrTree prev = T;
BiThrTree succ = NULL;
bool is_found=false,prev_exist = false, succ_exist = false;
while (p != T) {
    while (p->ltag == Link)
        p = p->lchild;
    if (prev_exist == false) {
        if (p->data == target) {
            prev_exist = true;
            is_found = true;
        }
        else {
            prev = p;
        }
    }
    else {
        if (succ_exist == false) {
            succ = p;
            succ_exist = true;
        }
    }
    while (p->rtag == Thread && p->rchild != T) {
        p = p->rchild;
        if (prev_exist == false) {
            if (p->data == target) {
                prev_exist = true;
```

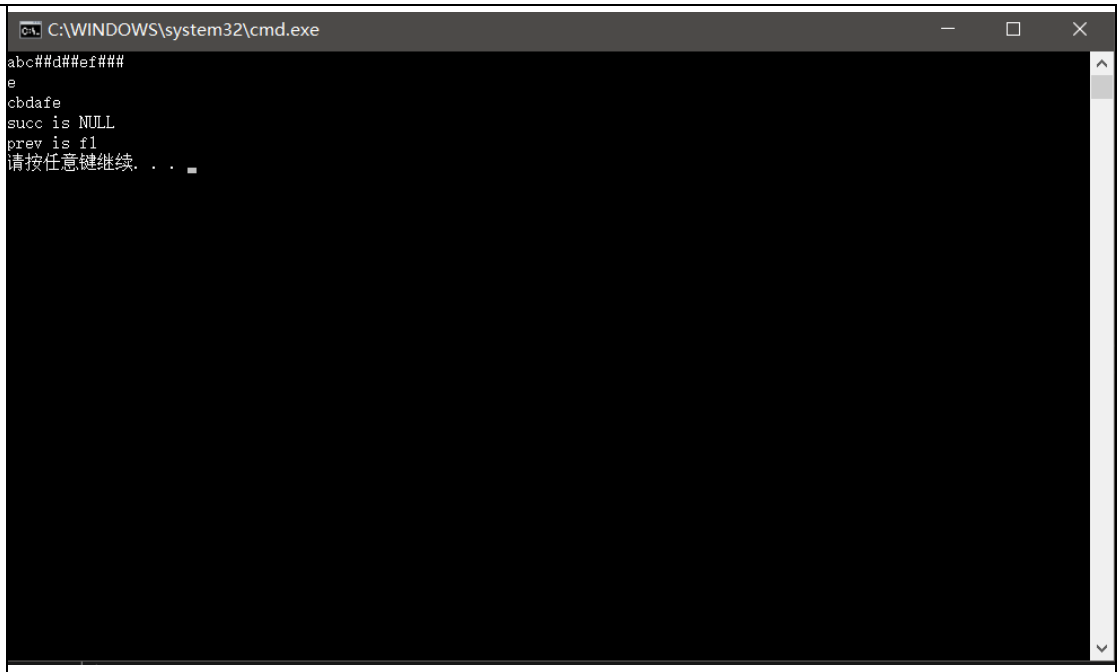
```
                    is_found = true;
                }
                else {
                    prev = p;
                }
            }
            else {
                if (succ_exist == false) {
                    succ = p;
                    succ_exist = true;
                }
            }
        }
        p = p->rchild;
    }
    if (is_found == true) {
        InOrderTraverse_Thr(T);
        cout << endl;
        cout << "succ is ";
        if (succ != NULL) {
            cout << succ->data << succ->rtag << endl;
        }
        else {
            cout << "NULL" << endl;
        }
        cout << "prev is ";
        if (prev != T) {
            cout << prev->data << prev->ltag << endl;
        }
        else {
            cout << "NULL" << endl;
        }
    }
    else {
        InOrderTraverse_Thr(T);
        cout << endl;
        cout << "Not found"<<endl;
    }
    return OK;
}
```

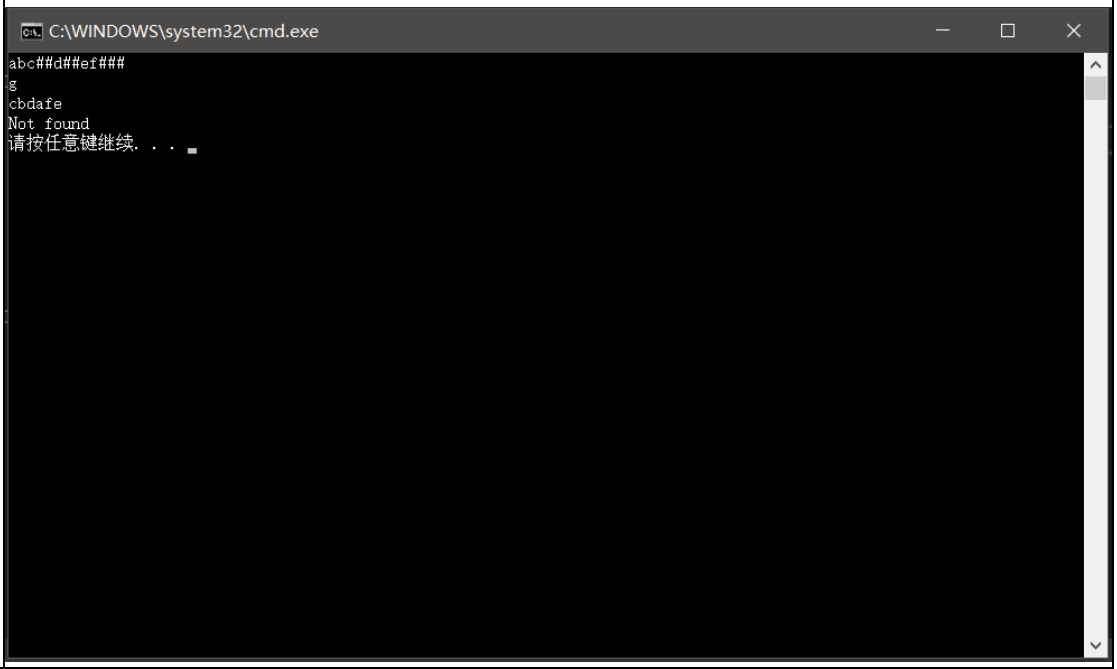
开发环境	Visual studio 2017

调试分析


C:\WINDOWS\system32\cmd.exe
abc##d##ef###
e01
f11
a00
d11
b00
c11
abcdef
请按任意键继续. . .


C:\WINDOWS\system32\cmd.exe
abc##d##ef###
b
cbdafe
succ is d1
prev is c1
请按任意键继续. . .





心得体会

1、 线索化设置头结点

对一棵二叉树加线索时，必须首先申请一个头结点，建立头结点与二叉树的根结点的指向关系，对二叉树线索化后，还需建立最后一个结点与头结点之间的线索。设置头结点有诸多好处，对后续的每一个结点进行遍历，寻找前驱后继时无需对根节点进行特殊操作。

2、 树型输出

树型输出的时候，要注意线索二叉树搜索下一个结点的方法是通过线索。所以不能直接去索引左右结点。必须先对左右线索进行判断，如果是 Link，说明下面的结点存在，直接去索引，否则不进行输出。

3、 题目问题

第二题中题目描述若不存在，输出一行 Not found，结束。但最后的 OJ 判断时需要加上遍历结果的输出。