4.12线索二叉树

存储
- 顺序存储
- 链式存储
- 线索二叉树

如何充分利用起来n+1个空指针域?

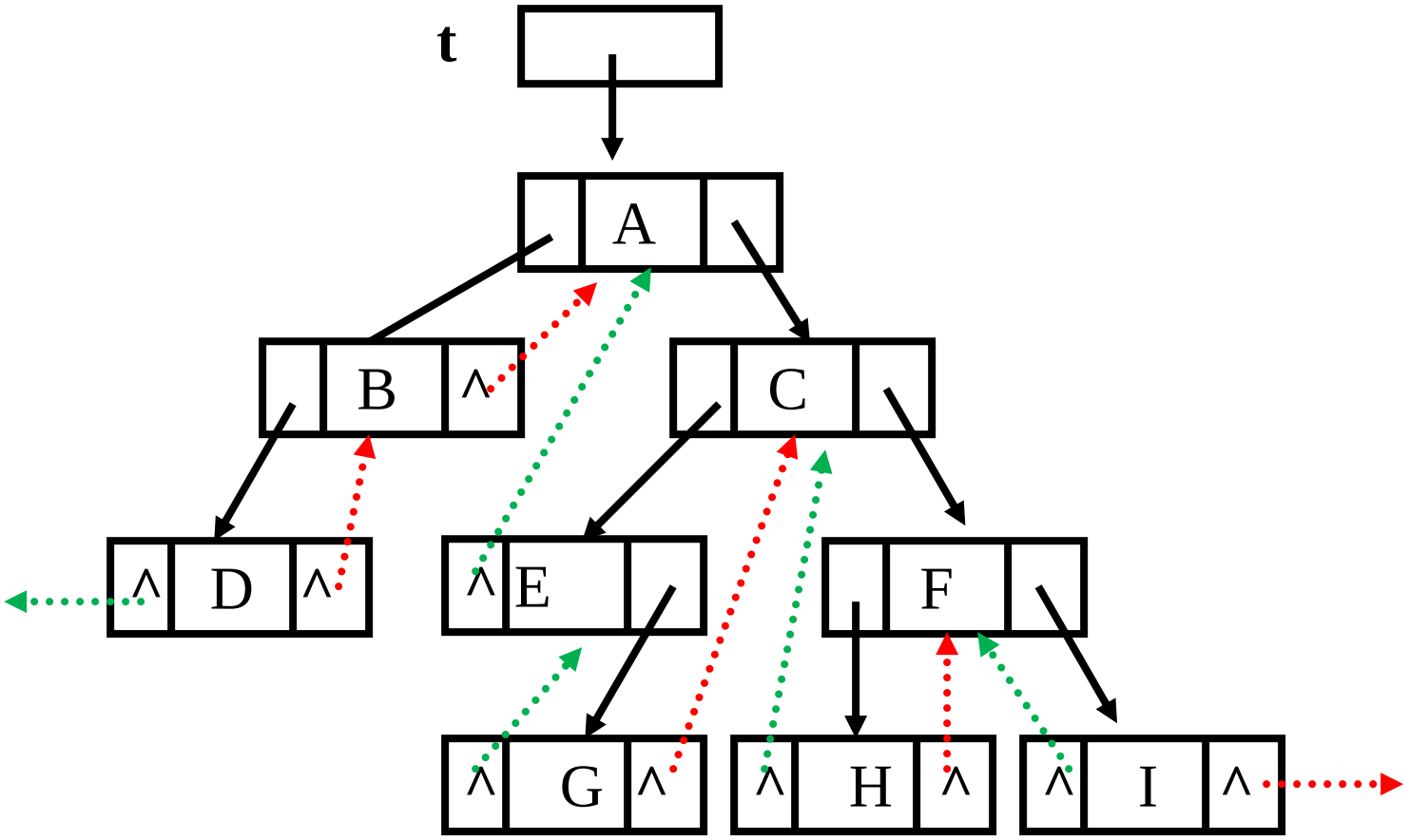
leftchild	lthread	data	rthread	rightchild

$$lthread = \begin{cases} 0 & \text{leftchild是指针，指向结点的左孩子} \\ 1 & \text{leftchild是线索，指向结点的前驱结点} \end{cases}$$

$$rthread = \begin{cases} 0 & \text{rightchild是指针，指向结点的右孩子} \\ 1 & \text{rightchild是线索，指向结点的后继结点} \end{cases}$$

线索二叉树

对称穿线树
先根穿线树
后根穿线树

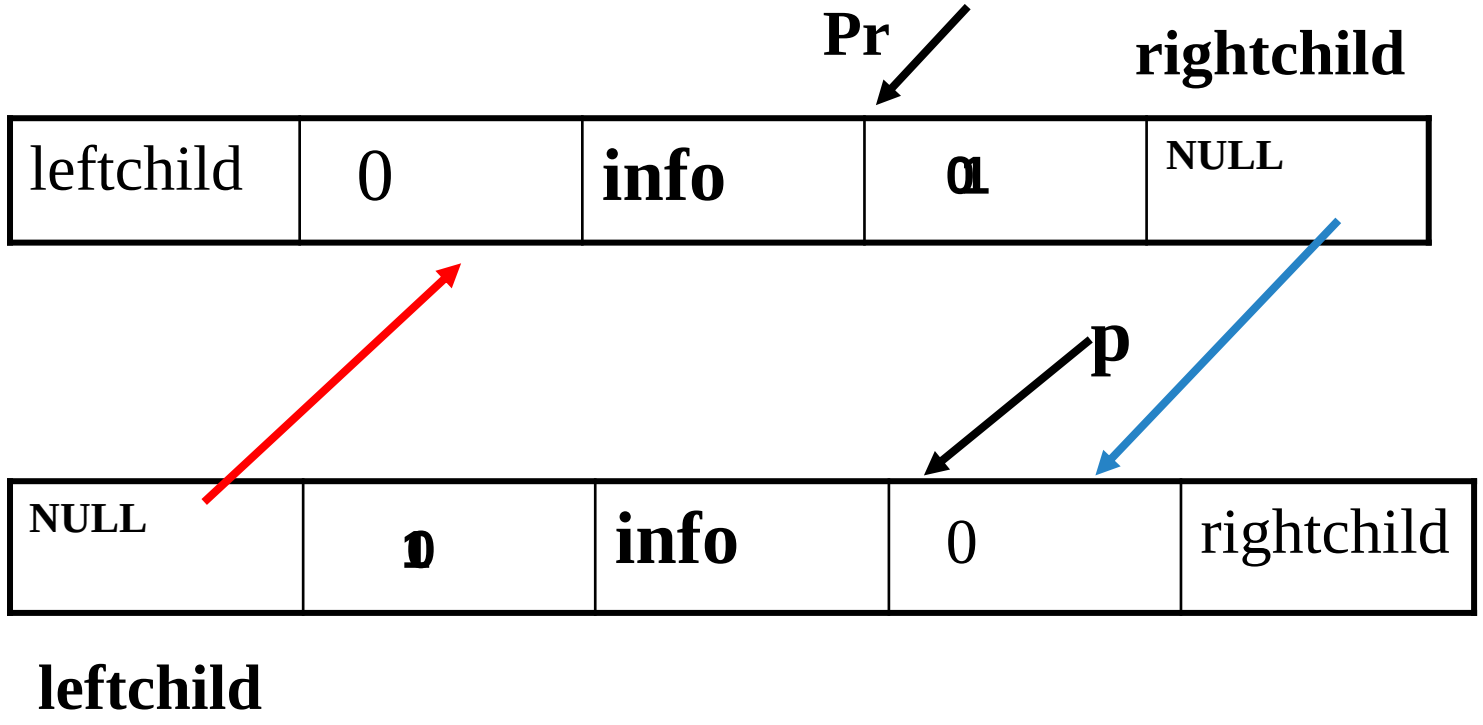


对称序列：D B A E G C H F I

如何建立中序线索树

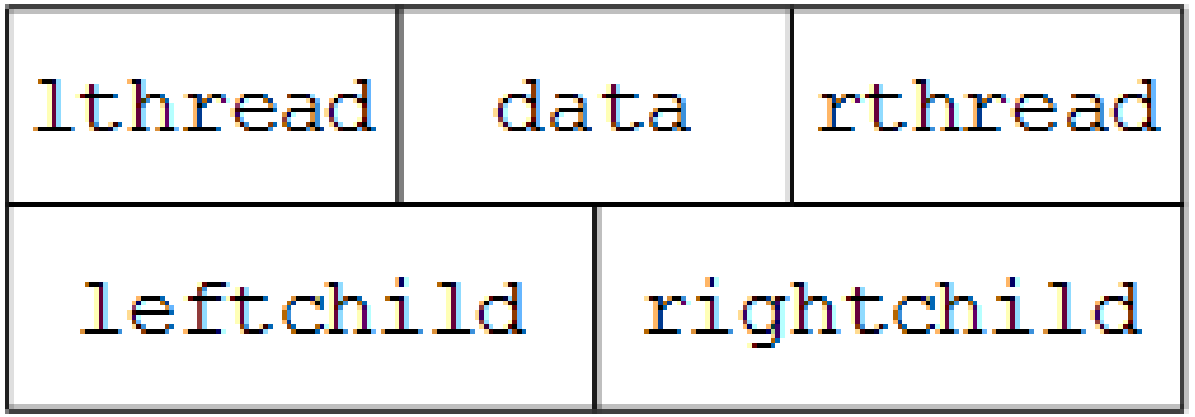
在中序遍历过程中修改结点的左右指针域，以保存当前访问结点的前驱和后继信息

遍历过程中，指针p指向当前正在访问的结点，附设指针pr，并始终保持指向p所指结点的对称前驱

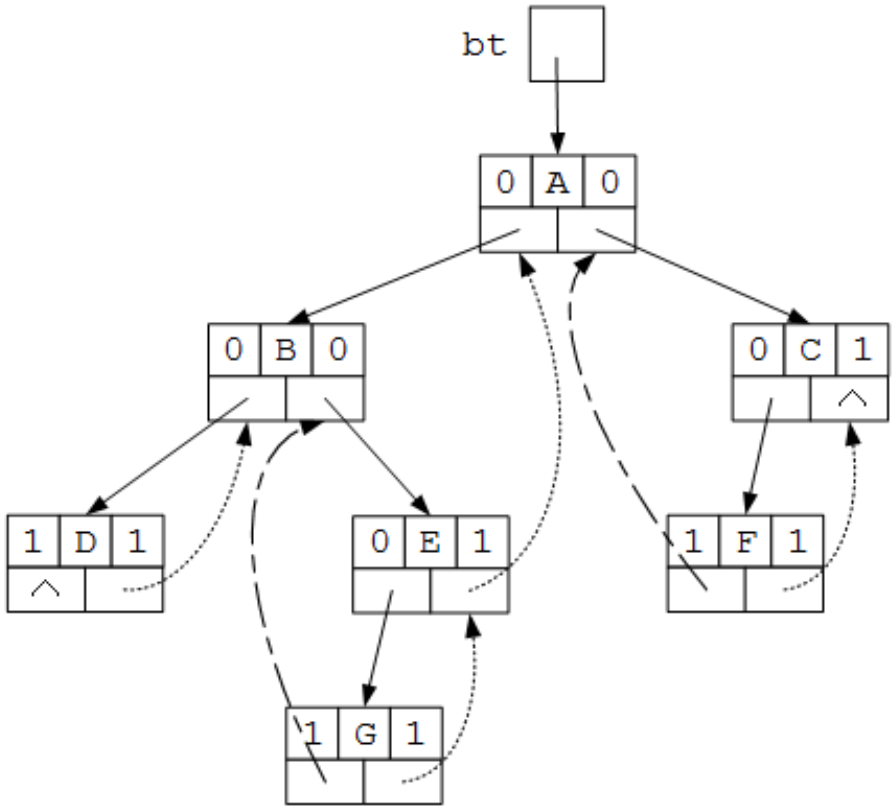


线索二叉树的存储表示

```
1   typedef char DataType;
2   typedef struct BTreeNode
3   {
4           DataType data;
5           struct BTreeNode *leftchild;
6           struct BTreeNode *rightchild;
7           int lthread;
8           int rthread;
9   }BinTreeNode;
10  typedef BinTreeNode *BinTree;
```



```
1  void Create_InorderThread(BinTree bt) //建立中序穿线树
2  {
3          LinkStack st = SetNullStack_Link();
4          BinTreeNode *p, *pr, *q;
5          if (bt == NULL) return;
6          p = bt;
7          pr = NULL;
8          do{
9                  while (p != NULL)
10                 {
11                         Push_link(st, p);
12                         p = p->leftchild;
13                 }
14                 p = Top_link(st);
15                 Pop_link(st);
16
17
18                                                          ad
19
20
21
22
23                 pr = p;
24                 p = p->rightchild;
25         } while (!IsNullStack_link(st) || p != NULL);
```



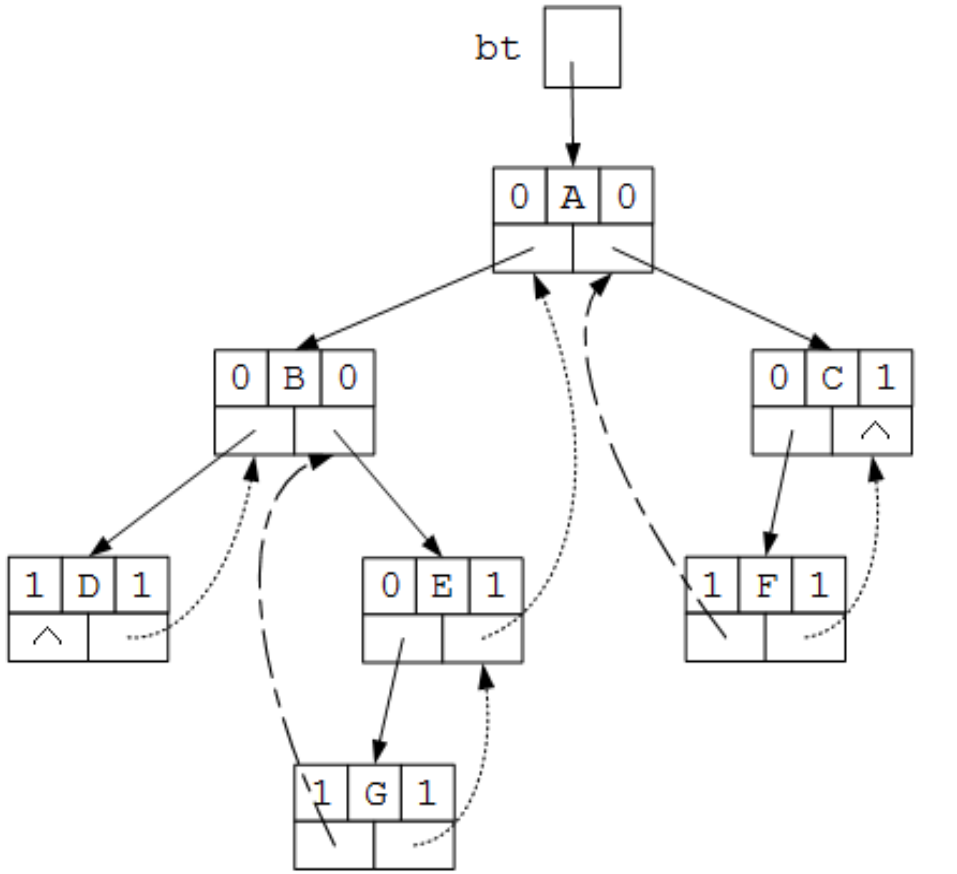
visit

算法4-19

建立中序穿线树

算法4-19

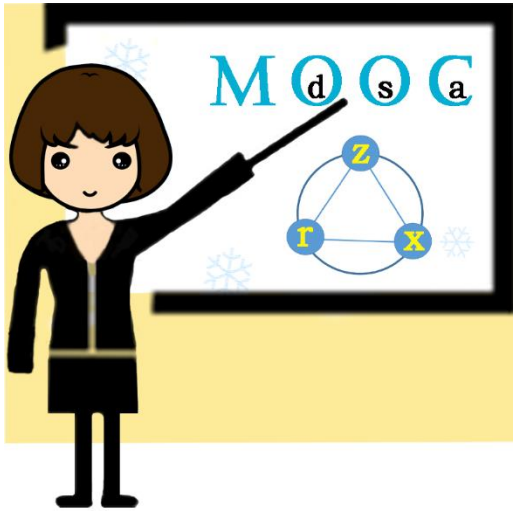
```
1   void Create_InorderThread(BinTree bt) //建立中序穿线树
2   {
…                ……
26              p = bt;
27              q = bt;
28             while (p->leftchild != NULL)
29              p = p->leftchild; //对中序遍历的第一个结点特殊处理
30              p->lthread = 1;
31            while (q->rightchild != NULL)
32              q = q->rightchild; //对中序遍历的最后一个结点特殊处理
33              q->rthead = 1;
34  }
```



建立线索二叉树意义

遍历

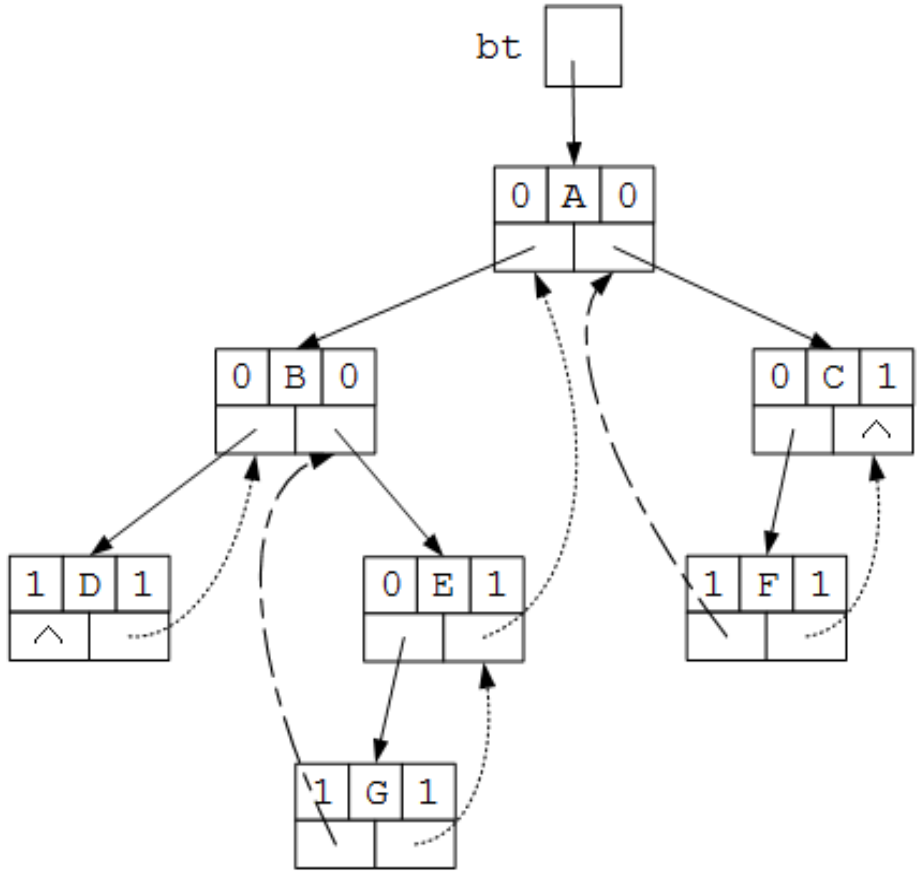
- 递归
- 非递归：栈
- 非递归：不用栈



中序遍历中序穿线树

- **如何找到对称序列的第一个结点?**
  - 从根结点出发，沿着左指针不断往下走，直到左指针为空，到达“**最左下**”的结点，这个结点就是中序序列的第一个结点

- **如何找到对称序列中结点的后继结点?** 有两种情况
  - 如果一个结点的右指针字段是**线索**，则该指针就指向该结点的**后继**
  - 如果**不是线索**，则它指向该结点右子树的根，那么继续找此右子树的最左下结点，它是当前结点的后继



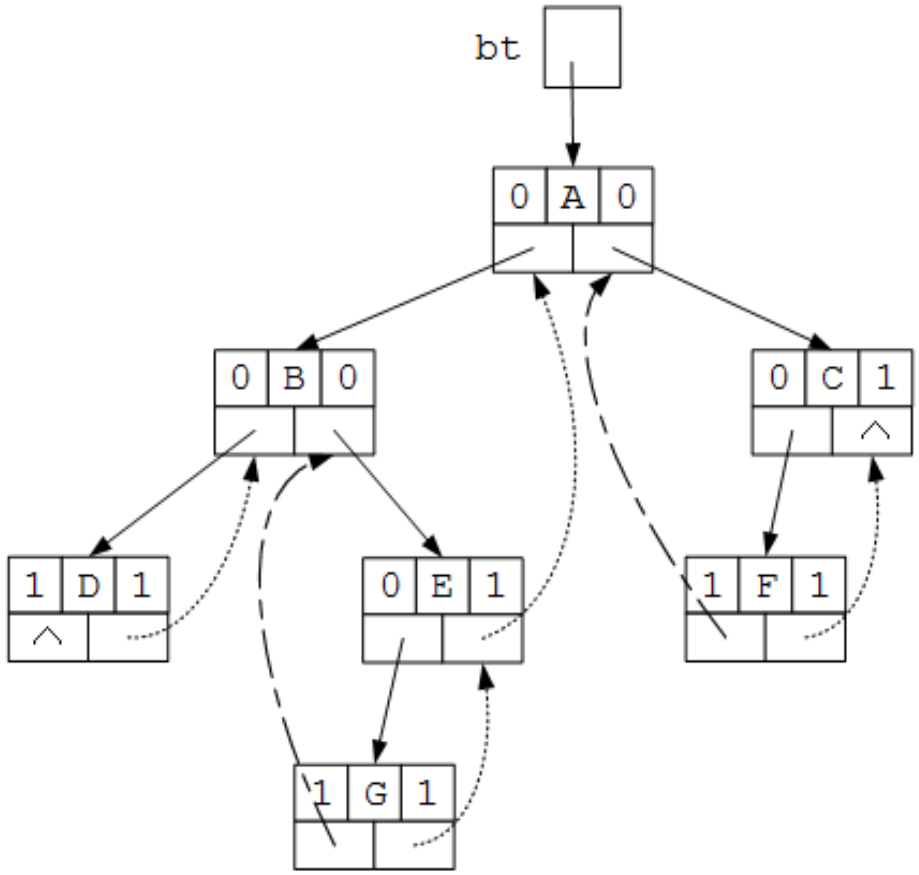
对称序遍历对称序线索二叉树

遍历的过程实质就是找到第一个结点后，不断查找后继的过程。

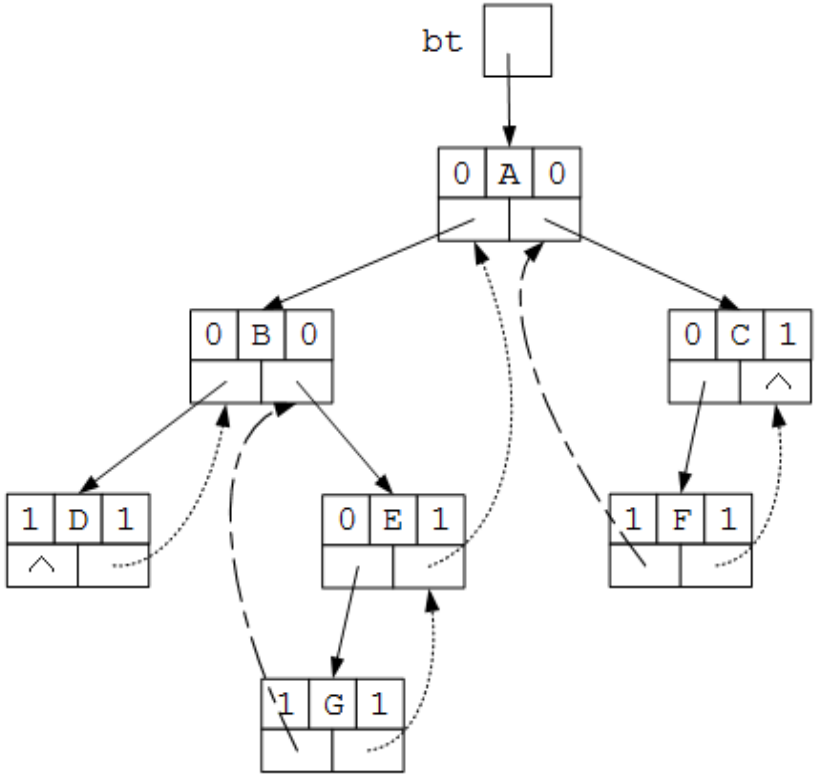
（1）中序遍历的第一个结点是沿着左分支的最下面的结点。

（2）在线索二叉树中，一个结点的后继分为两种情况：

- 如果该结点的rthread是线索，则右指针指向该结点的后继结点；
- 如果该结点的rthread不是线索，则右指针指向该结点的右子树的根结点，根据中序遍历的定义，该结点的后继是在右子树的最左下结点。



```
1  void Inorder_ThreadBinTree(BinTree bt)  //中序遍历中序穿线树
2  {
3          BinTreeNode  *p;
4          if (bt == NULL) return;
5          p = bt;
6         // 沿着左子树一直向下找第一个结点
7         while (p->leftchild != NULL && p->lthread == 0)
8               p = p->leftchild;
9          while (p != NULL)
10         {
11                 printf("%c  ", p->data);
12                 printf("%d  ", p->lthread);
13                 printf("%d\n", p->rthread);
14                 if (p->rightchild != NULL &&p->rthread == 0)  // 右子树不是线索
15                 {
16                         p = p->rightchild;
17                         // 顺右子树的左子树一直向下
18                         while (p->leftchild != NULL && p->lthread == 0)
19                                 p = p->leftchild;
20                 }
21                 else  p = p->rightchild;   // 顺线索向下
22         }
```



算法4-20