阅读：4,036　　作者：解学武

双向线索二叉树的建立及C语言实现

通过前一节对线索二叉树的学习，其中，在遍历使用中序序列创建的线索二叉树时，对于其中的每个结点，即使没有线索的帮助下，也可以通过中序遍历的规律找到直接前趋和直接后继结点的位置。

也就是说，建立的线索二叉链表可以从两个方向对结点进行中序遍历。通过前一节的学习，线索二叉链表可以从第一个结点往后逐个遍历。但是起初由于没有记录中序序列中最后一个结点的位置，所以不能实现从最后一个结点往前逐个遍历。

双向线索链表的作用就是可以让线索二叉树从两个方向实现遍历。

双向线索二叉树的实现过程

在线索二叉树的基础上，额外添加一个结点。此结点的作用类似于链表中的头指针，数据域不起作用，只利用两个指针域（由于都是指针，标志域都为 0 ）。

左指针域指向二叉树的树根，确保可以正方向对二叉树进行遍历；同时，右指针指向线索二叉树形成的线性序列中的最后一个结点。

这样，二叉树中的线索链表就变成了双向线索链表，既可以从第一个结点通过不断地找后继结点进行遍历，也可以从最后一个结点通过不断找前趋结点进行遍历。

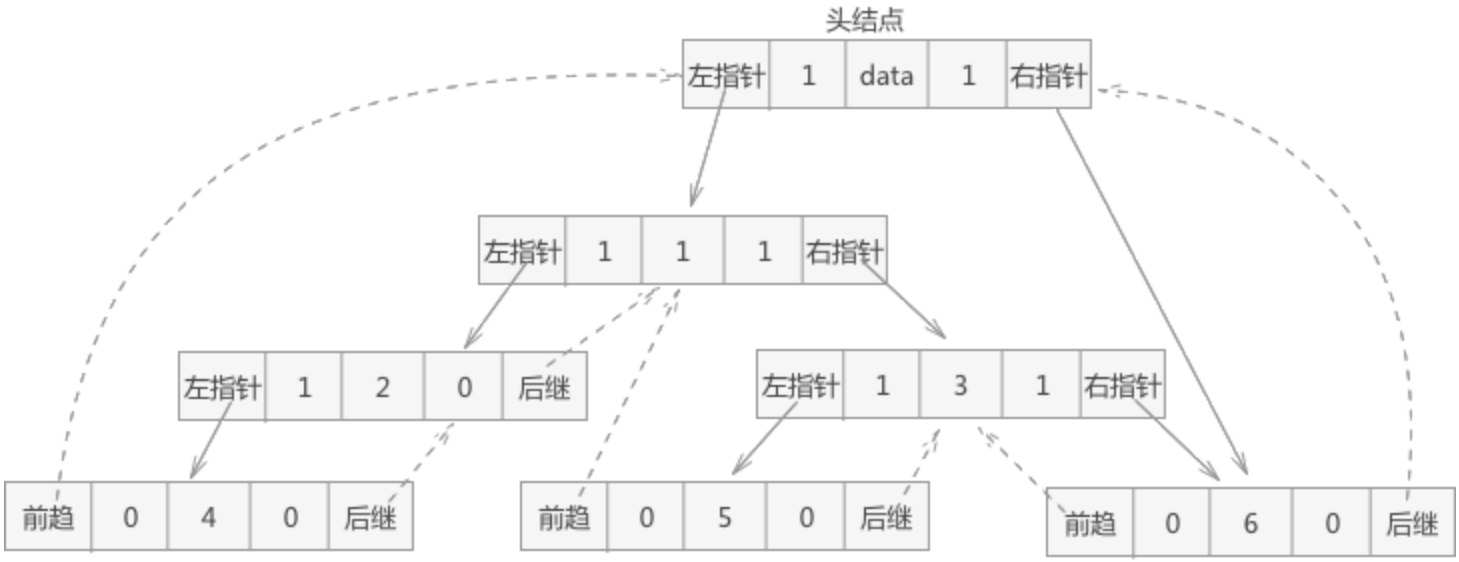

图1 双向线索二叉链表

代码实现：

```
//建立双向线索链表
void InOrderThread_Head(BiThrTree *h, BiThrTree t)
{
    //初始化头结点
    (*h) = (BiThrTree)malloc(sizeof(BiThrNode));
    if((*h) == NULL){
        printf("申请内存失败");
        return ;
    }
    (*h)->rchild = *h;
    (*h)->Rtag = Link;
    //如果树本身是空树
    if(!t){
        (*h)->lchild = *h;
        (*h)->Ltag = Link;
    }
    else{
        pre = *h;//pre指向头结点
        (*h)->lchild = t;//头结点左孩子设为树根结点
        (*h)->Ltag = Link;
        InThreading(t);//线索化二叉树，pre结点作为全局变量，线索化结束后，pre结点指向中序序列中最后一
        pre->rchild = *h;
        pre->Rtag = Thread;
        (*h)->rchild = pre;
    }
}
```

双向线索二叉树的遍历

双向线索二叉树遍历时，如果正向遍历，就从树的根结点开始。整个遍历过程结束的标志是：当从头结点出发，遍历回头结点时，表示遍历结束。

```
//中序正向遍历双向线索二叉树
void InOrderThraverse_Thr(BiThrTree h)
{
    BiThrTree p;
    p = h->lchild;            //p指向根结点
    while(p != h)
    {
        while(p->Ltag == Link)    //当ltag = 0时循环到中序序列的第一个结点
        {
            p = p->lchild;
        }
        printf("%c ", p->data);  //显示结点数据，可以更改为其他对结点的操作
        while(p->Rtag == Thread && p->rchild != h)
        {
```

```
            p = p->rchild;
            printf("%c ", p->data);
        }

        p = p->rchild;            //p进入其右子树
    }
}
```

逆向遍历线索二叉树的过程即从头结点的右指针指向的结点出发，逐个寻找直接前趋结点，结束标志同正向遍历一样：

```
//中序逆方向遍历线索二叉树
void InOrderThraverse_Thr(BiThrTree h){
    BiThrTree p;
    p=h->rchild;
    while (p!=h) {
        while (p->Rtag==Link) {
            p=p->rchild;
        }
        printf("%c",p->data);
        //如果lchild为线索，直接使用，输出
        while (p->Ltag==Thread && p->lchild !=h) {
            p=p->lchild;
            printf("%c",p->data);
        }
        p=p->lchild;
    }
}
```

完整代码实现

```
#include <stdio.h>
#include <stdlib.h>
#define TElemType char//宏定义，结点中数据域的类型
//枚举，Link为0，Thread为1
typedef enum {
    Link,
    Thread
}PointerTag;
//结点结构构造
typedef struct BiThrNode{
    TElemType data;//数据域
    struct BiThrNode* lchild,*rchild;//左孩子，右孩子指针域
    PointerTag Ltag,Rtag;//标志域，枚举类型
```

```
}BiThrNode,*BiThrTree;

BiThrTree pre=NULL;

//采用前序初始化二叉树
//中序和后序只需改变赋值语句的位置即可
void CreateTree(BiThrTree * tree){
    char data;
    scanf("%c",&data);
    if (data!='#'){
        if (!((*tree)=(BiThrNode*)malloc(sizeof(BiThrNode)))){
            printf("申请结点空间失败");
            return;
        }else{
            (*tree)->data=data;//采用前序遍历方式初始化二叉树
            CreateTree(&((*tree)->lchild));//初始化左子树
            CreateTree(&((*tree)->rchild));//初始化右子树
        }
    }else{
        *tree=NULL;
    }
}
//中序对二叉树进行线索化
void InThreading(BiThrTree p){
    //如果当前结点存在
    if (p) {
        InThreading(p->lchild);//递归当前结点的左子树，进行线索化
        //如果当前结点没有左孩子，左标志位设为1，左指针域指向上一结点 pre
        if (!p->lchild) {
            p->Ltag=Thread;
            p->lchild=pre;
        }
        //如果 pre 没有右孩子，右标志位设为 1，右指针域指向当前结点。
        if (pre&&!pre->rchild) {
            pre->Rtag=Thread;
            pre->rchild=p;
        }
        pre=p;//pre指向当前结点
        InThreading(p->rchild);//递归右子树进行线索化
    }
}
//建立双向线索链表
void InOrderThread_Head(BiThrTree *h, BiThrTree t)
{
    //初始化头结点
    (*h) = (BiThrTree)malloc(sizeof(BiThrNode));
    if((*h) == NULL){
```

```
61.         printf("申请内存失败");
62.         return ;
63.     }
64.     (*h)->rchild = *h;
65.     (*h)->Rtag = Link;
66.     //如果树本身是空树
67.     if(!t){
68.         (*h)->lchild = *h;
69.         (*h)->Ltag = Link;
70.     }
71.     else{
72.         pre = *h;//pre指向头结点
73.         (*h)->lchild = t;//头结点左孩子设为树根结点
74.         (*h)->Ltag = Link;
75.         InThreading(t);//线索化二叉树，pre结点作为全局变量，线索化结束后，pre结点指向中序序列中最后一
76.         pre->rchild = *h;
77.         pre->Rtag = Thread;
78.         (*h)->rchild = pre;
79.     }
80. }
81. //中序正向遍历双向线索二叉树
82. void InOrderThraverse_Thr(BiThrTree h)
83. {
84.     BiThrTree p;
85.     p = h->lchild;            //p指向根结点
86.     while(p != h)
87.     {
88.         while(p->Ltag == Link)    //当ltag = 0时循环到中序序列的第一个结点
89.         {
90.             p = p->lchild;
91.         }
92.         printf("%c ", p->data);  //显示结点数据，可以更改为其他对结点的操作
93.         while(p->Rtag == Thread && p->rchild != h)
94.         {
95.             p = p->rchild;
96.             printf("%c ", p->data);
97.         }
98.
99.         p = p->rchild;            //p进入其右子树
100.    }
101. }
102. int main() {
103.     BiThrTree t;
104.     BiThrTree h;
105.     printf("输入前序二叉树:\n");
106.     CreateTree(&t);
107.     InOrderThread_Head(&h, t);
```

```
108.      printf("输出中序序列:\n");
109.      InOrderThraverse_Thr(h);
110.      return 0;
111.  }
```

运行结果：

```
输入前序二叉树:
124###35##6##
输出中序序列:
4 2 1 5 3 6
```

> 程序中只调用了正向遍历线索二叉树的代码，如果逆向遍历，直接替换逆向遍历的函数代码到程序中即可。