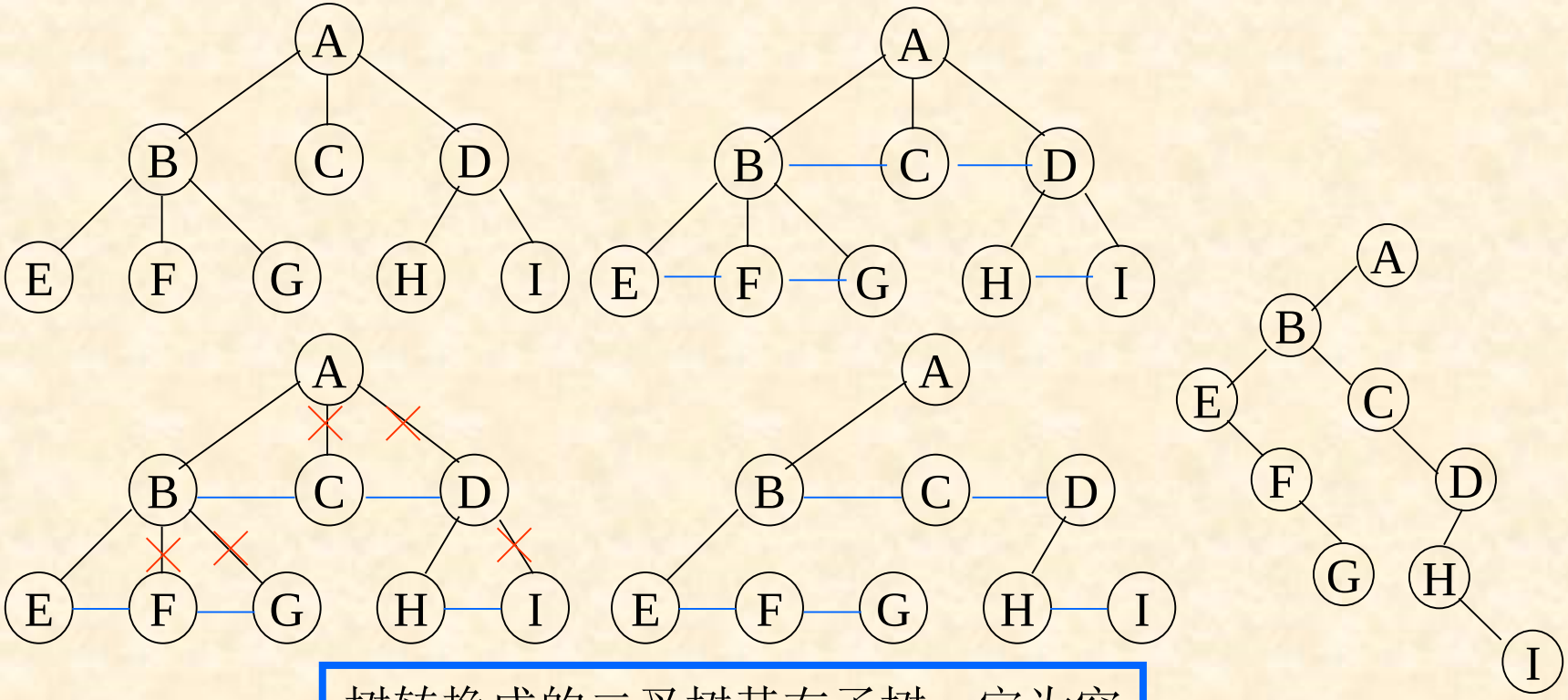
第六章　树

6.4 树和森林

将树转换成二叉树的步骤

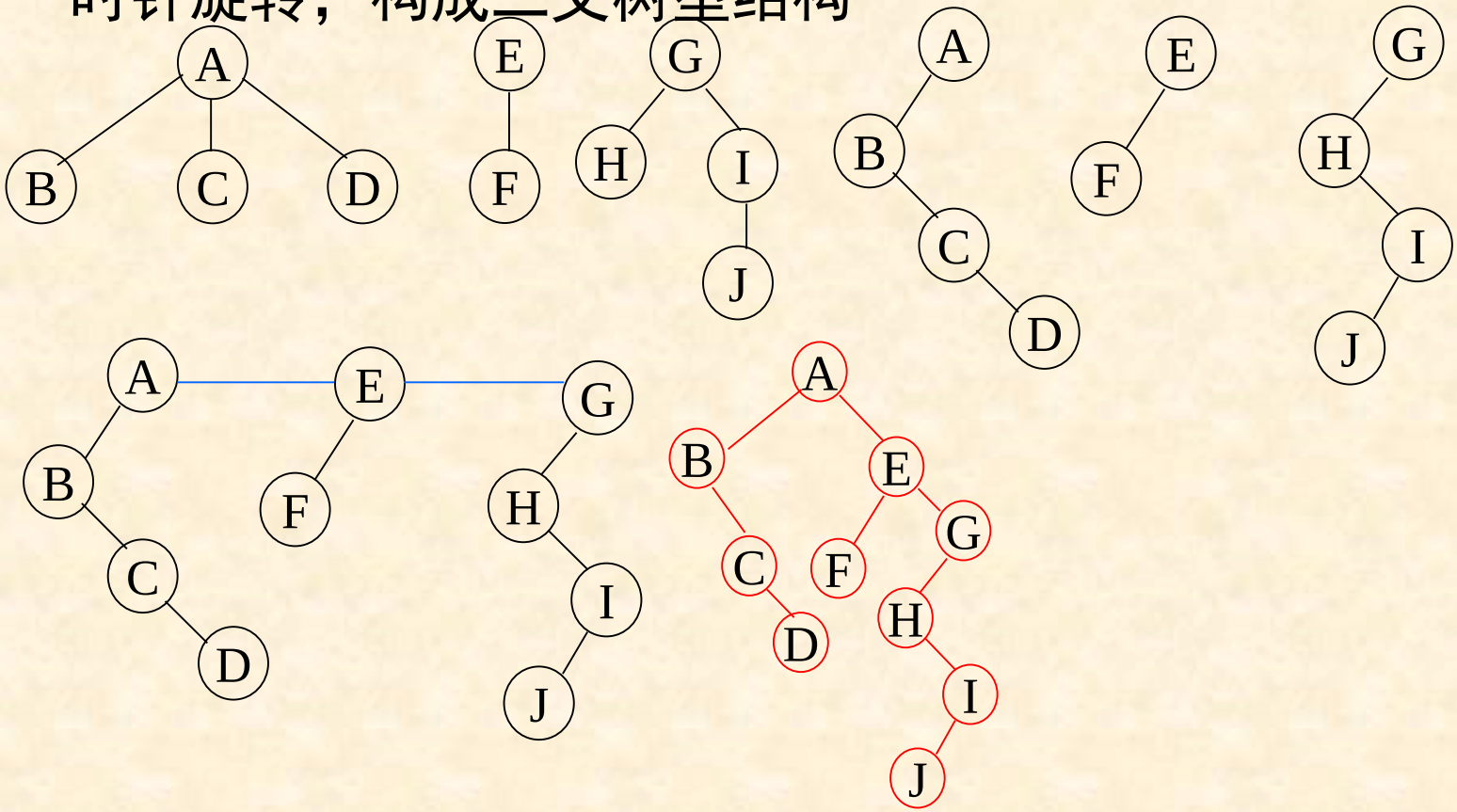
- 加线：在兄弟之间加一连线
- 抹线：对每个结点，除了其左孩子外，去除其与其余孩子之间的关系
- 旋转：以树的根结点为轴心，将整树顺时针转45°



树转换成的二叉树其右子树一定为空

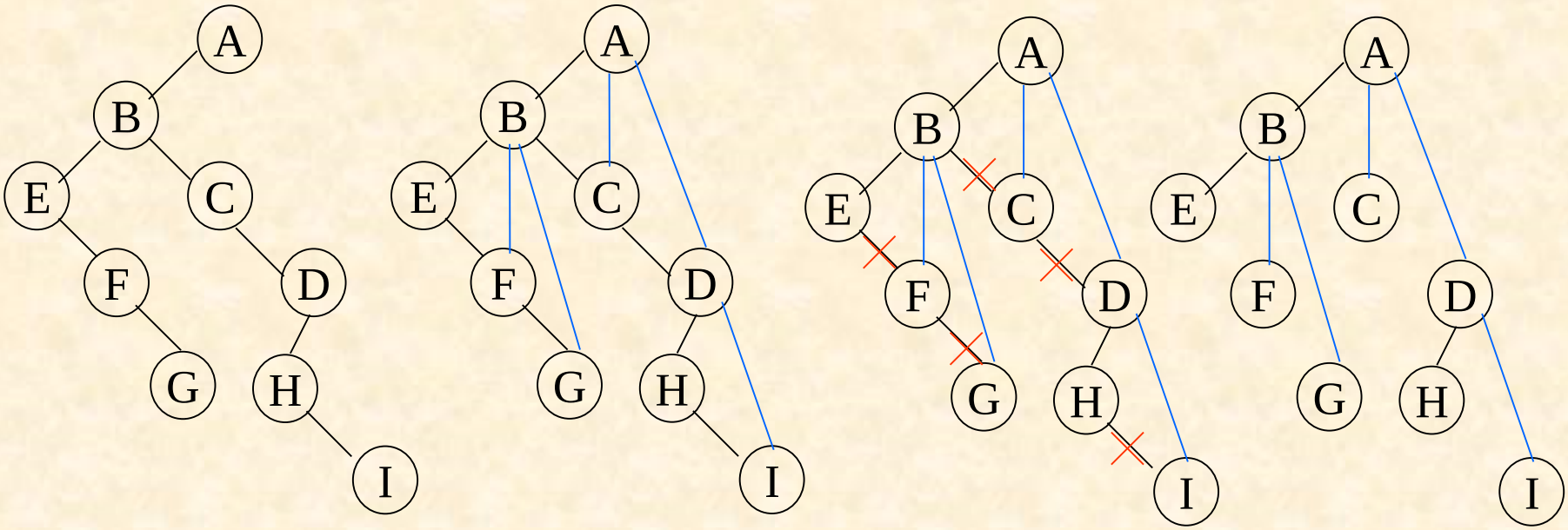
将森林转换成二叉树的步骤

- 将各棵树分别转换成二叉树
- 将每棵树的根结点用线相连
- 以第一棵树根结点为二叉树的根，再以根结点为轴心，顺时针旋转，构成二叉树型结构



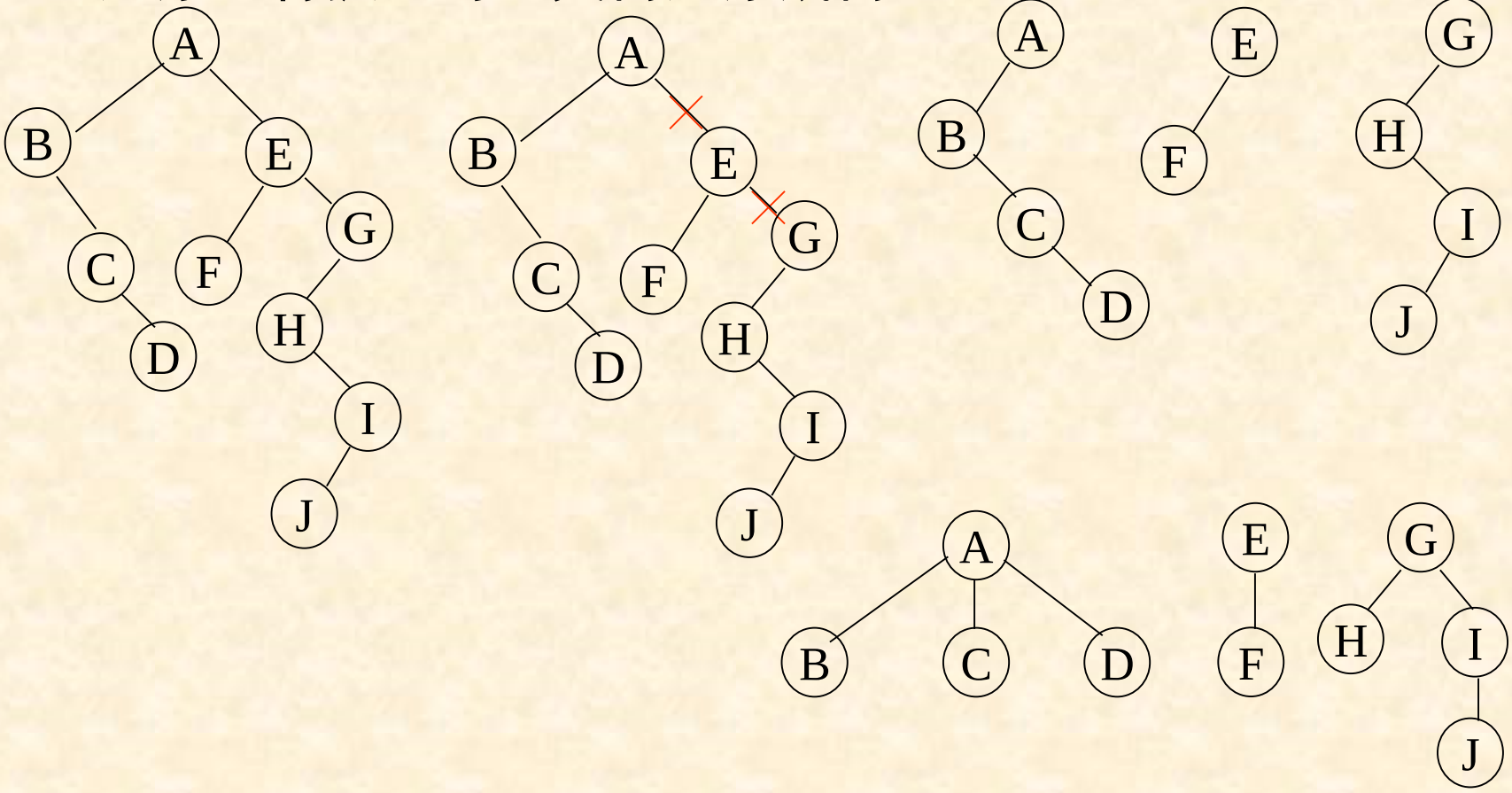
将二叉树转换成树的步骤

- 加线：若p结点是双亲结点的左孩子，则将p的右孩子，右孩子的右孩子，……沿分支找到的所有右孩子，都与p的双亲用线连起来
- 抹线：抹掉原二叉树中双亲与右孩子之间的连线
- 调整：将结点按层次排列，形成树结构



将二叉树转换成森林的步骤

- 抹线：将二叉树中根结点与其右孩子连线，及沿右分支搜索到的所有右孩子间连线全部抹掉,使之变成孤立的二叉树
- 还原：将孤立的二叉树还原成树



6.4 树和森林

6.4.1 树与二叉树的转换

6.4.2 树和森林的遍历

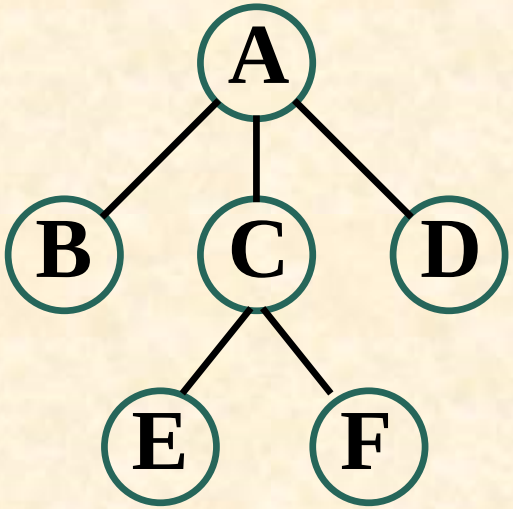
6.4.3 树的顺序存储

6.4.4 树的链接存储

- 树的遍历——按一定规律走遍树的各个顶点，且使每一顶点仅被访问一次，即找一个完整而有规律的走法，以得到树中所有结点的一个线性排列
- 常用方法
  - 先根（序）遍历：先访问树的根结点，然后依次先根遍历根的每棵子树
  - 后根（序）遍历：先依次后根遍历每棵子树，然后访问根结点
  - 按层次遍历：先访问第一层上的结点，然后依次遍历第二层，……第n层的结点

1、树的遍历

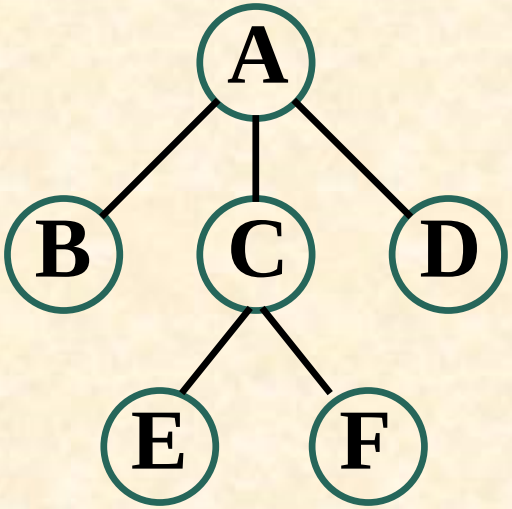
先根遍历：先访问树的根结点，然后依次先根遍历每棵子树。



先根遍历序列：A B C E F D

后根遍历：

先依次后根遍历每棵子树，然后访问树的根结点。



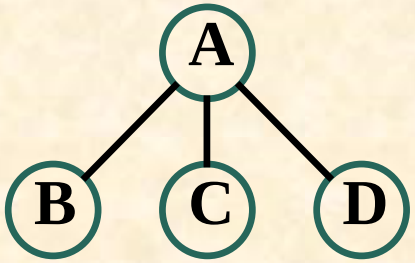
后根遍历序列：B E F C D A

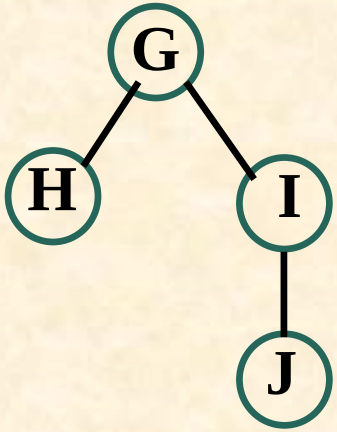
2、森林的遍历——先根遍历

① 访问森林中第一棵树的根结点；

② 先序遍历第一棵树中的诸子树；

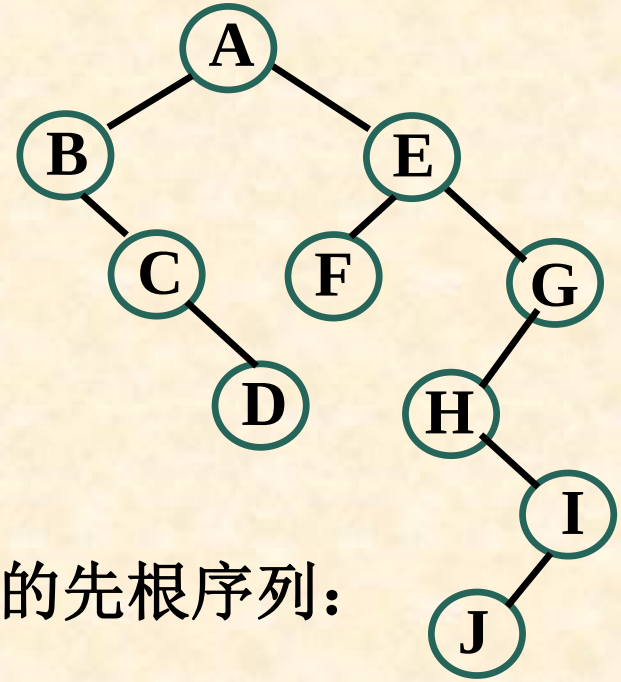
③ 先序遍历其余的诸树。





森林的先根遍历序列：

A B C D E F G H I J



二叉树的先根序列：

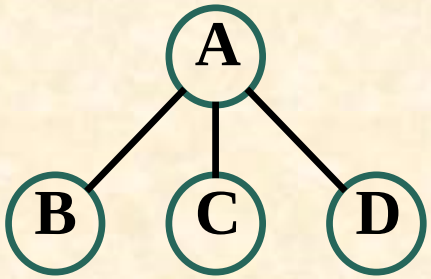
A B C D E F G H I J

后根遍历森林：

① 后序遍历第一棵树的诸子树；

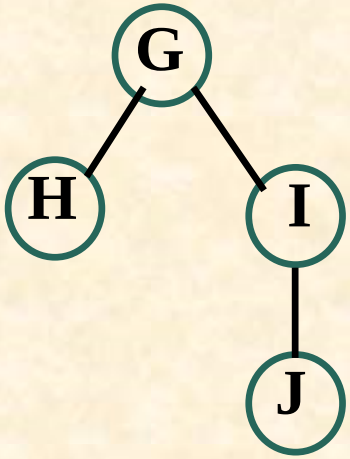
② 访问森林中第一棵树的根结点；

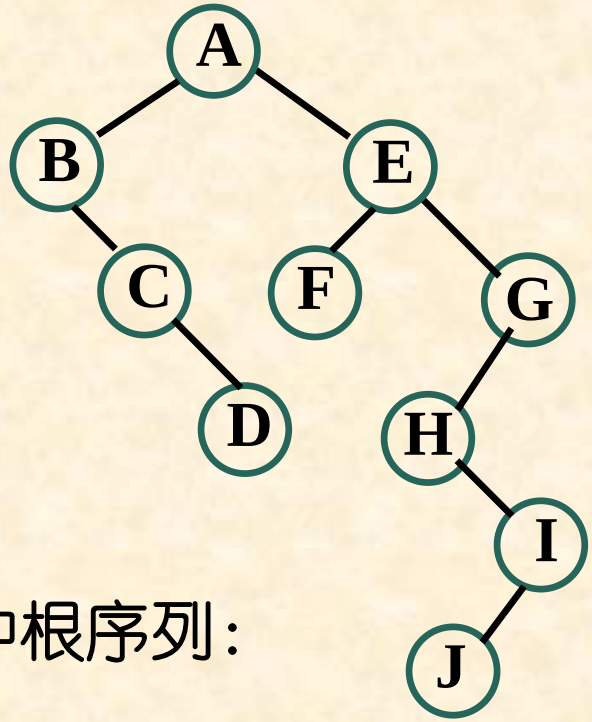
③ 后序遍历其余的诸树。



森林的后根遍历序列：

B C D A F E H J I G



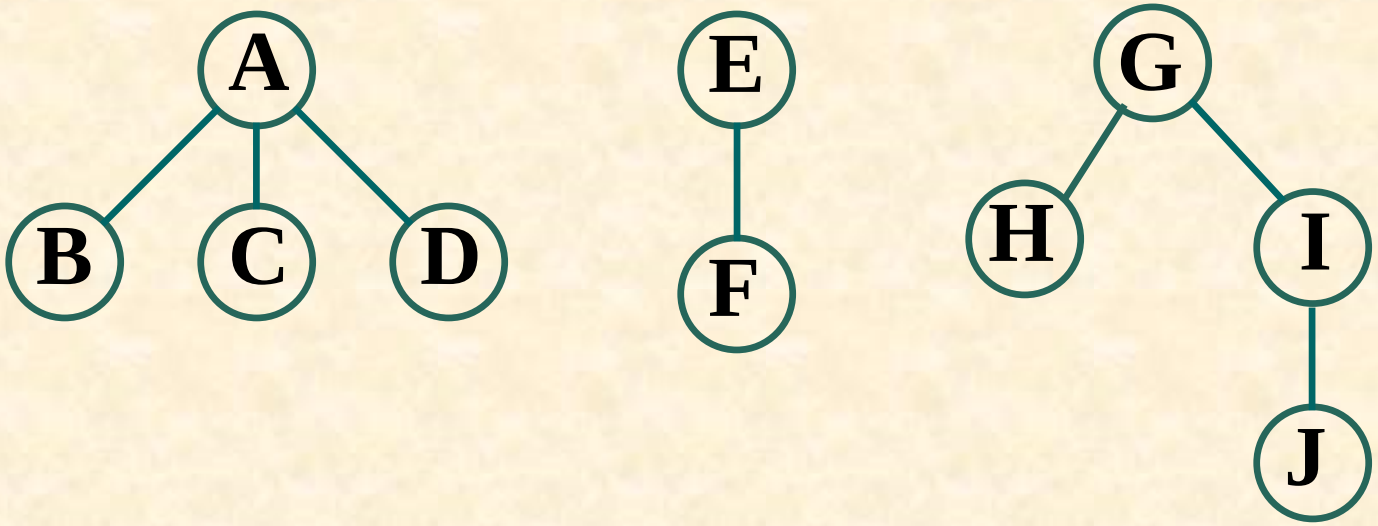


二叉树的中根序列：

B C D A F E H J I G

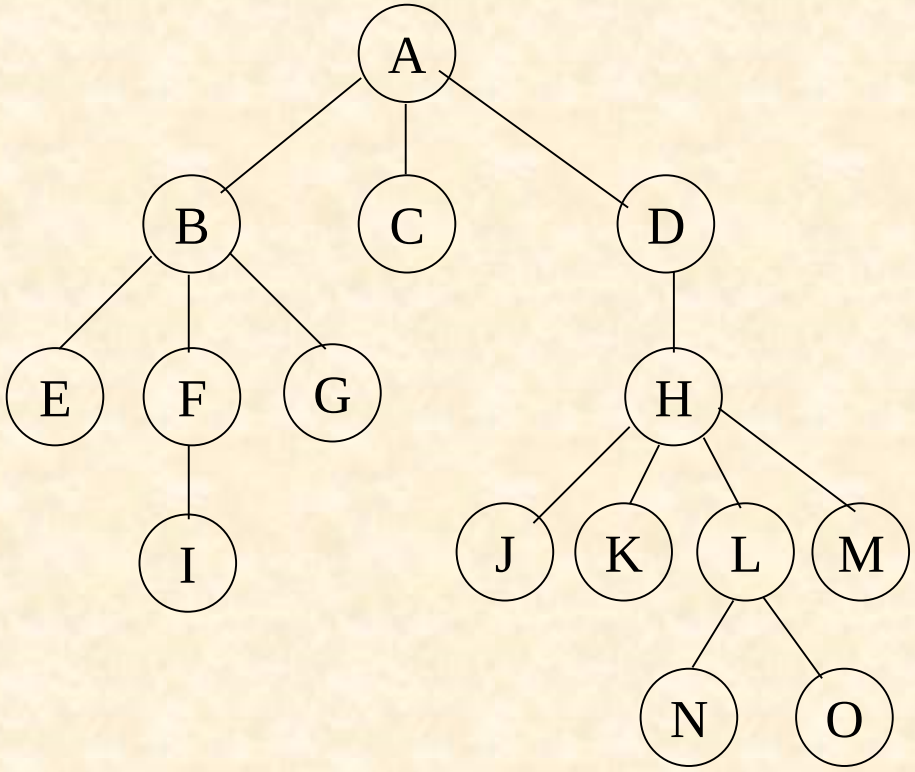
树和森林的层次遍历

[例]



森林的层次遍历序列：

A E G B C D F H I J



先序遍历：A B E F I G C D H J K L N O M

后序遍历：E I F G B C J K N O L M H D A

层次遍历：A B C D E F G H I J K L M N O

6.4 树和森林

6.4.1 树与二叉树的转换

6.4.2 树和森林的遍历

6.4.3 树的顺序存储

6.4.4 树的链接存储

树的顺序存储结构

1)先根序列及结点度数表示法

2)后根序列及结点度数表示法

3)层次序列及结点度数表示法

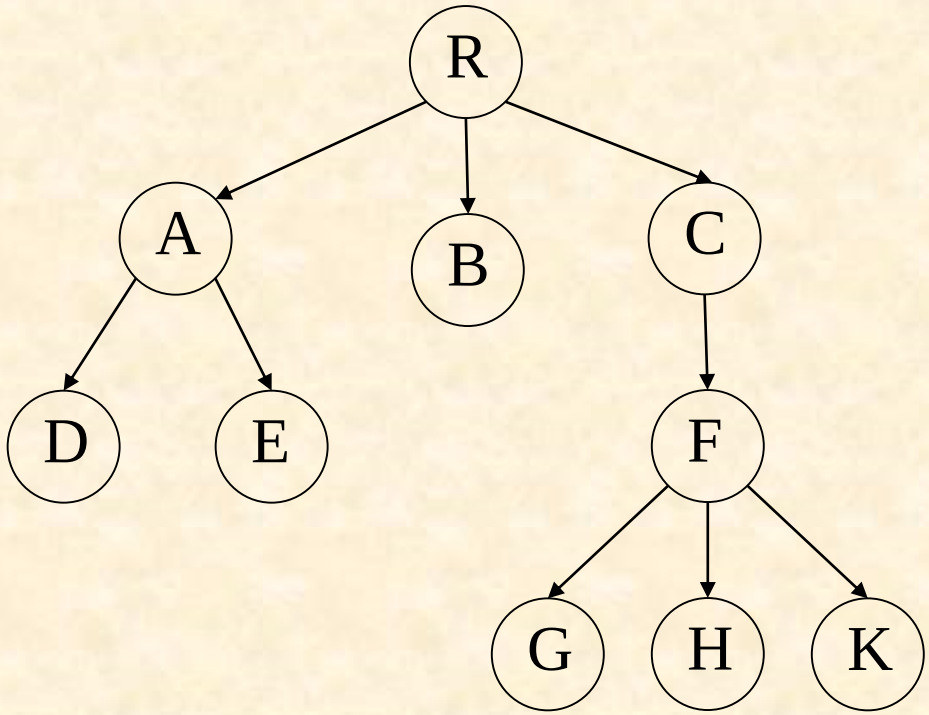
4)层次序列及Father数组表示法

1) 树的先根序列及结点度数表示法

树的先根遍历的定义

（1）访问根结点

（2）从左到右依次先根次序遍历树的诸子树



先根序列　**RADEBCFGHK**

定理6.2 如果已知一个树的先根序列和每个结点相应的度数，则能唯一确定该树的结构。

证明：用数学归纳法

1. 若树中只有一个结点，定理显然成立。

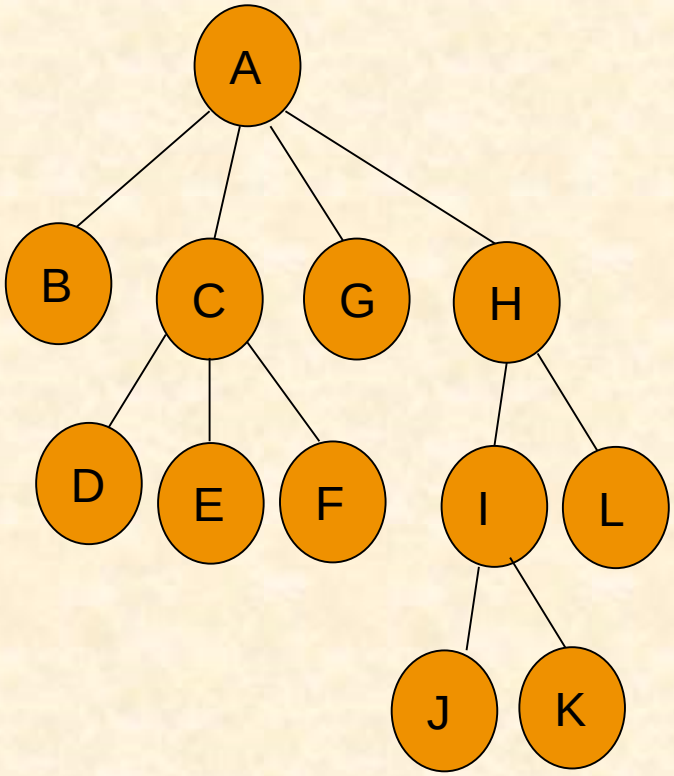
2. 假设树中结点个数小于n(n≥2)时定理成立。

3. 当树中有n个结点时，由树的先根序列可知，第一个结点是根结点，设该结点的次数为k，k≥1，因此根结点有k个子树。第一个子树排在最前面，第k个子树排在最后面，并且每个子树的结点个数小于n，由归纳假设可知，每个子树可以唯一确定，从而整棵树的树形可以唯一确定。

证毕。

例：先根序列： **A B C D E F G H I J K L**

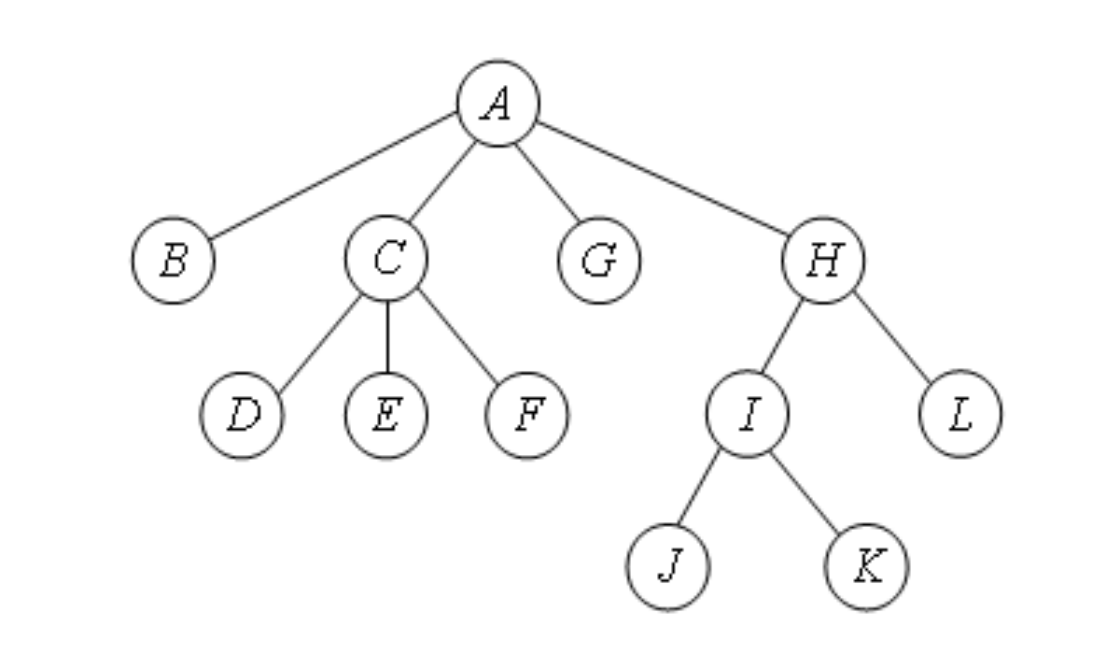
结点的度数： **4 0 3 0 0 0 0 2 2 0 0 0**



2) 后根序列和结点度数表示法

后根序列　　B D E F C G J K I L H A

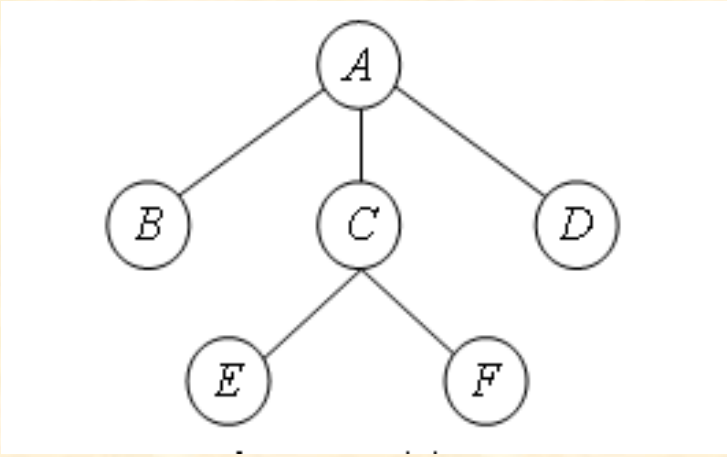
结点的度数　0 0 0 0 3 0 0 0 2 0 2 4



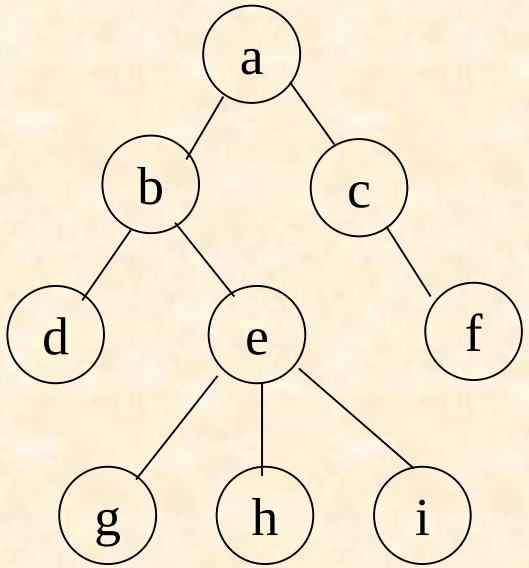
3) 层次序列和结点度数表示法

已知一个树的层次序列和每个结点度数，则能唯一确定该树的结构。

层次序列	A	B	C	D	E	F
结点的度数	3	0	2	0	0	0



4) 层次序列和Father数组表示法（双亲表示法）



	data	parent
0	0	9
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

0号单元不用或存结点个数

如何找孩子结点

双亲表示法数据结构定义

```
#define MAX_TREE_SIZE 100
typedef  struct PTNode {    //节点结构
    TElemType  data;
    int              parent;    //双亲位置域
} PTNode;
typedef struct {   //树结构
    PTNode nodes[MAX_TREE_SIZE];
    int  r, n;          //根的位置和节点数
} PTree;
```

6.4 树和森林

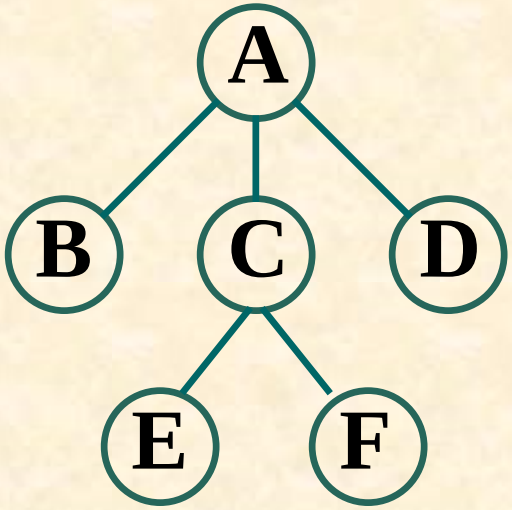
6.4.1 树与二叉树的转换

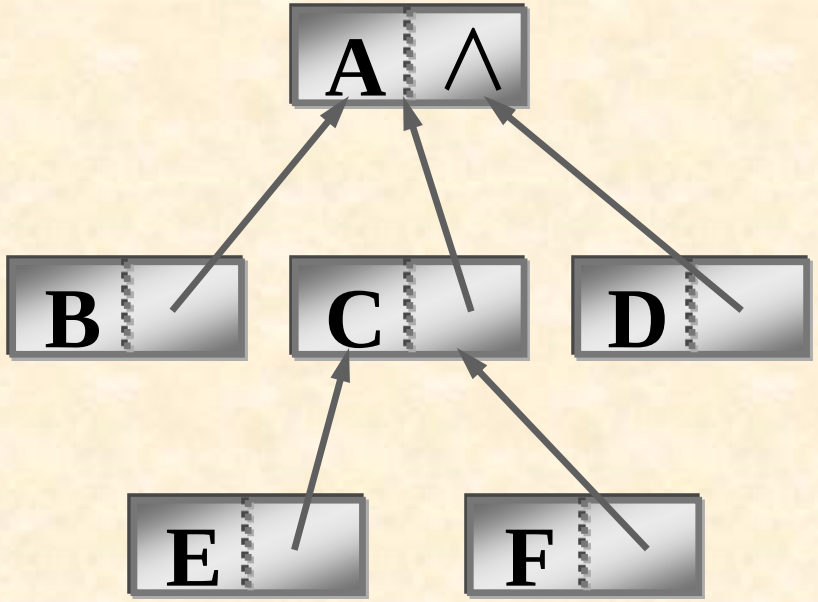
6.4.2 树和森林的遍历

6.4.3 树的顺序存储

6.4.4 树的链接存储

(1) Father链接结构:





*Father*链接提供了“向上”访问的能力，但很难确定一个结点是否为叶结点，也很难求结点的子结点，且不易实现遍历操作。

(2)孩子链接结构

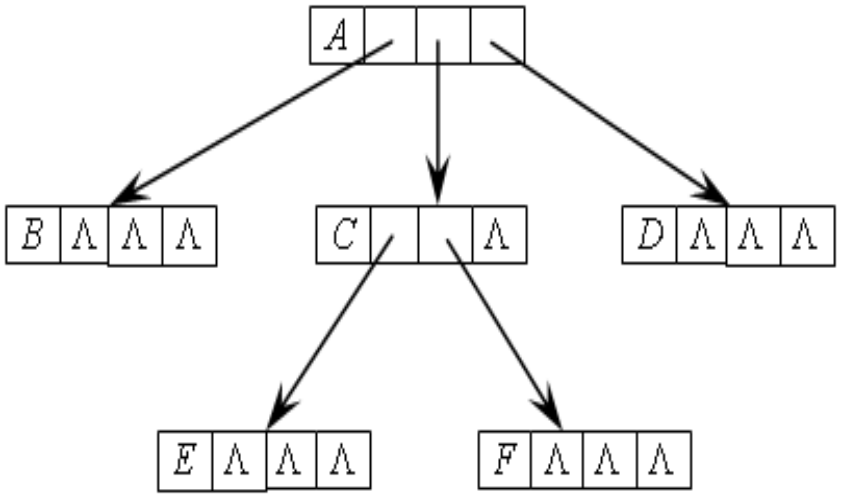


图 4.36 结点大小固定的链接结构

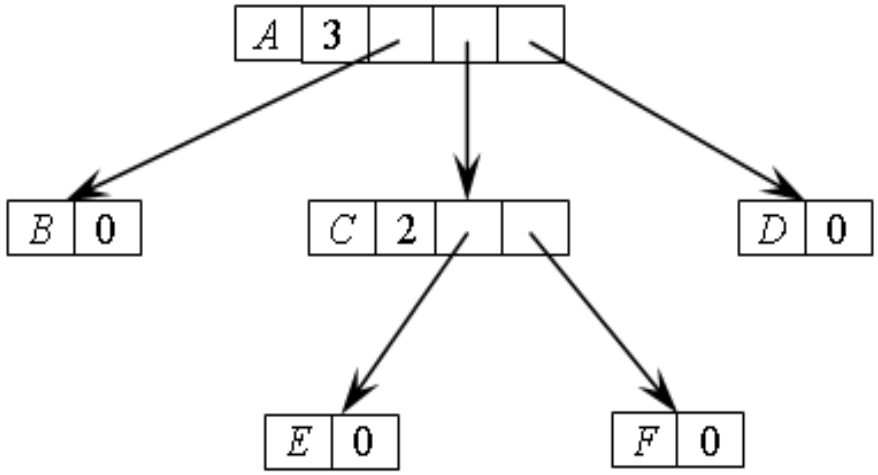


图 4.37 结点大小不固定的链接结构

*会出现大量指针为空的情况，浪费空间。

*节省了空间，但给操作和管理带来不便。

(3)孩子链表表示法

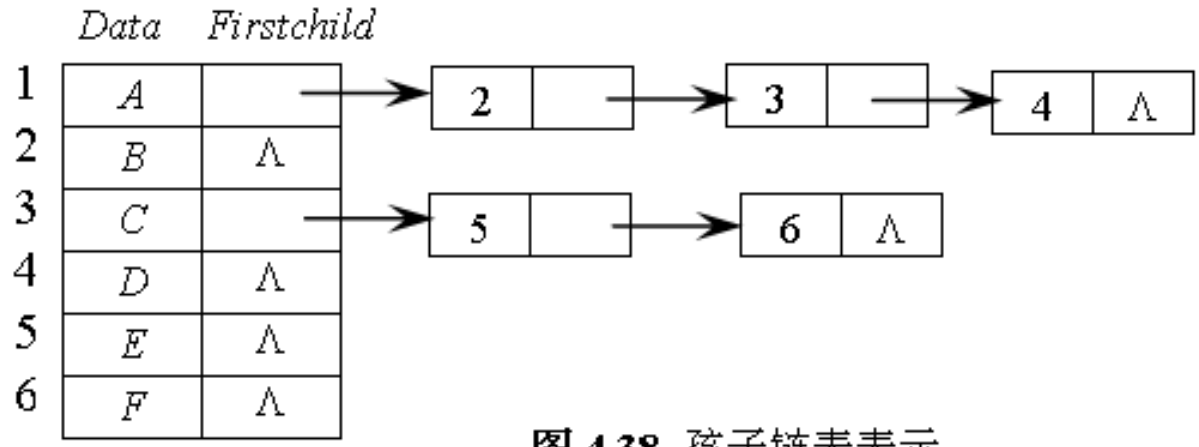
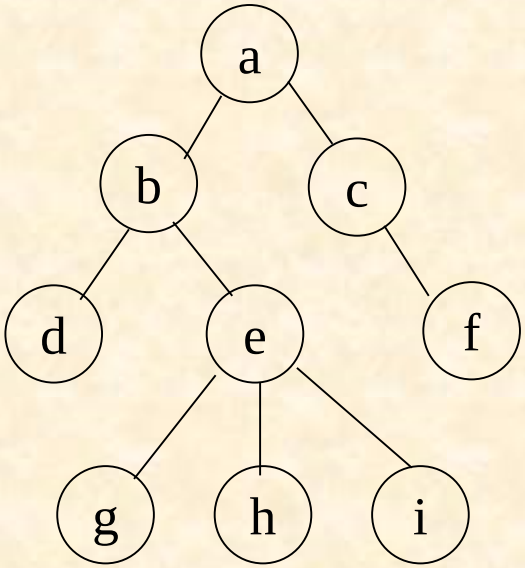


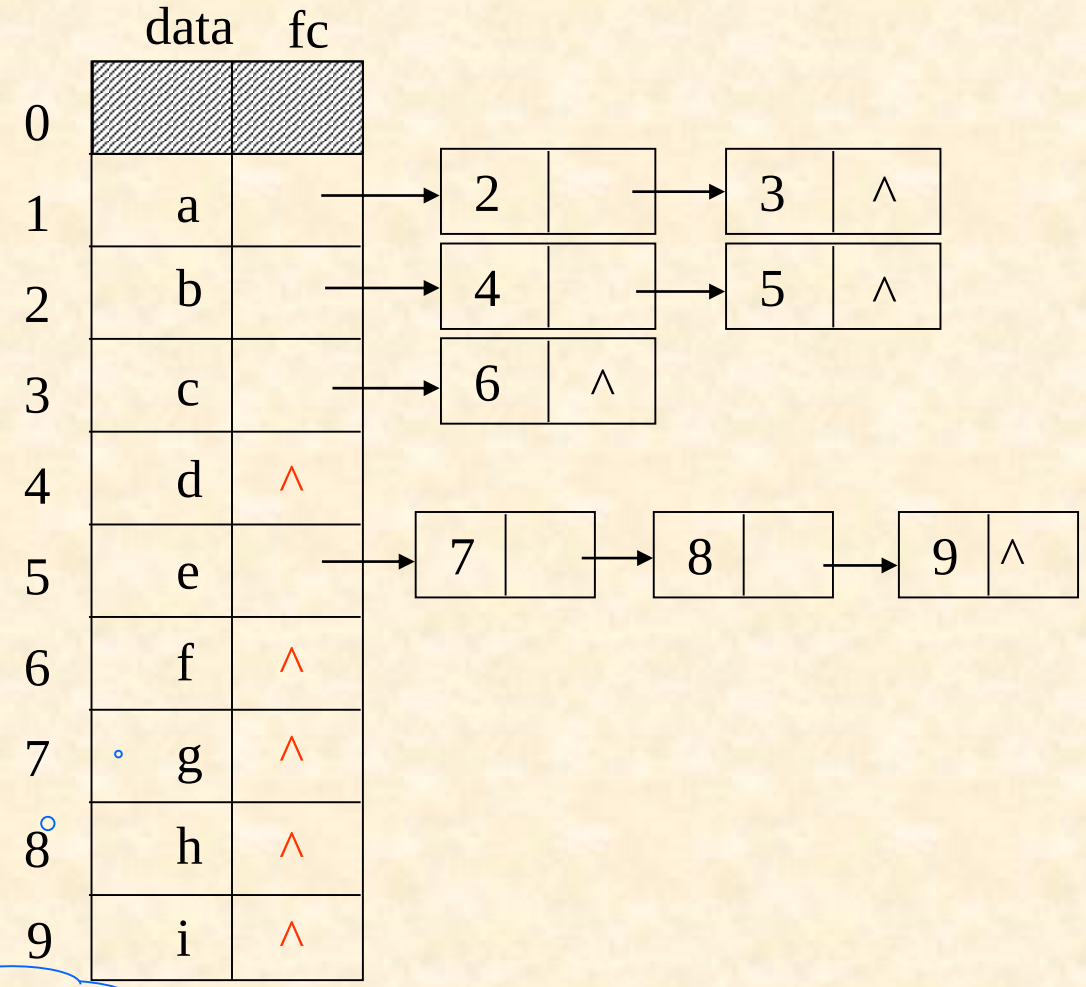
图 4.38 孩子链表表示

*孩子链表表示法是为树中每个结点设置一个孩子链表，并将这些结点及相应的孩子链表的头指针存放在一个数组中。孩子结点的数据域仅存放了它们在数组空间的下标。

*与Father链接表示法相反，孩子链表表示便于实现涉及孩子及其子孙的运算，但不便于实现与父结点有关的操作。

孩子表示法例

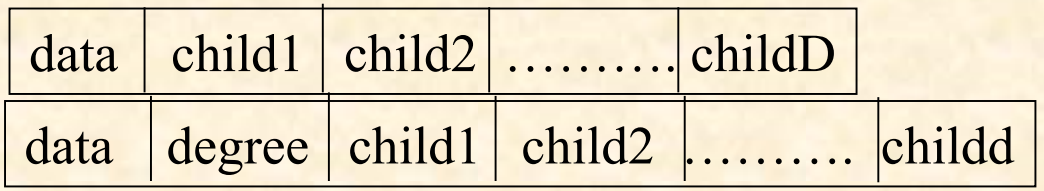




如何找双亲结点

树的孩子链表表示法

```
typedef  struct CTNode {    //孩子节点
    int  child;
    struct CTNode *next;
} *ChildPtr;
typedef struct {
    TElemType  data;
    ChildPtr firstchild;    //孩子链表头指针
} CTBox;
typedef struct {
    CTBox nodes[MAX_TREE_SIZE];
    int n, r;
} CTree;
```



(4)父亲孩子链表表示

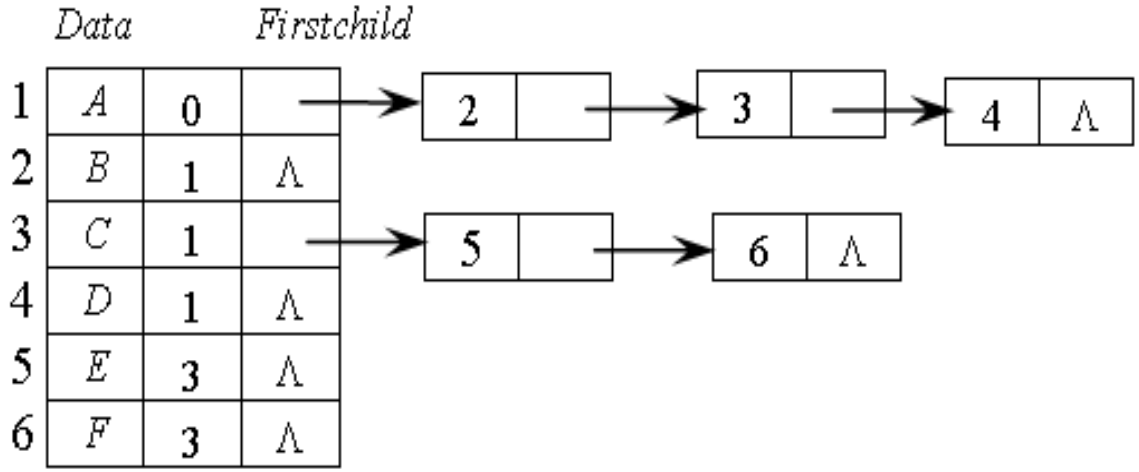
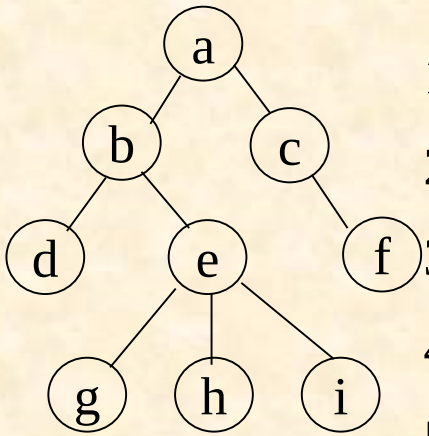


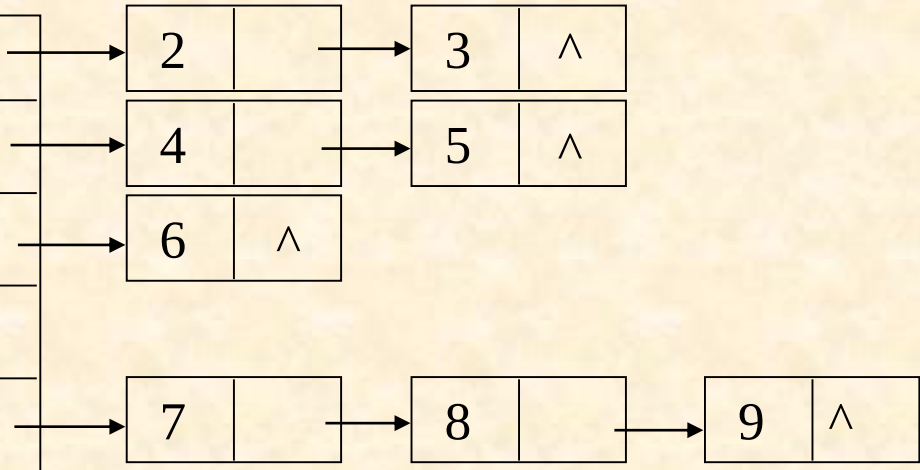
图 4.39 父亲孩子链表表示

*将*Father*数组表示法和孩子链表表示法结合起来，可形成父亲孩子链表表示法。

带双亲的孩子链表



	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	∧



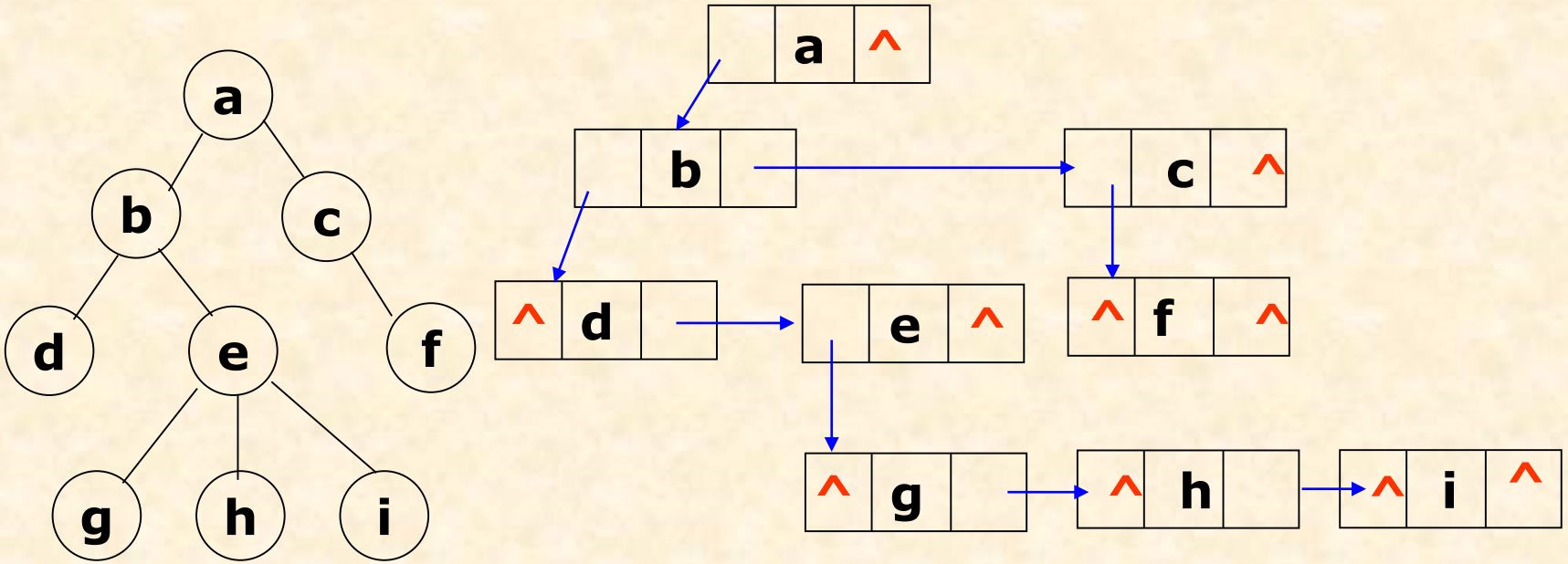
(5)左孩子-右兄弟链接结构（二叉树表示法）

结点结构

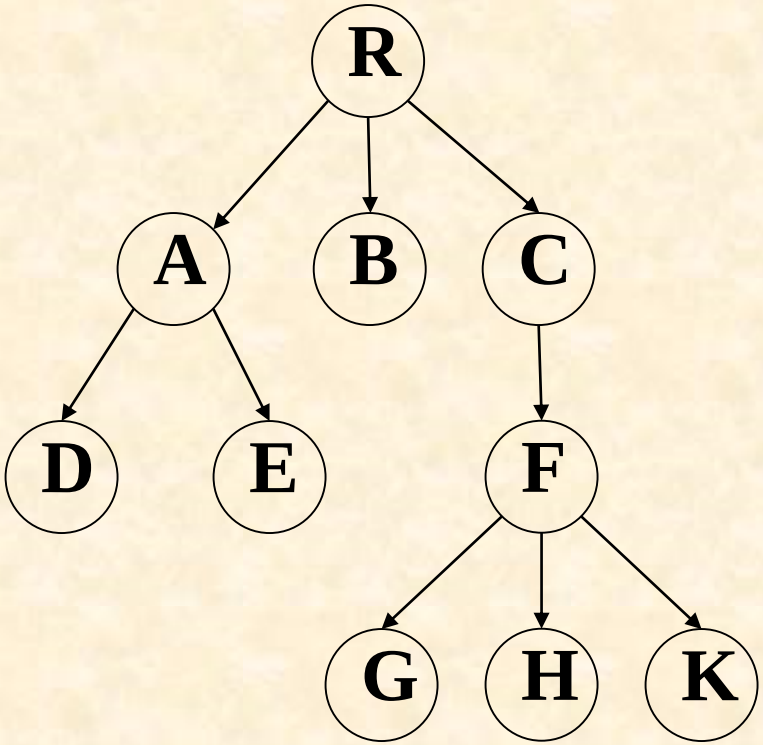
*这种存储结构的最大优点是：它和二叉树的二叉链表表示完全一样。可利用二叉树的算法来实现对树的操作。

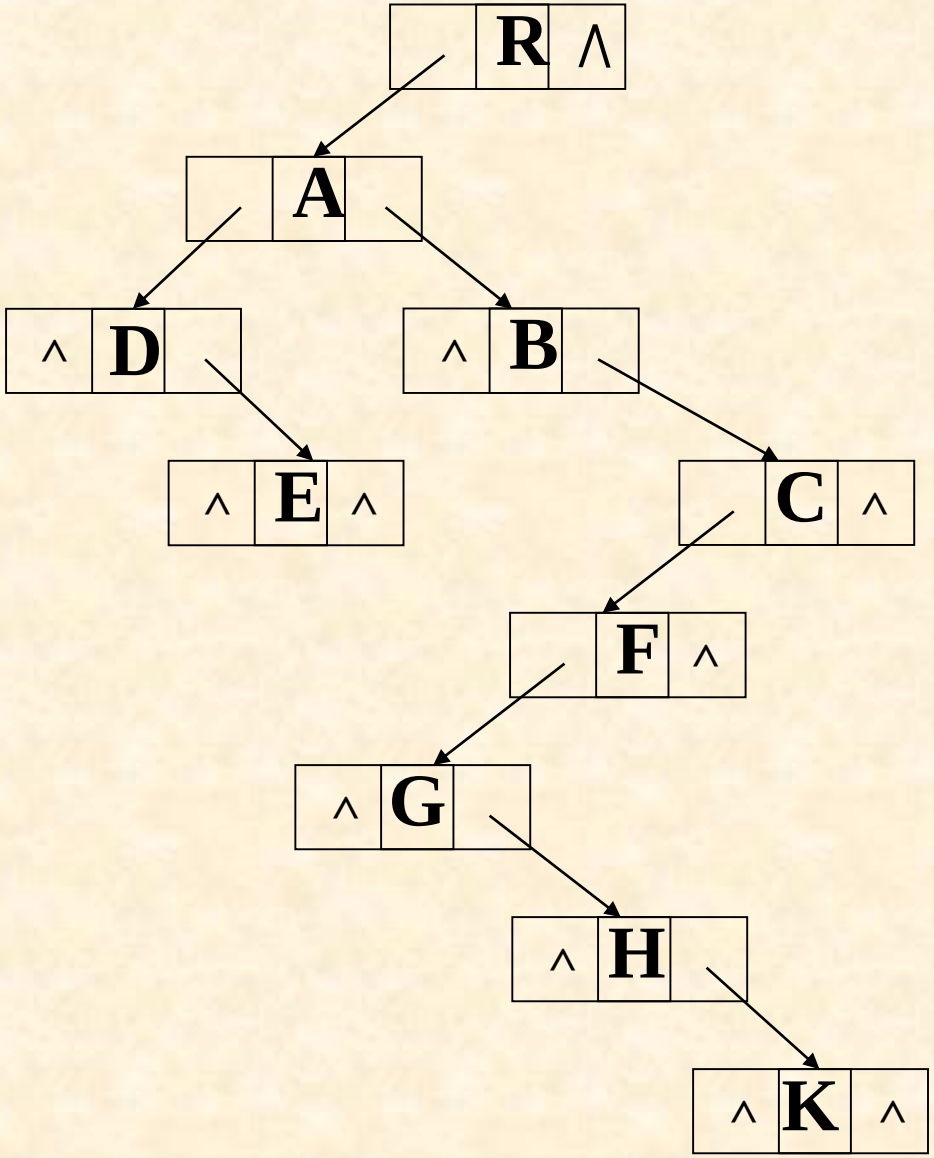
树的父子兄弟表示法

- 实现：用二叉链表作树的存储结构，链表中每个结点的两个指针域分别指向其第一个孩子结点和下一个兄弟结点
- 特点
  - 操作容易
  - 破坏了树的层次



左孩子-右兄弟表示法示例：

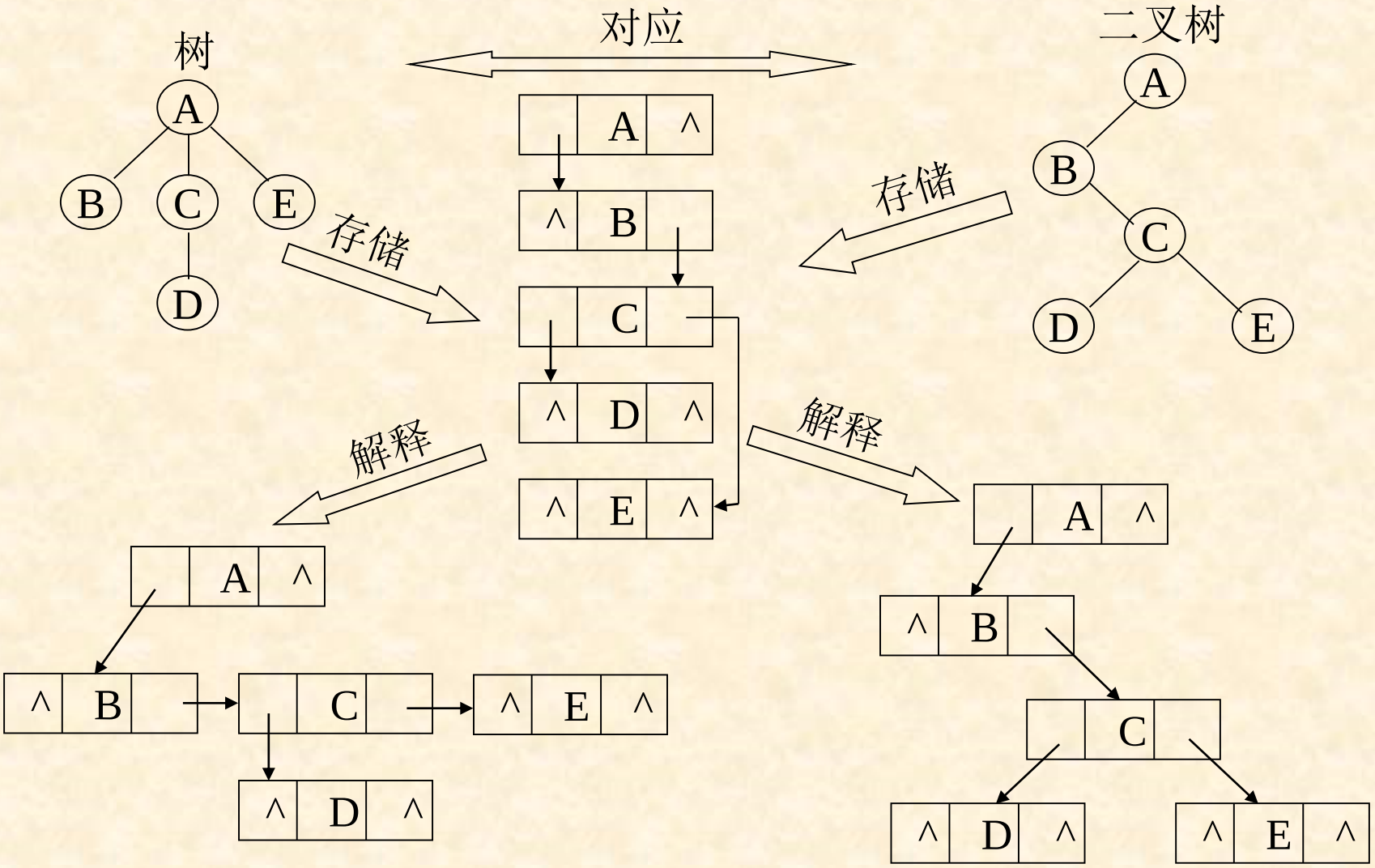




树的父子兄弟表示法

```
typedef  struct CSNode
{
    ElemType  data;
    struct CSNode *firstchild, *nextsibling;
} CSNode, *CSTree;
```

树与二叉树转换解释



树的存储结构小结

- 双亲表示法
  - 实现:定义结构数组存放树的结点,每个结点含两个域:
    - ✧数据域:存放结点本身信息
    - ✧双亲域:指示本结点的双亲结点在数组中位置
  - 特点:找双亲容易，找孩子难
- 孩子表示法
  - 多重链表:每个结点有多个指针域,分别指向其子树的根
    - ✧结点同构：结点的指针个数相等，为树的度D
    - ✧结点不同构：结点指针个数不等，为该结点的度d
  - 孩子链表：每个结点的孩子结点用单链表存储，再用含n个元素的结构数组指向每个孩子链表
- 父子表示法

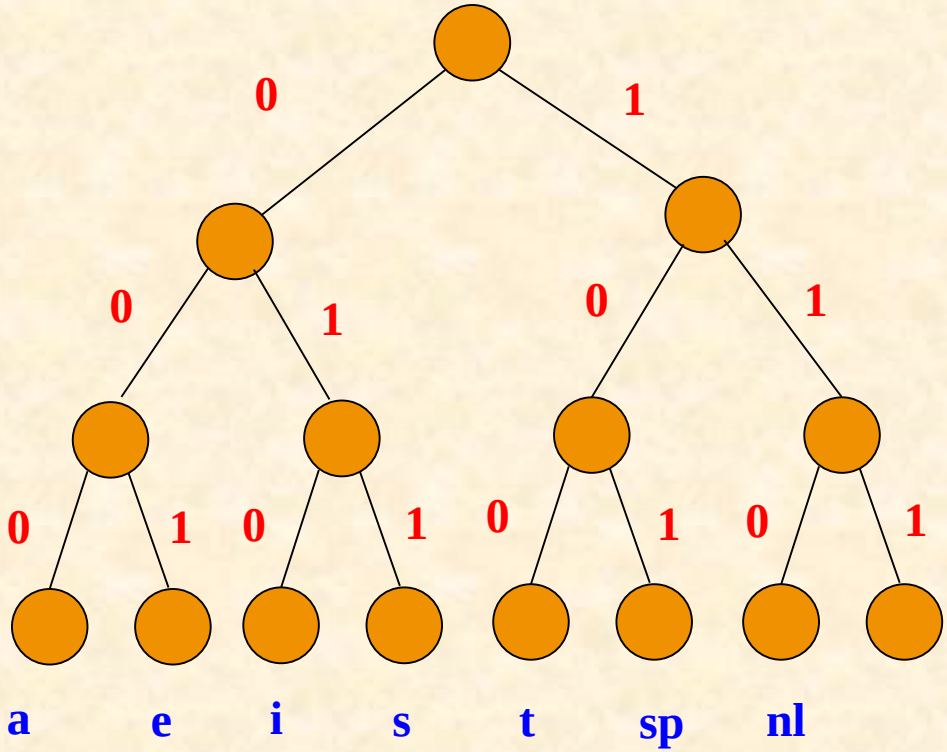
第六章　树

数据压缩是计算机科学中的重要技术。数据压缩过程称为编码，即将文件中的每个字符均转换为一个唯一的二进制位串。

数据解压过程称为解码，即将二进制位串转换为对应的字符。压缩的关键在于编码的方法，哈夫曼编码是一种最常用无损压缩编码方法。

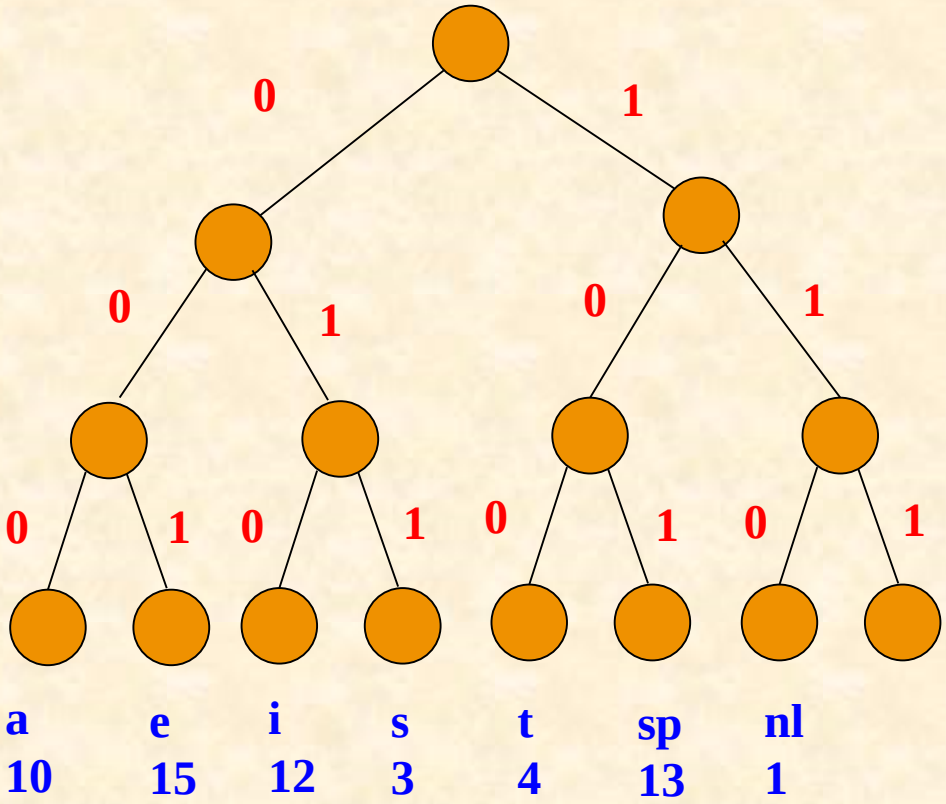
文件编码

假设有一个文件仅包含7个字符：*a*、*e*、*i*、*s*、*t*、*sp*(空格)和*nl*(换行)，且文件中有10个*a*，15个*e*，12个*i*，3个*s*，4个*t*，13个*sp*，1个*nl*。因为$\lceil \log_2 7 \rceil = 3$，所以，每个字符都至少由一个3位的二进制串表示。



文件的总位数是：

$10\times3 + 15\times3 + 12\times3 + 3\times3 + 4\times3 + 13\times3 + 1\times3 = 174$.



在实际应用的一些大文件中，字符被使用的比率是**非平均的**，即有些字符出现的**次数多**，而有些字符出现的次数却**非常少**。如果所有字符都由等长的二进制码表示，那么将造成**空间浪费**。

假设有一个文件仅包含7个字符：*a*、*e*、*i*、*s*、*t*、*sp*(空格)和*nl*(换行)，且文件中有10个*a*，15个*e*，12个*i*，3个*s*，4个*t*，13个*sp*，1个*nl*。

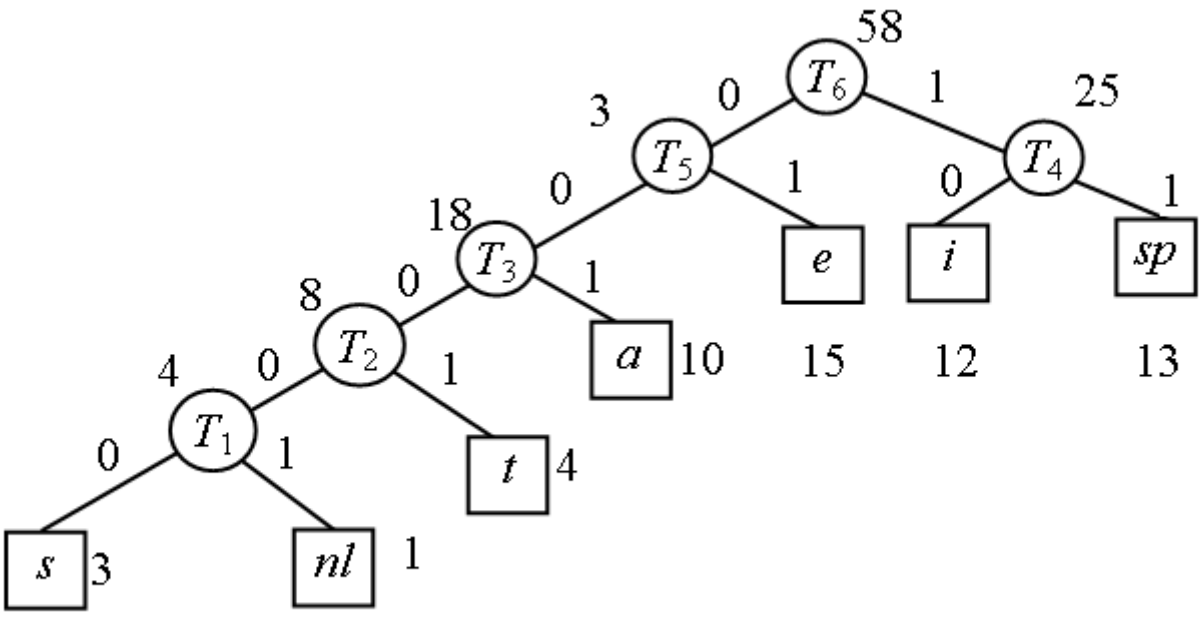


图 4.44 哈夫曼树

文件的总位数是：

$3\times5+1\times5+4\times4+10\times3+15\times2+12\times2+13\times2=146<176.$

如何才能减少不必要的空间浪费呢？文件压缩的通常策略是：**采用不等长的二进制码**，令文件中出现**频率高**的字符的**编码尽可能短**。

- 但是，采用不等长编码又可能会产生**多义性**。例如：如果用01表示*a*，10表示*b*，1001表示*c*，那么对于编码1001，我们无法确定它表示字符*c* ，还是表示字符串*ba* ，其原因是*b*的编码与*c*的编码的开始（前缀）部分相同。
- 为了避免出现多义性，必须要求字符集中**任何字符的编码都不是其它字符的编码的前缀**，满足这个条件的编码被称为**前缀码**。显然，**等长编码是前缀码**。

- 如何得到前缀码编码？——二叉树
- 怎样的前缀码才能使文件的总编码长度最短？

设组成文件的字符集$A=\{a_1,a_2,\ldots,a_n\}$，其中，a_i的编码长度为l_i；a_i出现的次数为c_i 。要使文件的总编码最短，就必须要确定l_i，使 $\sum_{i=1}^{n} c_i l_i$ 取最小值.

- 如何设计出使上式最小的前缀码？

Huffman于1952年提出了哈夫曼算法——最优二叉树

文件编码的基本思想

假设有一个文件仅包含7个字符：*a*、*e*、*i*、*s*、*t*、*sp*(空格)和*nl*(换行)，且文件中有10个*a*，15个*e*，12个*i*，3个*s*，4个*t*，13个*sp*，1个*nl*。

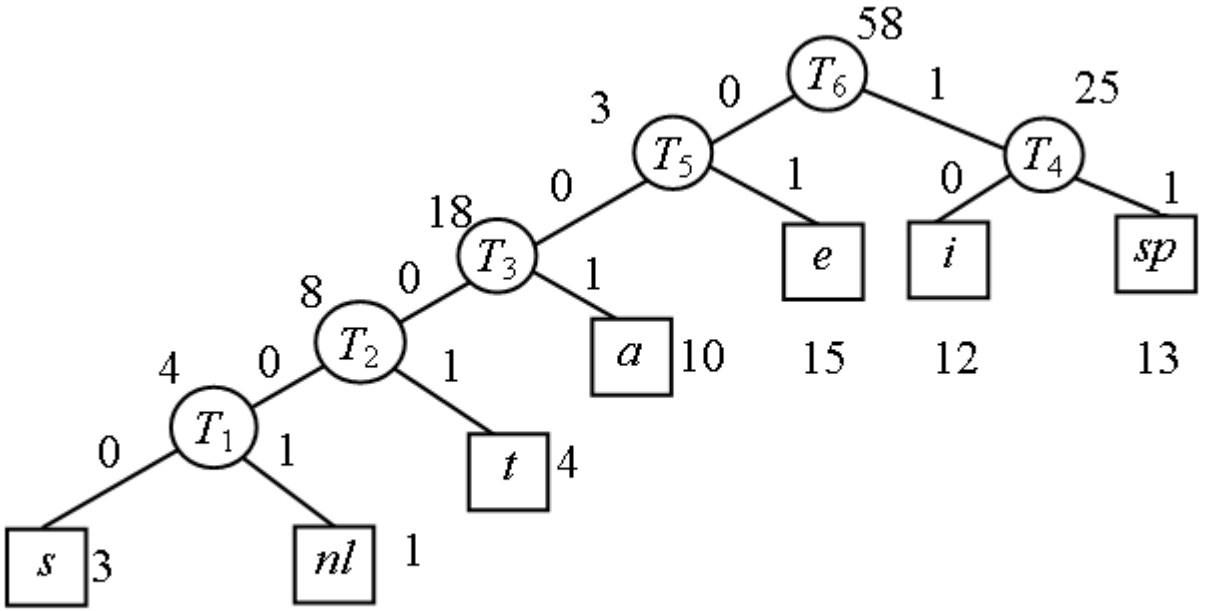


图 4.44 哈夫曼树

文件编码问题就变成**构造最优二叉树**问题，每个**叶子结点**代表一个**字符**，该**字符的频率**作为其**权值**，从根到外结点的**路径长度**就是该字符的**编码长度**．

为求得**最优二叉树**，哈夫曼巧妙的设计了**哈夫曼算法**，通过哈夫曼算法可以建立一棵**哈夫曼树**，从而为压缩文本文件建立**哈夫曼编码**。

哈夫曼树基本思想

■ **哈夫曼树(Huffman)——带权路径长度最短的树**

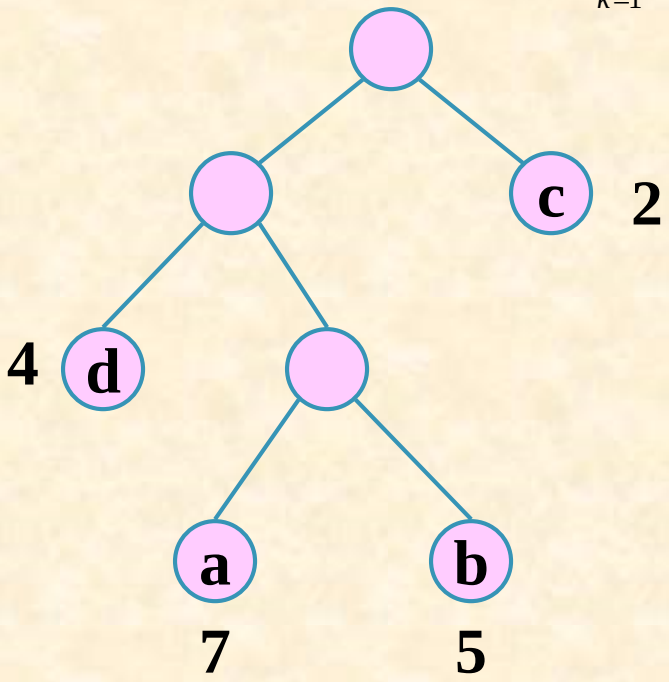
- 路径：从树中一个结点到另一个结点之间的分支构成这两个结点间的~
- 路径长度：路径上的分支数
- 树的路径长度：从树根到每一个结点的路径长度之和
- 树的带权路径长度：树中所有带权结点的路径长度之和

- Huffman树——设有n个权值$\{w_1,w_2,\ldots\ldots w_n\}$，构造一棵有n个叶子结点的二叉树，每个叶子的权值为w_i，**则树的带权路径长度wpl最小的二叉树叫哈夫曼树**

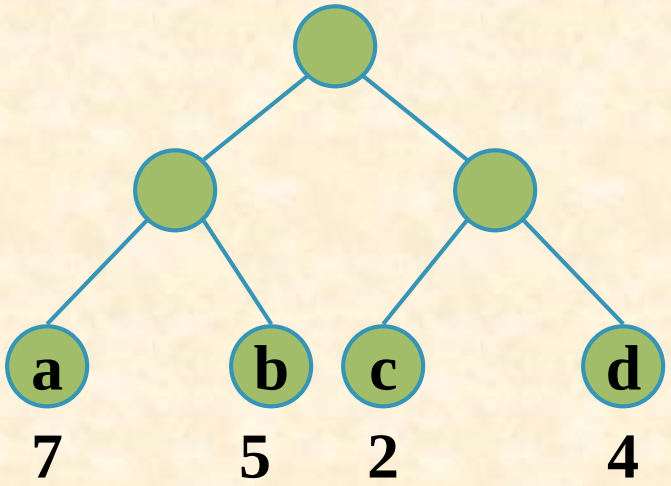
$$WPL=\sum_{k=1}^{n}W_K L_K$$

例　有4个结点，权值分别为7，5，2，4，构造有4个叶子结点的二叉树

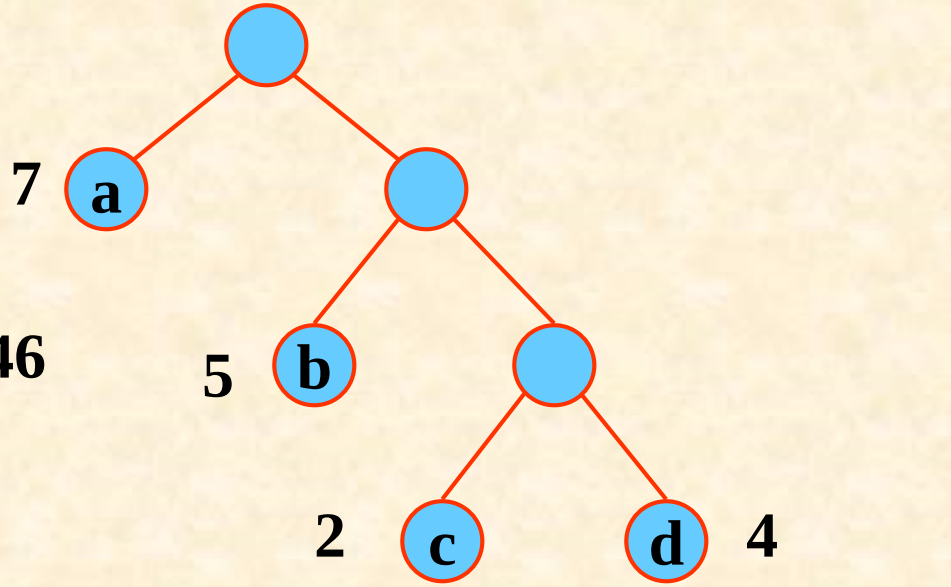
$$WPL = \sum_{k=1}^{n} W_K L_K$$



WPL=7*3+5*3+2*1+4*2=46



WPL=7*2+5*2+2*2+4*2=36

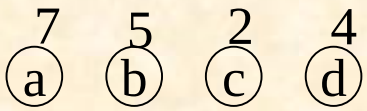


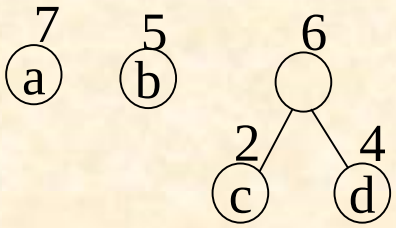
WPL=7*1+5*2+2*3+4*3=35

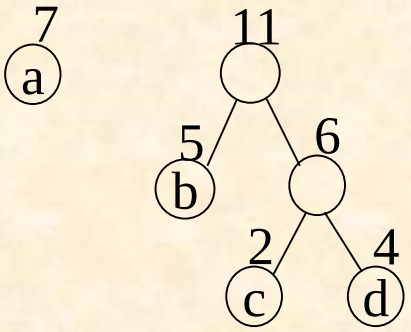
◆Huffman算法的基本原理

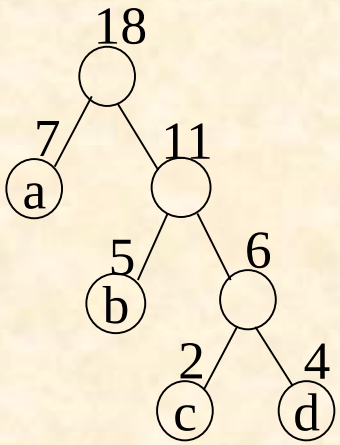
- ✧根据给定的n个权值{w_1,w_2,……w_n}，构造n棵只有根结点的二叉树，令其权值为w_j
- ✧在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- ✧在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- ✧重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树

例

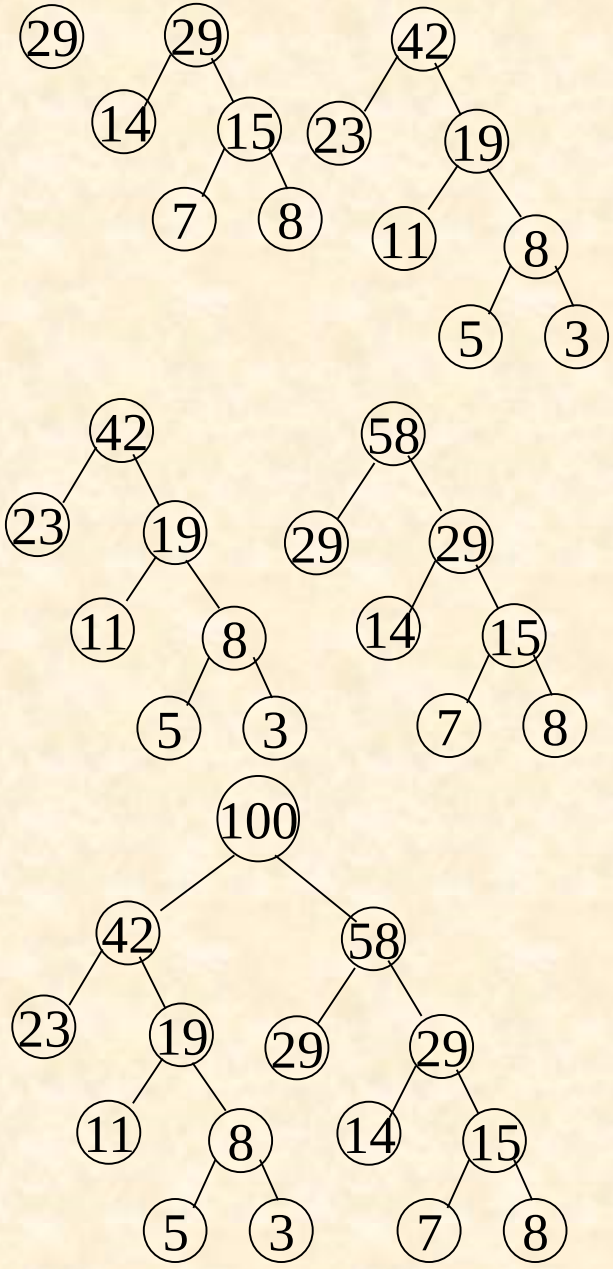








例　w={5, 29, 7, 8, 14, 23, 3, 11}



Huffman算法实现

✧一棵有n个叶子结点的Huffman树有2n-1个结点

✧采用顺序存储结构——一维结构数组

✧结点类型定义

```
typedef  struct {
    unsigned int weight;
    unsigned int parent, lchild, rchild;
} HTNode, *HuffmanTree;
```

	lc	data	rc	pa
1	0	7	0	0
2	0	5	0	0
3	0	2	0	0
4	0	4	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0

(1)

	lc	data	rc	pa
1	0	7	0	0
2	0	5	0	0
3	0	2	0	5
4	0	4	0	5
k→5	3	6	4	0
6	0	0	0	0
7	0	0	0	0

(2)

x1=3,x2=4

m1=2,m2=4

	lc	data	rc	pa
1	0	7	0	0
2	0	5	0	6
3	0	2	0	5
4	0	4	0	5
5	3	6	4	6
k→6	2	11	5	0
7	0	0	0	0

(3)

x1=2,x2=5

m1=5,m2=6

	lc	data	rc	pa
1	0	7	0	7
2	0	5	0	6
3	0	2	0	5
4	0	4	0	5
5	3	6	4	6
6	2	11	5	7
k→7	1	18	6	0

(4)

x1=1,x2=6

m1=7,m2=11

- **定理6.3** 在外结点权值分别为$w_0,w_1,\ldots,w_{n-1}$的扩充二叉树中，由哈夫曼算法构造出的哈夫曼树的带权路径长度最小，因此哈夫曼树为最优二叉树。

- 由观察可知，字符集中的字符所在的结点均是哈夫曼树中的外结点。哈夫曼树中没有度为1的结点。

哈夫曼编码：将哈夫曼树每个分支结点的**左分支标上0，右分支标上1**，把**从根结点到每个叶结点**的路径上的**标号连接起来**，作为**该叶结点所代表的字符的编码**，这样得到的编码称为**哈夫曼编码**。

根据定理6.3知道，对于所有的编码，**哈夫曼编码使文件的总编码长度最短**。实际上，哈夫曼算法应用很广，这里只是以哈夫曼编码为例来说明哈夫曼算法。

文件编码

假设有一个文件仅包含7个字符：*a*、*e*、*i*、*s*、*t*、*sp*(空格)和*nl*(换行)，且文件中有10个*a*，15个*e*，12个*i*，3个*s*，4个*t*，13个*sp*，1个*nl*。

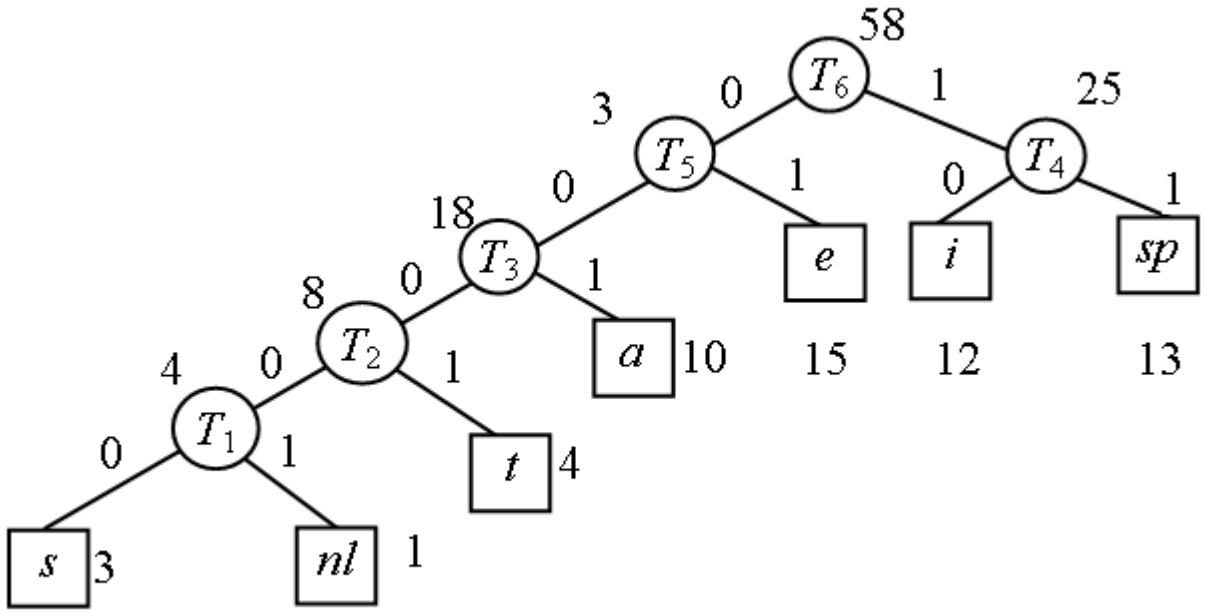


图 4.44 哈夫曼树

哈夫曼编码

哈夫曼编码的主要用途是实现数据压缩。

设给出一段报文：

CAST CAST SAT AT A TASA

字符集合是 **{ C, A, S, T }**

若给每个字符以等长编码

A : 00　T : 10　C : 01　S : 11

01001110 01001110 110010 0010 00 10001100

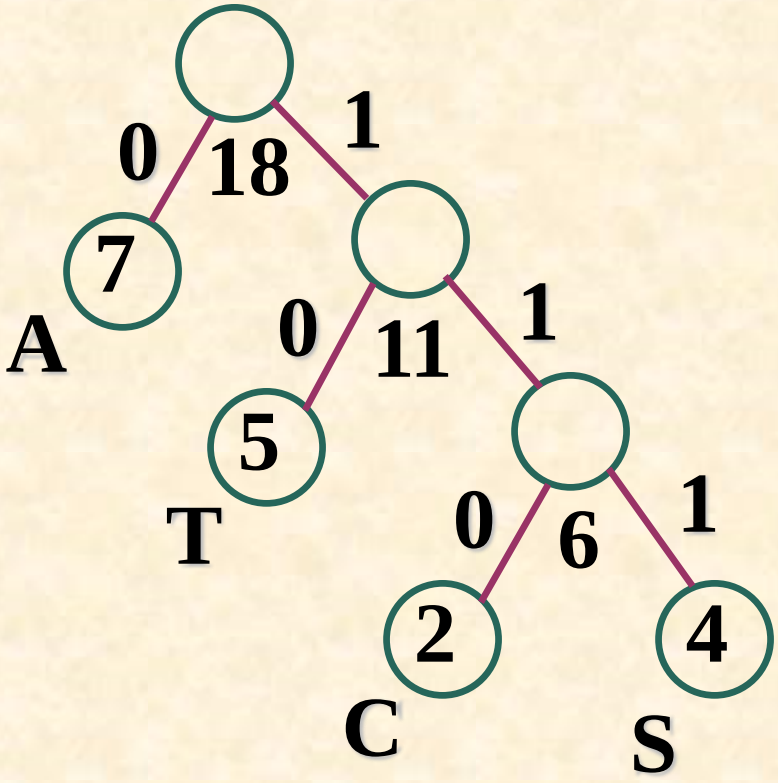
则总编码长度为 **18 * 2 = 36.**

[例] 哈夫曼编码

报文：CAST CAST SAT AT A TASA

字符集合是 { C, A, S, T },

各个字符出现的频度(次数)是 W＝{ 2, 7, 4, 5 }



编码

A (0)

T (10)

C (110)

S (111)

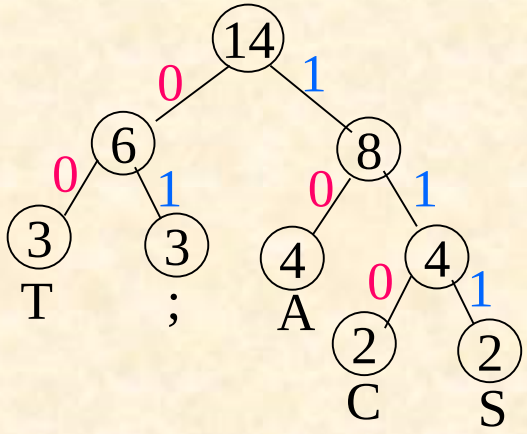
总编码长度：

1*7+2*5+3*2+3*4=35

- 在构造哈夫曼树的过程中，没有一片树叶是其他树叶的祖先，所以每个叶结点对应的编码不可能是其他叶结点对应的编码的前缀，由此可知哈夫曼编码是二进制的前缀码。
- 哈夫曼编码是否唯一？

- 编码：依次将数据文件中的字符按哈夫曼树转换成哈夫曼编码。
- 解码：依次读入文件的二进制码，从哈夫曼树的根结点出发，若当前读入0，则走向其左孩子，否则走向其右孩子，到达某一叶结点时，便可以译出相应的字符。

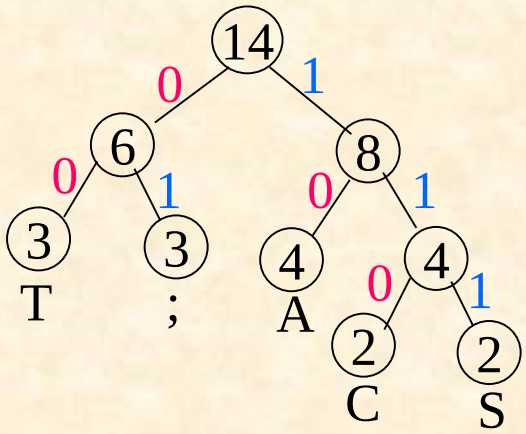
- Huffman编码：数据通信用的二进制编码
  - 思想：根据字符出现频率编码，使电文总长最短
  - 编码：根据字符出现频率构造Huffman树，然后将树中结点引向其左孩子的分支标“0”，引向其右孩子的分支标“1”；每个字符的编码即为从根到每个叶子的路径上得到的0、1序列



例 要传输的字符集 D={C,A,S,T, ; }
字符出现频率 w={2,4,2,3,3}

T : 00
; : 01
A : 10
C : 110
S : 111

✧译码：从Huffman树根开始，从待译码电文中逐位取码。若编码是“0”，则向左走；若编码是“1”，则向右走，一旦到达叶子结点，则译出一个字符；再重新从根出发，直到电文结束



T： 00
; ： 01
A： 10
C： 110
S： 111

例 电文是{CAS;CAT;SAT;AT}
其编码 “11010111011101000011111000011000”
电文为“1101000”
译文只能是“CAT”

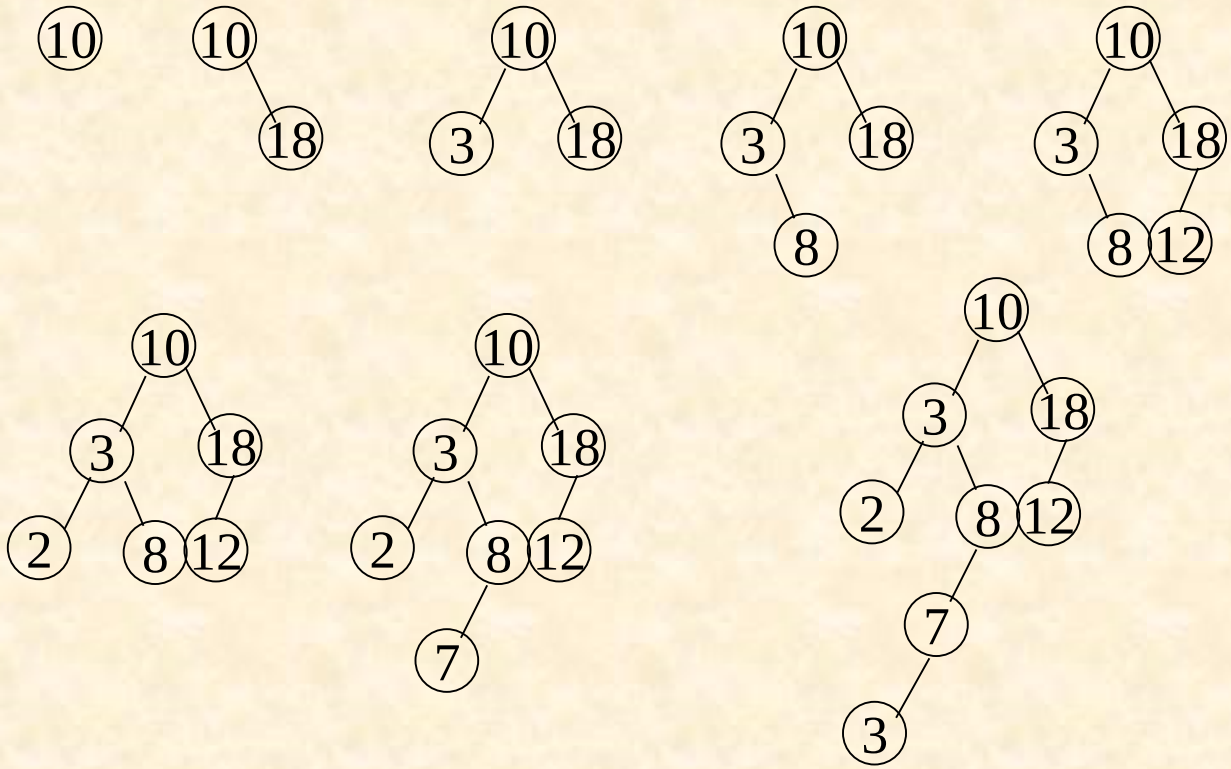
第六章　树

二叉排序树

- 定义：二叉排序树或是一棵空树，或是具有下列性质
  - 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值
  - 若它的右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值
  - 它的左、右子树也分别为二叉排序树
- 二叉排序树的插入
  - 插入原则：若二叉排序树为空，则插入结点应为新的根结点；否则，继续在其左、右子树上查找，直至某个叶子结点的左子树或右子树为空为止，则插入结点应为该叶子结点的左孩子或右孩子
  - 二叉排序树生成：从空树出发，经过一系列的查找、插入操作之后，可生成一棵二叉排序树

¤插入算法

例 {10， 18， 3， 8， 12， 2， 7， 3}



中序遍历二叉排序树可得到一个关键字的有序序列

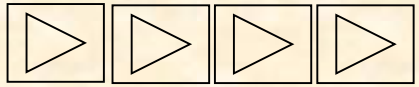
✧二叉排序树的删除

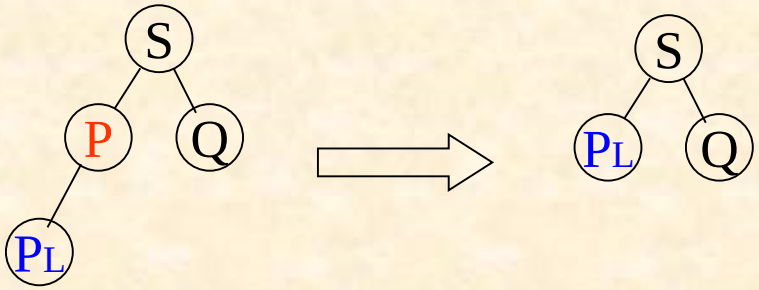
要删除二叉排序树中的p结点，分三种情况：

- p为叶子结点，只需修改p双亲f的指针

  f->lchild=NULL　　f->rchild=NULL

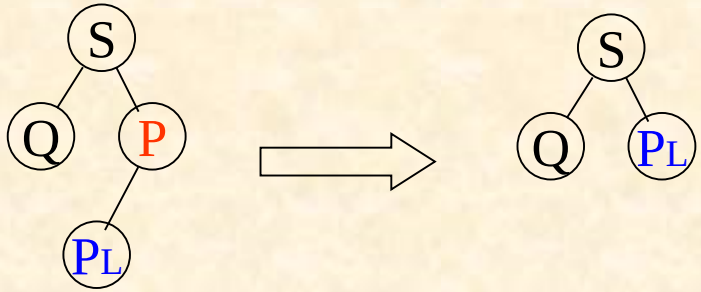
- p只有左子树或右子树
  - ✧p只有左子树，用p的左孩子代替p　　(1)(2)
  - ✧p只有右子树，用p的右孩子代替p　　(3)(4)
- p左、右子树均非空
  - ✧沿p左子树的根C的右子树分支找到S，S的右子树为空，将S的左子树成为S的双亲Q的右子树，用S取代p　(5)
  - ✧若C无右子树，用C取代p　　(6)





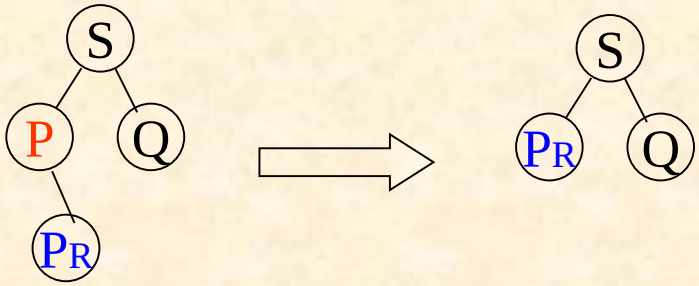
中序遍历：PL P S Q　　中序遍历：PL S Q

(1)



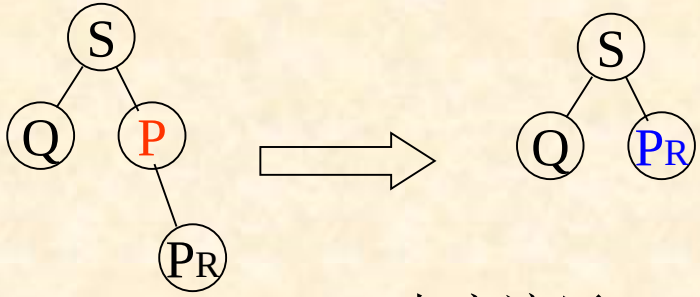
中序遍历：Q S PL P　　中序遍历：Q S PL

(2)



中序遍历：**P PR S** Q　　中序遍历：**PR S** Q

(3)



中序遍历：Q **S P PR**　　中序遍历：Q **S PR**

(4)

✧二叉排序树的删除

要删除二叉排序树中的p结点，分三种情况：

¤p为叶子结点，只需修改p双亲f的指针

f>lchild=NULL　　f->rchild=NULL

¤p只有左子树或右子树

✧p只有左子树，用p的左孩子代替p　　(1)(2)

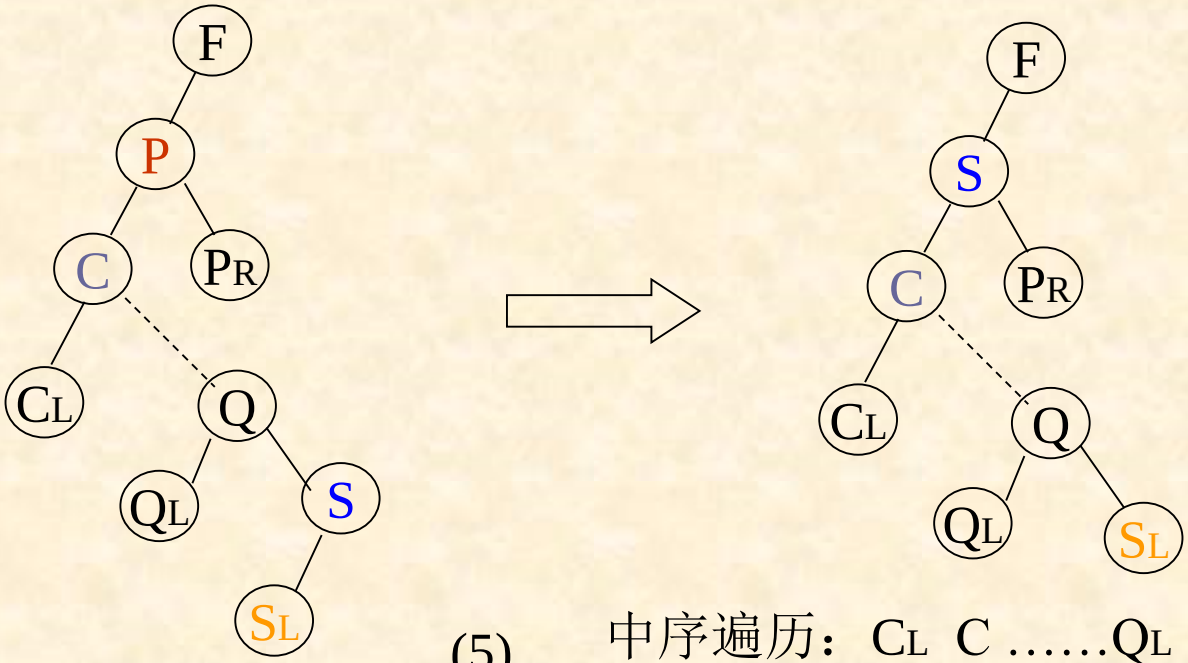
✧p只有右子树，用p的右孩子代替p　　(3)(4)

¤p左、右子树均非空

✧沿p左子树的根C的右子树分支找到S，S的右子树为空，将S的左子树成为S的双亲Q的右子树，用S取代p　(5)

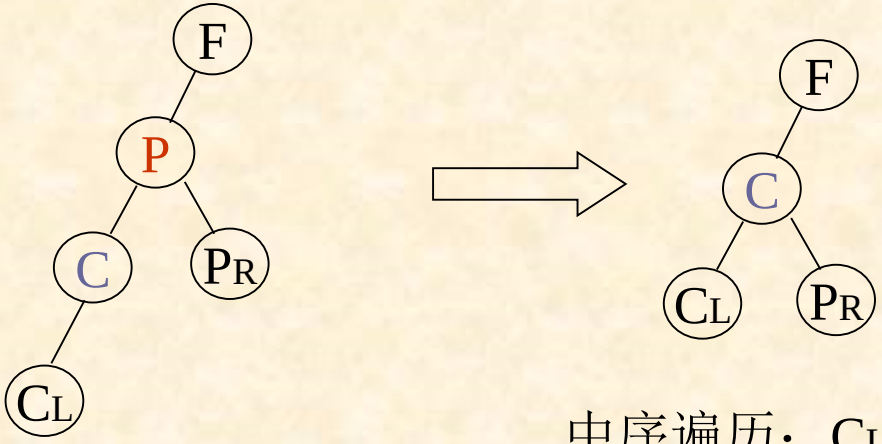
✧若C无右子树，用C取代p　　(6)

中序遍历：C_L C ……Q_L Q **S_L S P P_R** F



中序遍历：C_L C ……Q_L Q **S_L S P_R** F

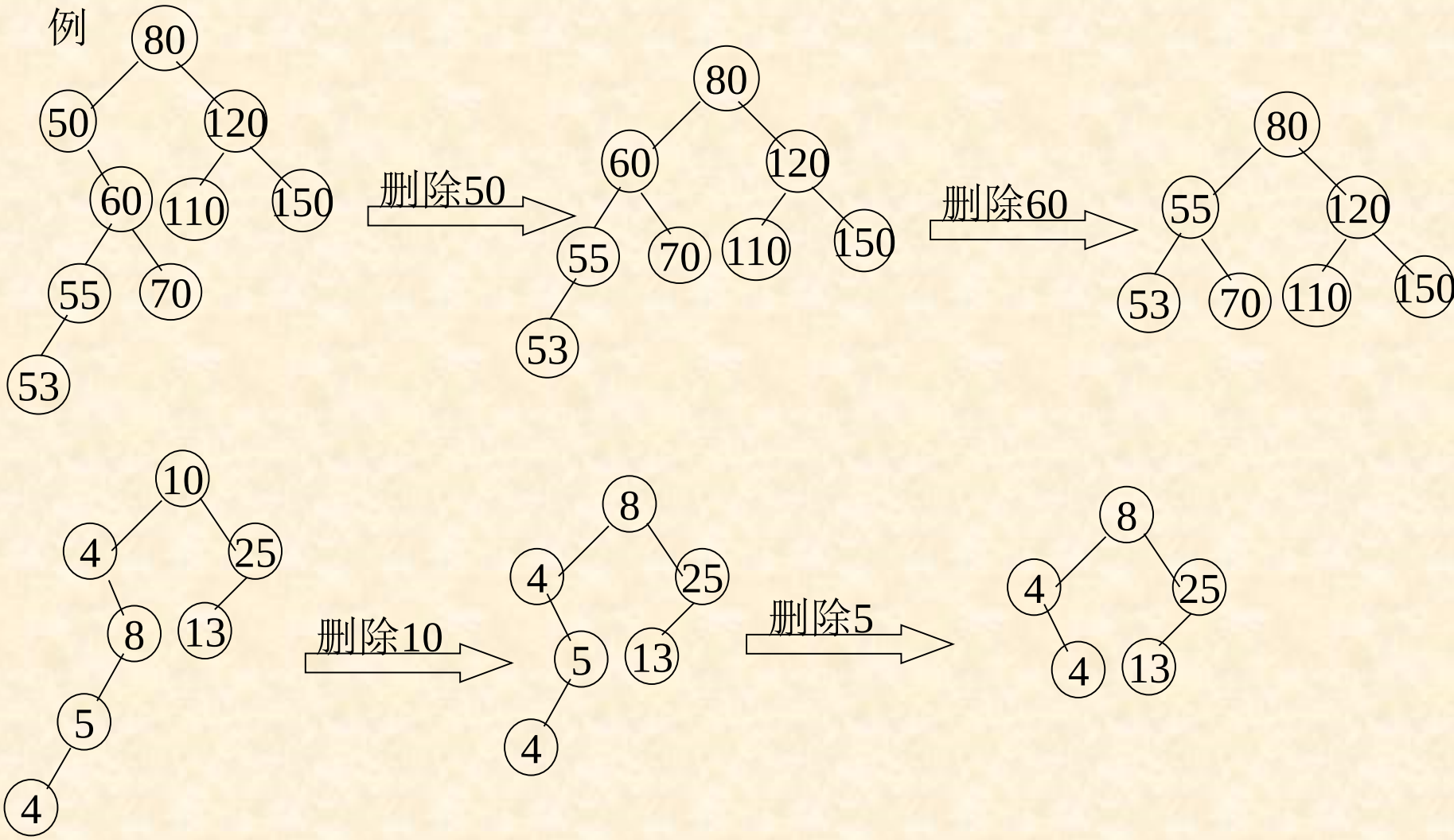
(5)



中序遍历：C_L C P_R F

中序遍历：C_L C P P_R F (6)

¤删除算法


例
80
50
120
60
110
150
55
70
53
删除50
80
60
120
55
70
110
150
53
删除60
80
55
120
53
70
110
150
10
4
25
8
13
5
4
删除10
8
4
25
5
13
4
删除5
8
4
25
4
13