第6章 树和二叉树

树的定义和基本术语

二叉树

二叉树的存储结构

遍历二叉树

线索化二叉树

树和森林

赫夫曼树

二叉树的计数*

小结

- 先、中、后序遍历：（深搜）
  时间复杂度：O(n)
  空间复杂度：O(m) m=二叉树的深度
- 层次遍历：（广搜）
  时间复杂度：O(n)
  空间复杂度：O(m) m=二叉树的宽度

6.4 线索二叉树（穿线树、线索树）(Threaded Binary Tree)

- 遍历的实质：

  对一个非线性结构进行线性化操作，使每个结点（除第一和最后一个外）在这些线性序列中有且仅有一个直接前驱和直接后继。

- 在二叉树遍历过程中将指针域值为空的指针指向其直接前驱或直接后继，该指针就称作线索。

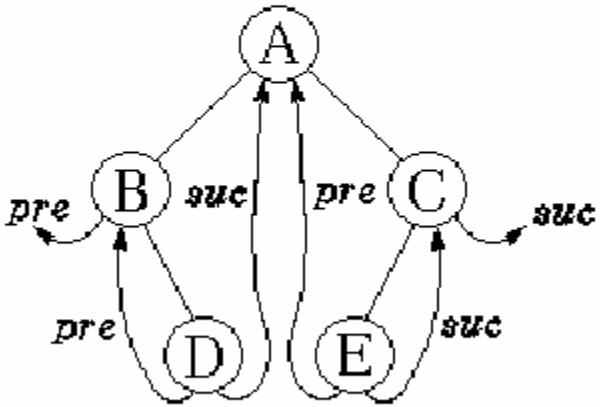
- 线索 (Thread): 指向结点前驱和后继的指针

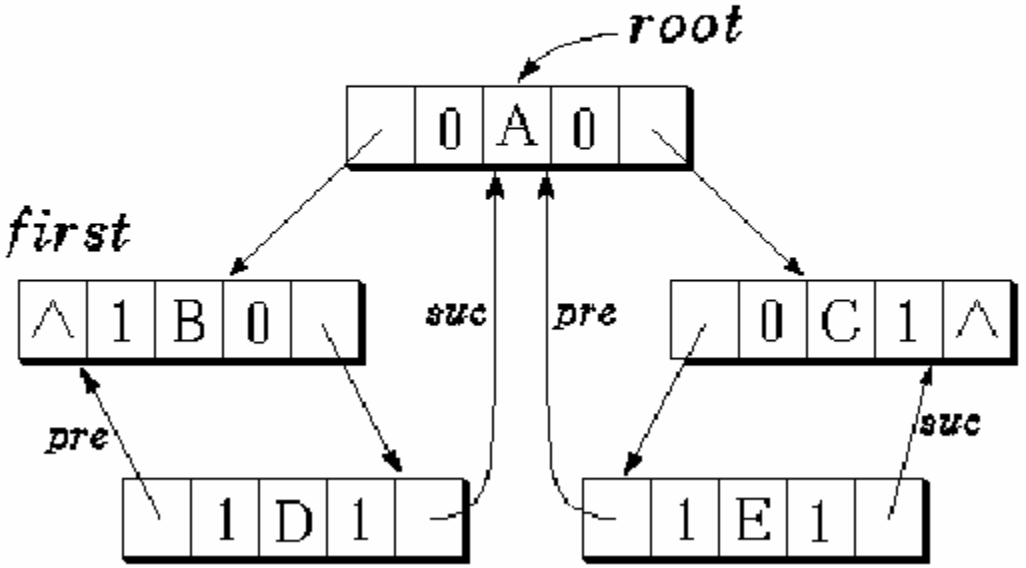
  - 若结点有左孩子，则lchild指示其左孩子，否则lchild中存储指向该结点的前驱结点的指针(线索)；
  - 若结点有右孩子，则rchild指示其右孩子，否则rchild中存储指向该结点的后继结点的指针(线索)
  - 在线索树中的前驱和后继是指按某种次序遍历所得到的序列中的前驱和后继。

线索二叉树及其线索链表的表示

lchild	LTag	data	RTag	rchild

中序线索树示意：





标志域：

tag=0, 孩子；

tag=1, 线索： lchild: 前驱线索， rchild: 后继线索

■概念:

线索：指向前驱或后继的指针

线索化：对二叉树以某种次序遍历加线索的过程

线索二叉树：加了线索的二叉树

线索链表：以下面结点构成的二叉链表作为二叉树的存储结构，叫做线索链表

lchild	ltag	data	rtag	rchild

标志域:

ltag = 0, lchild为左孩子指针

ltag = 1, lchild为前驱线索

rtag = 0, rchild为右孩子指针

rtag = 1, rchild为后继线索

线索二叉树的类型定义

- 类型定义：

```
typedef enum{Link,Thread}PointerTag;
//Link==0：指针，指向孩子结点
//Thread==1：线索，指向前驱或后继结点
typedef struct BiThrNode{
    TElemType data;
    struct BiThrNode *lchild,*rchild;
    PointerTag LTag,RTag;
}BiThrNode, *BiThrTree;
BiThrTree T;
```

遍历线索二叉树

- 从遍历的第一个结点来看

  先序序列中第一个结点必为根结点；

  中、后序序列中第一个结点的左孩子定为空

- 从遍历的最后一个结点来看

  先、中序序列中最后一个结点的右孩子必为空；

  后序序列中最后一个结点一定为根结点

- 线索树的作用

  对于遍历操作，线索树优于非线索树

  遍历线索树不用设栈

- 步骤

  1)找遍历的第一个结点

  2)不断地找遍历到的结点的后继结点，直到树中各结点都遍历到为止，结束。

寻找当前结点在中序下的后继

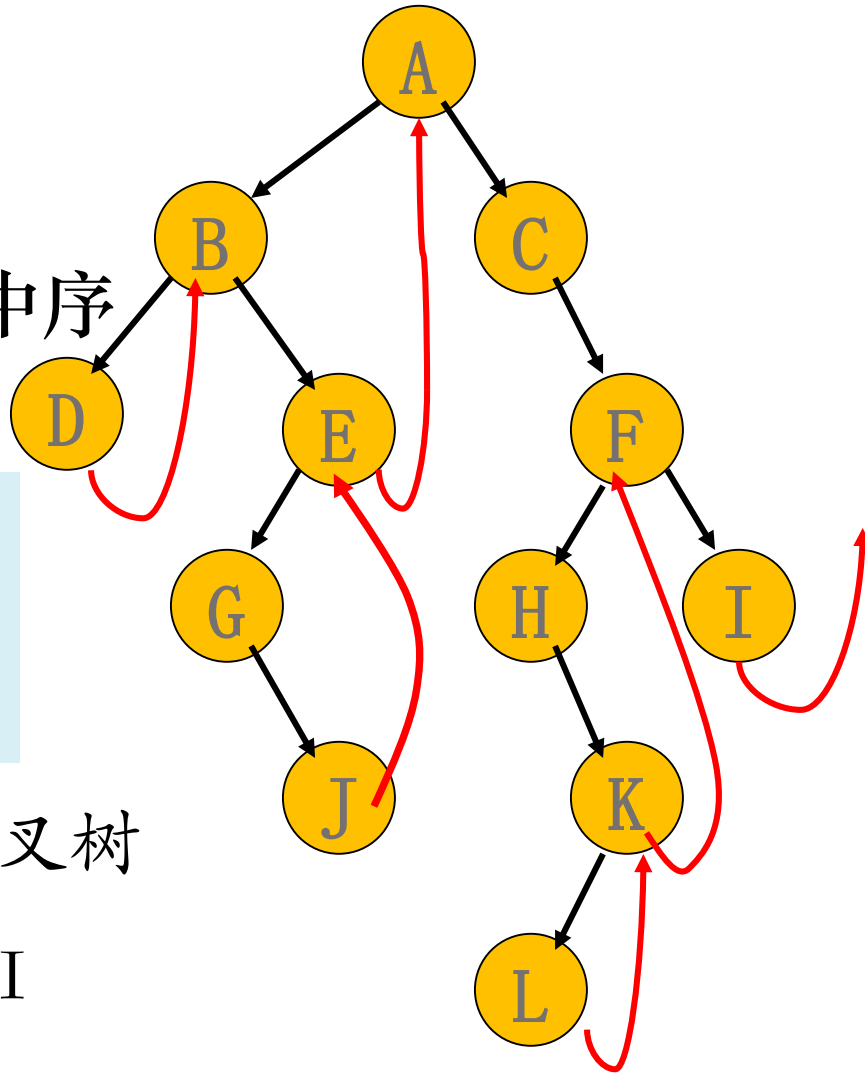
if (current.RTag ==Thread)

后继=current.rchild

else

后继=当前结点右子树的中序下的第一个结点

```
p=current->rchild;
while(p->LTag==Link)
        p=p->lchild;
```



中序后继线索二叉树

DBGJEACHLKFI

寻找当前结点在中序下的前驱

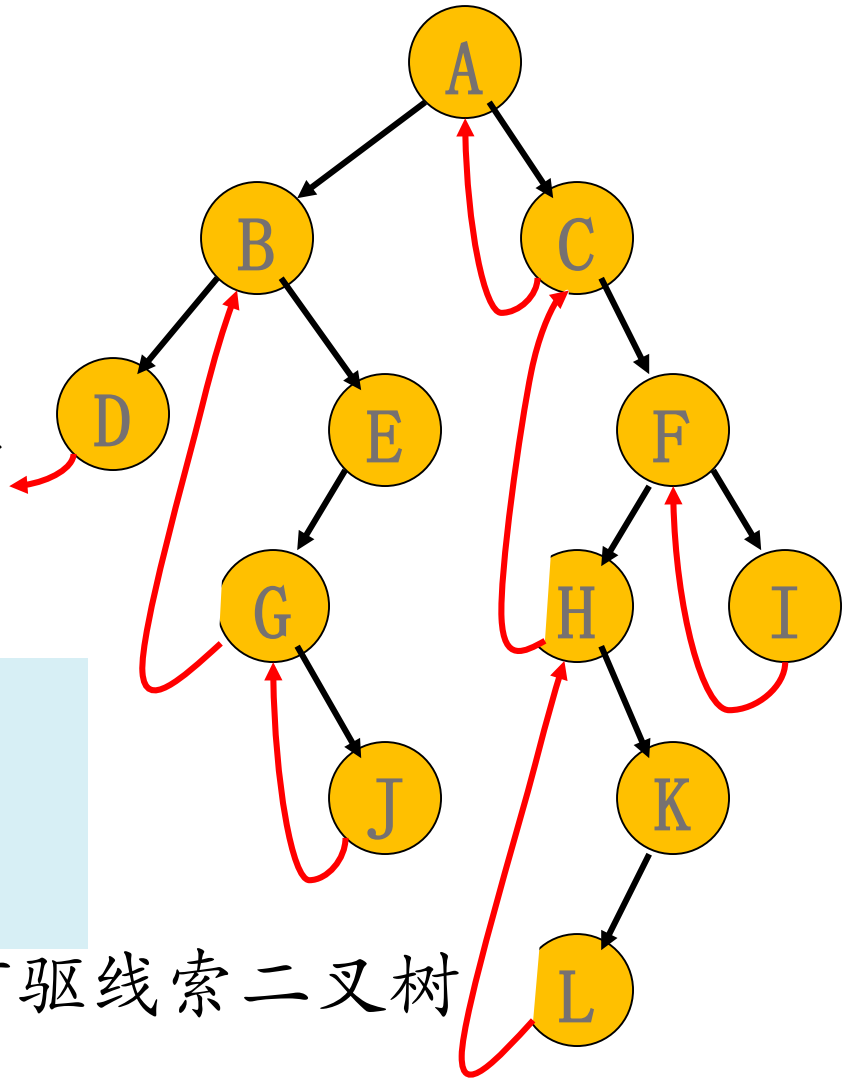
if (current->LTag ==Thread)

前驱=current->lchild

else

前驱=当前结点左子树的中序下的最后一个结点

```
p=current->lchild;
while(p->RTag==Link)
        p=p->rchild;
```



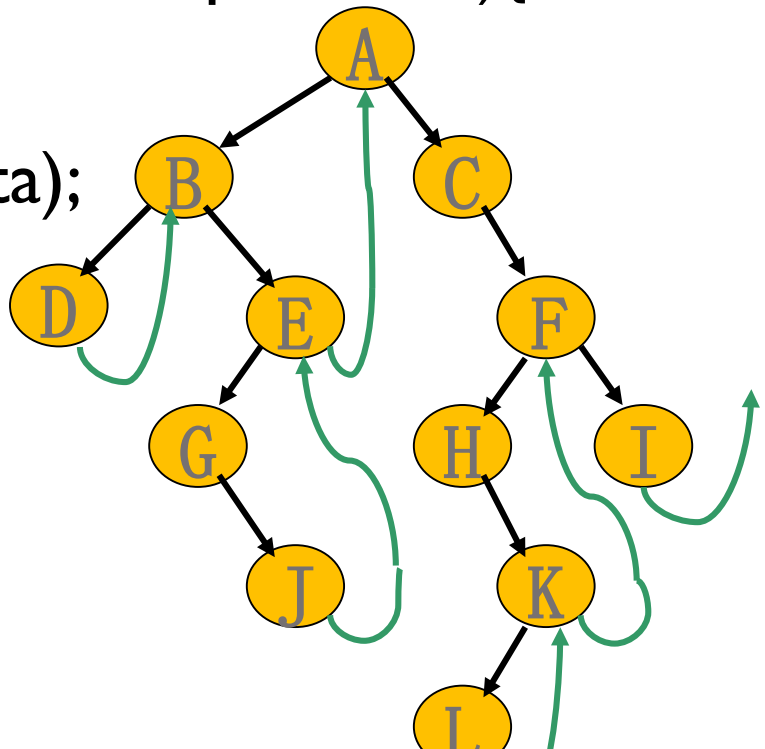
中序前驱线索二叉树

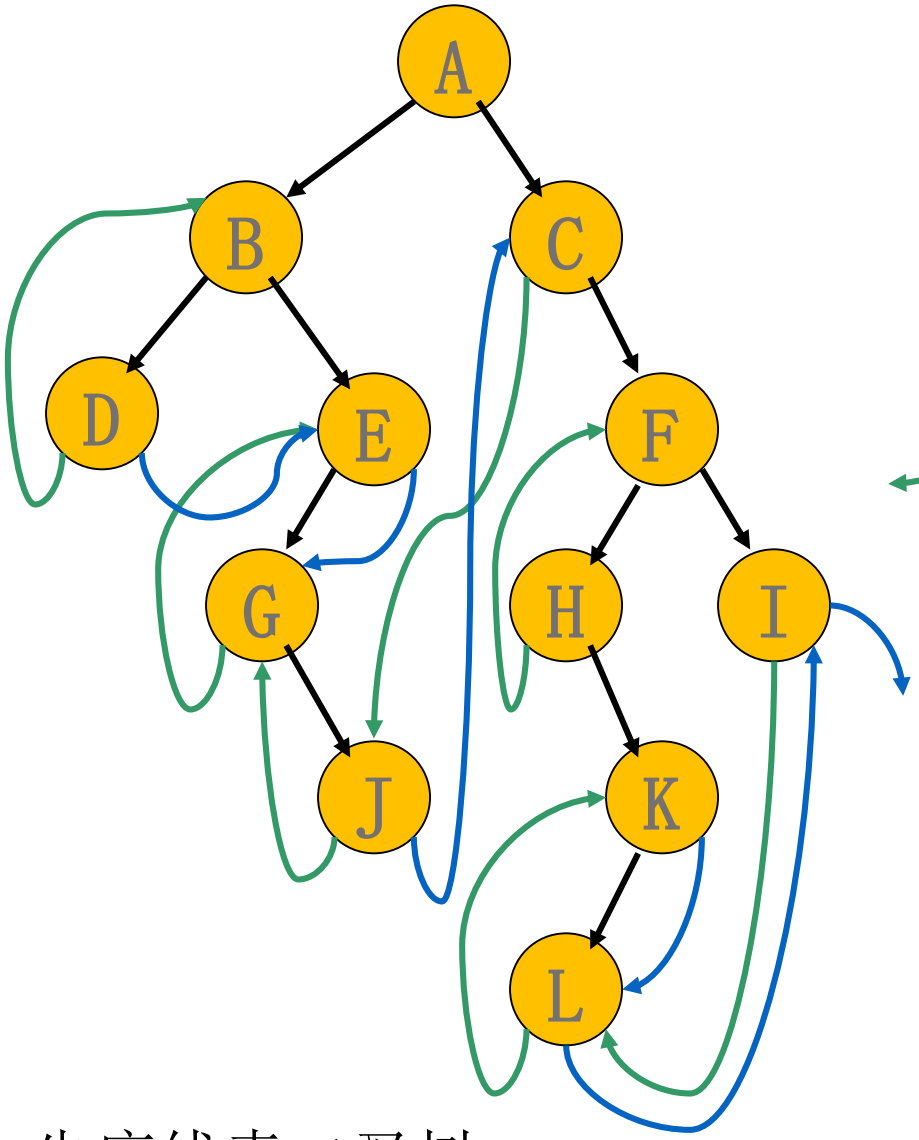
中序序列:
DBGJEACHLKFI

遍历中序线索二叉树(不带头结点)

```
void inorderI_Thr(BiThrTree T){
    BiThrTree p=T; if (p) return;
    while (p->LTag==Link) p=p->lchild;
    printf("%c",p->data);//中序遍历的第一个节点
    while (p->rchild){p不是最后一个结点
       if(p->RTag==Link){//有右子树
            p=p->rchild;
            while(p->LTag==Link) p=p->lchild;
        }else//无右子树
        p=p->rchild;          printf("%c",p->data);
    }
}
```

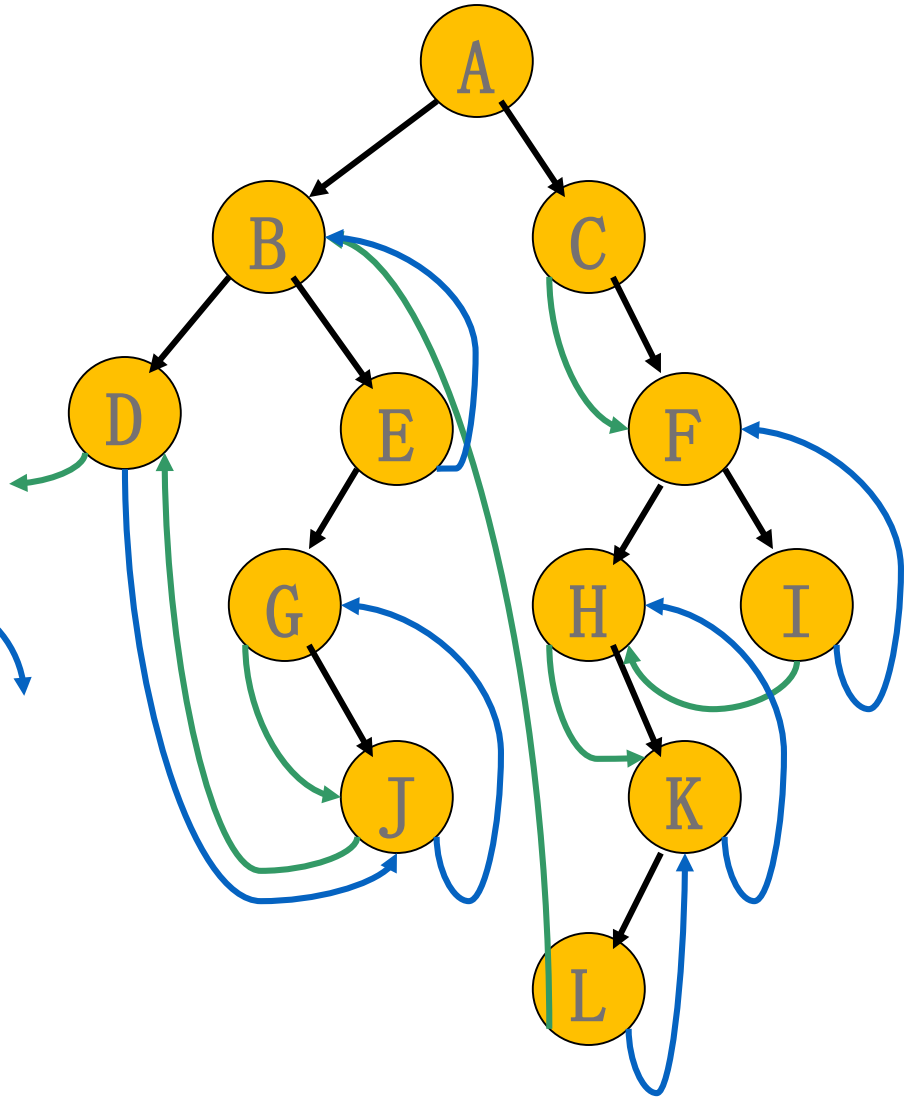
```
void inorder2_Thr(BiThrTree T){
        BiThrTree p=T;
        while (p){//
            while(p->LTag==Link) p=p->lchild;
            printf("%c",p->data);
            while(p->RTag==Thread && p->rchild){
                  p=p->rchild;
                  printf("%c",p->data);
            }
           p=p->rchild;
        }
}
```





先序线索二叉树

ABDEGJCFHKLI

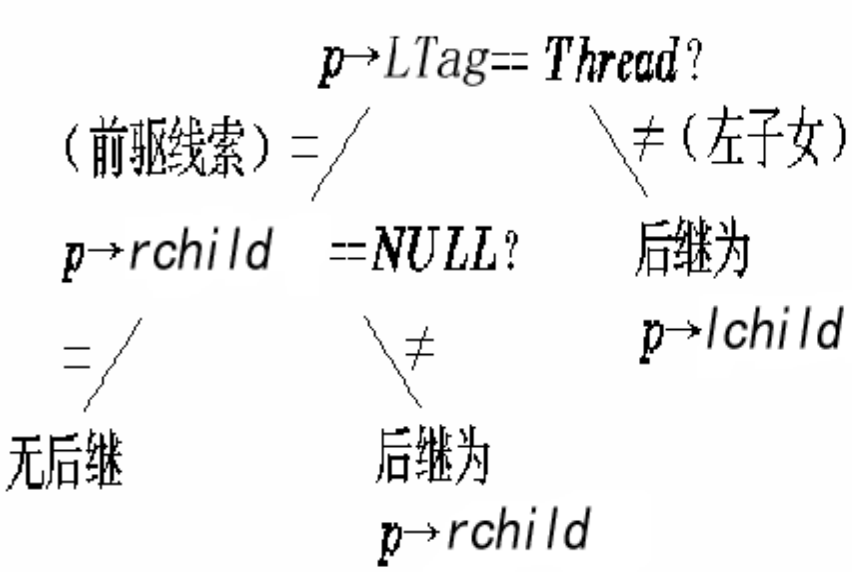


后序线索二叉树

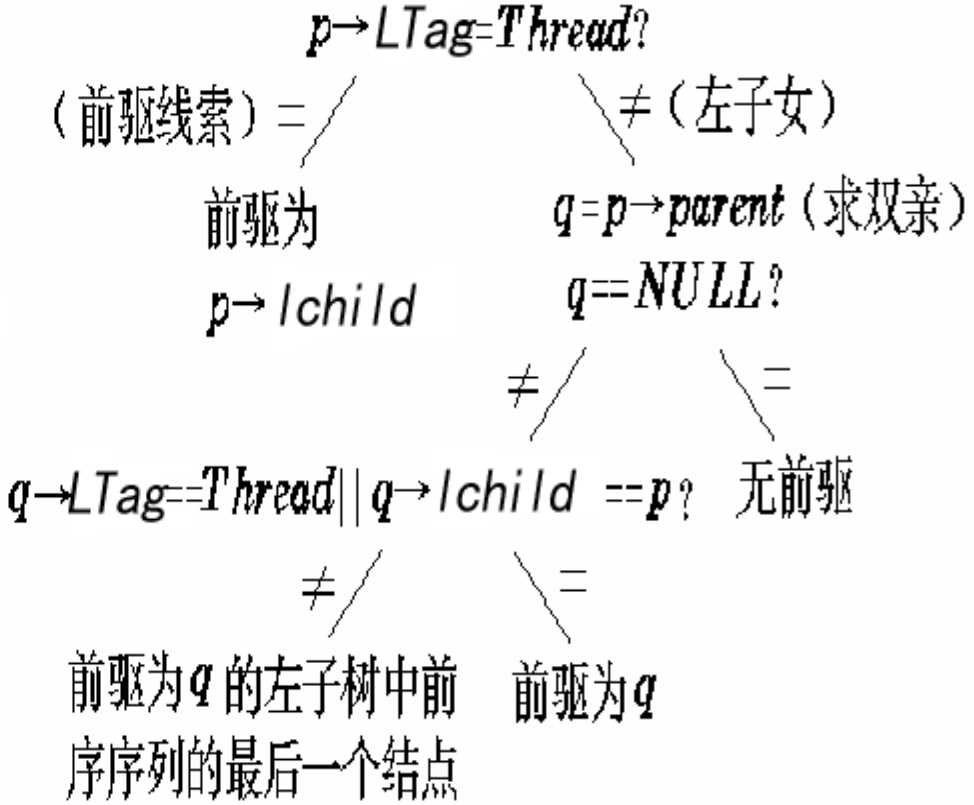
DJGEBLKHIFCA

先序线索二叉树

- 在先序线索二叉树中寻找当前结点的后继与前驱



（a）求结点 p 的后继



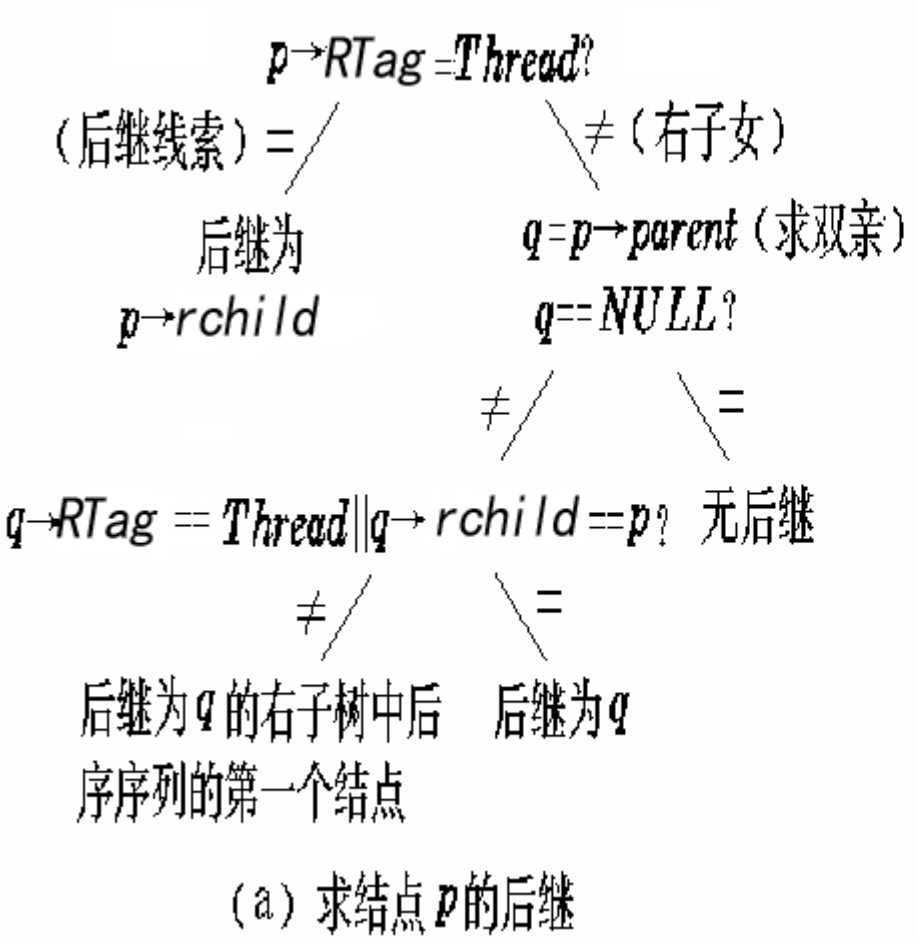
（b）求结点 p 的前驱

遍历先序线索二叉树(不带头结点)

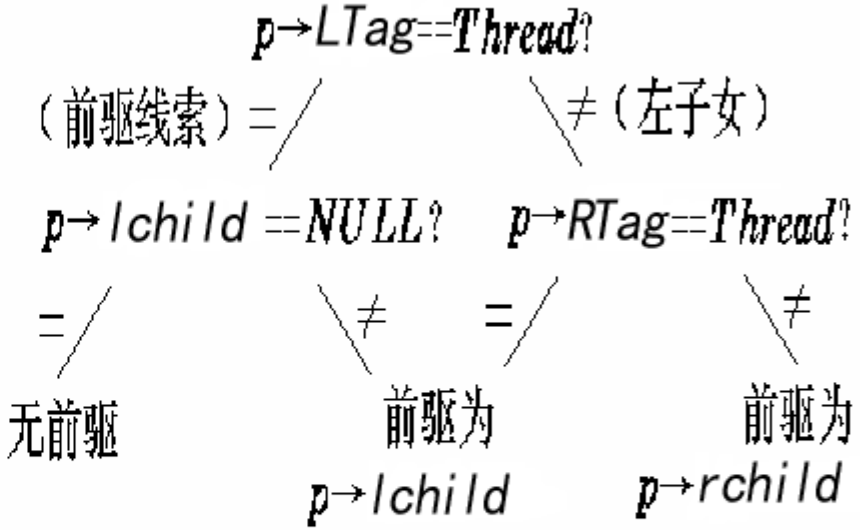
```
void preorder_Thr(BiThrTree T){
    BiThrTree p=T;
    printf("%c",p->data);
    while (p->rchild){
        if (p->LTag==Link)
            p=p->lchild;
        else
            p=p->rchild;
        printf("%c",p->data);
    }
}
```

后序线索二叉树

- 在后序线索化二叉树中寻找当前结点的后继



(a) 求结点 p 的后继



(b) 求结点 p 的前驱

遍历后序线索二叉树(不带头结点)

```
void postorder_Thr(TriThrTree T){
  TriThrTree f,p=T;
   do{
      while (p->LTag==Link)
            p=p->lchild;
      if (p->RTag==Link)
            p=p->rchild;
    } while (p->LTag!=Thread ||p-RTag!=Thread);
    printf("%c",p->data);
}
```

```
while (p!=T)
  { if (p->RTag==Link)
      { f=p->parent;
       if (f->RTag==Thread || p==f->rchild) p=f;
       else{ p=f->rchild;
           do{ while(p->LTag==Link) p=p->lchild;
               if (p->RTag==Link) p=p->rchild;
                   }while (p->LTag!=Thread || p-
>RTag!=Thread);
           }   }
    else p=p->rchild;
    printf("%c",p->data);
  }}
```

二叉树的线索化

- 将未线索过的二叉树给予线索

  在某种（先、中、后、层序）遍历的过程中，修改空指针的值为指向前驱或后继的线索。

- 中序线索化

  后继线索化——处理前驱结点

  - a. 如果无前驱结点或前驱结点的右指针域为非空，也不必加线索
  - b. 如果前驱结点的右指针域为空，则把当前结点的指针值赋给前驱结点的右指针域。

  前驱线索化——处理当前结点

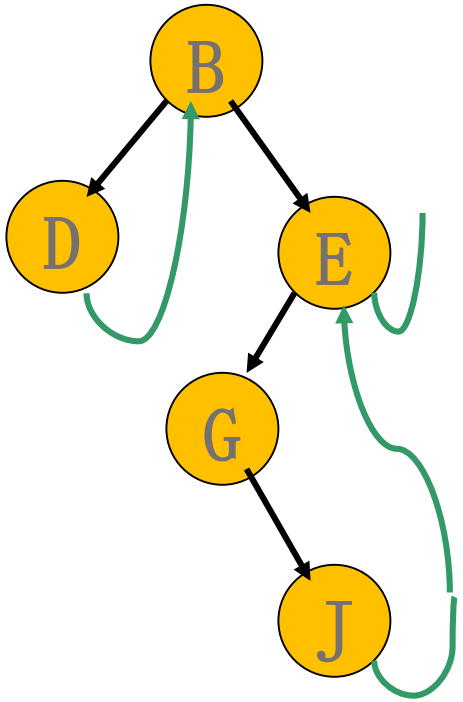
  - 如果当前结点的左指针域为空，则把前驱结点的指针值赋给当前结点的左指针域。

中序线索化

- 中序线索化递归程序

```
void InThreading(BiThrTree P){
  if (P){
    InThreading(P->lchild);
    后继线索化(处理前驱结点，给前驱结点加后继线索)
    前驱线索化 (处理当前结点，给当前结点加前驱线索)
     pre=P;//pre是前驱结点，P是当前结点
     InThreading(P->rchild);
   }
}
```

中序线索化



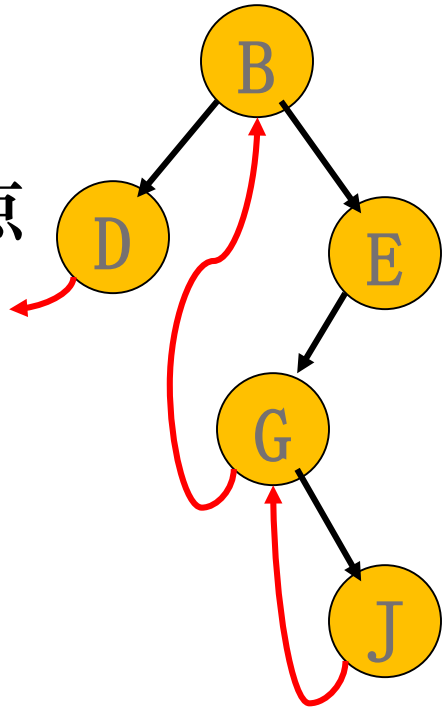
- 后继线索化——处理前驱结点

```
void InThreading(BiThrTree P){
  if (P){
    InThreading(P->lchild);
    if (pre && !pre->rchild){
        pre->RTag=Thread; pre->rchild=P;
    } //后继线索化
    前驱线索化。。。见下页
      pre=P;
      InThreading(P->rchild);
    }
}
```

中序线索化

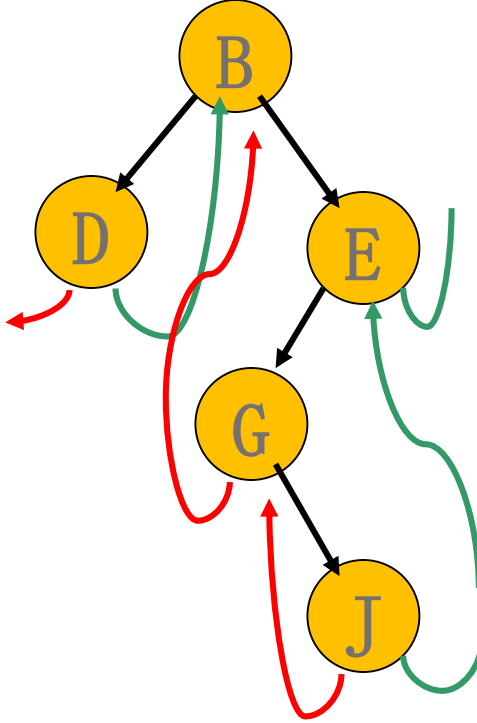
- 前驱线索化——处理后继（当前）结点

```
void InThreading(BiThrTree P){
  if (P){
      InThreading (P->lchild);
      后继线索化。。。见上页
        if (!P->lchild){
        P->LTag=Thread; P->lchild=pre;
      } //前驱线索化
        pre=P;
        InThreading(P->rchild);
    }
}
```



中序线索化

```
void InThreading(BiThrTree P){
  if (P){
      InThreading (P->lchild);
      if (!P->lchild){
         P->LTag=Thread; P->lchild=pre;
      } //前驱线索化
      if (pre && !pre->rchild){
         pre->RTag=Thread; pre->rchild=P;
      } //后继线索化
      pre=P;
       InThreading(P->rchild);
    }
}
```



先序线索化

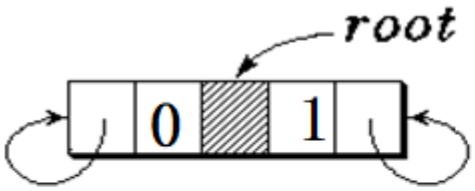
```
void PreThreading(BiThrTree P){
   if (P){
    if (!P->lchild){
       P->LTag=Thread;P->lchild=pre;}
      if (!P->rchild) P->RTag=Thread;
    if (pre && pre->RTag==Thread) pre->rchild=P;
      pre=P;
      if (P->LTag==Link) PreThreading(P->lchild);
      if (P->RTag==Link) PreThreading(P->rchild);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

后序线索化

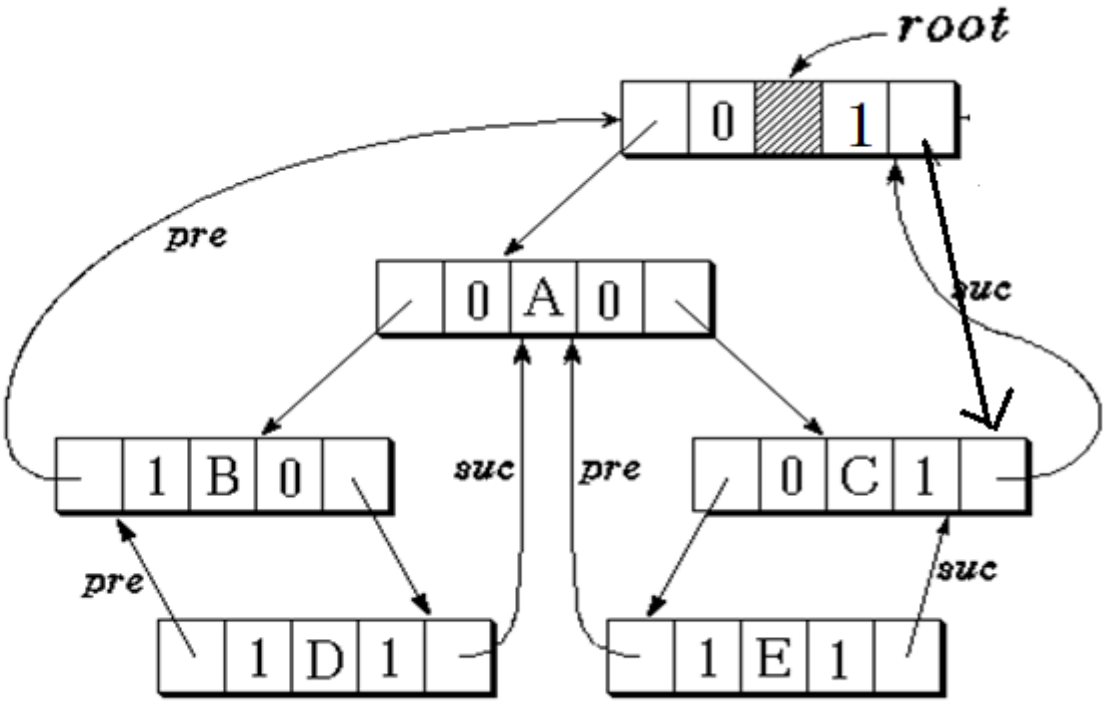
```
void PostThreading(TriThrTree P){
   if (P){
     PostThreading(P->lchild);
        PostThreading(P->rchild);
        if (!P->lchild){
        P->LTag=Thread; P->lchild=pre; }
        if (!P->rchild) P->RTag=Thread;
        if (pre && pre->RTag==Thread) pre->rchild=P;
        pre=P;
     }
 }
```

中序线索化

- 在二叉树的线索链表上加一个头结点，另其lchild域的指针指向根结点；其rchild域的指针指向中序遍历的最后一个结点。形成了一个双向线索链表。
- 二叉树中序序列中的第一个结点的lchild域的指针和最后一个结点的rchild域的指针均指向头结点。



（a）空二叉链表

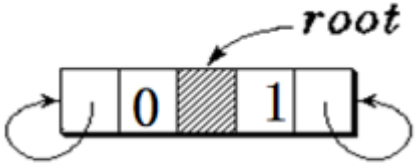


（b）非空二叉链表

带头结点的中序线索化

- 算法6.6： 指针pre始终指向当前访问结点的前驱结点

```
int InOrderThreading(BiThrTree &Thrt, BiThrTree T){
    if (!(Thrt = (BiThrTree)malloc(sizeof(BiThrNode))))
exit(OVERFLOW);
    Thrt->LTag = Link; Thrt->RTag=Thread;
    Thrt->rchild=Thrt;
    if (!T) Thrt->lchild = Thrt;
    else {Thrt->lchild=T;       pre=Thrt;
          InThreading(T);
          pre->rchild = Thrt; pre->RTag=Thread;
    Thrt->rchild = pre;
    }return OK;}
```



（a）空二叉链表

中序遍历带头结点的中序线索树

- 算法6.5

```
Status InOrderTraverse_Thr(BiThrTree T, Status
(*Visit)(TElemType e){
    p=T->lchild;
    while (p!=T){ //是否指向头结点
            while(p->LTag==Link) p=p->lchild;
            if (!Visit(p->data) return ERROR;
            while(p->RTag==Thread && p->rchild !=T){
                    p=p->rchild;
                    Visit(p->data);//访问后继结点
            }  p=p->rchild;
    }return OK;
}//InOrderTraverse_Thr
```

中序线索化(带头结点)

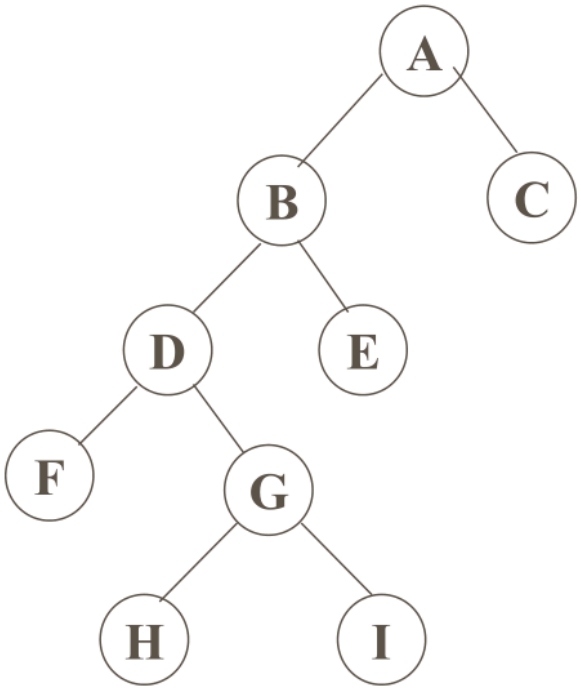
算法6.7

```
void InThreading2 (BiThrTree P){
  if(P){ InThreading2 (P->lchild);
          if (!P->lchild)
           { P->LTag=Thread; P->lchild=pre;}
          if (!P->rchild) P->RTag=Thread;
          if (pre->RTag==Thread) pre->rchild=P;
          pre=P;
          InThreading2 (P->rchild);
        }
  }//注：pre初始=头结点，见算法6.6。
```

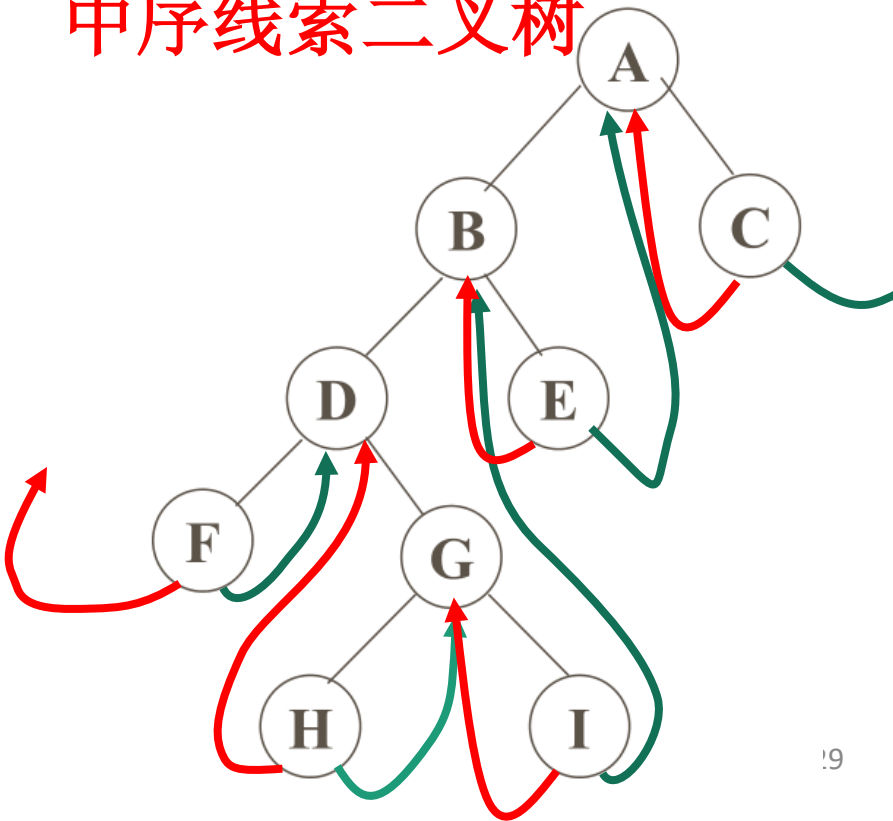
例1：画出线索二叉树

(1) 写出对下图所示的二叉树它们进行先序、中序和后序遍历时得到的结点序列;

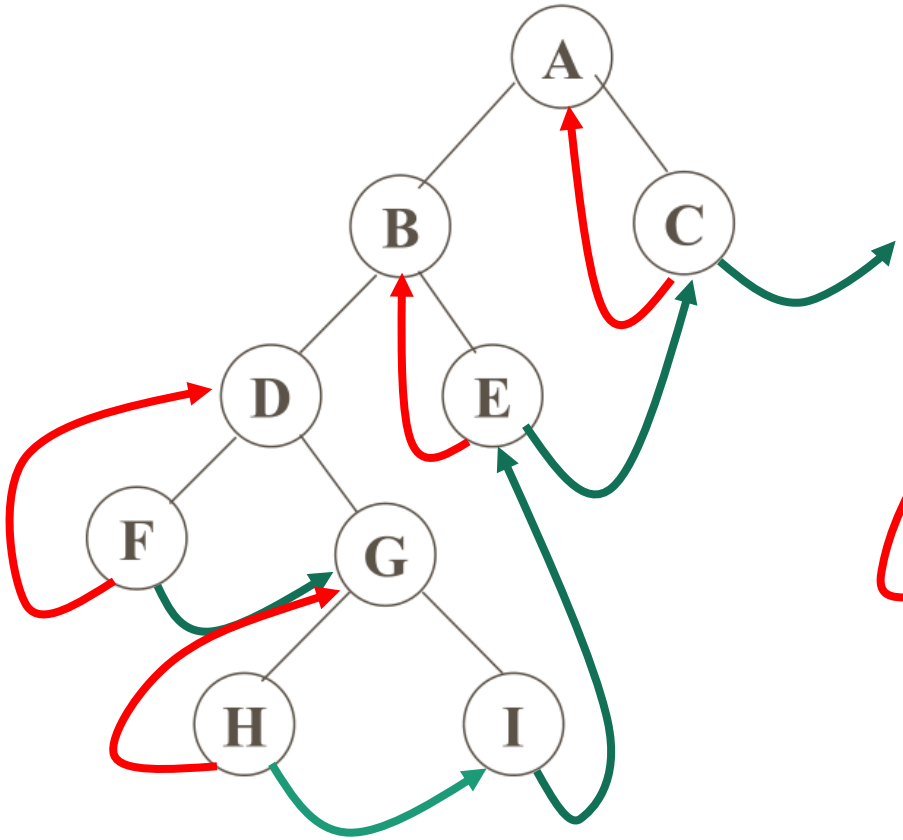
(2) 分别画出它们的先序、中序和后序线索二叉树。



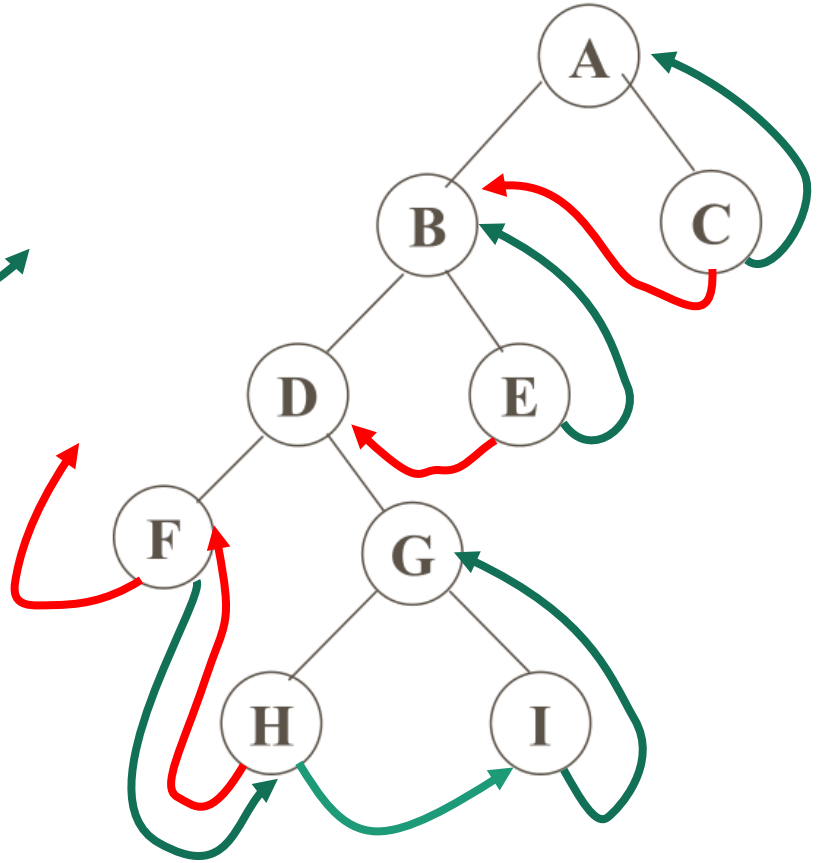
中序线索二叉树



先序线索二叉树



后序线索二叉树



6.5 树和森林

1. 树的存储结构

1)多重链表（标准存储结构）

定长结构 （n为树的度）指针利用率不高

data	$child_1$	$child_2$	$child_3$	·····	$child_n$

一般采用定长结构

不定长结构 d为结点的度，节省空间，但算法复杂

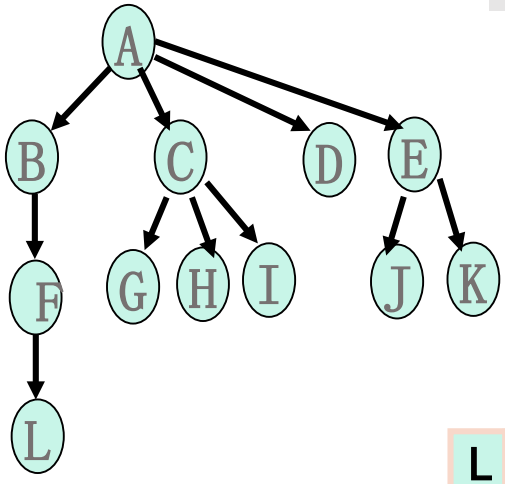
data	*d*	$child_2$	$child_3$	·····	$child_d$

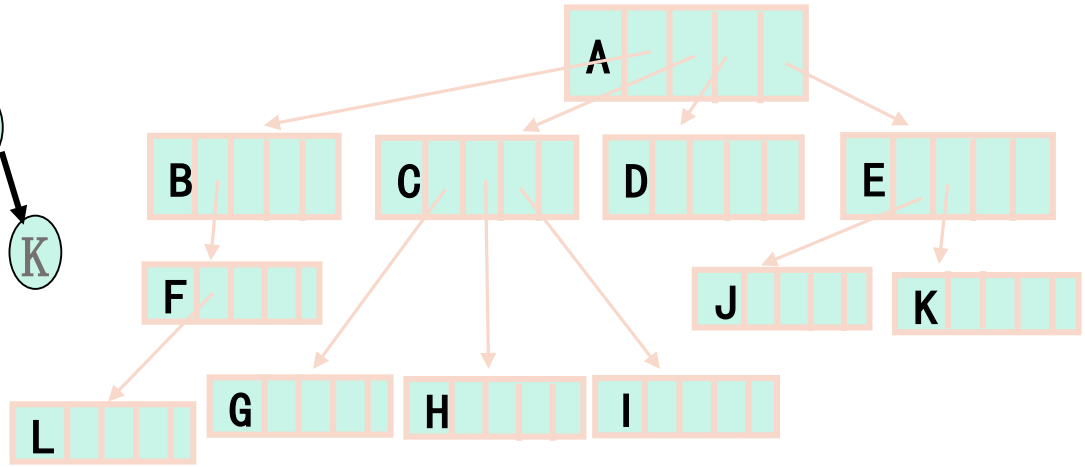
如有n个结点，树的度为k，则共有n*k个指针域，只有n-1个指针域被利用，而未利用的指针域为：n*k-(n-1) =n(k-1)+1，未利用率为：(n(k-1)+1)/nk ＞ n(k-1)/nk=(k-1)/k

二次树：1/2 ；三次树：2/3 ；四次树：3/4

树的度越高，未利用率越高，由于二叉树的利用率较其他树高，因此用二叉树。

定长结构

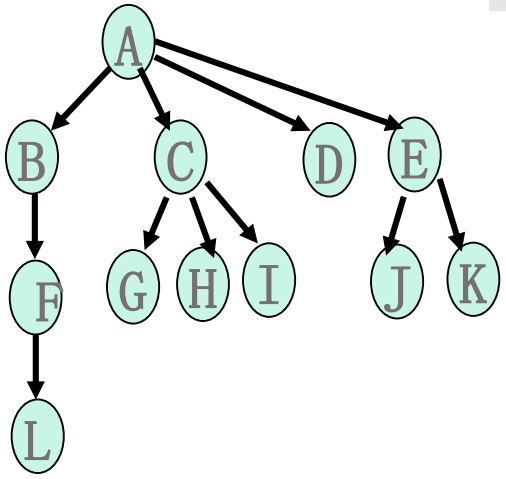


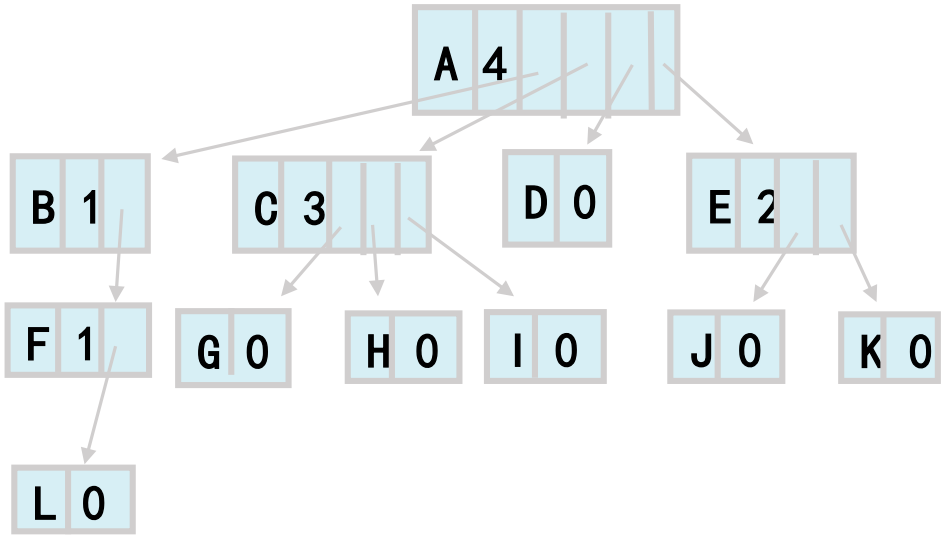


类型定义：

```
#define N 4
typedef struct dNode{
    ElemType data;
    struct dNode *child[N];
}dNode,*dTree;
```

不定长结构





不定长结构类型定义：（动态分配）

子节点数目不同，存储空间需动态分配

```
typedef struct dNode{
    ElemType data;
    int degree;
    struct dNode **child;
}dNode,*dTree;
```

2)常用的其他几种存储结构

- 双亲表示法

  用结构数组——树的顺序存储方式

  找双亲方便，找孩子难

- 孩子链表表示法

  顺序和链式结合的表示方法

  找孩子方便，找双亲难

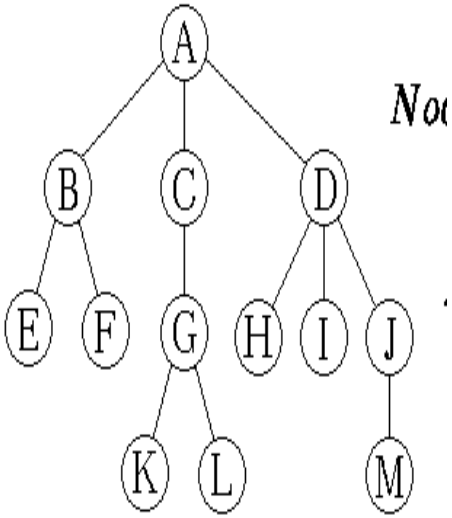
  若用带双亲的孩子链表表示，则找孩子找双亲都较方便

- 孩子兄弟表示法

  找孩子容易，若增加parent域，则找双亲也较方便。

■双亲表示类型定义

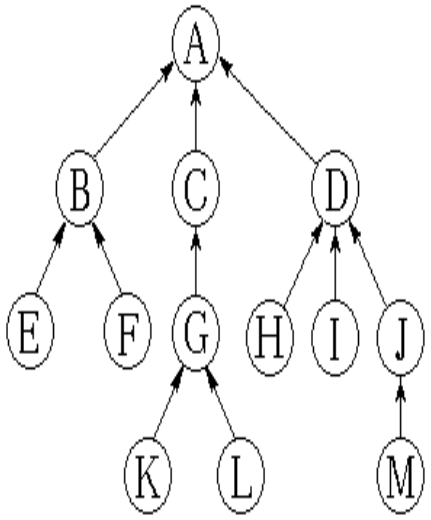
```
#define MAX_TREE_SIZE  100
typedef struct PTNode{
    TElemType data;
    int parent;
} PTNode;
typedef struct{
    PTNode nodes[MAX_TREE_SIZE];
    int r,n;  //根位置和结点数
}PTree;
```



(a) 树

NodeList	1	2	3	4	5	6	7	8	9	10	11	12	13
data	A	B	E	F	C	G	K	L	D	H	I	J	M
parent	0	1	2	2	1	5	6	6	1	9	9	9	12

(b) 双亲表示数组

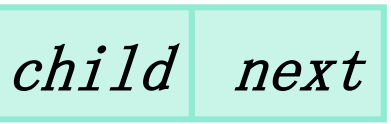


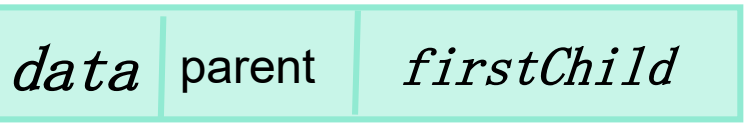
(c) 双亲表示图解

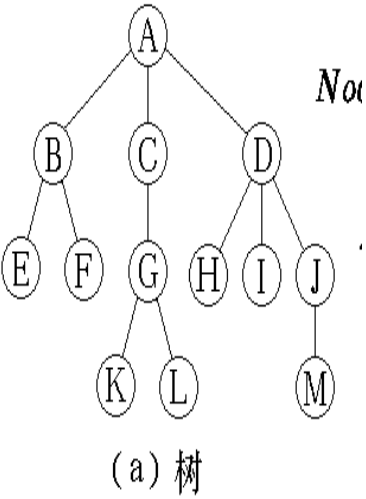
■孩子链表表示法

```
typedef struct CTNode{
int  child;
struct CTNode  *next;
}*ChildPtr;
typedef struct {
ElemType  data;
int parent;
ChildPtr  firstchild;
}CTBox;
```

```
typedef struct{
    CTBox nodes[MAX_TREE_SIZE];
    int n,r;
}CTree;
```

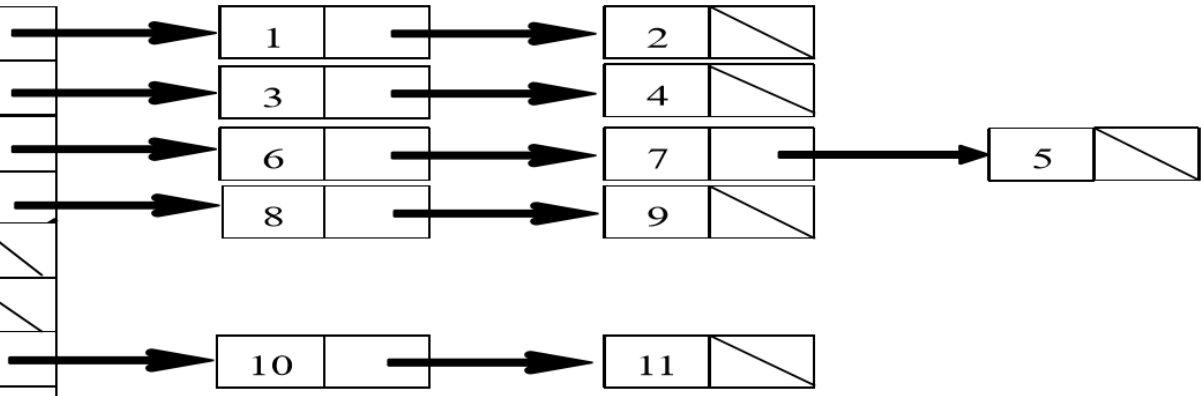






(a) 树

索引	值	父结点	子结点
0	A		
1	B	0	
2	C	0	
3	D	1	
4	E	1	
5	F	2	
6	G	2	
7	H	2	
8	I	3	
9	J	3	
10	K	6	
11	L	6	

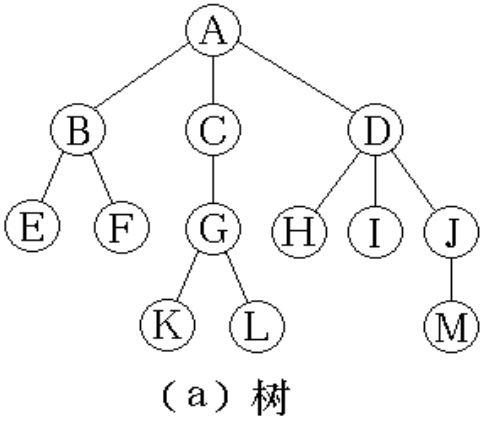


带双亲的孩子链表表示

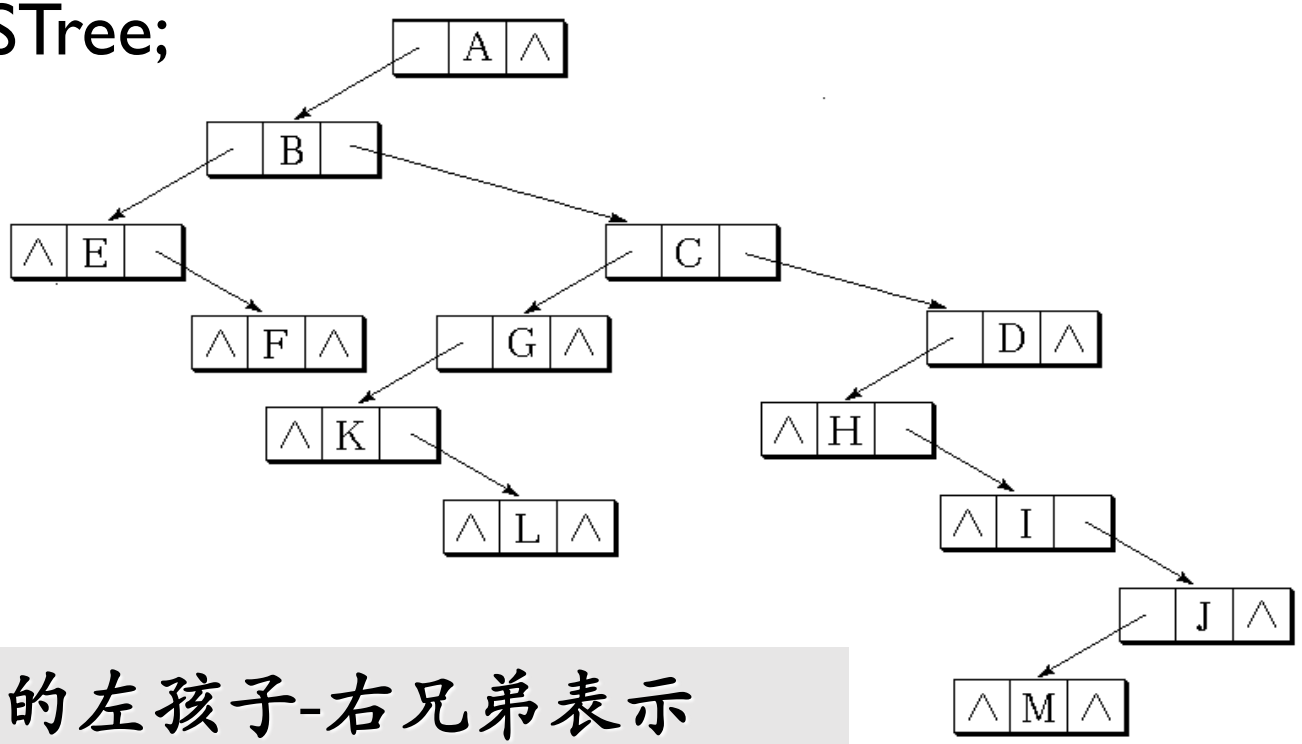
■孩子兄弟表示法（二叉链表表示法）

data	*firstChild*	*nextSibling*

```
typedef struct CSNode{
ElemType  data;
struct CSNode  *firstChild, *nextSibling;
}CSNode,*CSTree;
```

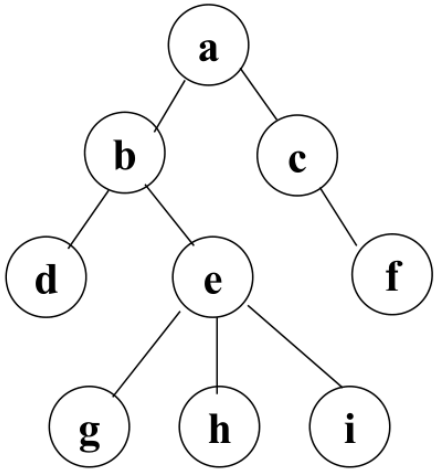


（a）树

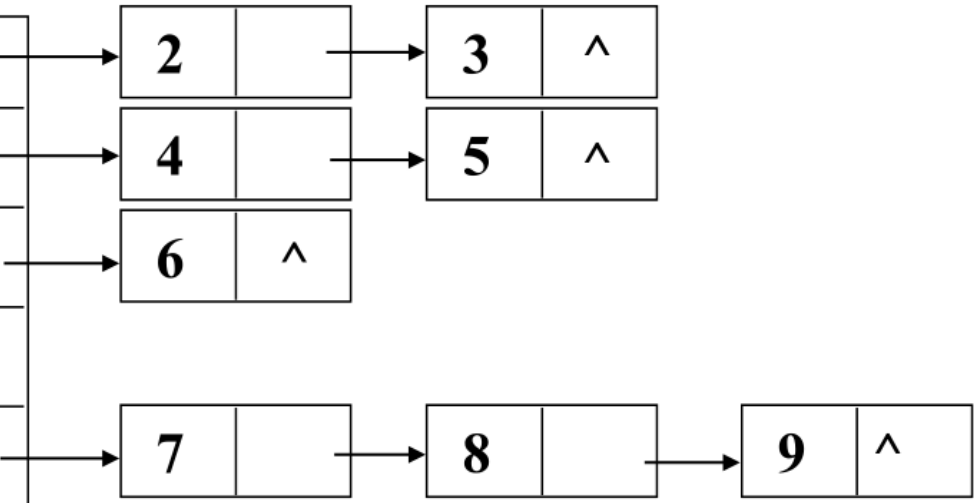


树的左孩子-右兄弟表示

例2.用带双亲的孩子链表表示下图所示的树

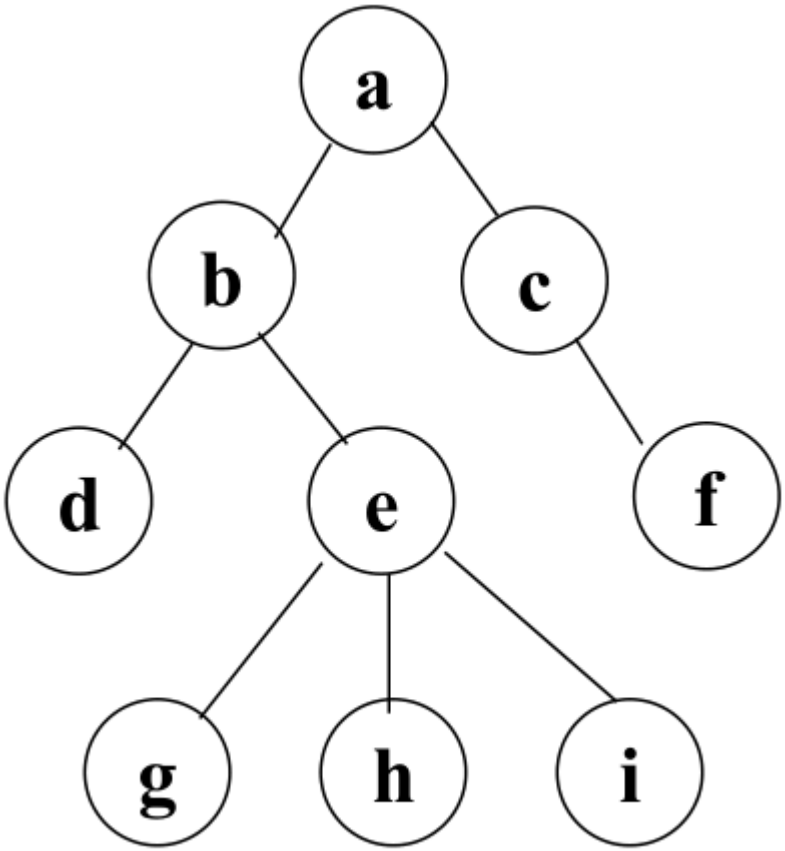


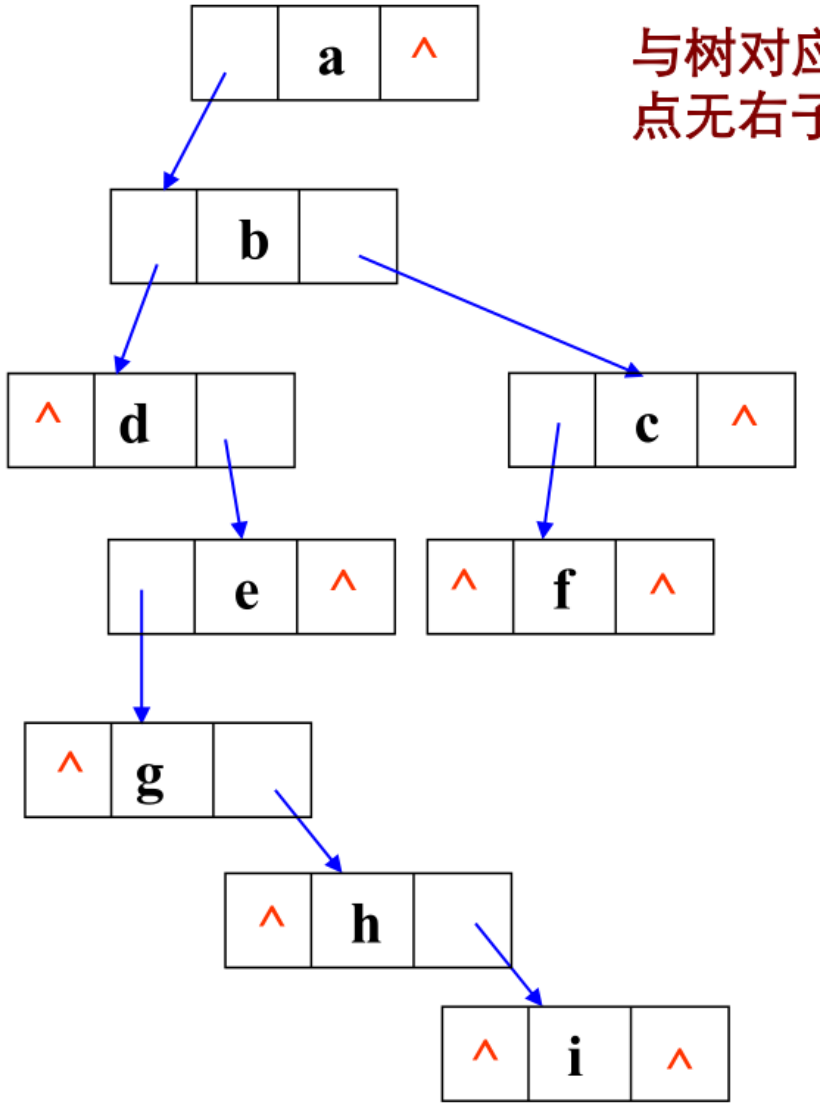
	data	parent	fc
1	a	0	
2	b	1	
3	c	1	
4	d	2	^
5	e	2	
6	f	3	^
7	g	5	^
8	h	5	^
9	i	5	^



PCTree.r=1
PCTree.n=9

例3：用孩子兄弟表示法表示下图所示的树(重点掌握)





2.森林、树与二叉树的转换

- 森林：

  m（m≥0）棵互不相交的树的集合。

- 树：

  Tree=(root,F), 其中：root是树的根结点，F是森林,是其子树的集合，F=(T1,T2,… Tm)，

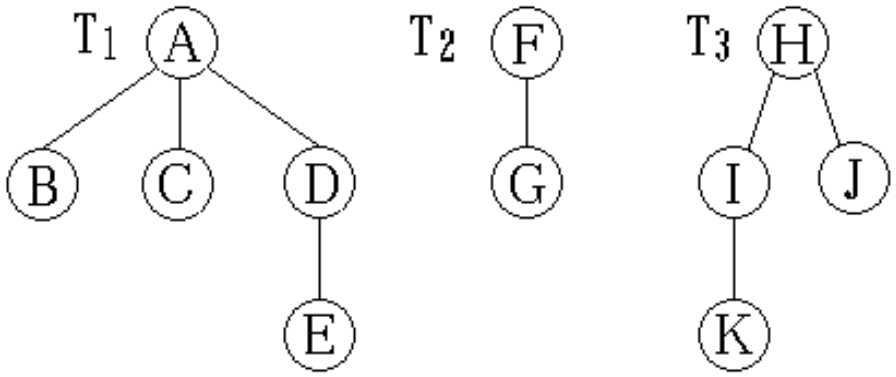
  其中Ti=(ri,Fi)称作根root的第i棵子树；

  当m≠0时，树根和子树森林存在下列关系：

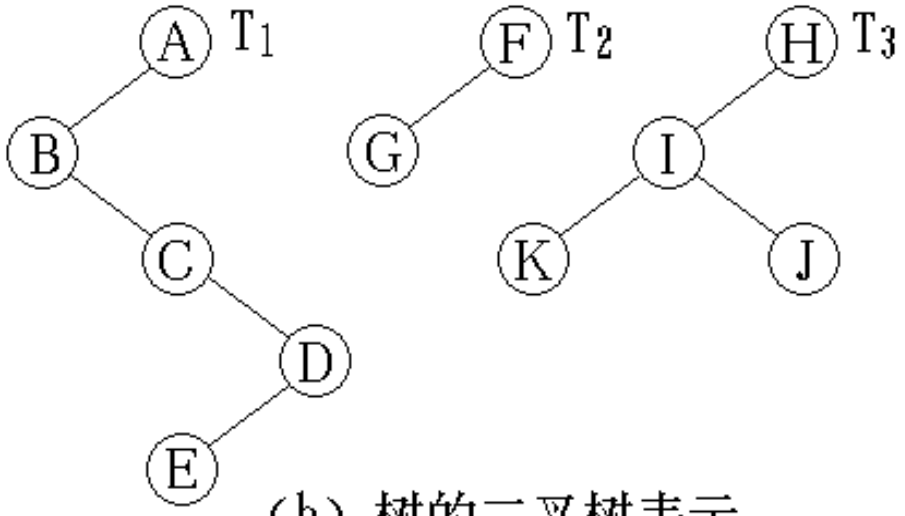
  RF= {<root,ri>|i=1,2,…,m, m>0}

- 森林和树存在相互递归的关系

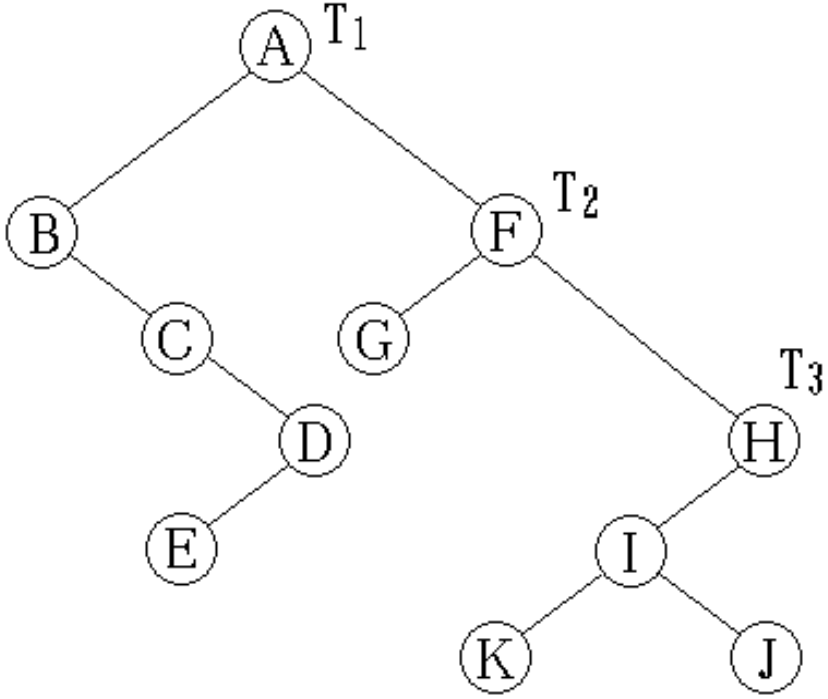
■森林与二叉树的对应关系



（a）3棵树的森林



（b）树的二叉树表示



（c）森林的二叉树表示

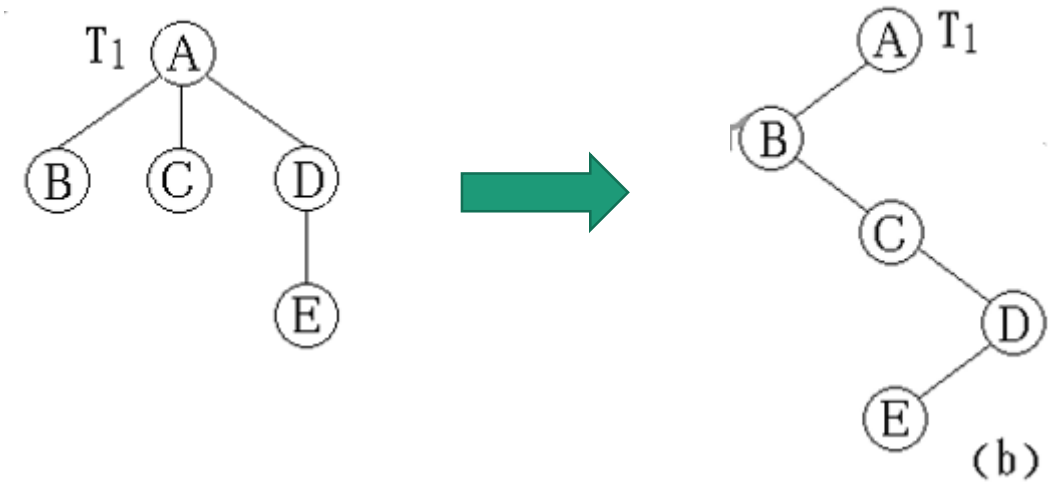
(I) 树转化成二叉树的简单方法

① 在同胞兄弟之间加连线；

② 保留结点与第一个孩子之间的连线，去掉其余连线；

③ 顺时针旋转45度。

④ 以根结点为轴；左孩子(只有一个孩子)不再旋转。

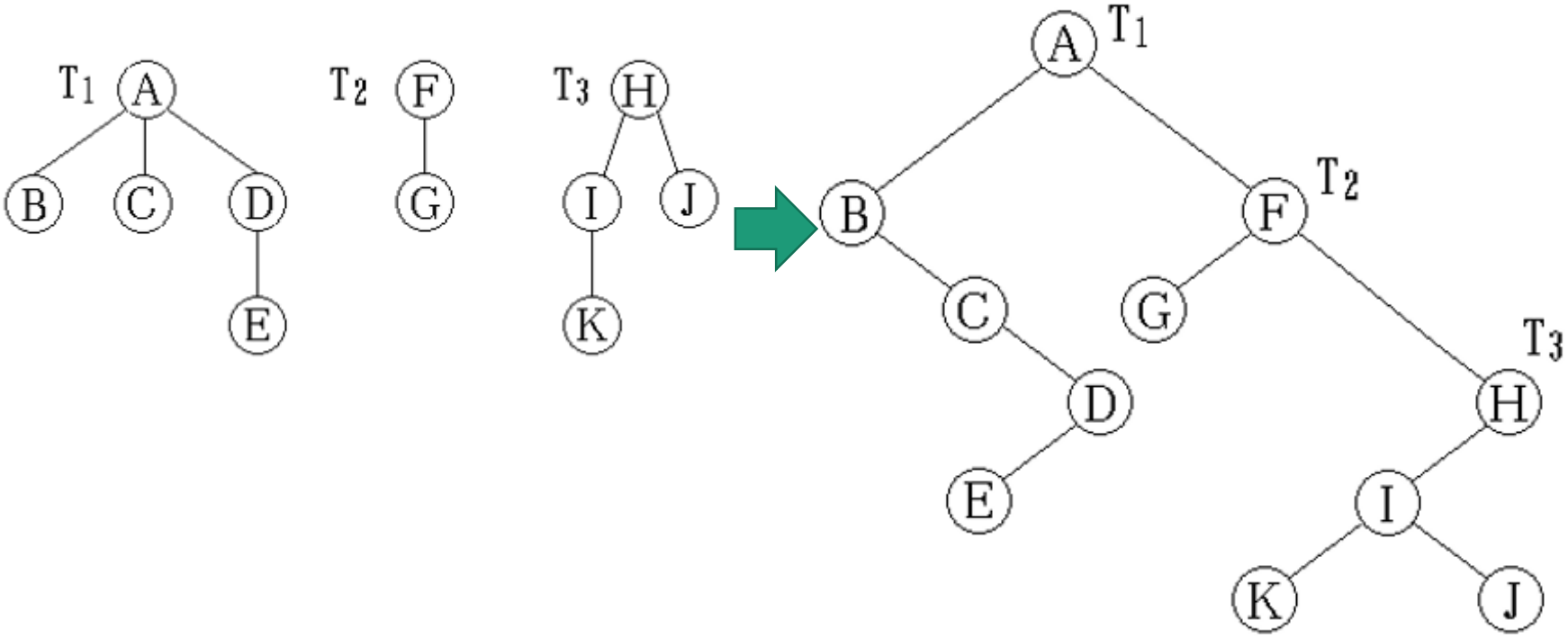


(2) 森林转化成二叉树

① 将森林中的每棵树转换成对应的二叉树；

② 将森林中已经转换成的二叉树的各个根视为兄弟，各兄弟之间自第一棵树根到最后一棵树根之间加连线；

③ 以第一棵树的根为轴，顺时针旋转45度。



(3) 森林转化成二叉树的规则

若F为空，即n = 0，则对应的二叉树B为空二叉树。

若F不空，则对应二叉树B的根root (B)是F中第一棵树T_1 的根root (T_1)；其左子树为B ($T_{11}, T_{12}, \ldots, T_{1m}$)，其中，$T_{11}, T_{12}, \ldots, T_{1m}$ 是root (T_1)的子树；其右子树为B ($T_2, T_3, \ldots, T_n$)，其中，$T_2, T_3, \ldots, T_n$ 是除T_1外其它树构成的森林。

(4) 二叉树转换为森林的规则

① 如果B为空，则对应的森林F也为空。

② 如果B非空，则F中第一棵树T_1的根为root；T_1的根的子树森林$\{T_{11}, T_{12}, \cdots, T_{1m}\}$是由root的左子树 LB 转换而来，F 中除了 T_1 之外其余的树组成的森林$\{T_2, T_3, \cdots, T_n\}$ 是由 root 的右子树 RB 转换而成的森林。

3. 树和森林的遍历

■树的遍历（先根遍历、后根遍历、层次遍历）

（1）先根遍历：先根访问树的根结点，然后依次先根遍历根的每棵子树　ABEFCGKLDHIJM

对应：二叉树的先序遍历

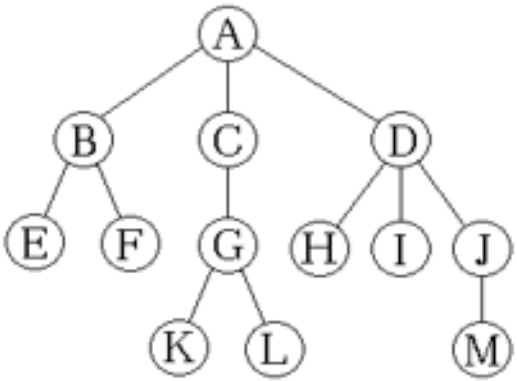
（2）后根遍历：先依次后根遍历每棵子树，然后访问根结点　EFBKLGCHIMJDA

对应：二叉树的中序遍历

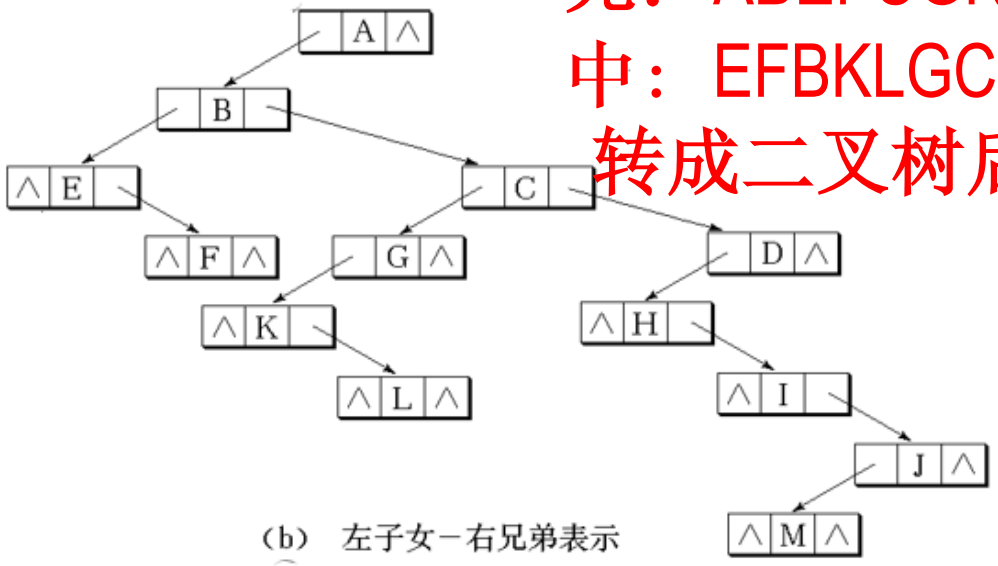
先：ABEFCGKLDHIJM

中：EFBKLGCHIMJDA

转成二叉树后的遍历

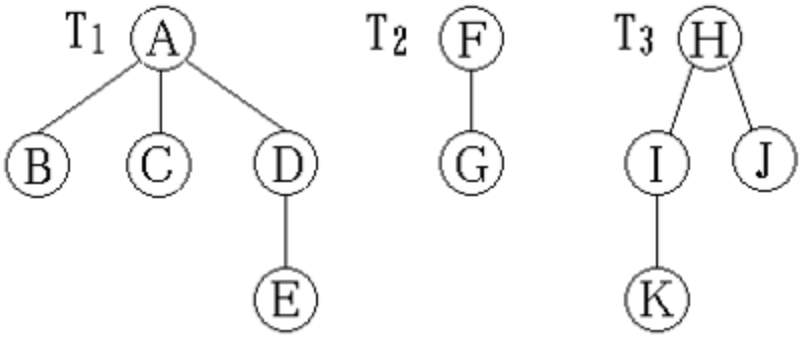


（a）树



（b） 左子女－右兄弟表示

■森林的遍历



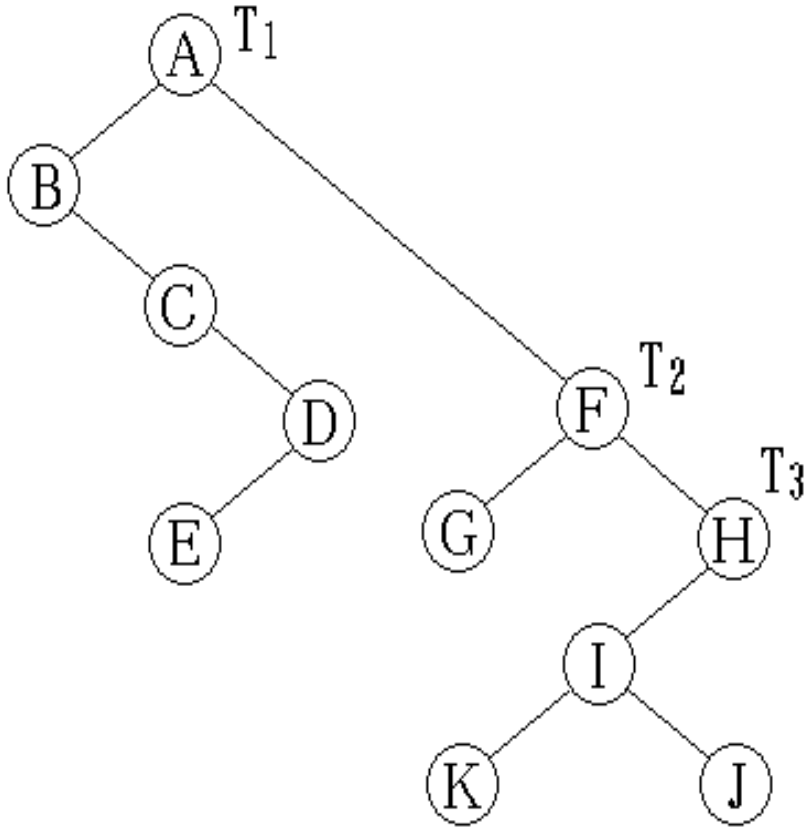
(I) 先根次序遍历的规则：

若森林F为空，返回；否则

① 访问F的第一棵树的根结点；

② 先根次序遍历第一棵树的子树森林；

③ 先根次序遍历其它树组成的森林。



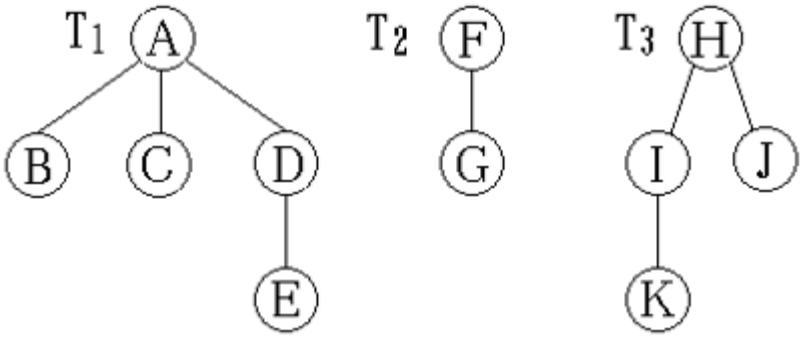
森林的二叉树表示

森林先根遍历：ABCDEFGHIKJ

转换成二叉树后的遍历：(先序)

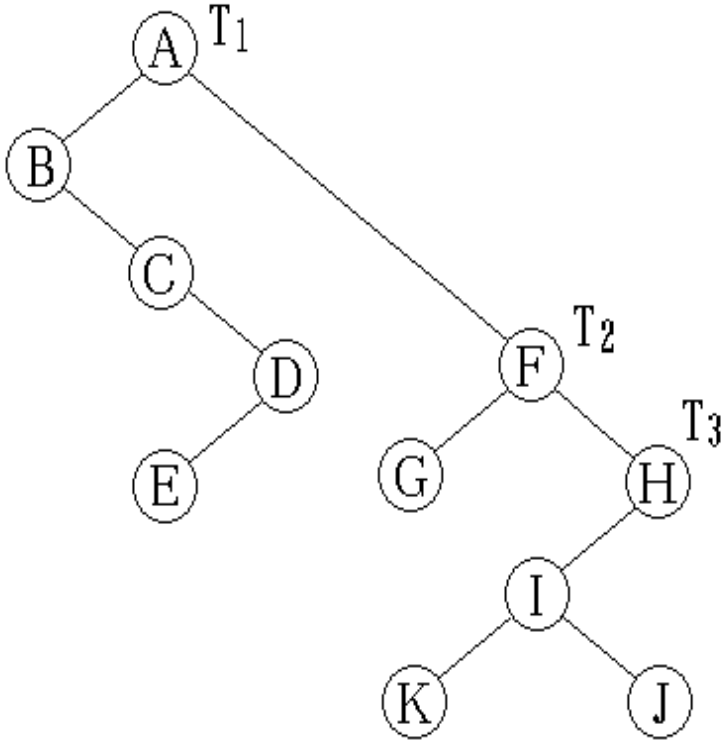
ABCDEFGHIKJ

(2) 中根次序遍历的规则：



若森林F为空，返回；
否则

① 中根次序遍历第一棵树的子树森林；
② 访问F的第一棵树的根结点；
③ 中根次序序遍历其它树组成的森林。

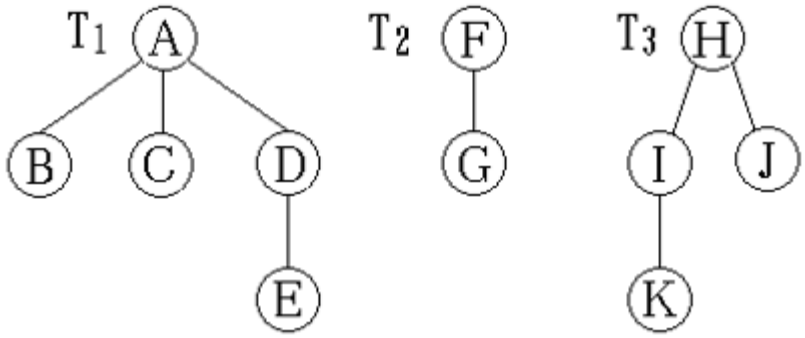


森林中根遍历：BCEDAGFKIJH

转换成二叉树后的遍历：(中序)

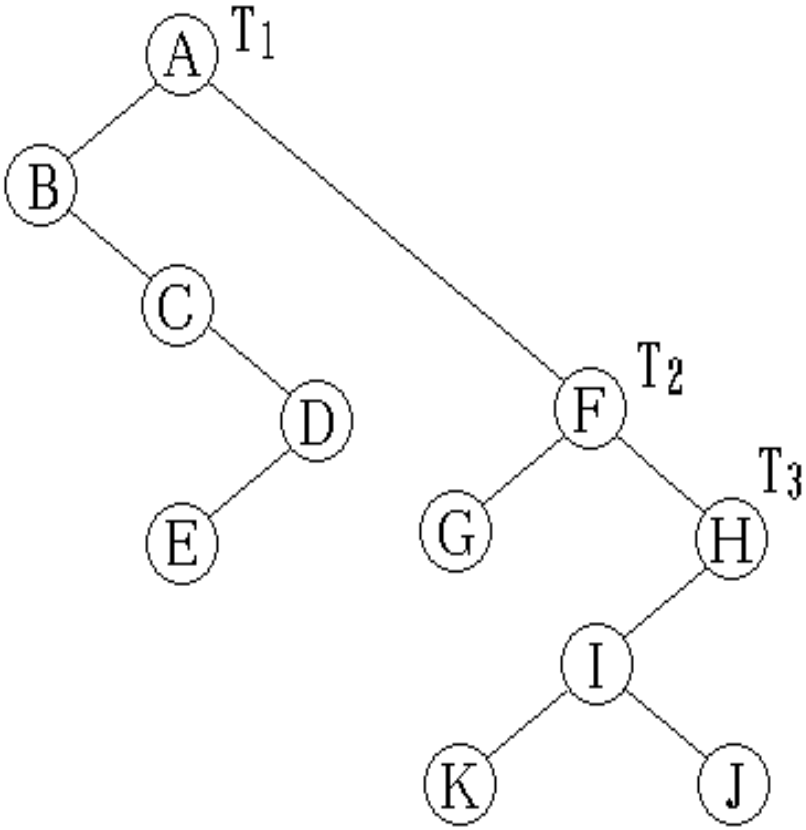
BCEDAGFKIJH

(3) 后根次序遍历的规则：



若森林F为空，返回；
否则

① 后根次序遍历第一棵树的子树森林；
② 后根次序遍历其它树组成的森林；
③ 访问F的第一棵树的根结点。

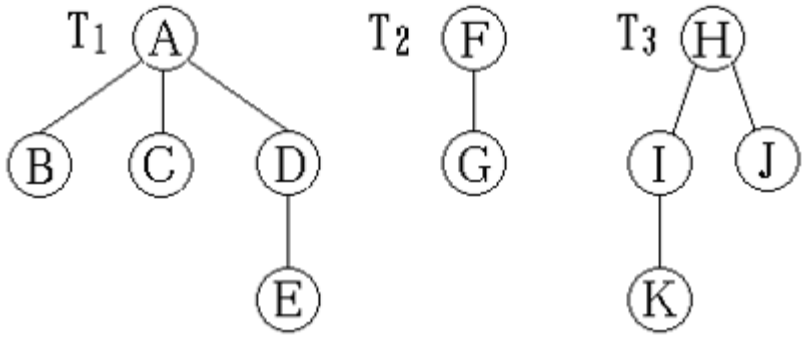


森林后根遍历：

E D C B G K J I H F A

转换成二叉树后的遍历：(后序) E D C B G K J I H F A

(4) 广度优先遍历(层次序遍历)：

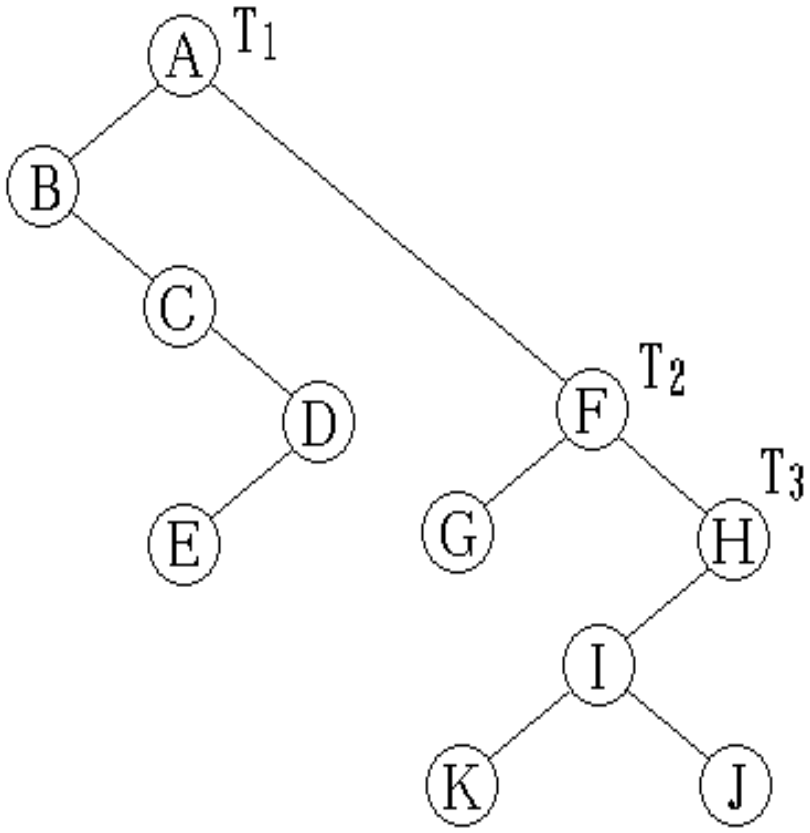


若森林F为空，返回；否则

① 依次遍历各棵树的根结点；

② 依次遍历各棵树根结点的所有孩子；

③ 依次遍历这些孩子结点的孩子结点。



森林的二叉树表示

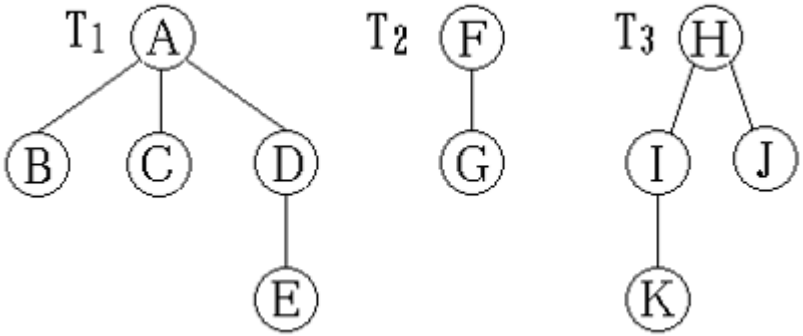
森林层次遍历：AFHBCDGIJEK

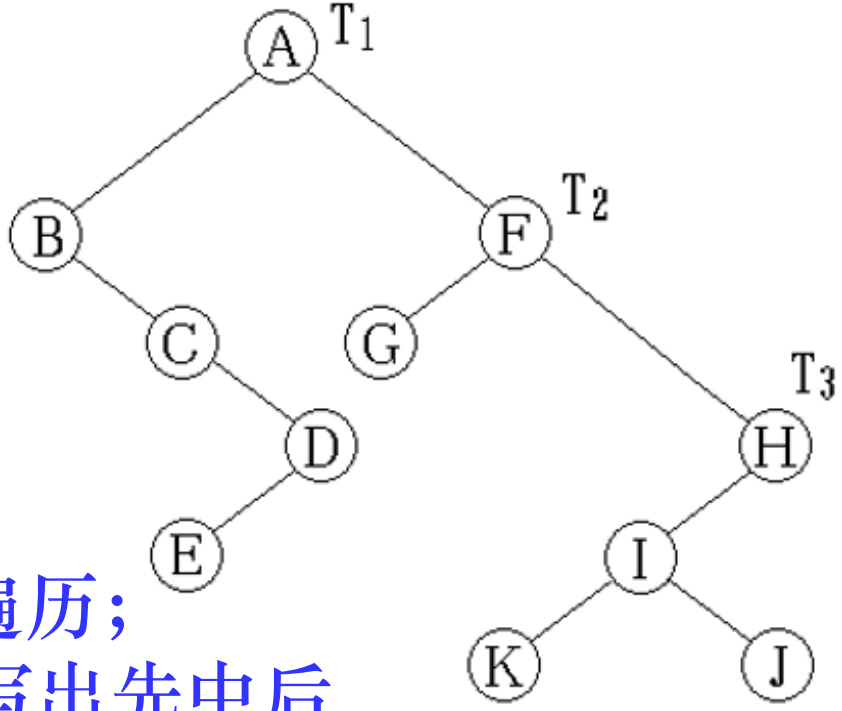
转换成二叉树后的遍历：(右子树看作一层，全部入队)

AFHBCDGIJEK

森林的遍历与转换后的二叉树的遍历

- 先根：对应二叉树的先序遍历
- 中根：对应二叉树的中序遍历
- 后根：对应二叉树的后序遍历





(1)分别写出森林的先中后根遍历；
(2)画出转换后的二叉树；(3)写出先中后序遍历；(4)验证两者之间的上述关系。

例4：树的遍历

- **将一棵树T转换为左孩子/右兄链表示的二叉树B，则T的后根次序遍历序列是B的相应_____序列。**

A. 前序遍历

B. 层次遍历

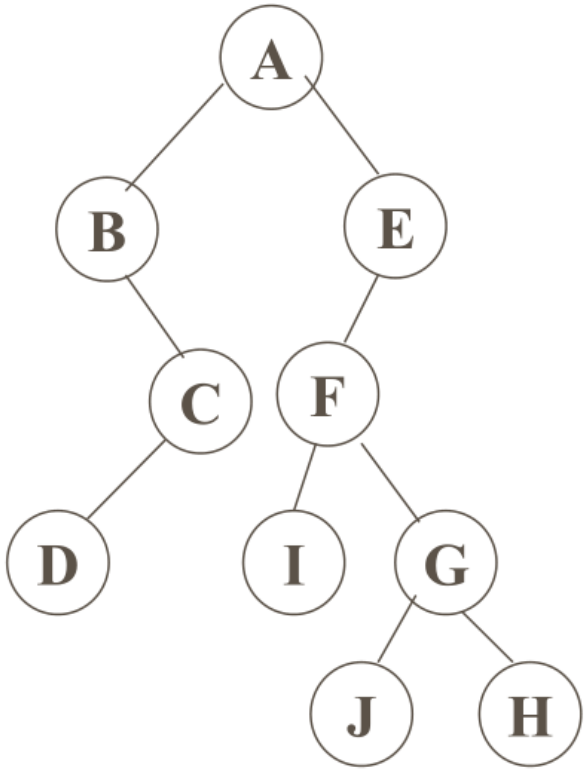
C. 中序遍历

D. 后序遍历

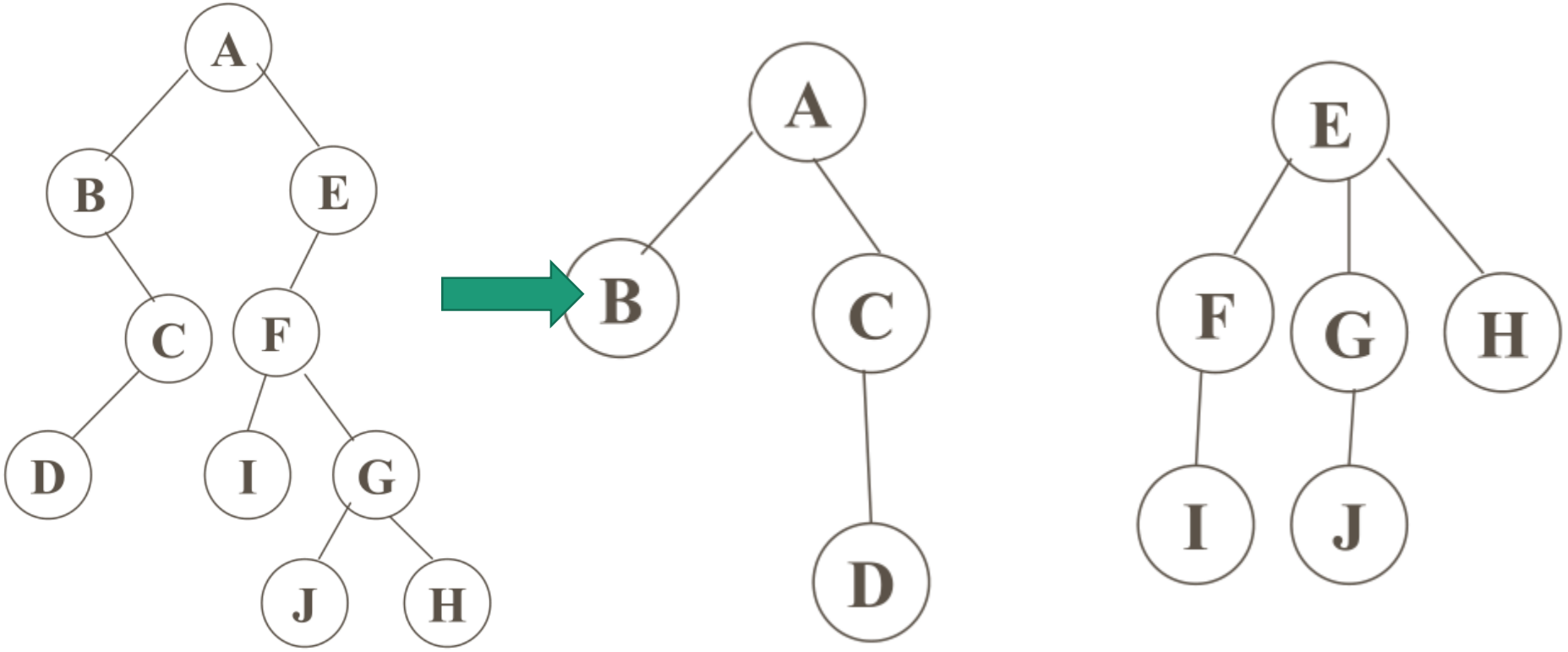
正确答案：C

例5：已知森林的前序序列和中序序列分别为ABCDEFIGJH和BDCAIFJGHE，请画出该森林。

- 解：由于森林的前序序列与其对应的二叉树前序序列相同，而森林的中序序列与其对应的二叉树中序序列相同。因此，根据二叉树前序序列ABCDEFIGJH和中序序列BDCAIFIGHE可画出二叉树如下图所示。



二叉树转换成森林



例6：树的概念

给出一棵树的逻辑结构T=(N,R),其中：
N={A,B,C,D,E,F,G,H,I,J,K}
R={r}
r={<A,B>,<B,E>,<B,F>,<F,G>,<F,H>,<A,C>,<C,I>,<C,J>,<J,K>,<A,D>}
试回答下列问题：
（1）哪个是根结点?
（2）哪些是F的孩子?
（3）结点K的层次是多少?

例7：树的基本性质

（1）已知一棵度为m的树有n_1个度为1的结点，n_2个度为2的结点，…，n_m个为m结点，问该树中有多少个叶子结点？

解： 设n为总结点个数，n_0为叶子结点(即度为0的结点个数)，则有:

$$n=n_0+n_1+n_2+\cdots+n_m \quad (1)$$

又有（分支总数）：$n-1=n_1*1+n_2*2+n_3*3+\cdots+n_m*m$ (2)

（因为一个结点对应一个分支）

式(2)-(1)得:

$1=n_0-n_2-2n_3-\cdots-(m-1)n_m$

则有:$n_0=1+n_2+2n_3+\cdots+(m-1)n_m$

（2）设树T的度为4，其中度为1，2，3和4的结点个数分别为4，2，1，1则T中的叶子数为（ 8 ）

补充：树的基本性质

2、树的基本性质

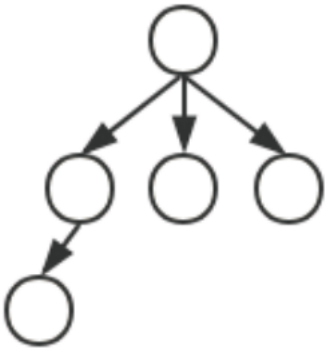
（2）度为 m 的树中第i层至多有 m^{i-1} 个结点

（3）深度为h的m叉树至多有 $m^h - 1$ 个结点 （满m叉树时， 等比数列求和）

（4）具有n个结点的m次树的最小高度为 $\lceil \log_m(n(m-1)+1) \rceil$

（5）树的结点总个数确定下， “完全”m叉树时，高度最小

（6）树的结点总个数确定下，单边树，可以使高度最大



“完全”m叉树

例8：森林与二叉树的转换

- **设F是由T1,T2,T3三棵树组成的森林，其中T1,T2,T3的结点数分别为n1,n2和n3，与F对应的二叉树为B，则二叉树B的左子树中有_____个结点**

A. n2-1

B. n3-1

C. n2

D. n1-1

正确答案：D

例9：森林与二叉树的转换

- **设F是由T1,T2,T3三棵树组成的森林，其中T1,T2,T3的结点数分别为n1,n2和n3，与F对应的二叉树为B，则二叉树B的右子树中有_____个结点**

A. n1+n3

B. n2+n3

C. n2

D. n3

正确答案：B