7.8 线索二叉树

7.8.1 线索二叉树的概念

回顾：

- 对于具有n个节点的二叉树，采用二叉链存储结构时，每个节点有两个指针域，总共有$2n$个指针域。
- 其中只有$n-1$个节点被有效指针所指向，即有$n-1$个非空指针域。
- 所以共有$2n-(n-1) = n+1$个空链域。

相关概念：

- 采用某种方法遍历二叉树的结果是一个节点的线性序列。
- 修改空链域改为存放指向节点的前趋和后继节点的地址。
- 这样的指向该线性序列中的“前趋”和“后继”的指针，称作线索（thread）。
- 创建线索的过程称为线索化。
- 线索化的二叉树称为线索二叉树。
- 显然线索二叉树与采用的遍历方法相关，有先序线索二叉树、中序线索二叉树和后序线索二叉树。
- 线索二叉树的目的是提高该遍历过程的效率。

在节点的存储结构上增加两个标志位来区分这两种情况：

左标志ltag
- 0　表示lchild指向左孩子节点
- 1　表示lchild指向前趋节点，即左线索

右标志rtag
- 0　表示rchild指向右孩子节点
- 1　表示rchild指向后继节点，　即右线索

这样，每个节点的存储结构如下：

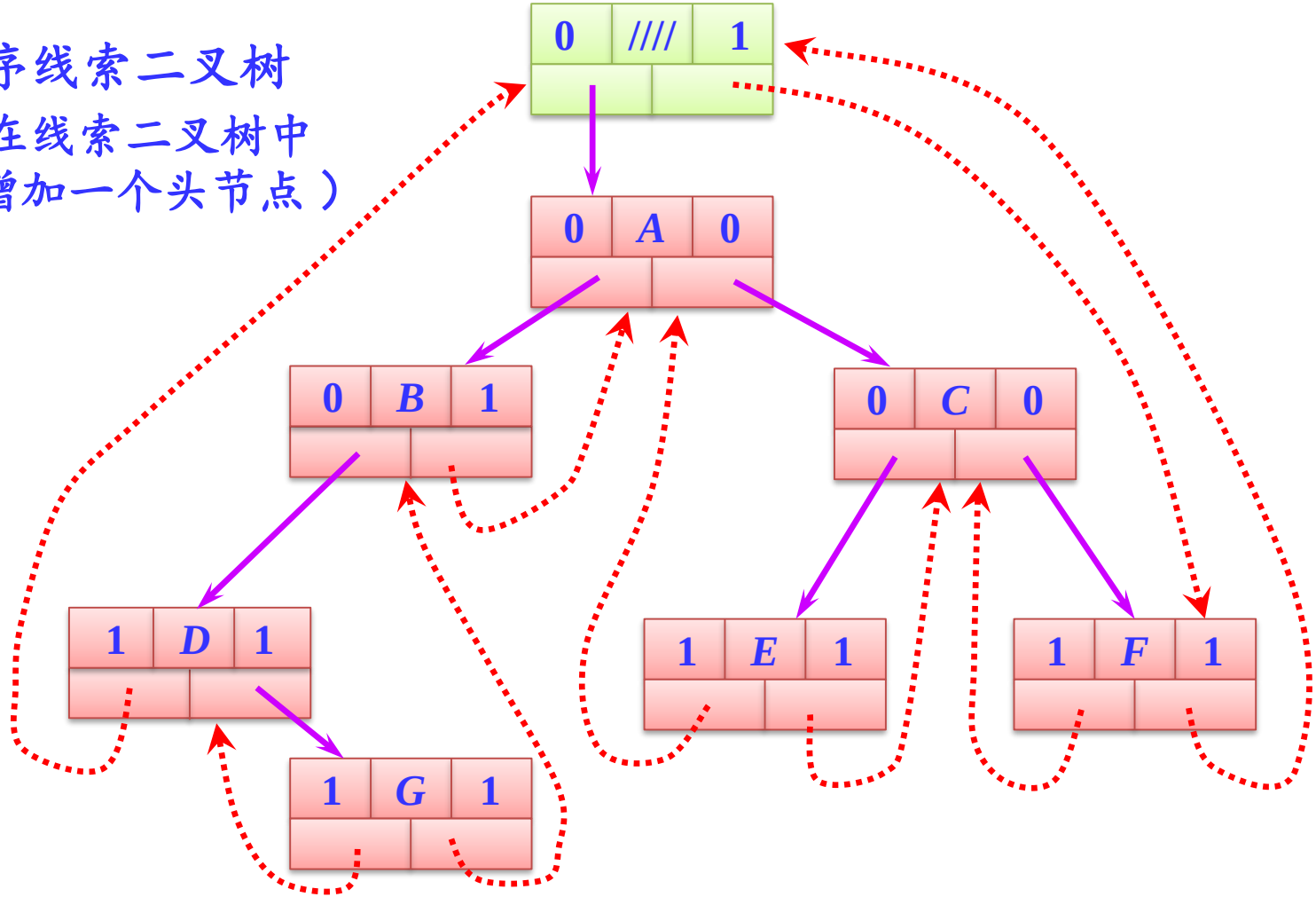
ltag	lchild	data	rchild	rtag

为了方便算法设计，在线索二叉树中再增加一个头节点。

线索化二叉树中节点的类型定义如下：

```
typedef struct node
{     ElemType data;              //节点数据域
      int ltag,rtag;              //增加的线索标记
      struct node *lchild;        //左孩子或线索指针
      struct node *rchild;        //右孩子或线索指针
} TBTNode;                        //线索树节点类型定义
```

中序线索二叉树
（在线索二叉树中
再增加一个头节点）



7.8.2 线索化二叉树

建立某种次序的线索二叉树过程：

- 以该遍历方法遍历一棵二叉树。
- 在遍历的过程中，检查当前访问节点的左、右指针域是否为空：
  - 如果左指针域为空，将它改为指向前趋节点的线索；
  - 如果右指针域为空，将它改为指向后继节点的线索。

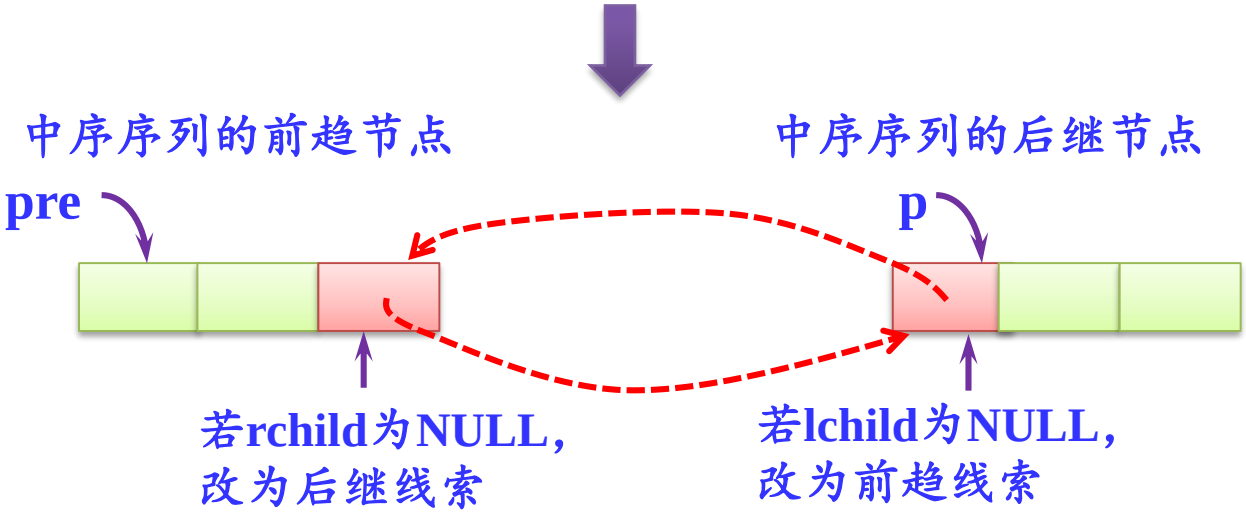
以中序线索二叉树为例，讨论建立线索二叉树的算法。

建立中序线索二叉树的算法

- **CreatThread(b)算法：对以二叉链存储的二叉树b进行中序线索化，并返回线索化后头节点的指针root。**
- **Thread(p)算法：对以*p为根节点的二叉树子树的中序线索化。**

在中序遍历中：

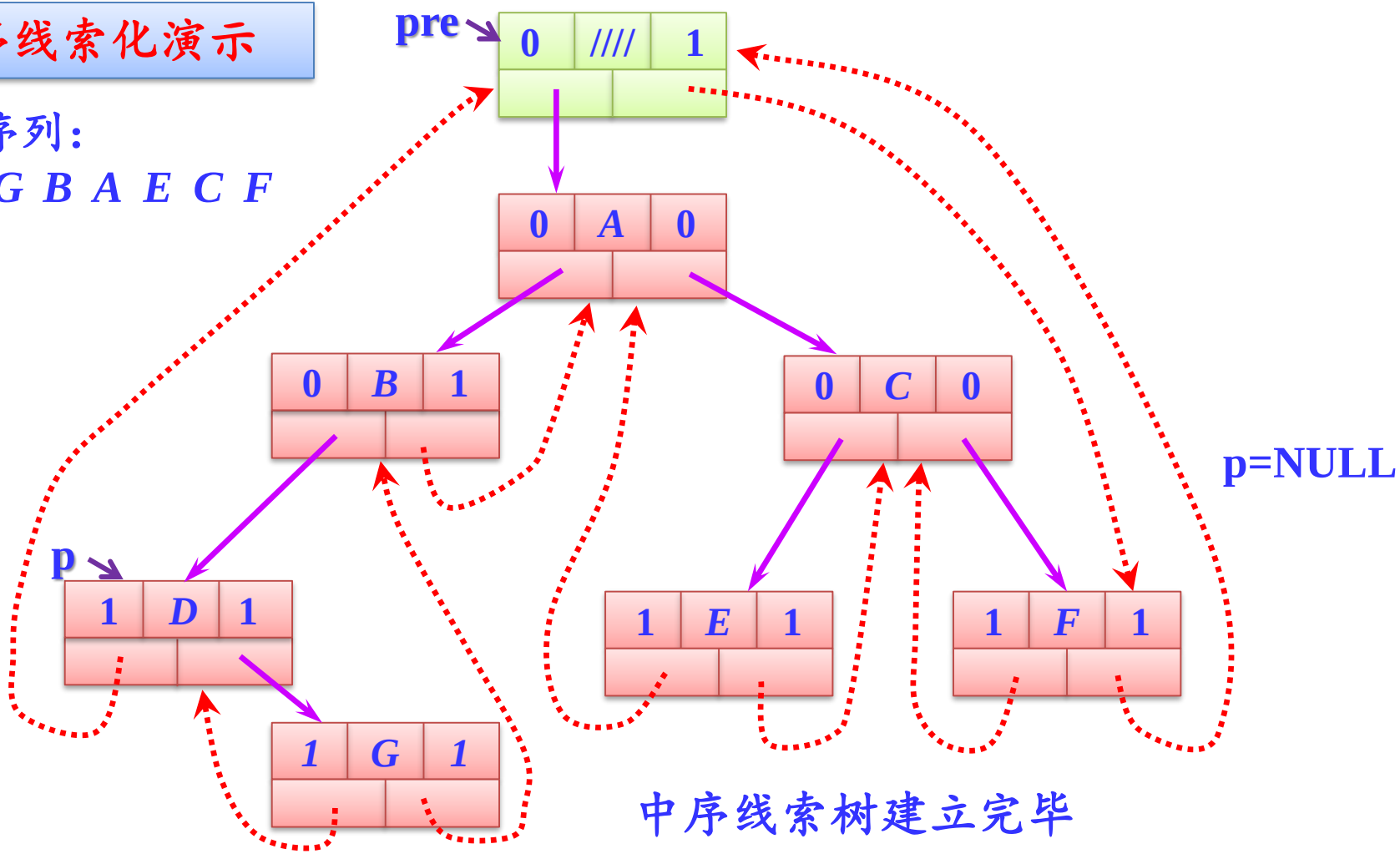
- p总是指向当前线索化的节点。
- pre作为全局变量，指向刚刚访问过的节点。
- *pre是*p的中序前趋节点，*p是*pre的中序后继节点。



中序线索化演示

中序序列：

D G B A E C F



```
TBTNode *pre;                                   //全局变量
TBTNode *CreatThread(TBTNode *b)                //中序线索化二叉树
{     TBTNode *root;
      root=(TBTNode *)malloc(sizeof(TBTNode));  //创建头节点
      root->ltag=0; root->rtag=1;  root->rchild=b;
      if (b==NULL) root->lchild=root;           //空二叉树
      else
      {       root->lchild=b;
              pre=root;                         //pre是*p的前趋节点,供加线索用
              Thread(b);                        //中序遍历线索化二叉树
              pre->rchild=root;                 //最后处理,加入指向头节点的线索
              pre->rtag=1;
              root->rchild=pre;                 //头节点右线索化
      }
      return root;
}
```

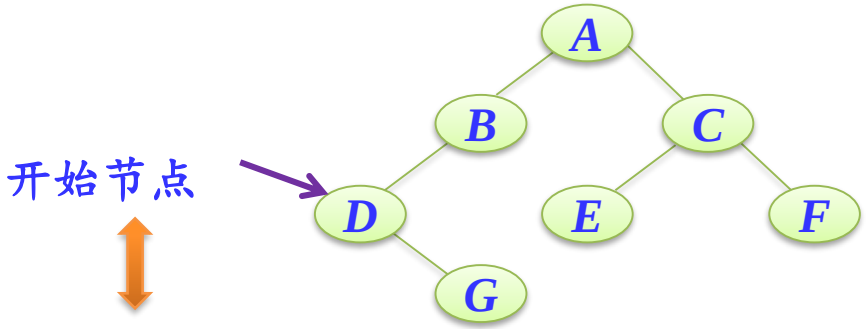
中序遍历递归算法

```
void  Thread(TBTNode *&p)                    //对二叉树b进行中序线索化
{   if (p!=NULL)
    {
      Thread(p->lchild);                     //左子树线索化
      if (p->lchild==NULL)                   //前趋线索化
      {    p->lchild=pre; p->ltag=1;  }      //建立当前节点的前趋线索
      else  p->ltag=0;
      if  (pre->rchild==NULL)                //后继线索化
      {    pre->rchild=p;pre->rtag=1;}       //建立前趋节点的后继线索
      else  pre->rtag=0;
      pre=p;
      Thread(p->rchild);                     //递归调用右子树线索化
    }
}
```

7.8.3 遍历线索化二叉树

遍历某种次序的线索二叉树，就是从该次序下的开始节点出发，反复找到该节点在该次序下的后继节点，直到头节点。

以中序线索二叉树为例，开始节点是根节点的最左下节点。



在中序线索二叉树中，开始节点的左指针域为线索，即ltag=1

找开始节点的算法：

```
TBTNode *p=tb->lchild;
//p指向根节点
while (p->ltag==0)
    p=p->lchild;
```

在中序线索二叉树中中序遍历的过程：

```
p指向根节点；
while p ≠root时循环
{
        找开始节点*p；
        访问*p节点；
        while (*p节点有右线索)
                一直访问下去；
        p转向右孩子节点；
}
```

中序线索二叉树的中序遍历示例演示

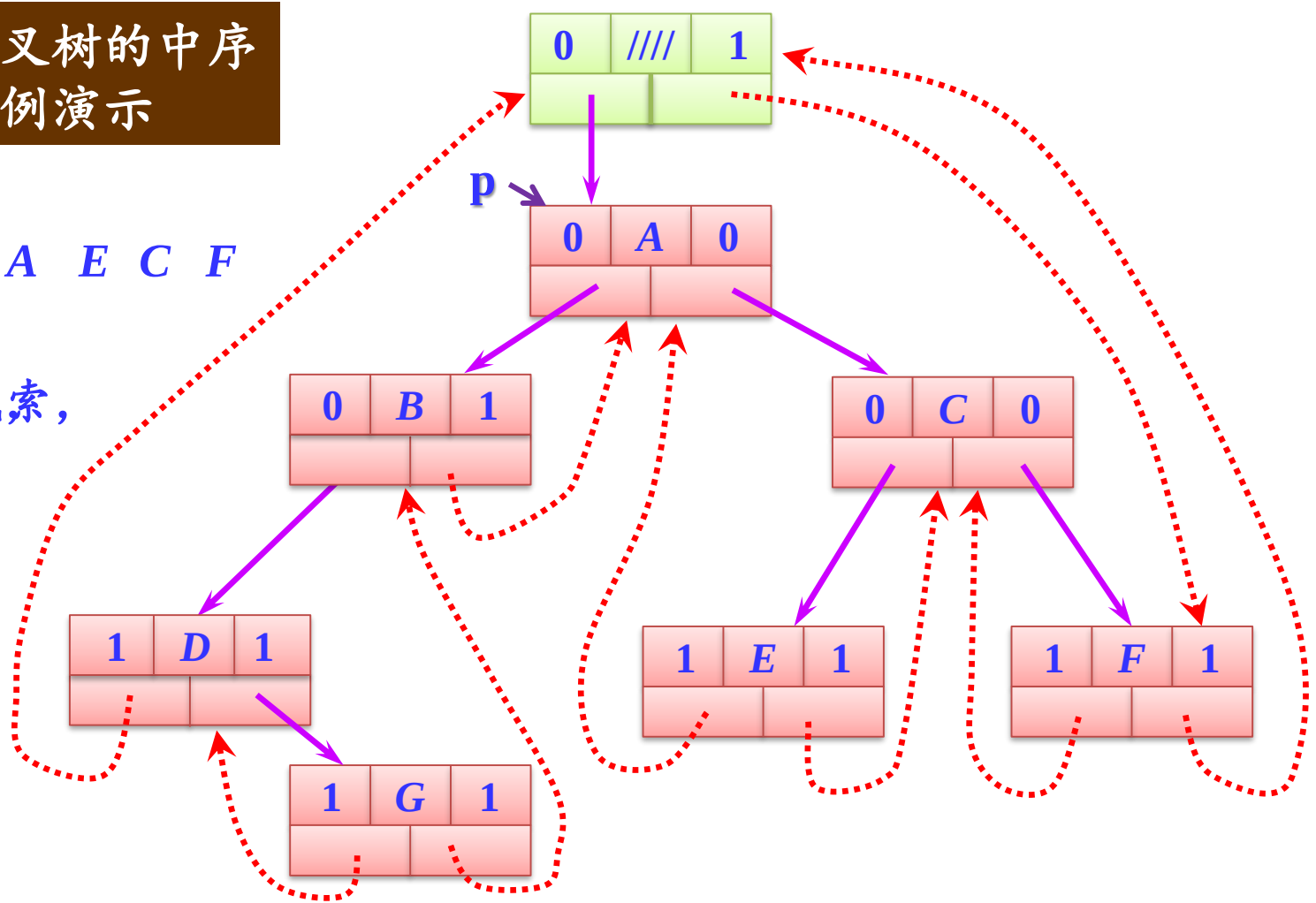
中序序列：

D G B A E C F

操作：

找指向开始结点，

沿着线索访问



```
void ThInOrder(TBTNode *tb)
{       TBTNode *p=tb->lchild;                          //p指向根节点
        while (p!=tb)
        {
              while (p->ltag==0)   p=p->lchild;         //找开始节点
              printf("%c",p->data);                     //访问开始节点
              while (p->rtag==1 && p->rchild!=tb)
              {     p=p->rchild;
                    printf("%c",p->data);
              }
              p=p->rchild;
        }
}
```

如果是线索就一直访问下去

优点：中序遍历算法既没有递归也没有用栈，空间效率得到提高。

思考题：

❶ 中序线索二叉树可以提高先序遍历的效率吗？

❷ 中序线索二叉树可以提高后序遍历的效率吗？

——本讲完——