树的存储结构

- 双亲表示法

采用**一组连续空间**来存储每个结点,同时在**每个结点中增设一个伪指针**,指示其**双亲结点在数组中的位置**.根结点的下标为0,其伪指针域为-1.

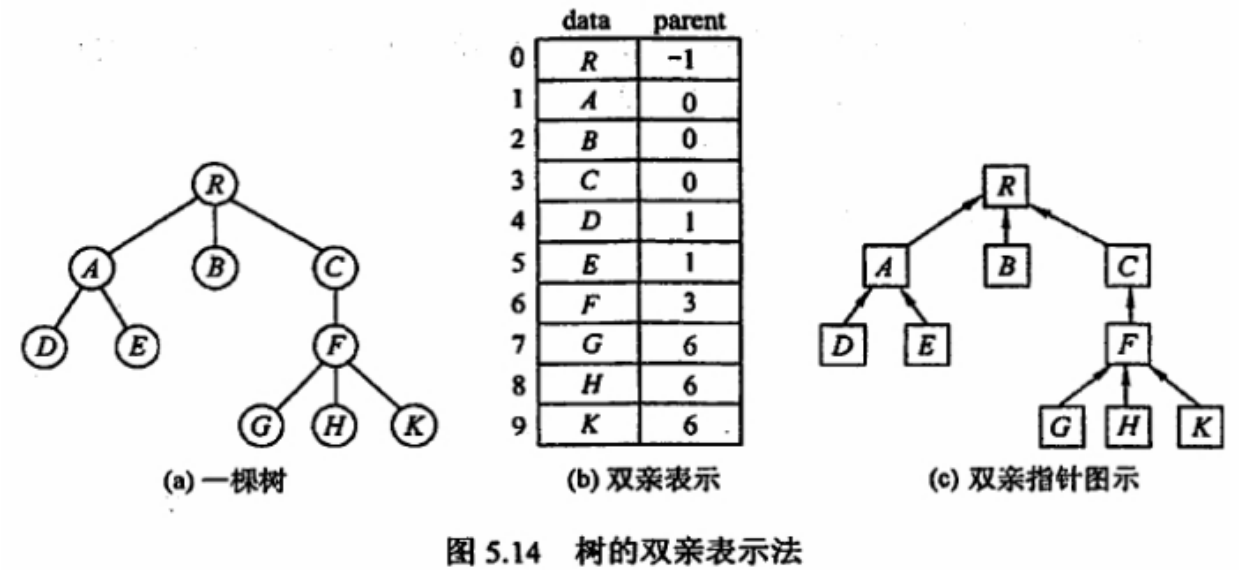


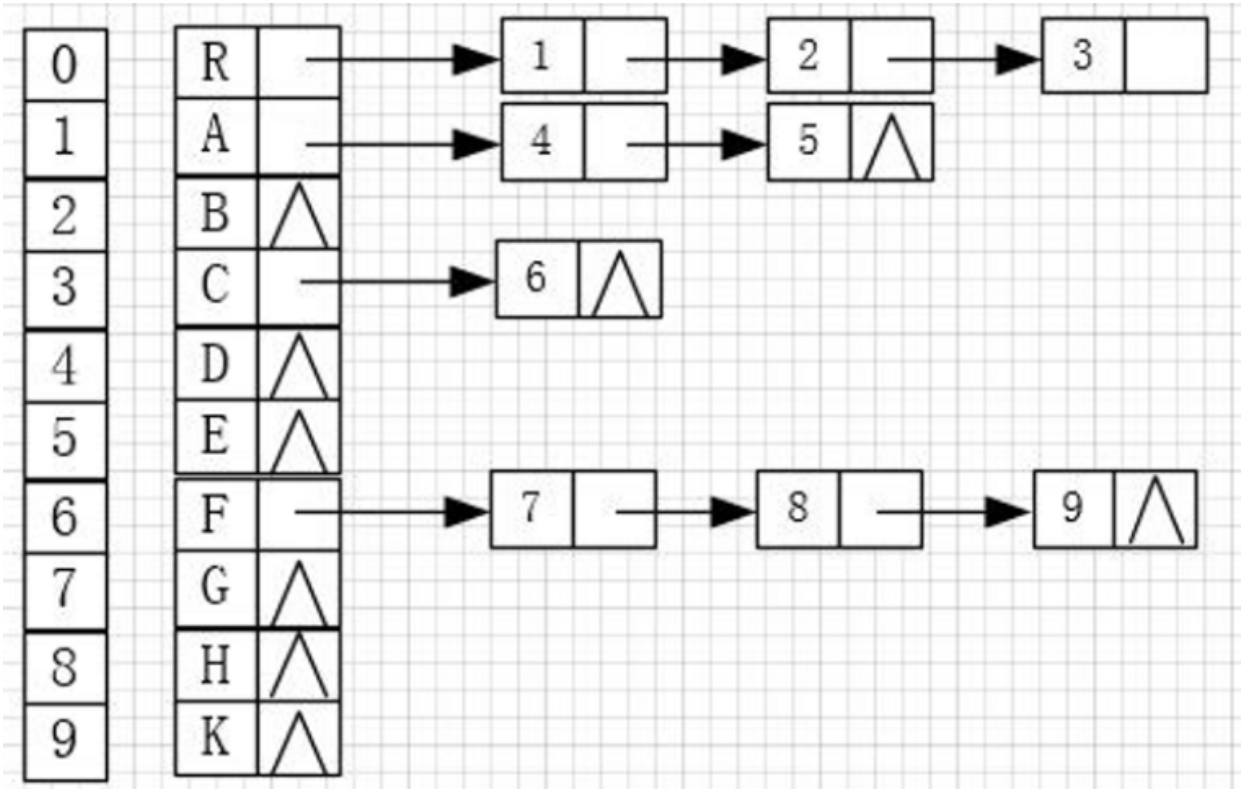
图 5.14　树的双亲表示法

```c
#define MAX_TREE_SIZE 100
typedef int ElemType;

typedef struct{//树的结点定义
    ElemType data;//数据元素
    int parent;//双亲位置域
}PTNode;

typedef struct{
    PTNode nodes[MAX_TREE_SIZE];//双亲表示
    int size;//结点个数
}PTree;
```

查指定结点的**双亲**很方便,查指定结点的**孩子**不方便(只能**从头遍历**)

- 孩子表示法

将每个结点的孩子结点都用单链表链接起来形成一个线性结构,此时 n个结点就有 n个孩子链表(**叶子结点**的孩子链表为**空表**)

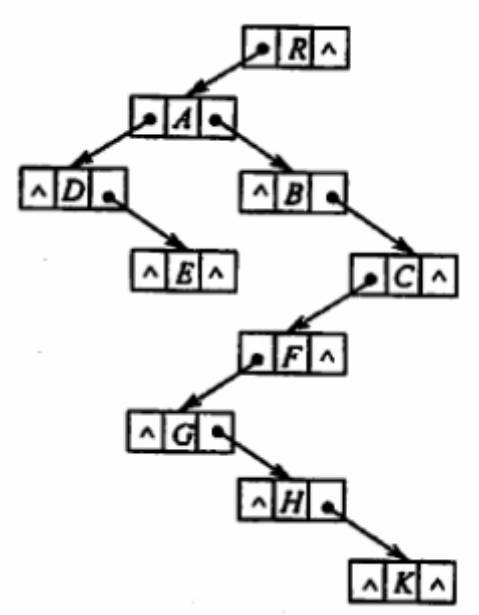



```c
#define MAX_TREE_SIZE 100
typedef char ElemType;

struct ChildNode
{//链表中每个结点的定义
    //链表中每个结点存储的不是数据本身,而是数据在数组中存储的位置下标
    int child;//孩子结点在数组中的位置
    struct ChildNode *next;//下一个孩子
};

typedef struct
{//树中每个结点的定义
    ElemType data;
    ChildNode *firstchild;//孩子链表头指针
}CHNode;

typedef struct
{
    CHNode nodes[MAX_TREE_SIZE];
    int size;
}CTree;
```

孩子表示法这种存储方式**寻找子女的操作非常直接**,而**寻找双亲**的操作需要遍历 n个结点中孩子链表指针域所指向的 n个孩子链表.

- 孩子兄弟表示法(**使用较多**)

孩子兄弟表示法又称**二叉树表示法**,即以**二叉链表**作为树的存储结构.孩子兄弟表示法使每个结点包括三部分内容：结点值、**指向结点第一个孩子结点的指针**、**指向结点下一个兄弟结点的指针**（沿此域可以找到结点的所有兄弟结点）。




```
#define MAX_TREE_SIZE 100
typedef char ElemType;

typedef struct CSNode
{
    ElemType data;
    struct CSNode *firstchild,*nextsibling;//第一个孩子和右兄弟节点
    //类似于二叉树中左孩子(第一个孩子),右孩子(右兄弟节点)
}CSNode,*CSTree;
```

孩子兄弟存储表示法比较灵活,其最大的**优点**是可以方便的**实现树转换为二叉树**的操作,易于**查找结点的孩子**等;**缺点**是从当前结点**查找其双亲结点**比较麻烦.

若为每一个结点增设一个 $parent$ 域指向其父结点,则查找结点的父结点也很方便.

通过孩子兄弟表示法,任意一棵普通树都可以相应转化为一棵二叉树,也就是说,任意一棵普通树都有唯一一颗二叉树与之对应;而森林通过**相邻树连边**变为大树,最后亦可转化为二叉树.

树的遍历

树的遍历主要有**先根遍历**和**后根遍历**.

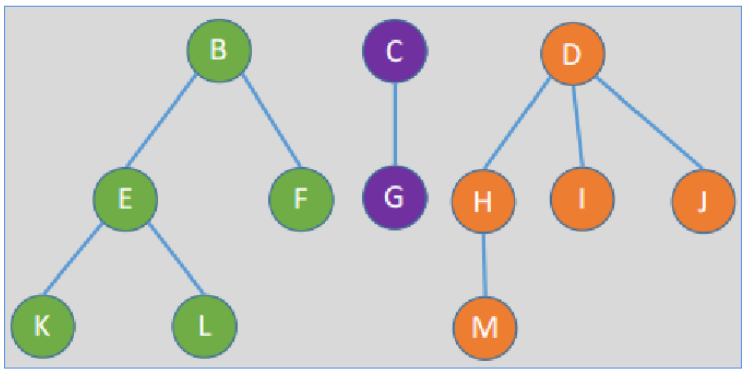
- 先根遍历:若树非空,则先访问根结点,再按照从左到右的顺序遍历根结点的每一棵子树.**这个访问顺序与这棵树对应的二叉树的** `先序遍历顺序` **相同**.
- 后根遍历:若树非空,则按照从左到右的顺序遍历根结点的每一棵子树,之后再访问根结点.**其访问顺序与这棵树对应的二叉树的** `中序遍历` **顺序相同**.
- 层次遍历:与二叉树的层次遍历思想基本相同,即按**层序依次访问**各结点.

森林的遍历

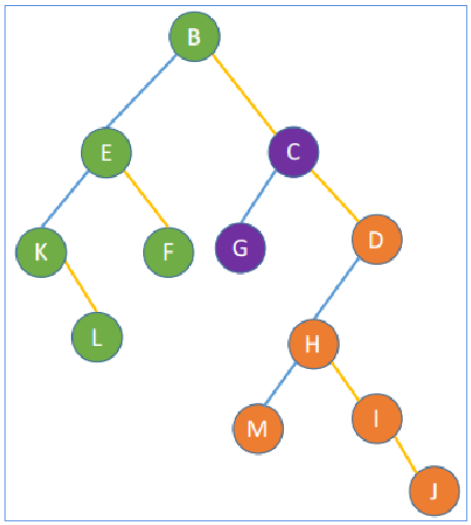
- **先序遍历森林**

> 若森林为非空，则按如下规则进行遍历：
>
> （1）访问森林中**第一棵树**的根结点。

> （2）先序遍历第一棵树中根结点的子树森林。
>
> （3）先序遍历除去第一棵树之后剩余的树构成的森林。
>
> 对森林的先序遍历，效果等同于依次对各个树进行**先根**遍历，实质上，等同于依次对**二叉树的先序遍历**。



森林

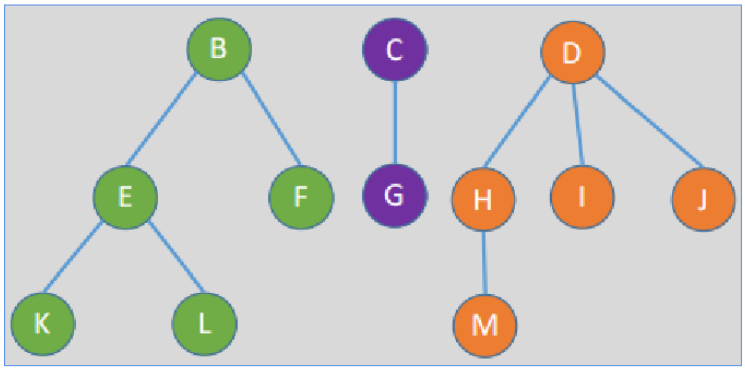


二叉树

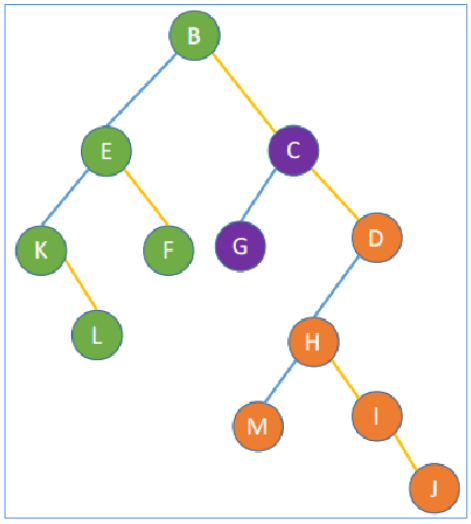
先序遍历森林结果为: $BEKLFCGDHMIJ$

- **中序遍历森林**

> 若森林为非空，则按如下规则进行遍历：
>
> （1）中序遍历森林中第一棵树的根结点的子树森林。
>
> （2）访问第一棵树的根结点。
>
> （3）中序遍历除去第一棵树之后剩余的树构成的森林。
>
> 对森林的先序遍历，效果等同于依次对各个树进行**后根**遍历，实质上，等同于依次对**二叉树的中序遍历**。



森林



二叉树

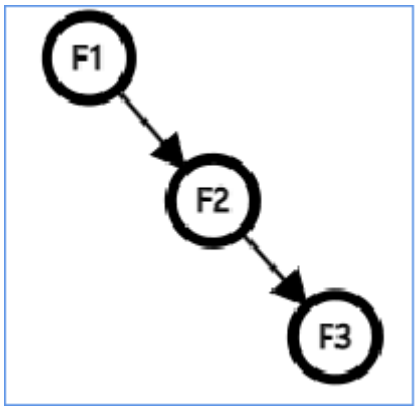
中序遍历森林结果为: $KLEFBGCMHIJD$

总结

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

> 设森林 F中有 3棵树，第一、第二、第三棵树的结点个数分别为 M_1，M_2和 M_3。与森林 F对应的二叉树根结点的右子树上的结点个数是().
>
> A. M_1
>
> B. M_1+M_2
>
> C. M_3
>
> D. M_2+M_3

设第一、第二、第三棵树的根结点分别为 F_1, F_2和 F_3,转化为二叉树为如下图所示:



故森林 F对应的二叉树根结点为 F_1, F_1的右子树包含第二和第三棵树,故结点数为 M_2+M_3.

> 设 F是一个森林， B是由 F变换来的二叉树。若 F中有 n个非终端结点，则 B中右指针域为空的结点有()个。
>
> A. $n-1$
>
> B. n
>
> C. $n+1$
>
> D. $n+2$

根据森林转换为二叉树的"**左孩子右兄弟**"的表示法，即对于每棵二叉树，每个结点的**右指针**指向其**右邻兄弟**.

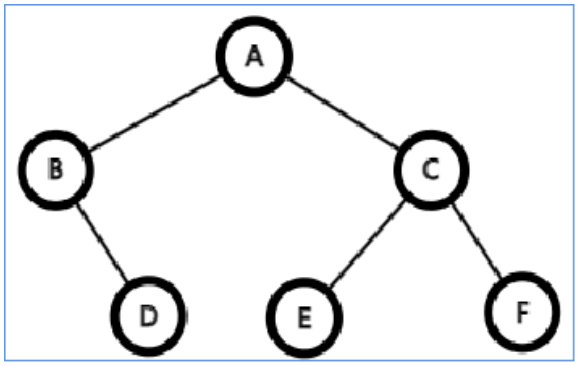
针对每一个非终端结点，一定会**有且仅有一个**孩子结点没有右邻兄弟，即右指针域为空。因此 n个非终端结点，就有 n个右指针域为空。(**一对一**的关系)

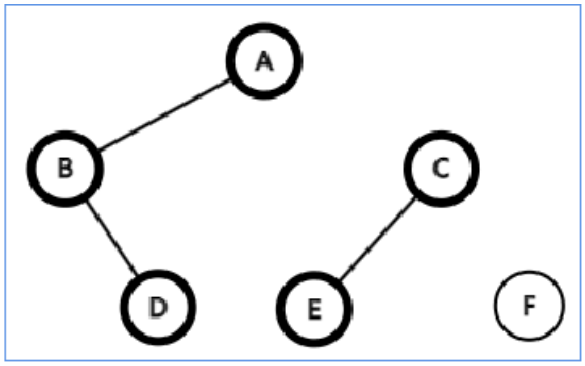
看完单棵二叉树，再来看这些二叉树是怎么连接成一棵二叉树的。原理是：将后一棵二叉树的**根节点**作为前一棵二叉树的**右孩子**连接起来，所以只有`最后一棵二叉树的根结点`没有右孩子，即右指针域为空。

综上,故在 B中有 $n+1$个右指针域为空的结点.

> 某二叉树结点的中序序列为 $BDAECF$,后序序列为 $DBEFCA$,则该二叉树对应的森林包括()棵树.
>
> A. 1
>
> B. 2
>
> C. 3
>
> D. 4

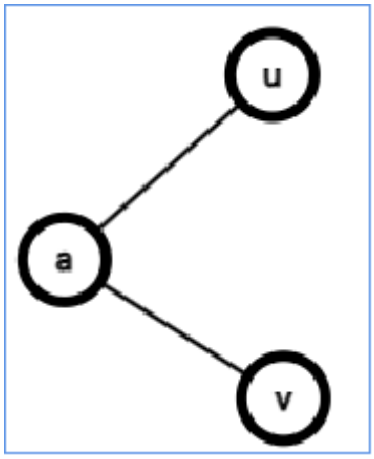
通过二叉树的中序序列和后序序列可构造出如下左图所示,对应的森林如下右图所示,可看出有 3棵树.

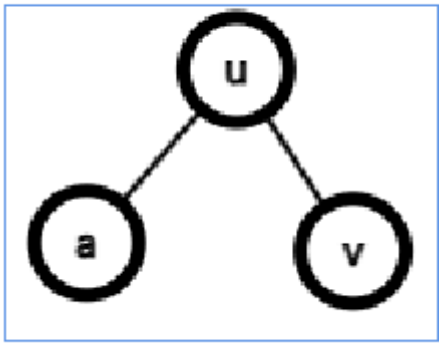




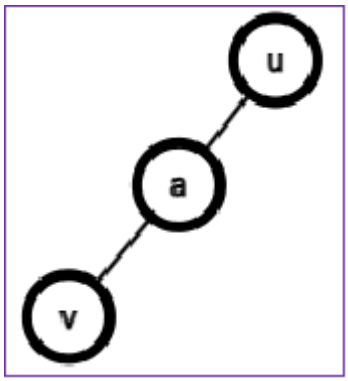
> **2009统考真题**：将森林转换为对应的二叉树，若在二叉树中，结点 u是结点 v的父结点的父结点,则在原来的森林中， u和 v可能具有的关系是(B).
>
> Ⅰ. 父子关系
>
> Ⅱ. 兄弟关系
>
> Ⅲ. u的父结点与 v的父结点是兄弟关系
>
> A. 只有Ⅱ
>
> B. Ⅰ和Ⅱ
>
> C. Ⅰ和Ⅲ
>
> D. Ⅰ、Ⅱ和Ⅲ

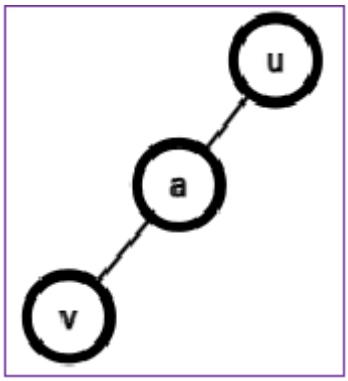
- 方式一(**父子关系**)



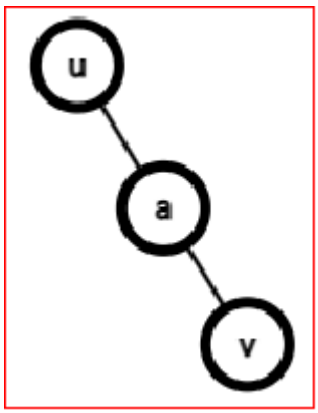


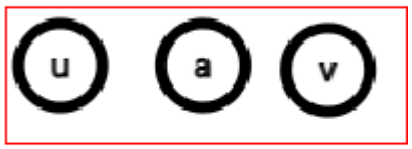
- 方式二(**爷孙关系**)

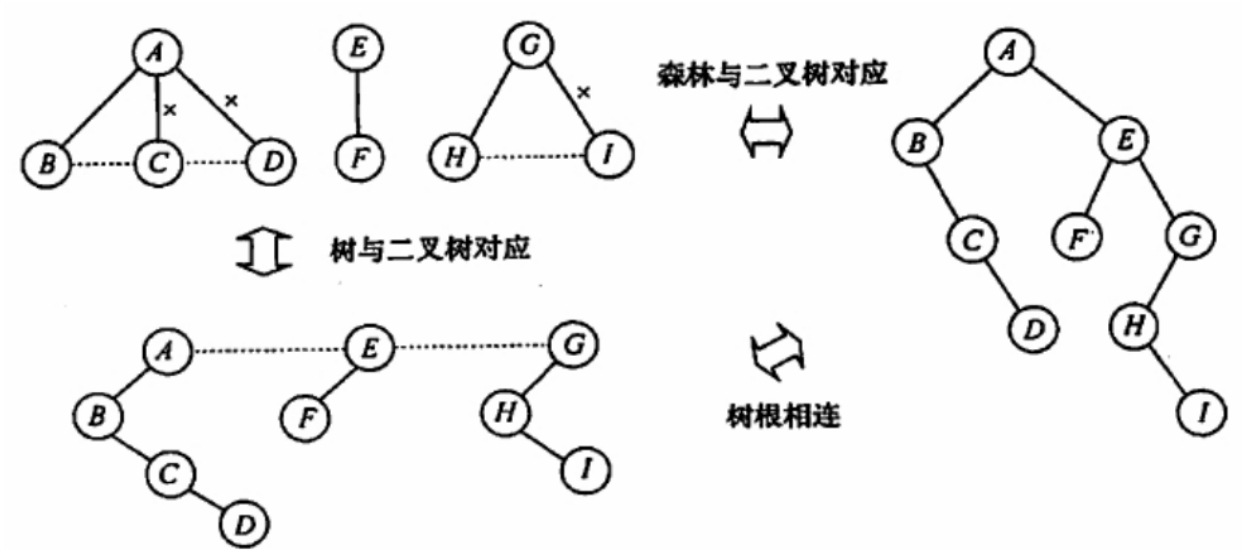




- 方式三(可能**兄弟关系**,或者**没有关系(位于不同的树上)**)

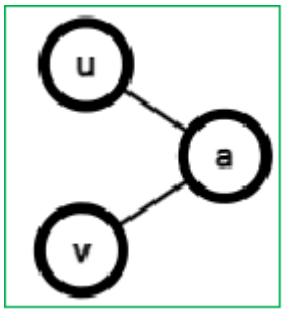


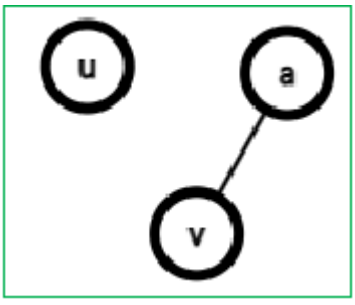




例如图中 B和 D在二叉树中构成爷孙关系,而在森林中是**兄弟关系**;而 A和 G也在二叉树中构成爷孙关系,而在森林中**没有关系**.

- 方式四(叔侄关系或**没有关系(u为某树的根结点)**)





当 u为某树的根结点时, u和 v分别位于不同的树上,此时在森林中**没有关系**;反之,此时在森林中 a为 u的右兄弟,故 u和 v构成**叔侄关系**.

综上,只有 Ⅰ和Ⅱ满足条件.

> **2011统考真题**：已知一棵有 2011个结点的树，其叶结点个数为 116，该树对应的二叉树中无右孩子的结点个数是().
>
> A. 115
>
> B. 116
>
> C. 1895
>
> D. 1896

由前面知,针对每一个非终端结点(**非叶子结点**)，一定会**有且仅有一个**孩子结点没有右邻兄弟，即右指针域为空。因此 n个非终端结点，就有 n个右指针域为空。(**一对一**的关系),故非终端结点的个数是 $2011-116=1895$个,而它本身是个**森林**,仍需考虑最后一棵树(有且仅有一棵树),它没有右孩子，即右指针域为空,故需在非终端结点的基础上加 1,结果为 1896个.

> **2016统考真题**：若森林 F有 15条边、 25个结点，则 F包含树的个数是().
>
> A. 8
>
> B. 9
>
> C. 10
>
> D. 11

假设令 15条边都在同一棵树上,此时该数共使用 16个(树中`结点数=边数+1`)结点,剩下共有 $25-16=9$个结点,分别各构成一棵树,故森林共有 $1+9=10$棵树.

书中解释:

解法一：树有一个很重要的性质，即在 n 个结点的树中有 $n-1$ 条边，“那么对于每棵树，其结点数比边数多 1”。题中的森林中的结点数比边数多 10（即 $25-15=10$），显然共有 10 棵树。

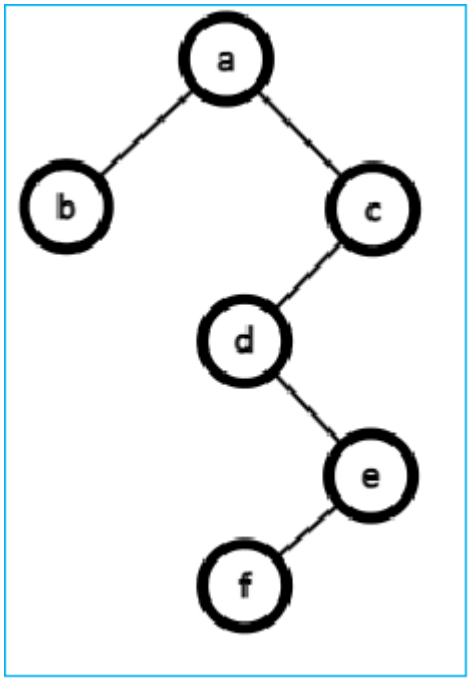
解法二：仔细分析后发现，此题考查的也是图的某些方面的性质：生成树和生成森林。此时对于图的生成树有一个重要的性质，即图中顶点数若为 n，则其生成树含有 $n-1$ 条边。对比解法一中树的性质，不难发现两种解法都用到了性质“树中结点数比边数多 1”，接下来的分析如解法一。

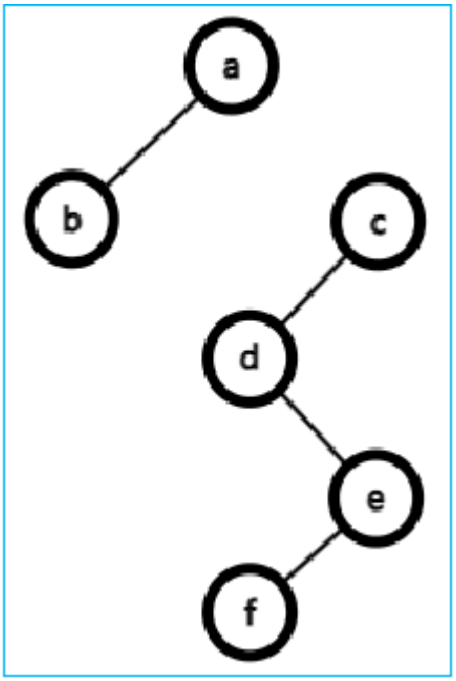
> **2020统考真题**：已知森林 F及与之对应的二叉树 T,若 F的先根遍历序列是 a,b,c,d,e,f，中根遍历序列是 b,a,d,f,e,c，则 T的后根遍历序列是().
>
> A. b,a,d,f,e,c
>
> B. b,d,f,e,c,a
>
> C. b,f,e,d,c,a
>
> D. f,e,d,c,b,a

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

注意：部分教材也将森林的中根遍历称为后根遍历，称中根遍历是相对其二叉树而言的，称后根遍历是因为根确实是最后才访问的，如遇到这两种称谓，那么都可以理解为同一种遍历方法。

由表知,此时**二叉树**的 `先序`(或 `中序`)遍历与**森林**的 `先序`(或 `中(后)序`)遍历相同,故可通过该先序和中序构造出二叉树,进而得到二叉树的后序遍历.

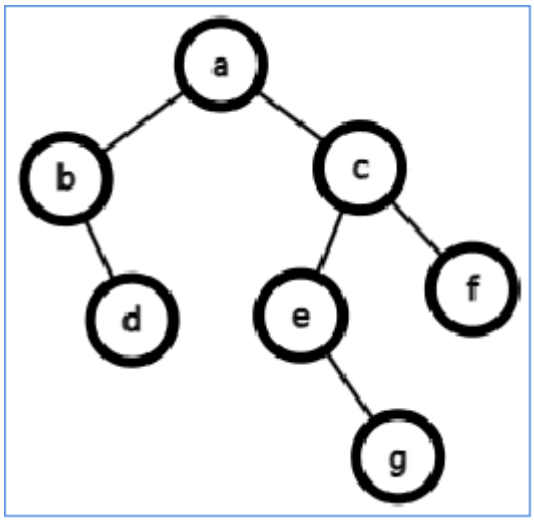


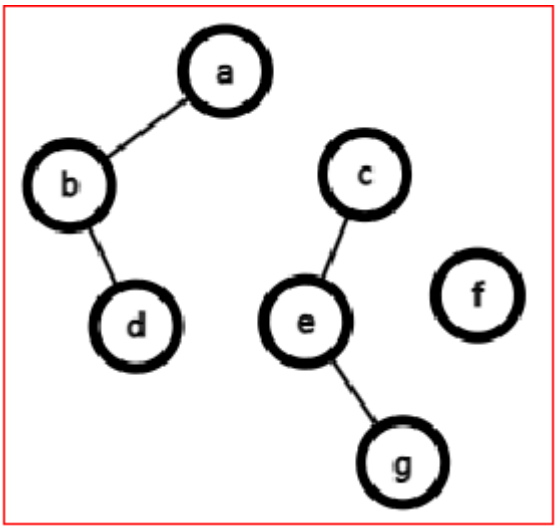


故二叉树的后序遍历为 b, f, e, d, c, a

> **2021统考真题**：某森林 F对应的二叉树为 T,若 T的先序遍历序列为 a, b, d, c, e, g, f,中序遍历序列是 b, d, a, e, g, c, f,则 F中树的棵树是().
>
> A. 1
>
> B. 2
>
> C. 3
>
> D. 4

根据二叉树的先序和中序遍历序列生成二叉树,再转化为森林即可.





由上图知,该森林 F共有 3棵树.

> **2022统考真题**：如果三叉树 T中有 244个结点(叶结点的高度为 1),那么 T的高度至少是()
>
> A. 8
>
> B. 7
>
> C. 6
>
> D. 5

套结论：具有 n个结点的 m叉树(或度为 m的树)的最小高度为 $\lceil \log_m(n \times (m-1)+1) \rceil$;

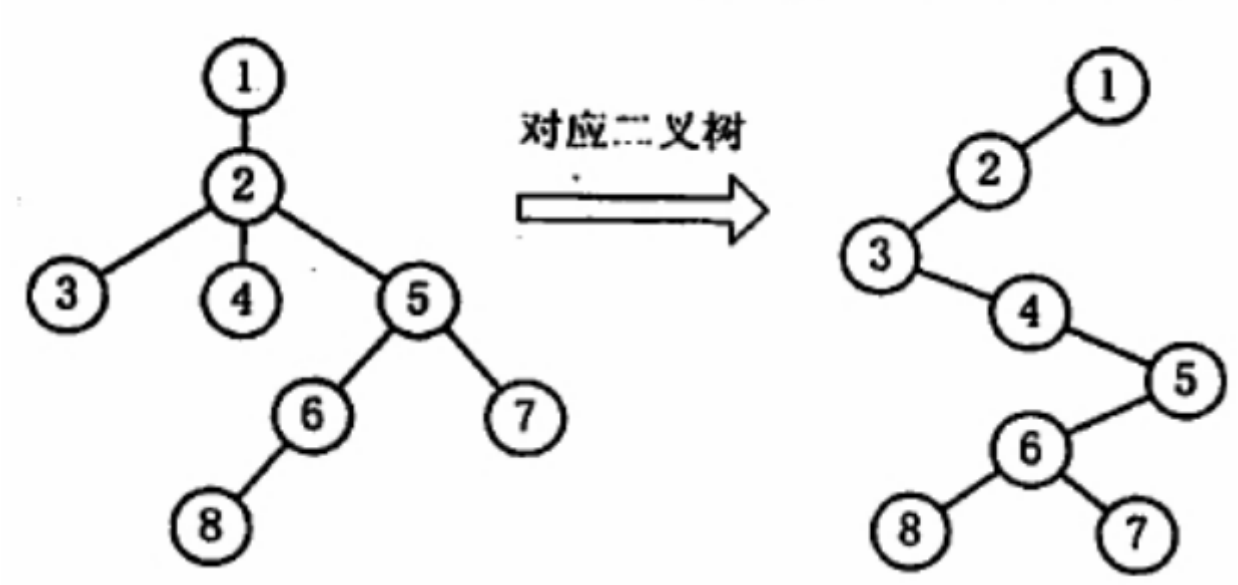
T的高度至少为 $\lceil \log_3(244 \times 2+1) \rceil = 6$

5.4.1

> 给定一棵树的先根遍历序列和后根遍历序列，能否唯一确定一棵树?若能，请举例说明;若不能，请给出反例。

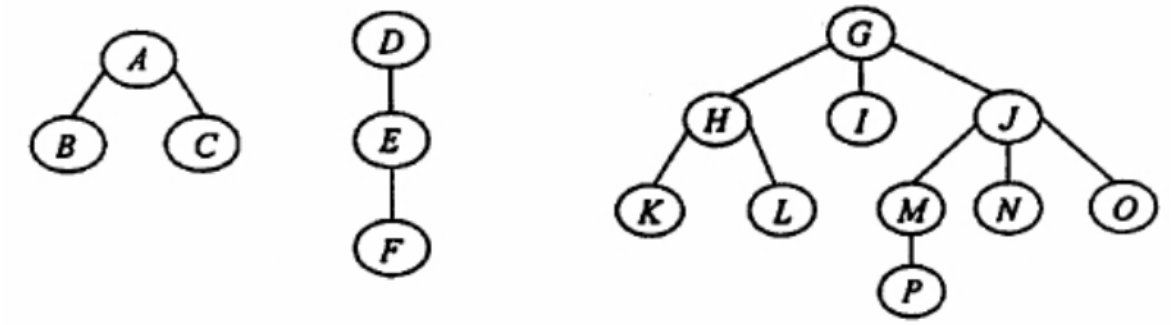
一棵树的**先根遍历**结果与其对应二叉树的**先序遍历**结点相同,树的**后根遍历**结果与其对应二叉树表示的**中序遍历**结果相同.

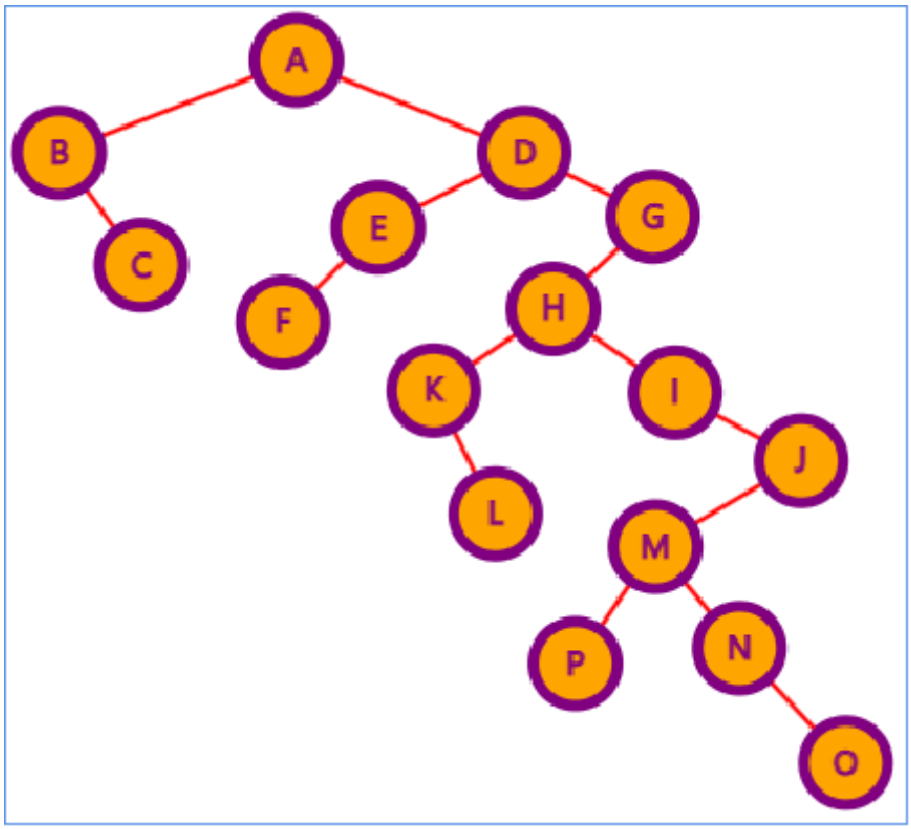
对应二叉树的先序序列为 $1,2,3,4,5,6,8,7$,中序序列为 $3,4,8,6,7,5,2,1$;原树的先根遍历序列为 $1,2,3,4,5,6,8,7$,后根遍历为 $3,4,8,6,7,5,2,1$.



5.4.2

> 将下面一个由 3棵树组成的森林转换为二叉树。

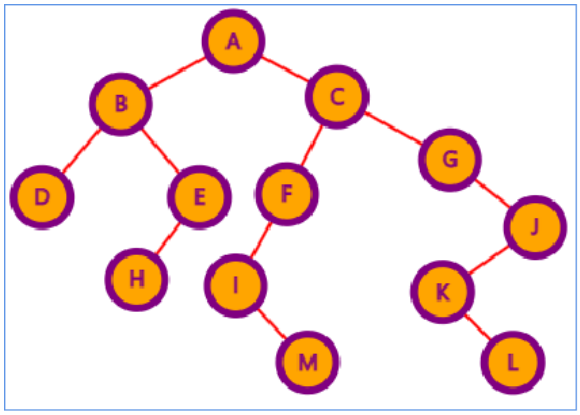


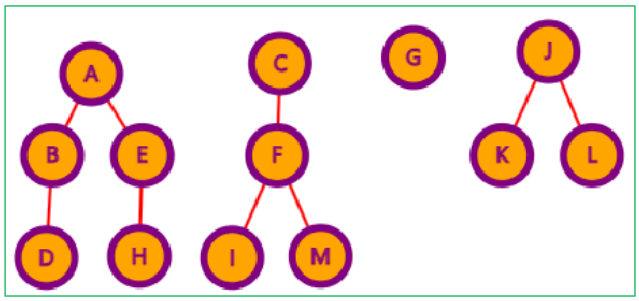


5.4.3

> 已知某二叉树的先序序列和中序序列分别为 $ABDEHCFIMGJKL$和 $DBHEAIMFCGKLJ$,请画出这棵二叉树,并画出二叉树对应的森林.

左图为二叉树,右图为对应的森林.





5.4.4

> 编程求以孩子兄弟表示法存储的森林的叶结点数.

当森林(树)以孩子兄弟表示法存储时,若结点的**指向结点第一个孩子结点的指针为空**,则说明为叶子结点.

```
void Get_Leaves(CSTree t,int &tot)//通过引用传值
{
    if(t!=NULL)
    {
        Get_Leaves(t->firstchild,tot);
        if(t->firstchild==NULL)
            tot++;
        Get_Leaves(t->nextsibling,tot);
    }
}
```

5.4.5

> 以孩子兄弟链表为存储结构,请设计递归算法求树的深度.

孩子兄弟表示法中采用`左孩子右兄弟`,故只需递归比较`左孩子的深度+1`和`右兄弟的深度`即可.

```
int Get_Height(CSTree t)
{
    if(t==NULL)
        return 0;
    else
    {
        int hchild=Get_Height(t->firstchild);
        int hsibing=Get_Height(t->nextsibling);
        return hchild+1>hsibing?hchild+1:hsibing;
    }
}
```

5.4.6

> 已知一棵树的层次序列及每个结点的度，编写算法构造此树的孩子-兄弟链表.

由于层次遍历的顺序可以根据节点的度去充分地遍历每一个节点的兄弟，所以可以每次都把每一层节点的兄弟关系先链接好，并且由于父子关系只能查找一次，所以按照层序序列经过节点时，需要把其孩子节点链接上。
所以按照层序次序遍历节点时，先链接上**第一个孩子节点**，然后链接上该节点的**全部兄弟节点。**

```
void Init_Tree_By_LevelOrder(CSTree &t,ElemType e[],int degree[],int n)
{
    CSTree *temp=new CSTree[MAX_TREE_SIZE];

    for(int i=0;i<=n-1;i++)//初始化赋值和置空
    {
        temp[i] = new CSNode;//一定记得初始化
        temp[i]->data=e[i];
        temp[i]->firstchild=temp[i]->nextsibling=NULL;
    }

    int pos=0;
    for(int i=0;i<=n-1;i++)//遍历至第i个结点
```

```
    {
        if(degree[i]>0)//如果有孩子
        {
            temp[i]->firstchild=temp[++pos];//第一个孩子赋给孩子指针
            for(int j=2;j<=degree[i];j++)
            {
                temp[pos]->nextsibling=temp[pos+1];//其余孩子处理:后一个赋给前一个的兄弟指针
                pos++;
            }
        }
    }
    t=temp[0];
    //delete []temp;
}
```

5.4.7

> **2016统考真题**：若一棵非空 $k(k\geq 2)$ 叉树 T中的每个非叶结点都有 k个孩子，则称 T为正则 k叉树。请回答下列问题并给出推导过程。
>
> 1)若 T有 m个非叶节点,则 T中的叶结点有多少个?
>
> 2)若 T的高度为 h(单节点的树 $h=1$),则 T的结点数最多有多少个?最少为多少个?

1)设叶子结点的个数为 n_0,已知非叶子结点个数为 m,总个数为 n,则 $n=n_0+m$;

又因为`边数=总结点数-1`,且边数由每个非叶子结点作出 k条边,故 $n-1=m\times k$;

两式联立: $m\times k+1=n_0+m$,即叶子结点的个数为 $n_0=m\times(k-1)+1$

2)当该树为满 k叉树时,此时 T的结点数最多为 $1+k^1+k^2+\cdots+k^{h-1}=\frac{1(1-k^h)}{1-k}=\frac{k^h-1}{k-1}$;

当第 1层只有根结点,第 2到第 $h-1$层仅含 1个分支结点和 $k-1$个叶结点,第 h层有 k个叶结点(即即除根外第 2到第 h层中每层的结点数均为 k),此时 T的结点数最少,结果为 $1+k\times(h-1)$