数据结构与算法

Data Structures and Algorithms

第三部分 树

回顾：树--1

1. 基本术语

三种不同的定义

根，互不相交的划分（分支）、子树

2. 二叉树（左右之分）

（1）概念、定义、性质

（2）二叉树的遍历（ADT操作）

先根顺序遍历

中根顺序遍历

后根顺序遍历

如何唯一确定一个二叉树？

三种遍历的递归算法

二叉树的性质

性质1： 在二叉树中的第 i 层的结点数最多为：2^{i-1} 。

性质2： 高度为k的二叉树其结点总数最多为$2^k-1\ (k \geq 1)$ 。

性质3： 对任意的非空二叉树 T，如果叶结点的个数为 n_0，而其度为 2 的结点数为 n_2，则：$n_0 = n_2 + 1$ 。

性质4： 具有 n 个结点的完全二叉树的深度为$\lfloor log_2 n \rfloor + 1$。

性质5： 如果对一棵有n个结点的完全二叉树的结点按层序编号，则对任一结点i有：

(1) 如果 $i=1$，则结点i是二叉树的根，无双亲；
如果 $i>1$，则其双亲结点是$\lfloor i/2 \rfloor$；

(2) 如果 $2i>n$，则结点i无左孩子结点，否则其左孩子结点是 $2i$；

(3) 如果 $2i+1>n$，则结点i无右孩子结点，否则其右孩子结点是 $2i+1$。

*二叉树的遍历的非递归过程

中根顺序遍历二叉树：

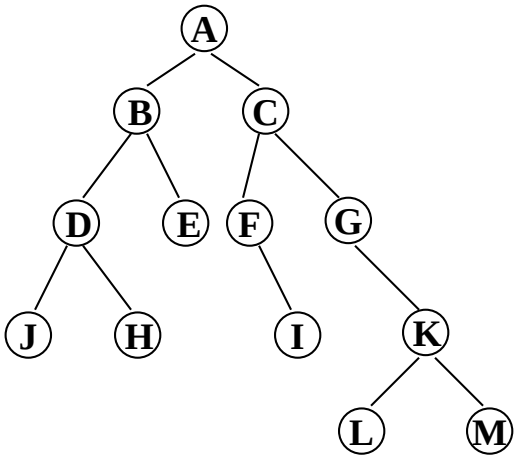
若二叉树非空则：

{ 中根顺序遍历左子树；

访问根结点；

中根顺序遍历右子树；

}；



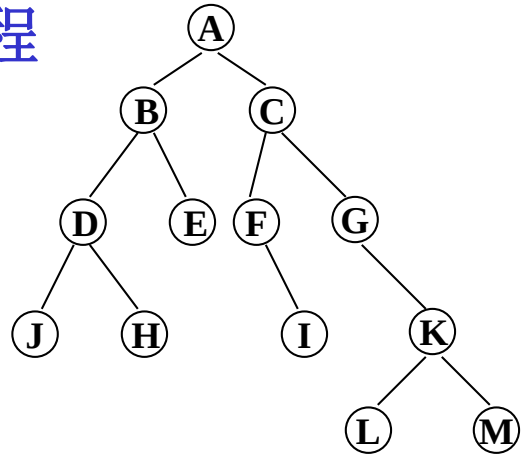
思想：每次在第二次从左边回来的时候再访问，即先碰到，但后访问，为了辨识回来的路，将这种路径结构用栈来存储；

算法:

```
Loop:
{
   if (BT 非空)
      { 进栈; 左一步;}
   else
      { 退栈; 输出; 右一步;}
};
```

*二叉树的遍历的非递归过程

例




```
Void  InOrder ( BT )
BTREE  BT ;
{   if ( ! IsEmpty ( BT ) )
    {  InOrder ( Lchild ( BT ) ) ;
       visit ( Data ( BT ) ) ;
       InOrder ( Rchild ( BT ) ) ;
     }
}
```

先序：A B D J H E C F I G K L M

中序：J D H B E A F I C G L K M

后序：J H D E B I F L M K G C A

No.	指针BT	栈	输出
1	A →	#	
2	B →	#A	
3	D →	#AB	
4	J →	#ABD	
5	^	#ABDJ →	J
6	^	#ABD →	D
7	H →	#AB	
8	^	#ABH →	H
9	^	#AB →	B
10	E →	#A	
11	^	#AE →	E
12	^	#A →	A
13	C →	#	
14	F →	#C	
15	^	#CF →	F
16	I →	#C	
17	^	#CI →	I
18	^	#C →	C
19	G →	#	
20	^	#G →	G
21	K →	#	
22	L →	#K	
23	^	#KL →	L
24	^	#K →	K
25	M →	#	
26	^	#M →	M
27	^	#	结束

数据结构:

设栈S:
用以保留
当前结点;

算法:

```
Loop:
{
   if  (BT 非空)
      { 进栈;
        左一步;}
   else
      { 退栈;
        输出;
        右一步;}
};
```

二叉树的遍历的非递归过程

```
Void   NInOrder( BT )
BTREE  BT;
{   STACK  S ;  BTREE  T ;
    MakeNull( S ) ;
    T = BT ;
    while ( !IsEmpty( T ) || Empty ( S ) )
      if  ( !IsEmpty ( T ) )
          {   Push( T ,S );
              T = Lchild ( T ) ;  }
      else
          {  T = TOP ( S ) ;  POP ( S ) ;
             visit( Data( T ) ) ;
             T = Rchild ( T ) ;   }
}
```

进栈; 左走一步

退栈; 右走一步

先序遍历非递归算法

```
Loop:
{
   if (BT 非空)
      { 输出;
        进栈;
        左一步;}
   else
      { 退栈;
        右一步;}
};
```

中序遍历非递归算法

```
Loop:
{
   if (BT 非空)
      { 进栈;
        左一步;}
   else
      { 退栈;
        输出;
        右一步;}
};
```

后序遍历非递归算法

```
Loop:
{
   if (BT 非空)
      { 进栈;
        左一步;}
   else
      {  当栈顶指针
         所指结点的
         右子树不存
         在或已访问,
         退栈并输出;
         否则右一步; }
};
```

中序遍历非递归算法：

```
void  InOrder(BTREE *root)
 {   BTREE *stack[MAX];
    int top=0;
    do{       while(root!=Null)
              {   top++;
                  if(top>MAX)
                        printf("栈满!\n");
                  else
                         stack[top]=root;
                root=Lchild(root);
               }
              if(top!=0)
              {
                   root=stack[top];
                   top--;
                   printf("%c ",Data(root));
                   root=Rchild(root);
              }
     }while((top!=0)||(root!=Null));
}
```

```
Loop:
{
   if  (BT 非空)
      { 进栈;
        左一步;}
   else
      {  退栈;
         输出;
         右一步;}
};
```

先序遍历非递归算法：

```
void  PreOrder(BTREE *root
{     BTREE *stack[MAX];
      int top=0;
      do{   while(root!=Null)
            {
                  printf("%c ",Data(root));
                   top++;
                   if(top>MAX)
                       printf("栈满!\n");
                   else stack[top]=root;
                         root=Lchild(root);
            }
             if(top!=0)
            {
                  root=stack[top];
                   top--;
                   root=Rchild(root);
            }
       }while((top!=0)||(root!=Null));
}
```

```
Loop:
{
   if  (BT 非空)
      { 输出;
        进栈;
        左一步;}
   else
      { 退栈;
        右一步;}
};
```

后序遍历非递归算法：

```
void  PostOrder(BTREE *root)   //后序遍历，非递归
{     BTREE *stack[MAX],*p;
      int top=0,b;
      do{   while(root!=Null)
            {  top++;
               if(top>MAX)
                    printf("栈满!\n");
               else
                    stack[top]=root;
               root=Lchild(root);
            }
             p=Null;
             b=1;
             while((top!=0)&&b)  //右子树不存在或已访问
             {  root=stack[top];
                if(root->rchild==p)
                {   printf("%c ",Data(root));   //访问根结点
                    top--;
                    p=root;
                }                          //p指向刚访问
                else
                {      root=Rchild(root);
                       b=0;
                }
             }
         }while(top!=0);
}
```

```
Loop:
{
   if  (BT 非空)
       { 进栈;
         左一步;}
   else
       {   当栈顶指针
           所指结点的
           右子树不存
           在或已访问,
           退栈并输出;
           否则右一步; }
};
```

3.2.3 二叉树的表示

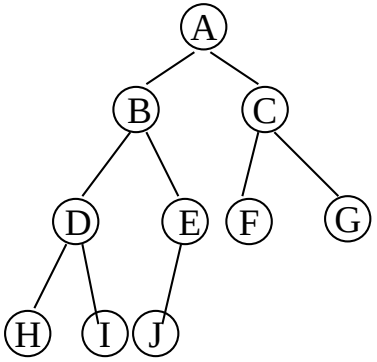
1、顺序存储

（1）完全（或满）二叉树

根据性质5，如已知某结点的层序编号i,则可求得该结点的双亲结点、左孩子结点和右孩子结点。

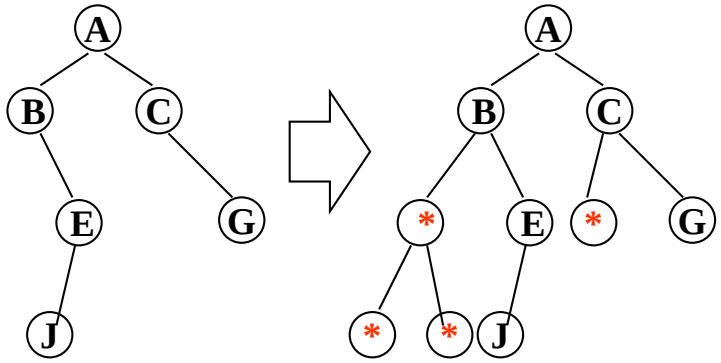
采用一维数组，按层序顺序依次存储二叉树的每一个结点。如下图所示：

A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10



（2）一般二叉树

通过虚设部分结点，使其变成相应的完全二叉树。

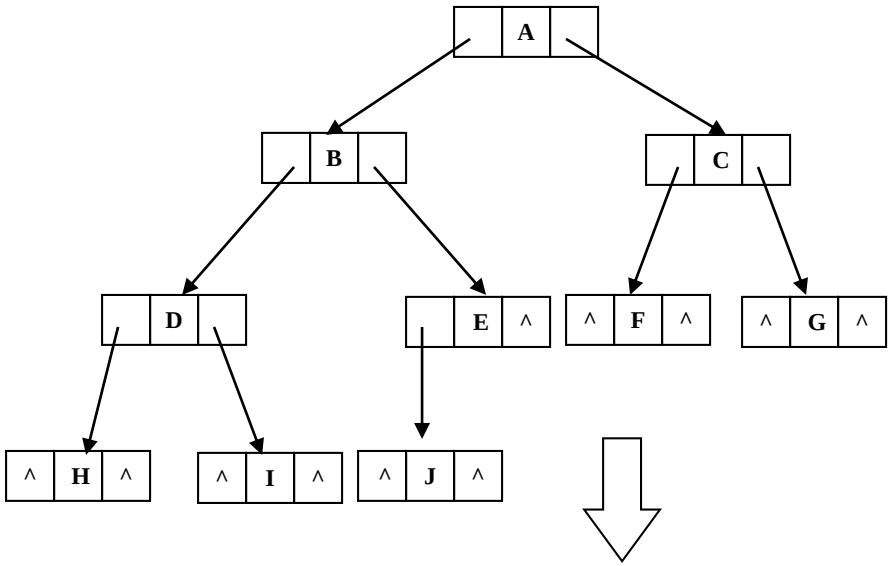


A	B	C	*	E	*	G	*	*	J
1	2	3	4	5	6	7	8	9	10

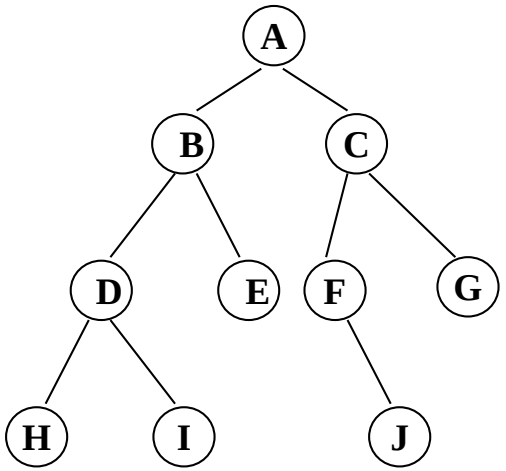
根据性质5，如已知某结点的层序编号i,则可求得该结点的双亲结点、左孩子结点和右孩子结点，然后检测其值是否为虚设的特殊结点*。

2、二叉树的左右链表示

lchild	data	rchild



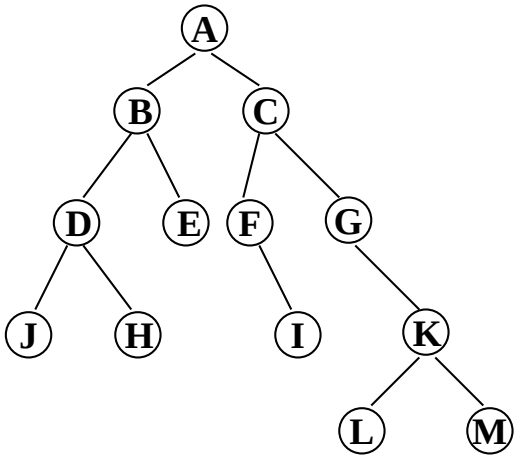

```
Struct  Node  {
    Struct  Node  *lchild ;
    Struct  Node  *rchild ;
          datatype  data ;  } ;
Typedef  struct   Node    * BTREE ;
```



ADT

```
BTREE CreateBT(v , ltree ,  rtree )
datatype   v ; BTREE  ltree , rtree ;
{  BTREE  root ;
   root = New Node ;
   root->data = v ;
   root->lchild = ltree ;
   root->rchild = rtree ;
   return  root ;
}
```

证明：n 个结点的二叉树中，共有 n+1 个空链接域。



结点总数：13

空链域的个数：14

证：设其空链域数为 x

分支数 $B_{入} = n - 1$

$B_{出} = 2 \times n - x$

$\because B_{入} = B_{出}$

$\therefore n - 1 = 2 \times n - x$

得出 $x = n + 1$

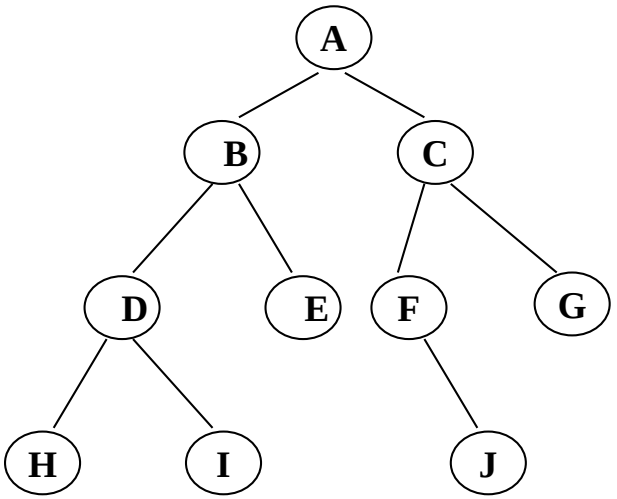
【例3-4】二叉树建立方法之一

按先序序列建立二叉树的左右链结构。

如下图所示二叉树，输入：

ABDH##I##E##CF#J##G##

其中:#表示空




```
BTREE *CreateTree1()
{   BTREE *bt;
    char ch;
    fflush(stdin);
    scanf("%c",&ch);
    if(ch=='#')
            bt=Null;
    else
    { bt=New BNODE;
      if(!bt) exit(0);
      bt->data=ch;
      bt->lchild= CreateTree1();
      bt->rchild= CreateTree1();
     }
     return(bt);
}
```

【例3-5】二叉树的遍历（二叉链表结构）

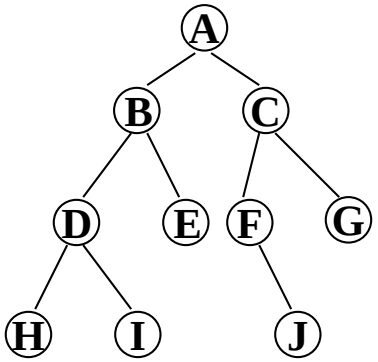
```
Void PreOrder( BTREE T)
{
  if(T){
        visit(T->data);
        PreOrder(T->lchild);
        PreOrder(T->rchild);
      }
}
```

```
Void InOrder( BTREE T)
{
  if(T){
        InOrder(T->lchild);
        visit(T->data);
        InOrder(T->rchild);
      }
}
```

```
Void PostOrder( BTREE T)
{ if(T){
        PostOrder(T->lchild);
        PostOrder(T->rchild);
        visit(T->data);
       }
}
```

【例3-6】按层序遍历二叉树（二叉链表结构）

从二叉树的第一层开始，直到最后一层，每层从左到右访问每一个结点，是的每个结点只能被访问一次。



右图二叉树按层序遍历结果： **ABCDEFGHIJ**

算法思想：

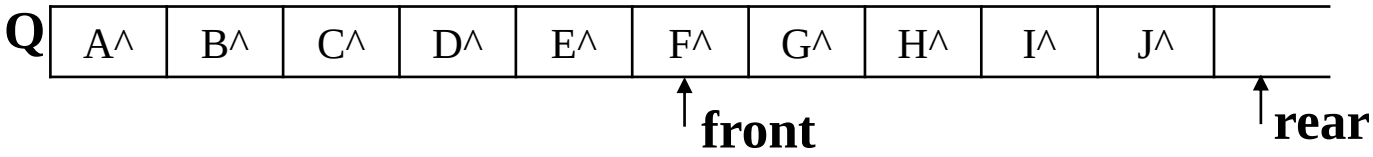
设立一个队列，队列元素为结点的指针；

（1）将根结点指针排队；

（2）当队列非空时，从队首结点出队列并访问这个结点，如果该结点的左子树非空，则其左孩子结点指针排队；

（3）如果该结点的右子树非空，则其右孩子结点指针排队；

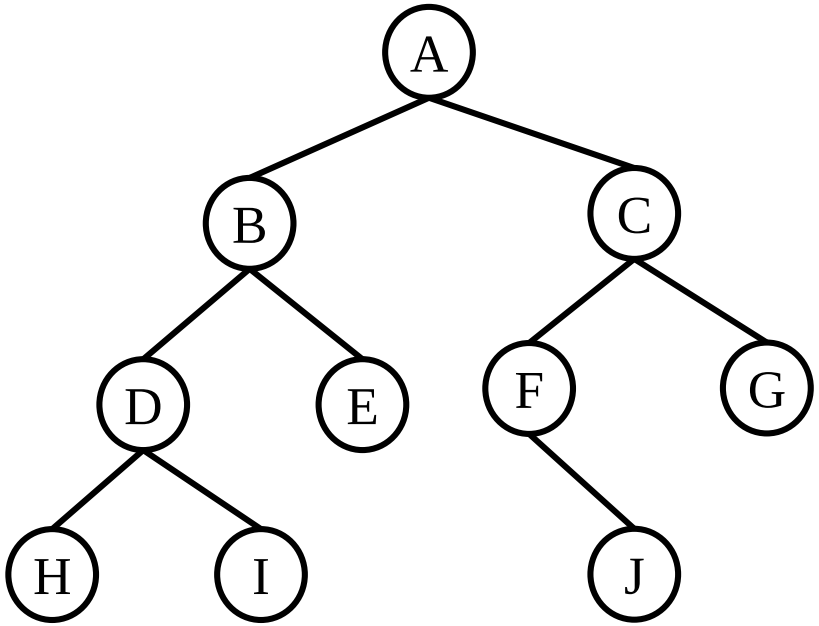
（4）重复上述 (2)-(3) 过程，直到队列为空。



```
Struct  node  {
    Struct  node  *lchild ;
    Struct  node  *rchild ;
          datatype  data ;
} ;
Typedef  struct   node    * BTREE ;


Struct  QUEUE {
    Struct  node  * data[maxlength] ;
    int   front ;
    int   rear ;
};
```

```
Void  LeverList(BTREE T)
{
  QUEUE Q;
  BTREE p=T;
  MakeNull(Q);
  if(T) {  EnQueue(p, Q);
           while(!Empty(Q))
           {  p=DeQueue(Q);
              visit(p->data);
              if(p->lchild)
                   EnQueue(p->lchild);
              if(p->rchild)
                   EnQueue(p->rchild);
            }
}
```



Q	A^	B^	C^	D^	E^	F^	G^	H^	I^	J^
层	1	2	2	3	3	3	3	4	4	4
高										4
宽				3	3	3	3			
双亲	--	A	A	B	B	C	C	D	D	F
祖先										

【例3-7】一棵二叉树的先序、中序和后序序列分别如下，其中一部分未显示出来，试求出空格处的内容，并画出该二叉树。

先序：_B_F_ICEH_G;　　先序：ABDFKICEHJ G;

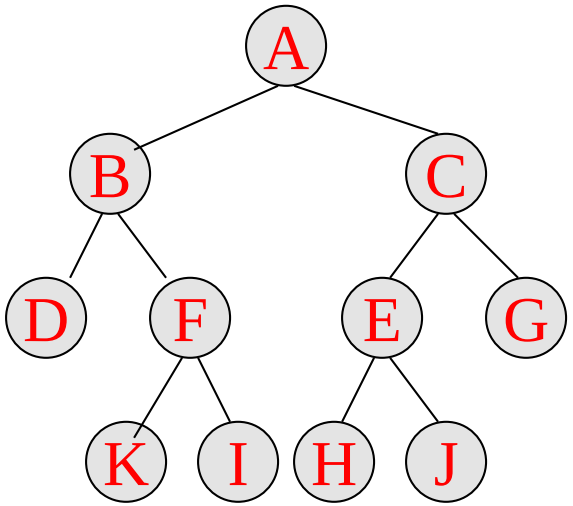
中序：D_KFIA_EJC_;　　中序：DB KFIAHEJCG;

后序：_K_FBHJ_G_A;　　后序：_ K_ FBHJ_ G_ A;

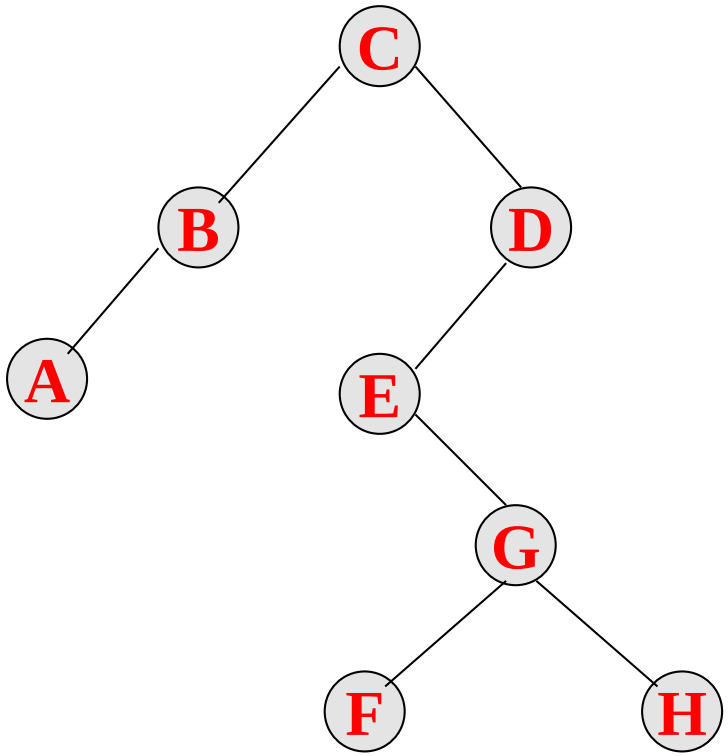
先序：ABDFKICEHJG

中序：DBKFIAHEJCG

后序：DKIFBHJEGCA



【例3-8】二叉树中序序列为：ABCEFGHD，
后序序列为：ABFHGEDC。
请画出此二叉树。



已知：

①已知先序和中序？

②已知中序和后序？

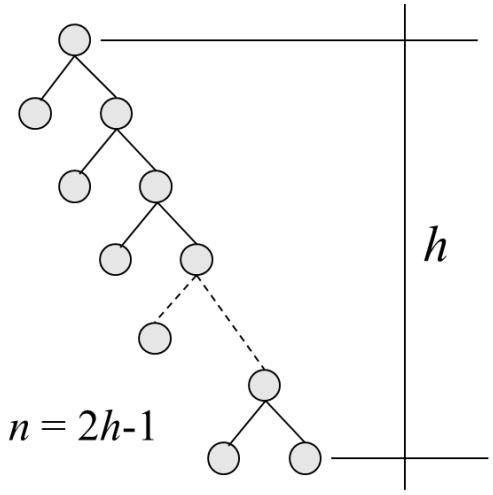
③已知先序和后序？

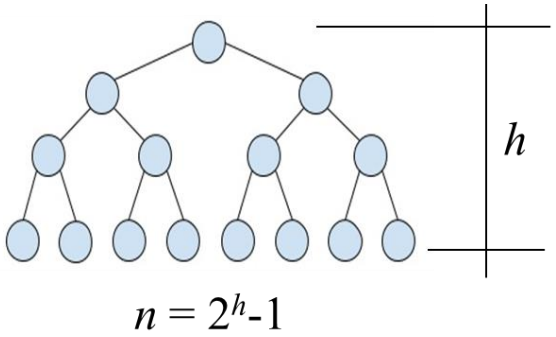
能否唯一还原二叉树？

【例3-9】试举出：先序遍历和中序遍历相同的二叉树？
先序遍历和后序遍历相同的二叉树？
中序遍历和后序遍历相同的二叉树？

【例3-10】完全二叉树的某结点若无左孩子结点，则它必是叶结点，为什么？

【例3-11】设高为*h*的二叉树只有度为*0*和度为*2*的结点，则此类二叉树的结点数至少为____，至多为____。





【例3-12】一棵有124个叶子结点（n_0）的完全二叉树，最多有__？__个结点（n）？

因为：$n_0=n_2+1$　　（性质3）

$n=n_0+n_1+n_2$　　（结点总数）

所以有：$n=n_1+2n_0-1$

在完全二叉树中，

n_1 不是 0 就是 1

只有$n_1=1$ 时，n 取最大值为 $2n_0$

【例3-13】证明任一棵满二叉树T中的分支数 B 满足：

$B=2(n_0-1)$ ，其中 n_0 为叶子结点数。

证明：

满二叉树中不存在度为1的节点，设度为2的结点数为n_2

则： $n=n_0+n_2$

又： $n=B+1$

所以有： $B=n_0+n_2-1$ ， 而 $n_0=n_2+1$, $n_2=n_0-1$

$B=n_0+n_0-1-1=2(n_0-1)$

【例3-14】具有 n 个结点的满二叉树，其叶子结点的个数为多少？（n和n0的关系）

方法一：设满二叉树的高度为 h；

则根据二叉树的性质，叶子结点数为2^{h-1}；

二叉树总结点数$n=2^h-1$；

可导出：$2^{h-1}=(n+1)/2$；

方法二：结点总数：$n=n_0+n_1+n_2$；

但对满二叉树，除有$n_0=n_2+1$外，还有$n_1=0$；

故有：$n=n_0+n_0-1$

$n_0=(n+1)/2$

【例3-15】n个结点的完全二叉树，其叶子结点的个数为多少？

【例3-16】二叉树建立方法之二

```
BTREE *CreateTree2(BTREE *bt, int n)      //交互问答方式创建二叉树
{   char ch;
    if(n==0)   printf("根结点:");
    fflush(stdin);scanf("%ch",&ch);fflush(stdin);
    if(ch!='#')
   {    n=1;
        bt=New BNODE;
        bt->data=ch;
        bt->lchild=Null;
        bt->rchild=Null;
        printf("%c 的左孩子是:",bt->data);
        bt->lchild= CreateTree2(bt->lchild,n);
        printf("%c 的右孩子是:",bt->data);
        bt->rchild= CreateTree2(bt->rchild,n);
    }
    return(bt);
}
```

交互地问根、左右孩子

【例3-17】
求任意二叉树的宽度。

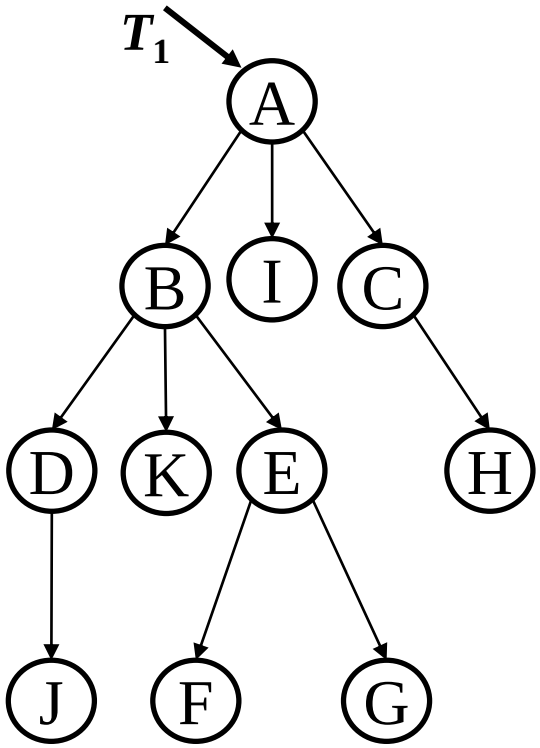
```
int Width(BTREE *T)
 {    //求二叉树的宽度
      int i,n=0,front=0,rear=0,max=0,lev=1, maxlev[10]={0};
      struct W{
           BTREE *Node;
                int Nodelev;    } Q[50];
       Q[front].Node=T;      Q[front].Nodelev=1;
       while(front<=rear)
       {   if(Q[front].Node->lchild)
           {   Q[++rear].Node=Q[front].Node->lchild;
               Q[rear].Nodelev=Q[front].Nodelev+1;        }
           if(Q[front].Node->rchild)
           {   Q[++rear].Node=Q[front].Node->rchild;
               Q[rear].Nodelev=Q[front].Nodelev+1;        }
            front++;
       }
       for(i=0;i<=rear;i++)    maxlev[Q[i].Nodelev]++;
       for(i=0;i<10;i++)
            if(max<maxlev[i])          max=maxlev[i];
       return(max);
}
```

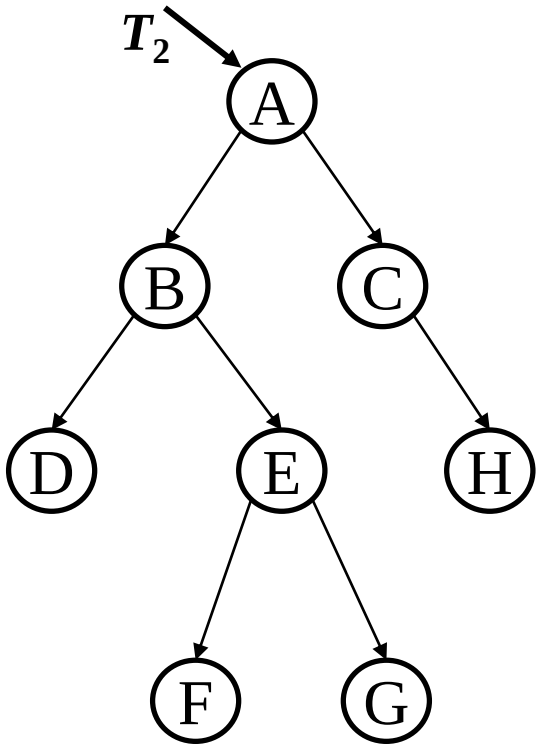
【例3-18】求任意二叉树的深度。

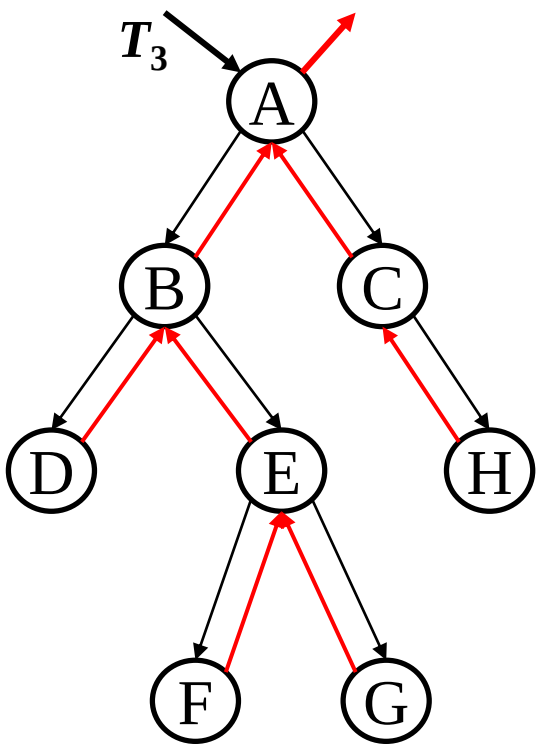
```
int Depth(BTREE *bt)    //求二叉树的深度
 {
      int ldepth,rdepth;
      if(bt==Null)
           return(0);
       else
      {
          ldepth=Depth(bt->lchild);
          rdepth=Depth(bt->rchild);
          if(ldepth>rdepth)
               return(ldepth+1);
           else
               return(rdepth+1);
       }
 }
```

求：最大路长
　　最小路长

讨论：先、中、后序遍历都需要栈，以及回溯策略，如果让你设计一个算法去除栈，你会怎么做？

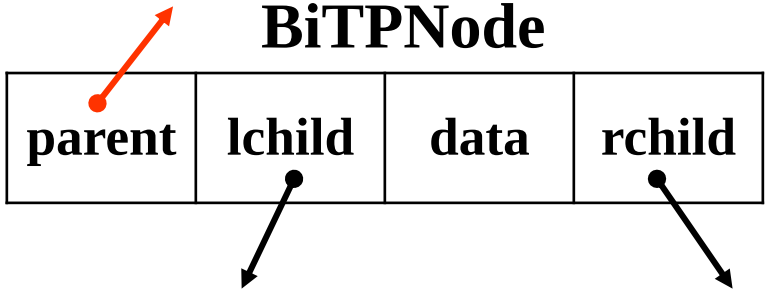






三叉树？

二叉树的三叉链表存储表示




```
typedef struct BiTPNode
{
    ElementType data;
    struct BiTPNode *parent,*lchild,*rchild;
} *BiPTree;
```

◆ *n*个结点的二叉树有*n*+2个空指针！

很显然：

- 相对二叉链表表示的二叉树，除了找父结点的操作变得很容易外，其它基本操作没有什么变化。
- 对二叉树的先序/中序/后序的非递归遍历，不需要再使用栈。

不用栈非递归遍历

```
void InOrder(TriTree PT, void (*visit)(TElemType))
{   TriTree p=PT, pr;
    while(p)
    {   if (p->lchild)   p = p->lchild;  //找最左结点
        else {   visit(p->data);          //访问最左节点
                 if (p->rchild)   p =p->rchild; //若有右子树，找右子树最左结点
                 else {    pr = p;          //否则返回其父结点
                           p = p->parent;
                           while (p && (p->lchild != pr ||!p->rchild))
                           {   if (p->lchild == pr)   visit(p->data);
                               pr = p; //父结点已被访问，故返回上一级
                               p = p->parent;
                           }
                           if (p){   visit(p->data);
                                     p = p->rchild;
                                     }
                 }
        }
    }
}
```

//该while循环沿双亲链一直查找，若无右孩子则访问，直至找到第一个有右孩子的结点为止（但不访问该结点，留给下步if语句访问）

//若其不是从左子树回溯来的，或左结点的父结点并没有右孩子

//访问父结点，并转到右孩子（经上步while处理，可以确定此时p有右孩子）

思考题：

（1）如何判断一颗任意二叉树是否为满二叉树？

（2）如何判断一颗任意二叉树是否为完全二叉树？

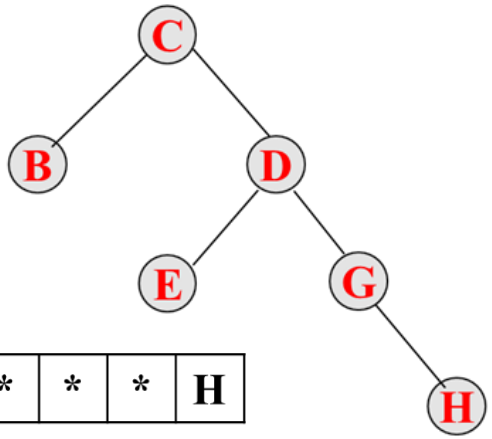
（3）求二叉树任意结点所在的层？

（4）求任意结点的所有祖先结点（根到该结点的路径）

（5）统计任意二叉树中的结点个数？

总结点、度为2、度为1、度为0的结点个数。

（6）二叉链表存储的二叉树转换到按照完全二叉树存储的数组中。‘*’表示空结点。



C	B	D	*	*	E	G	*	*	*	*	*	*	*	H

3.2.4 线索二叉树

问题的提出：

（1）在n个结点的二叉树左右链表示中，有n+1个空链域。如何利用n+1个空链域，使二叉树的操作更加方便；

（2）在二叉树左右链表示中，为求某个结点的（中序）前驱 $P 或（中序）后继 p$，每次都要从树根开始进行查找，很不方便。

思想：考虑利用这些空链域来存放遍历后结点的前驱和后继信息，这就是线索二叉树构成的思想。

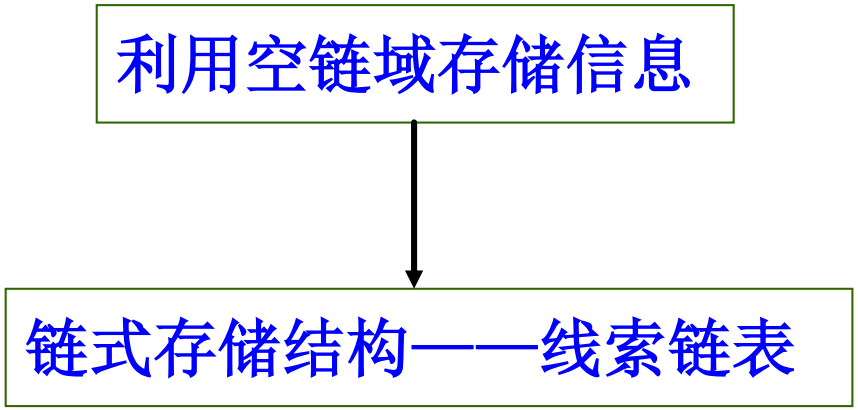
概念：采用既可以指示其前驱又可以指示后继的双向链接结构的二叉树被称为线索二叉树。

3.2.4 线索二叉树

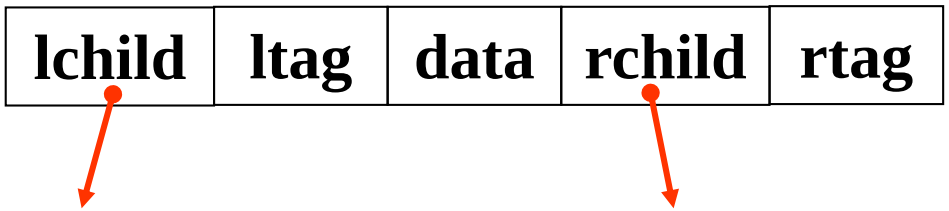
【定义】

若结点p有左孩子，则p->lchild指向其左孩子结点，否则令其指向其（中序）前驱；

若结点p有右孩子，则p->rchild指向其右孩子结点，否则令其指向其（中序）后继。



结点类型 LNode




```
Struct  LNode {
  ElementType   data ;
  Struct  LNode
  *lchild , *rchild ;
   int  ltag , rtag ; //左右线索标志
}
```

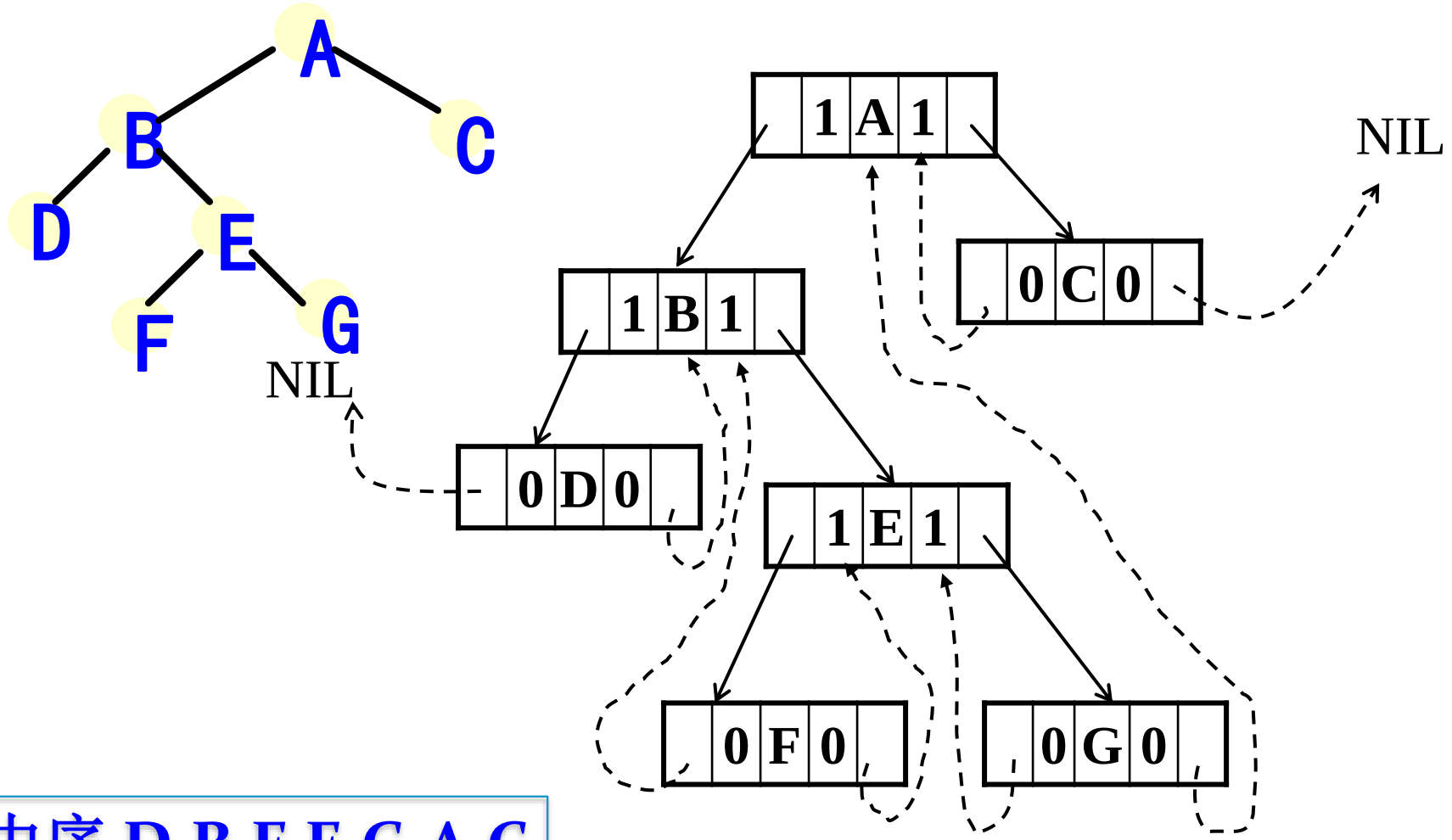
Typdef Struct LNode * THTREE;

$$p\text{->ltag} = \begin{cases} 1 & p\text{->lchild 指向左孩子} \\ 0 & p\text{->lchild 指向（中序）前驱} \end{cases}$$

$$p\text{->rtag} = \begin{cases} 1 & p\text{->rchild 指向右孩子} \\ 0 & p\text{->rchild 指向（中序）后继} \end{cases}$$

讨论：为方便操作利用了 n+1 个指针，但为实现操作却多用了 2n 个标志位，如何理解？

二叉树存储结构--线索链表

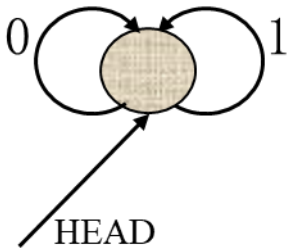


中序 D,B,F,E,G,A,C

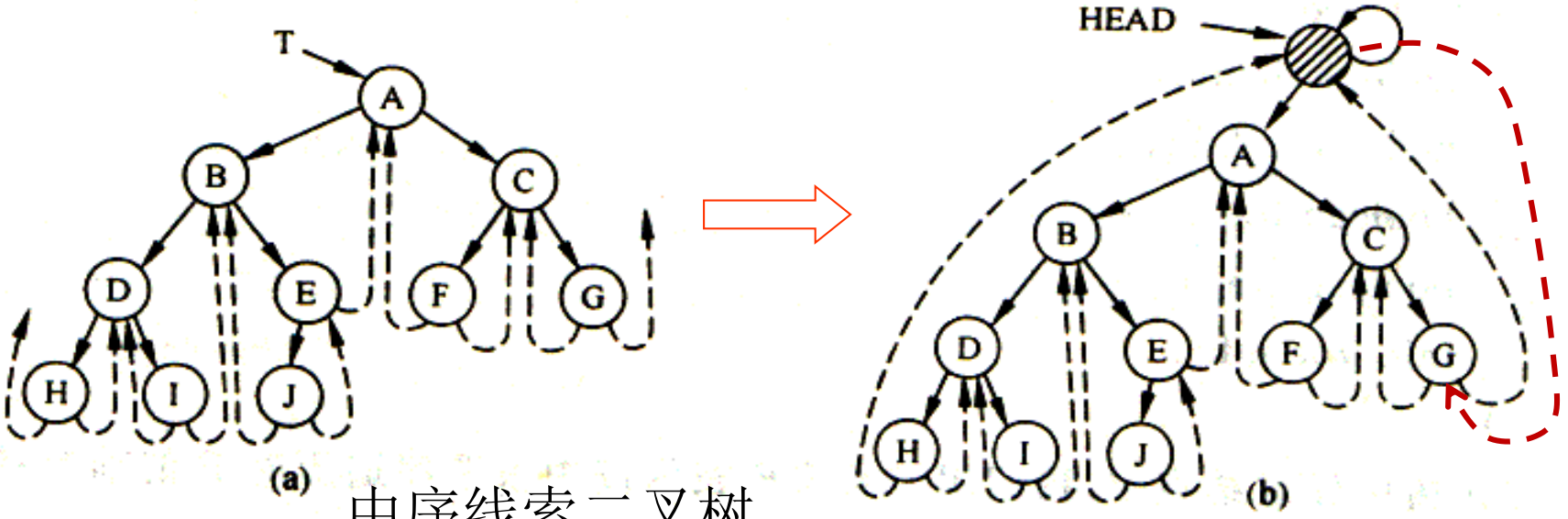
类似线性链表，为每个线索树增加一个**头结点**。并设：

head->lchild = T (二叉树的根) ;
head->rchild = head ;
head->ltag = 1 ;
head->rtag = 1 ;

当线索树为空时：



head->lchild = head ;
head->rchild = head ;
head->ltag = 0 ;
head->rtag = 1 ;

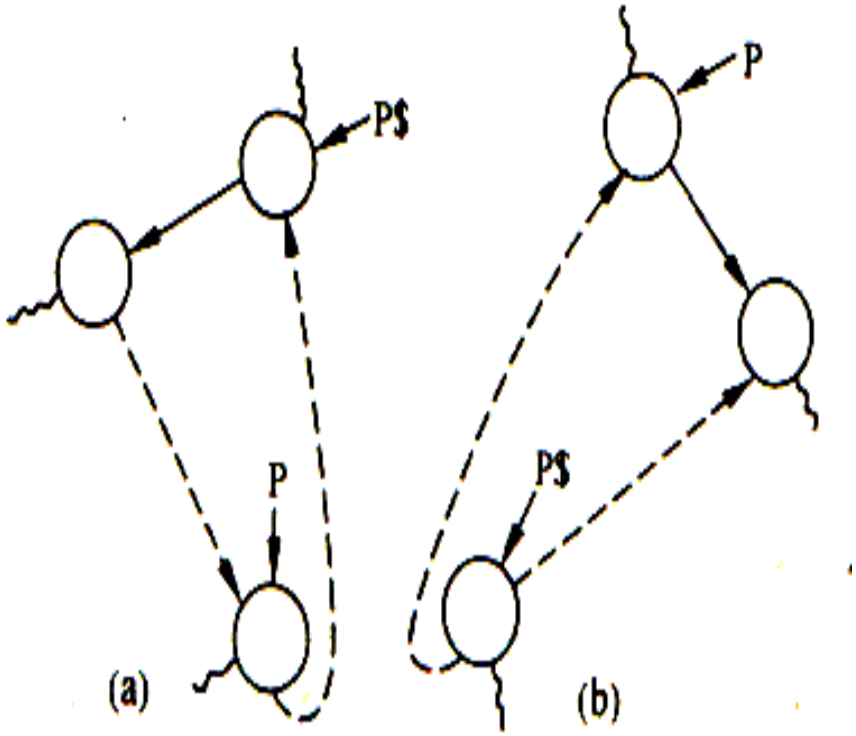


中序线索二叉树

【例3-19】求p$（中序后继）

分析：(1) 当p->rtag = 0时，p->rchild 即为所求（线索）；

(2) 当p->rtag = 1时，p$为p的右子树的最左结点。

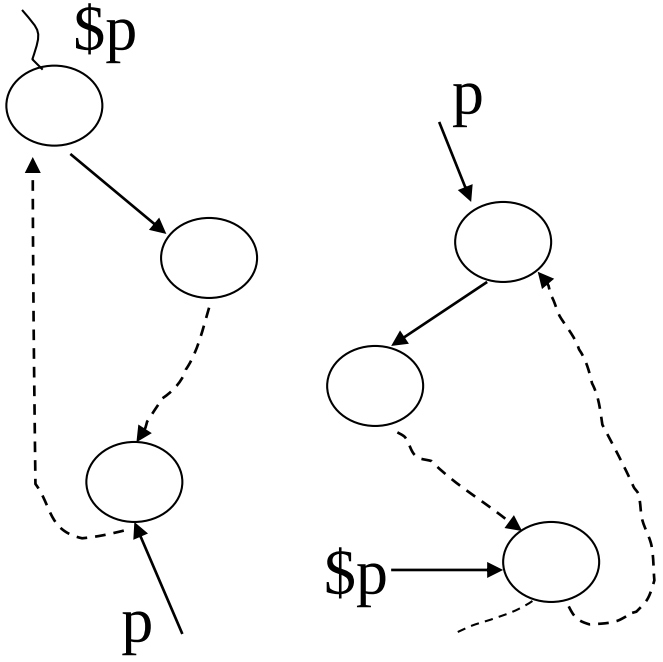



```
THTREE  InNext( THTREE  p)
{  THTREE  Q ;
    Q=p->rchild ;//直接赋值
    if (p->rtag = = 1 )
      while(Q->ltag = = 1)
        Q = Q->lchild ;
    return ( Q ) ;
}
```

【例3-20】求$p（中序前驱）

分析：(1) 当p->ltag = 0时，p->lchild 既为所求（线索）；

(2) 当p->ltag = 1时，$p为p的左子树的最右结点。

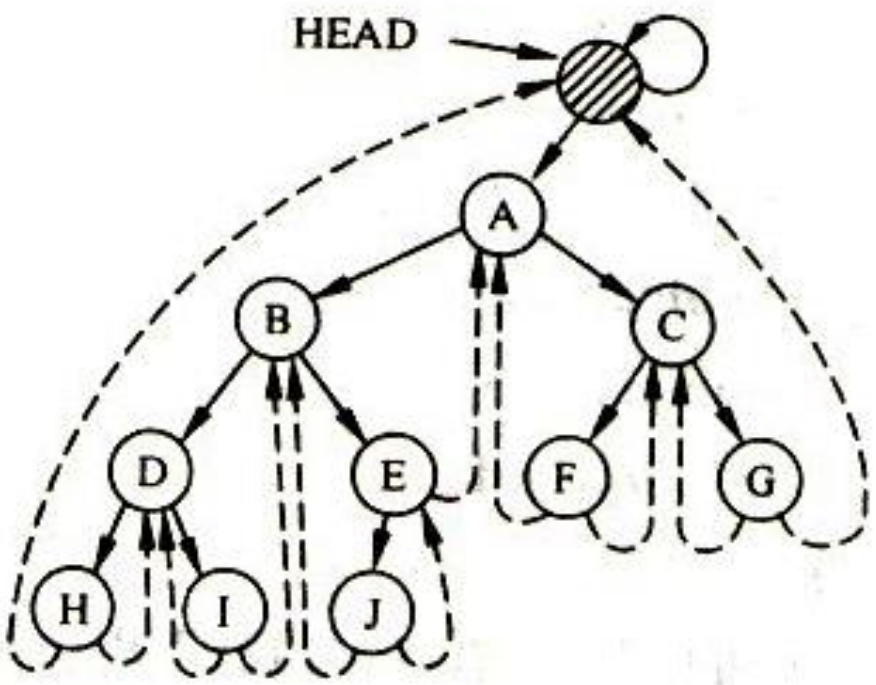



```
THTREE  InPre( THTREE  p)
{  THTREE  Q ;
    Q=p->lchild ;
    if (p->ltag = = 1 )
      while(Q->rtag = = 1)
        Q = Q->rchild ;
    return ( Q ) ;
}
```

【例3-21】利用InNext算法，中序遍历线索二叉树。

```
Void   THInOrder( THTREE   HEAD)
{   THTREE    temp ;
    temp = HEAD ;
    do {
          temp = InNext ( temp ) ;
          if ( temp != HEAD )
             visit ( temp -> data ) ;
     } while ( temp != HEAD ) ;
}
```

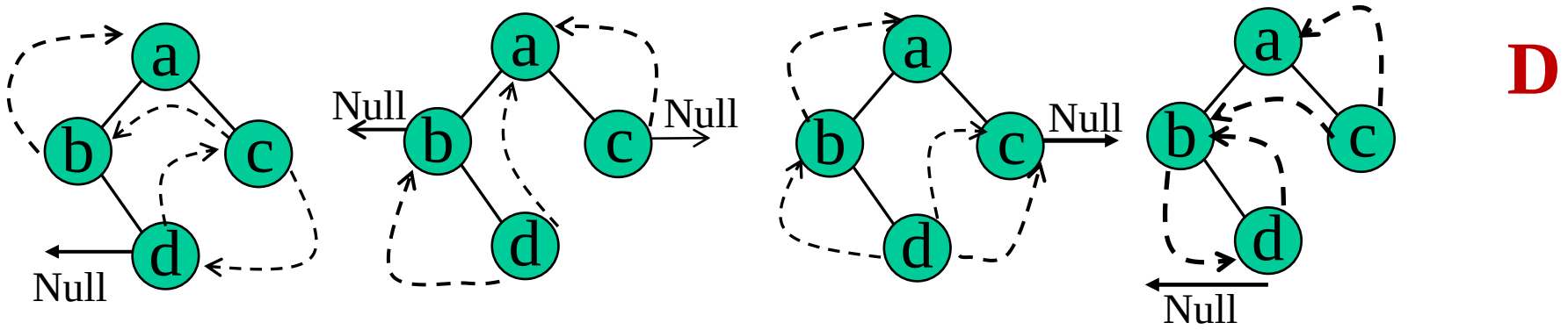
```
head->lchild = T
head->rchild = head ;
head->ltag = 1 ;
head->rtag = 1 ;
```



```
void InOrderTraverse_Thr(THTREE T)
{
   p=T->lchild;
   while(p!=T)
   {
     while(p->ltag==1) p=p->lchild;
     visit(p->data);
     while(p->rtag==0 && p->rchild!=T)
     {
         p=p->rchild;
         visit(p->data);
      }
      p=p->rchild;
}
```

非递归，不利用栈，
中序遍历（中序）线索二叉树。

例题 下列线索二叉树中(用虚线表示线索), 符合后序线索树的是( )



D

思路：后序遍历dbca; d无前驱及左子树，左链为空，右链连b，验证其它。

例题 若X是后序线索二叉树的叶结点，且X存在左兄弟结点Y，则X的右线索指向( )

A. X的父结点　　**B**. 以Y为根的子树的最左结点

C. X的左兄弟结点Y　　**D**.以Y为根的子树的最右结点

A

例题 若对右图二叉树中序线索化，则X的左、右线索分别指向( )

A. e,c **B**. e,a **C**. d,c **D**. b,a

D