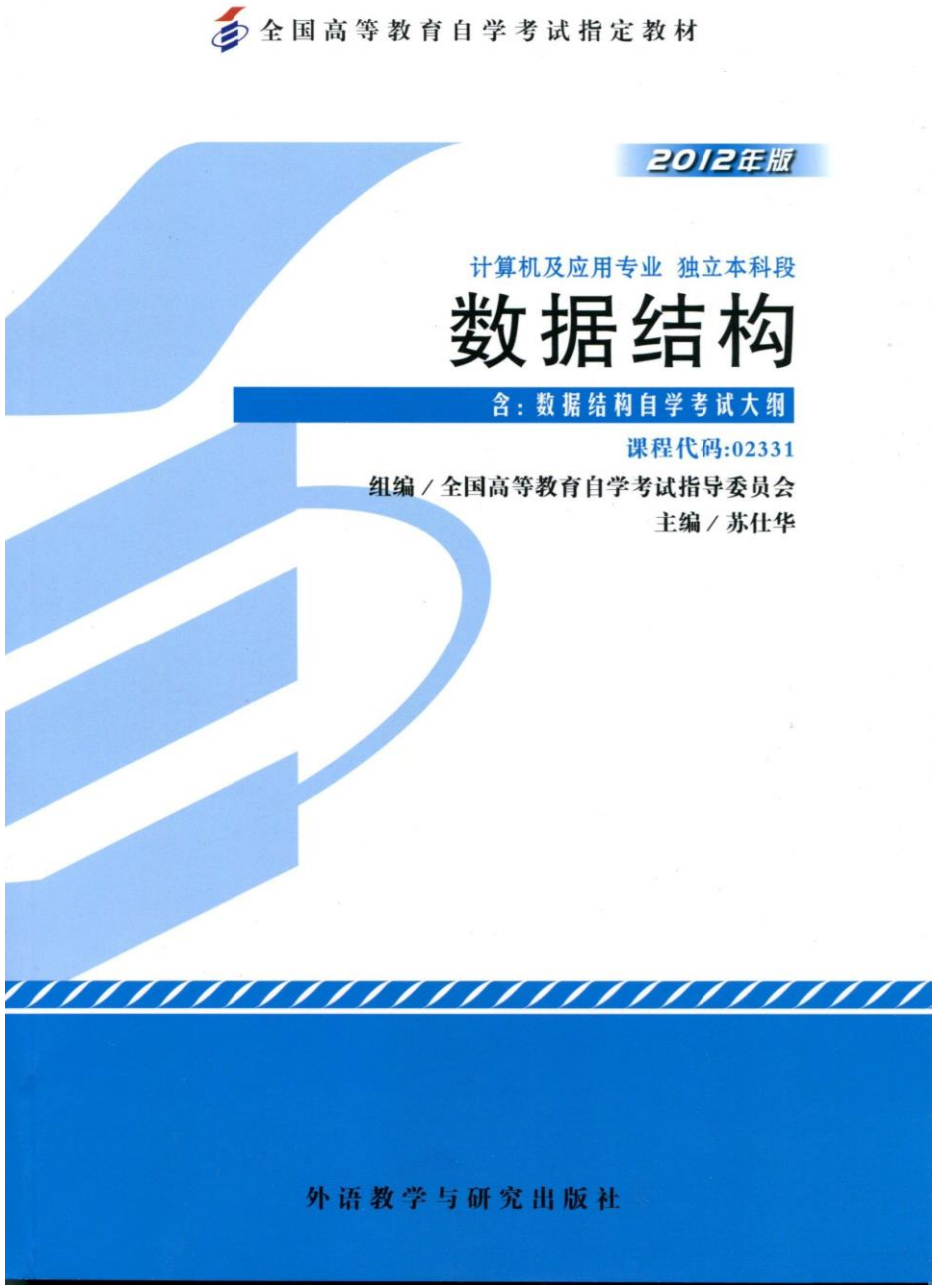
尚德机构

数据结构

主讲：王老师

学习是一种信仰！ IN LEARING WE TRUST

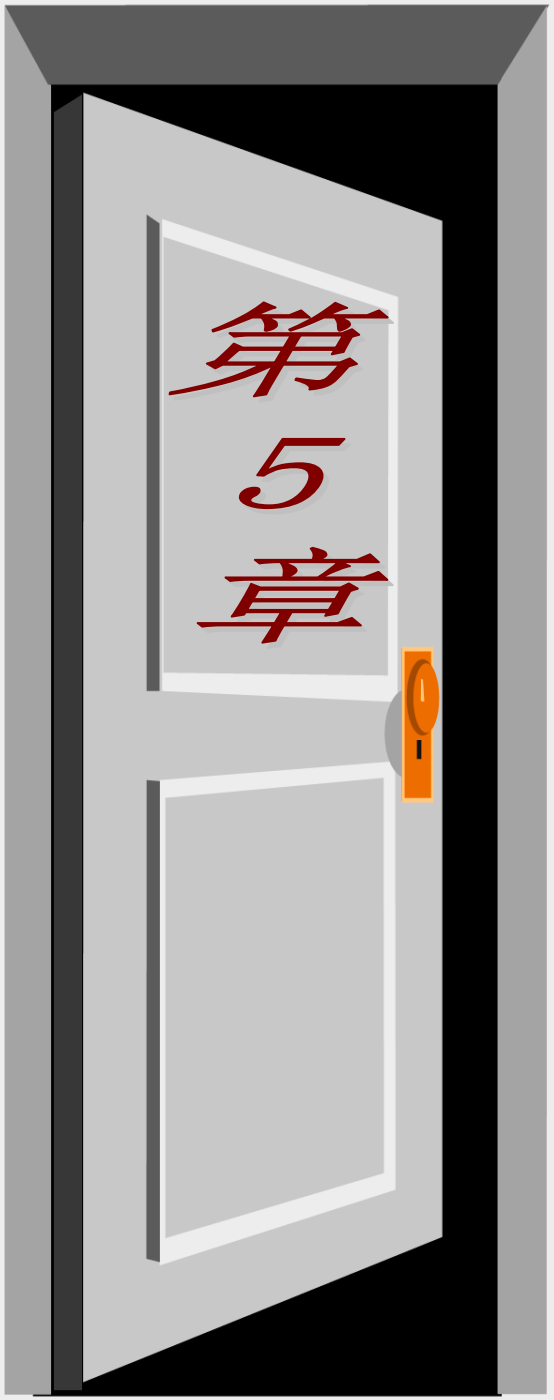


全国高等教育自学考试指定教材 2331数据结构

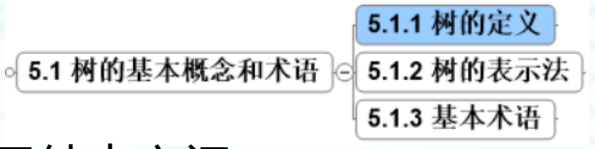
主　编　　苏仕华

出版社　　外语教学与研究出版社

出版时间　2012年3月



树和二叉树



树形结构是一类重要的非线性数据结构，树中结点之间具有明确的层次关系，并且结点之间有分支，非常类似于真正的树。

图 5.1 树形结构示意图

1. 树的定义

树（Tree）是n(n≥0)个结点的有限集T。它或是空集（空树即n=0），或者是非空集。对于任意一棵非空树：

（1）有且保有一个特定的称为根（Root）的结点；

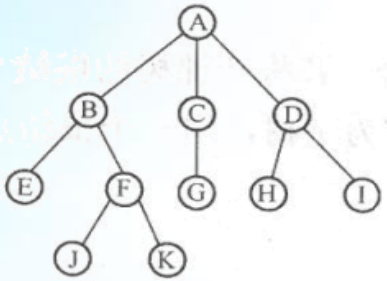
（2）当n>1时，其余的结点可分为m(m>0)个互不相交的有限集T_1, T_2, …, T_m，其中每个集合本身又是一棵树，并称为根的子树。例如，图5.2(a)表示的是一个有11个结点的树，其中A是根结点，其余的结点分成三棵互不相交的子集：T_1={ B, E，F，J，K }，T_2={ C, G }，T_3={ D，H，I }；T_1、T_2和T_3都是根A的子树，且本身也是一棵子树。

5.1.2二、树的表示法

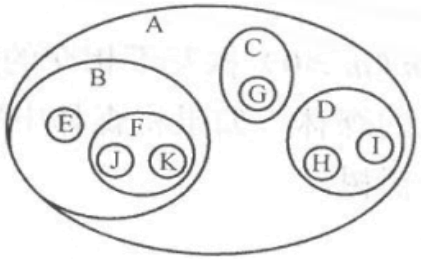
2. 树的表示法

树的表示法有多种，但最常用的是树形图表示法。在树形图表示中，结点通常用圆圈表示，结点名一般写在圆圈内或写在圆圈旁，如图5.2(a)所示。这种表示法非常直观，因此在书中多数情况下，都是用它来表示树结构的。

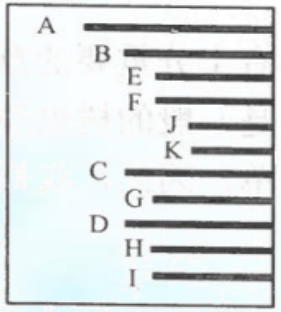
在不同的应用场合，树的表示方法也不尽相同。除了树形表示法外，通常还有三种表示方法，如图5.2(a)中所示的树可以用图（b）、（c）和（d）来表示，其中图5.2(b)是以嵌套集合的形式表示的；图5.2(c)用的是凹形表示法；图5.2(d)是以广义表的形式表示的，树根作为由子树森林组成的表的名字写在表的左边。



（a）树形表示



（b）嵌套集合表示



（c）凹形表表示

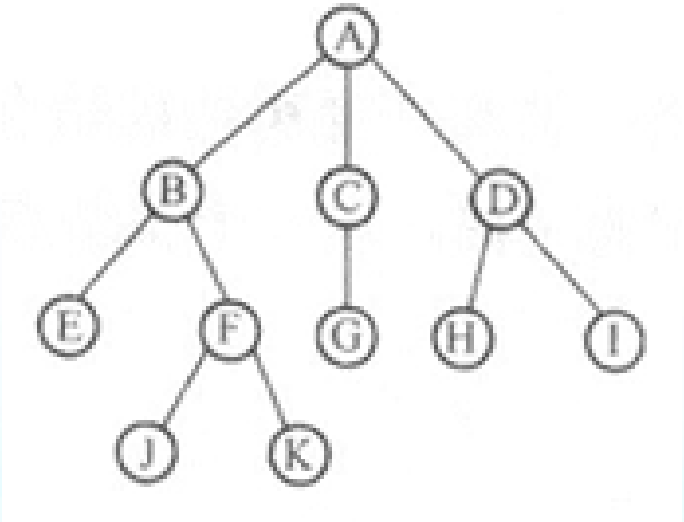
（A（B（E，（F（J，K））），C（G），D（H，I））

（d）广义表表示

图 5.2　图的表示法示意图

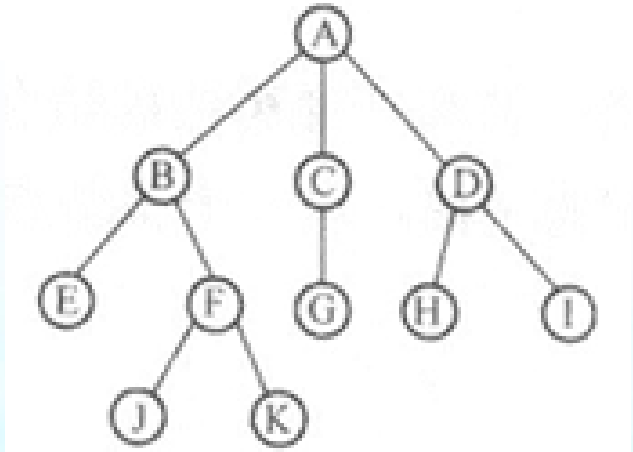
5.1.3三、基本术语

树的结点包含一个数据元素及若干个指向其子树的分支。一个结点拥有的子树数称为该结点的度（Degree）。例如，图5.2(a)中根结点A有三棵子树，因此它的度为3，C的度为1，E的度为0。一棵树中结点的最大度数称为该树的度，如图5.2(a)中树的度为3。度数为零的结点称为叶子结点或终端结点，图5.2(a)中的结点E、J、K、G、H、I都是树的叶子结点。度数不为零的结点称为非终端结点或分支结点。除根结点之外，分支结点也称为内部结点，而根结点又称为开始结点。

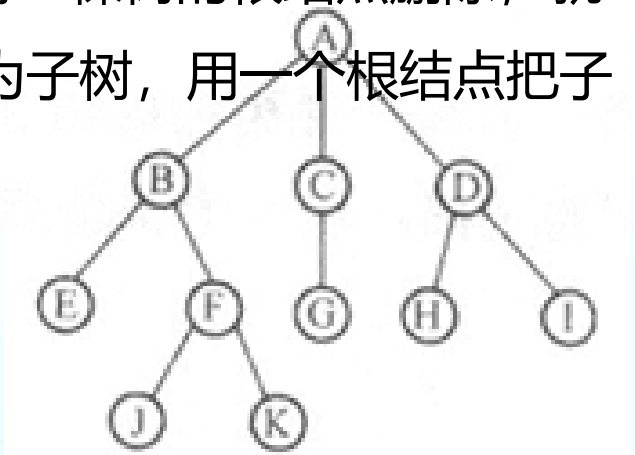


树中某个结点子树的根称为该结点的孩子（Child），相应地，该结点称为孩子结点的双亲（Parent）或父结点。例如，在图5.2(a)所示的树中，B是子树T_1的根，则B是A的孩子，而A则是B的双亲，同一个双亲的孩子之间互为兄弟。又如，结点B、C、D之间互为兄弟。若将上述这种双亲关系进一步推广，可以认为结点B是结点K的祖父。

若在一棵树中存在着一个结点序列$k_1, k_2, \ldots, k_j$使得k_i是k_{i+1}的父结点（$1 \le i \le j$），则称该结点序列是从k_1到k_j的一条路径。例如，在图5.2(a)中，结点A到K有一条路径ABFK，它的路径长度是3。显然，从树根到树中其余结点均存在唯一的一条路径。



- 若树中结点k_i到k_j存在一条路径，则称结点k_i是k_j的祖先，结点k_j是k_i的子孙。例如，在图5.2(a)所示的树中，结点J的祖先是A、B和F，结点B的子孙有E、F、J和K。
- 树中结点的层次（Level）是从根开始算起，根为第一层，其余结点的层次等于其双亲结点的层数加1。树中结点的最大层次称为树的深度（Depth）或高度。如图5.2(a)表示的树，其结点F的层次为3，而该树的高度是4。
- 如果将树中结点的各子树看成是从左至右依次有序且不能交换，则称该树为有序树，否则称为无序树。
- 森林（Forest）是m(m>0)棵互不相交的树的集合。若将一棵树的根结点删除，就得到该树的子树所构成的森林；如果将森林中所有树作为子树，用一个根结点把子树都连起来，森林就变成一棵树。

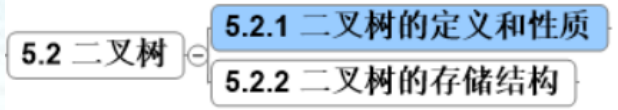


1. 二叉树定义

二叉树（Binary Tree）是n(n≥0)个元素的有限集合，它的每个结点至多只有两棵子树。它或者是空集，或者是由一个根结点及两棵互不相交的分别称作这个根的左子树和右子树的二叉树组成。

从上面的二叉树定义可以看出它是递归的，根据这个定义可以导出二叉树如图5.3所示的五种基本形态。其中，图5.3(a)为空二叉树，图5.3(b)为只有单个根结点的二叉树。

图 5.3 二叉树的五种基本形态



2. 二叉树的性质

二叉树具有几个非常重要的性质，如下所述。

性质1 在二叉树的第i层上至多有2^{i-1}个结点（i≥1）。

利用数学归纳法证明如下：

（1）当i=1时，只有一个根结点，即$2^{i-1}=2^0=1$，命题成立。

（2）假设对所有的j(1≤j<n-i)，命题成立，即第j层上最多有2^{j-1}个结点。

（3）由归纳假设，第i-1层上至多有2^{i-2}个结点。由于二叉树的每个结点的度至多为2，因此，在第i层上的结点数至多是第i-1层上最大结点数的2倍，即$2\times2^{i-2}=2^{i-1}$个结点，所以命题成立。

性质2 深度为k（k≥1）的二叉树至多有2^k-1个结点。

性质3 对任何一棵二叉树T，若其终端结点数为n_0，度数为2的结点数为n_2，则$n_0=n_2+1$。

满二叉树和完全二叉树是两种特殊情形的二叉树。

（1）满二叉树：一棵深度为k且有2^k-1个结点的二叉树称为满二叉树。如图5.4(a)所示的二叉树是一棵深度为4的满二叉树，这种树的特点是每一层上的结点数都达到最大值，因此不存在度数为1的结点，且所有叶子结点都在第k层上。

（2）完全二叉树：若一棵深度为k的二叉树，其前k-1层是一棵满二叉树，而最下面一层（即第k层）上的结点都集中在该层最左边的若干位置上，则称此二叉树为完全二叉树。如图5.4(b)所示的二叉树就是一棵完全二叉树，而图5.4(c)所示就不是一棵完全二叉树，因为最下面一层上的结点L不是在最左边的若干位置上。显然，满二叉树一定是完全二叉树，但完全二叉树则不一定是满二叉树。

性质4 具有n个结点的完全二叉树的深度为$\lfloor \log n \rfloor+1$或($\lceil \log (n+1) \rceil$)。

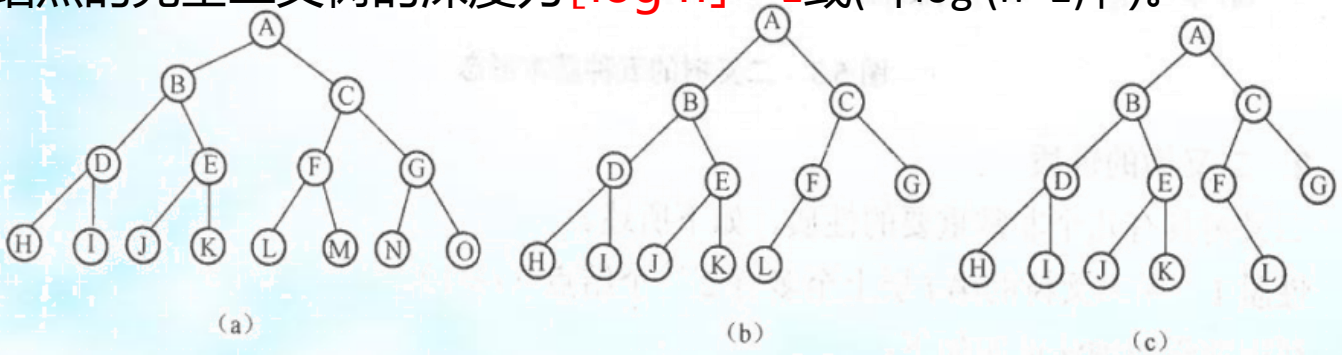
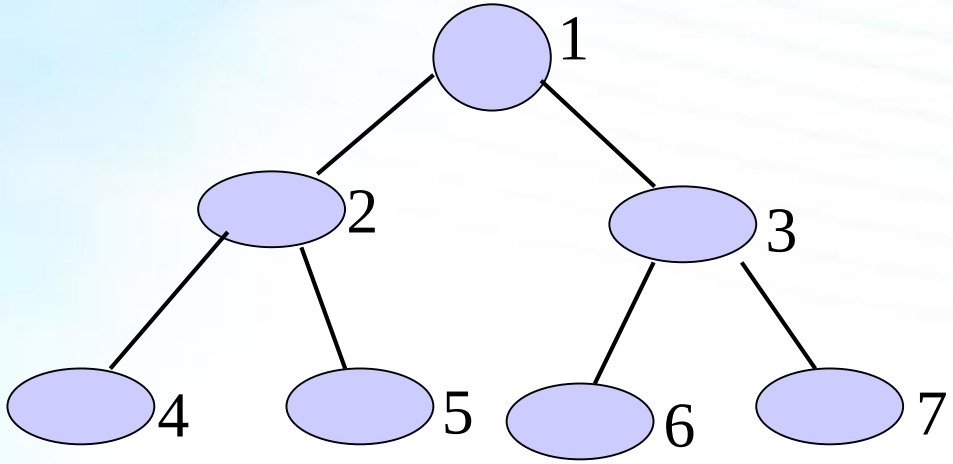


图 5.4 满二叉树和完全二叉树示意图

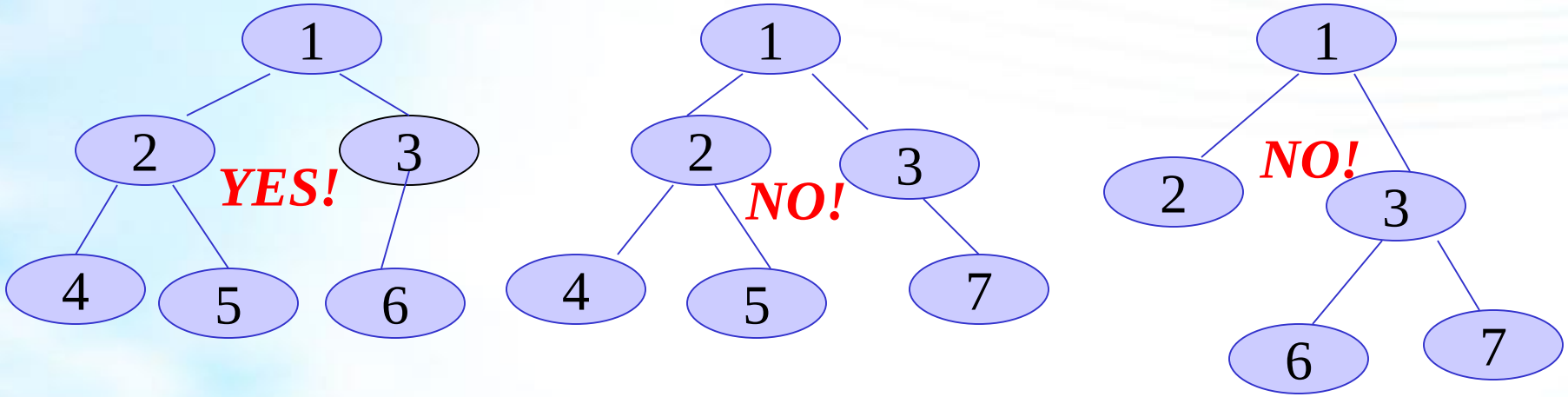
▲满二叉树——深度为k(k>=1)且有2^k-1个结点的二叉树。



满二叉树（结点数=$2^3-1=7$）

▲满二叉树中结点顺序编号：即从第一层结点开始自上而下，从左到右进行连续编号。

▲完全二叉树——深度为K的二叉树中，K-1层结点数是满的(2^{k-2})，K层结点是左连续的(即结点编号是连续的)。如下图（a）



(a)完全二叉树　　(b)非完全二叉树　　(c)非完全二叉树

注：满二叉树是完全二叉树的特例。

真题演练

已知完全二叉树T的第4层有5个叶结点，则T的结点个数最多是（　）。

A: 12

B: 20

C: 21

D: 36

真题演练

已知完全二叉树T的第4层有5个叶结点，则T的结点个数最多是（　）。

A: 12

B: 20

C: 21

D: 36

答案：C

真题演练

已知在一棵度为3的树中,度为2的结点数为4,度为3的结点数为3,则该树中的叶子结点数为(　)。

A: 5

B: 8

C: 11

D: 18

真题演练

已知在一棵度为3的树中,度为2的结点数为4,度为3的结点数为3,则该树中的叶子结点数为(　)。

A: 5

B: 8

C: 11

D: 18

答案：C

真题演练

已知一棵完全二叉树T的第5层上共有5个叶结点，则T中叶结点个数最少是(　)。

A:5

B: 8

C: 10

D: 27

真题演练

已知一棵完全二叉树T的第5层上共有5个叶结点，则T中叶结点个数最少是(　)。

A:5

B: 8

C: 10

D: 27

答案：C

真题演练

深度为4的完全二叉树的结点数至少为(　)。

A: 4

B: 8

C: 13

D: 15

真题演练

深度为4的完全二叉树的结点数至少为(　)。

A: 4

B: 8

C: 13

D: 15

答案：B

真题演练

若一棵二叉树有10个度为2的结点，5个度为1的结点，则度为0的结点数是(　)。

A: 9

B: 11

C: 15

D: 不确定

真题演练

若一棵二叉树有10个度为2的结点，5个度为1的结点，则度为0的结点数是(　)。

A: 9

B: 11

C: 15

D: 不确定

答案：B

真题演练

在一棵二叉树中，度为2的结点数为15，度为1的结点数为3，则叶子结点数为(　)。

A: 12

B: 16

C: 18

D: 20

真题演练

在一棵二叉树中，度为2的结点数为15，度为1的结点数为3，则叶子结点数为(　)。

A: 12

B: 16

C: 18

D: 20

答案：B

真题演练

若一棵二叉树中度为1的结点个数是3，度为2的结点个数是4，则该二叉树叶子结点的个数是(　)。

A: 4

B: 5

C: 7

D: 8

真题演练

若一棵二叉树中度为1的结点个数是3，度为2的结点个数是4，则该二叉树叶子结点的个数是(　)。

A: 4

B: 5

C: 7

D: 8

答案：B

真题演练

若根结点的层数为1，则具有n个结点的二叉树的最大高度是(　)。

A: n

B: $\lfloor \log_2 n \rfloor$

C: $\lfloor \log_2 n \rfloor + 1$

D: n/2

真题演练

若根结点的层数为1，则具有n个结点的二叉树的最大高度是(　)。

A: n

B: $\lfloor \log_2 n \rfloor$

C: $\lfloor \log_2 n \rfloor + 1$

D: n/2

答案：A

真题演练

一棵二叉树的第7层上最多含有的结点数为(　)。

A: 14

B: 64

C: 127

D: 128

真题演练

一棵二叉树的第7层上最多含有的结点数为(　)。

A: 14

B: 64

C: 127

D: 128

答案：B

真题演练

设深度为k(k≥1)的二叉树中只有度为0和度为2的结点，则该二叉树中所包含的结点数至少是(　)。

A: k+1

B: 2k+1

C: 2k-1

D: 2k

真题演练

设深度为k(k≥1)的二叉树中只有度为0和度为2的结点，则该二叉树中所包含的结点数至少是(　)。

A: k+1

B: 2k+1

C: 2k-1

D: 2k

答案：C

真题演练

设高度为h的二叉树中，只有度为0和2的结点，则此类二叉树包含的结点数至少是（　）。

A: 2h

B: 2h-1

C: 2h+1

D: h+1

真题演练

设高度为h的二叉树中，只有度为0和2的结点，则此类二叉树包含的结点数至少是（ ）。

A: 2h

B: 2h-1

C: 2h+1

D: h+1

答案：B

真题演练

一棵完全二叉树T的全部k个叶结点都在同一层中且每个分支结点都有两个孩子结点。

T中包含的结点数是(　)。

A: k

B:2k-1

C: k^2

D: 2^k-1

真题演练

一棵完全二叉树T的全部k个叶结点都在同一层中且每个分支结点都有两个孩子结点。

T中包含的结点数是(　)。

A: k

B:2k-1

C: k^2

D: 2^k-1

答案：B

真题演练

若完全二叉树T包含20个终端结点，则T的结点数最多是(　)。

A: 38

B: 39

C: 40

D: 41

真题演练

若完全二叉树T包含20个终端结点，则T的结点数最多是(　)。

A: 38

B: 39

C: 40

D: 41

答案：C

真题演练

已知一棵高度为4的完全二叉树T共有5个叶结点，则T中结点个数最少是(　)。

A: 9

B: 10

C: 11

D: 12

真题演练

已知一棵高度为4的完全二叉树T共有5个叶结点，则T中结点个数最少是(　)。

A: 9

B: 10

C: 11

D: 12

答案：A

1. 顺序存储结构

在顺序存储一棵具有k结点的完全二叉树时，只要从树根开始自上到下，每层从左至右地给该树中每个结点进行编号（假定编号从0开始），就能够得到一个反映整个二叉树结构的线性序列，如图5.5(a)所示。然后以各结点的编号为下标，把每个结点的值对应存储到一个一维数组bt中，如图5.5(b)所示。

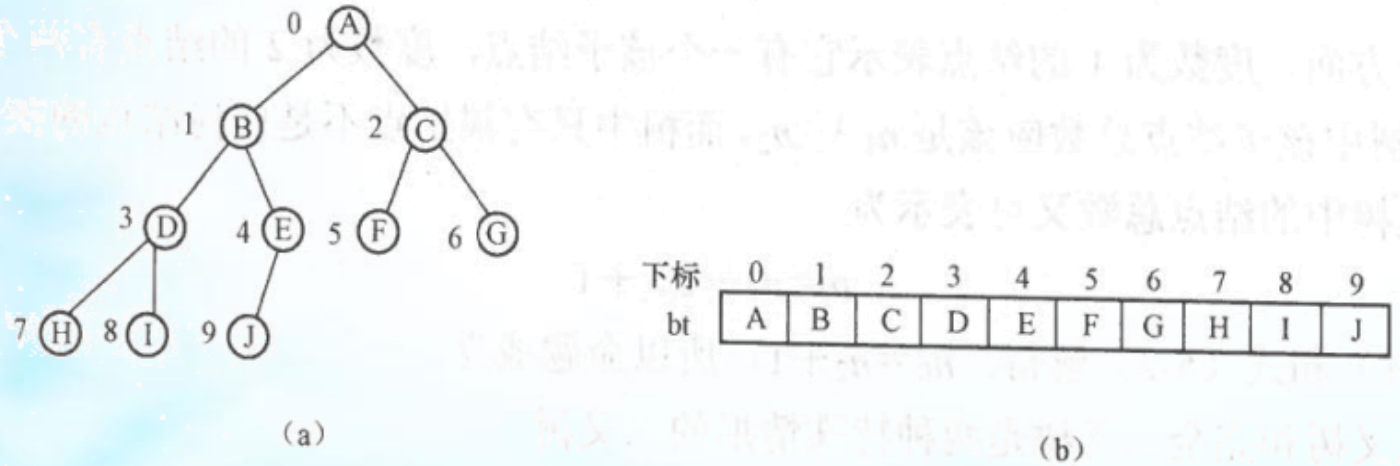
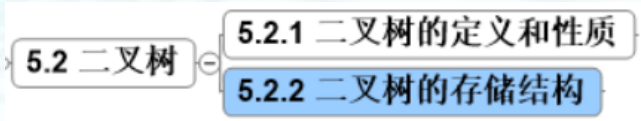


图 5.5　完全二叉树顺序存储结构示意图



由完全二叉树定义可知，在完全二叉树中除最下面一层外，各层结点都达最大值，每一层上结点个数恰好是上一层结点个数的2倍。因此，从一个结点的编号就可以推得其双亲及左、右孩子等结点的编号。例如，假设编号为i的结点q_i（$0 \le i < n$），那么，

①若i=0，则q_i为根结点，无双亲；否则q_i的双亲结点编号为$\lfloor (i-1)/2 \rfloor$。

②若$2i+1<n$，则q_i的左孩子结点编号为$2i+1$；否则q_i无左孩子，即q_i必定是叶子结点。

③若$2i+2<n$，则q_i的右孩子结点编号为$2i+2$；否则，q_i无右孩子。

显然，对于完全二叉树而言，使用顺序存储结构既简单又节省存储空间。但对于一般的二叉树来说，采用顺序存储时，为了使用结点在数组中的相对位置来表示结点之间的逻辑关系，就必须增加一些虚结点使其成为完全二叉树的形式。这样存储二叉树中的结点会造成存储空间上的浪费，在最坏的情况下，一棵深度为k且只有k个结点的单支二叉树却需要2^k-1个结点的存储空间。例如，只有4个结点的二叉树，将其添加一些实际上不存在的虚结点“@”，使之成为如图5.6(a)所示的完全二叉树，其相应的顺序存储结构如图5.6(b)所示。

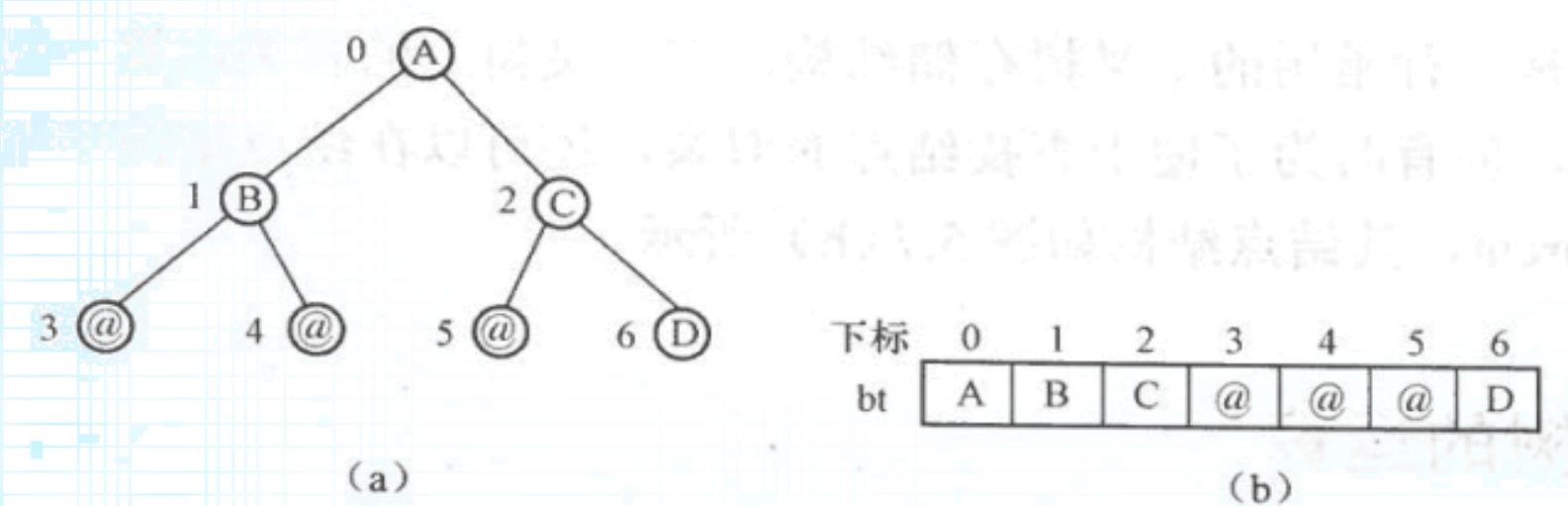


图 5.6　非完全二叉树的顺序存储结构示意图

2. 链式存储结构

从上面介绍的二叉树顺序存储结构来看，它仅适用于完全二叉树，但对于一般的二叉树来说，不但会浪费存储空间，而且当经常在二叉树中进行插入或删除结点操作时，需要移动大量的结点。因此，在一般情况下，多采用链式存储方式来存储二叉树。设计不同的（结点）结构可构成不同形式的链式存储结构。在二叉树的链式存储表示中，通常采用的方法是：每个结点设置三个域，即值域、左指针域和右指针域，用data表示值域，lchild和rchild分别表示指向左右子树（孩子）的指针域，如图5.7(3)所示。

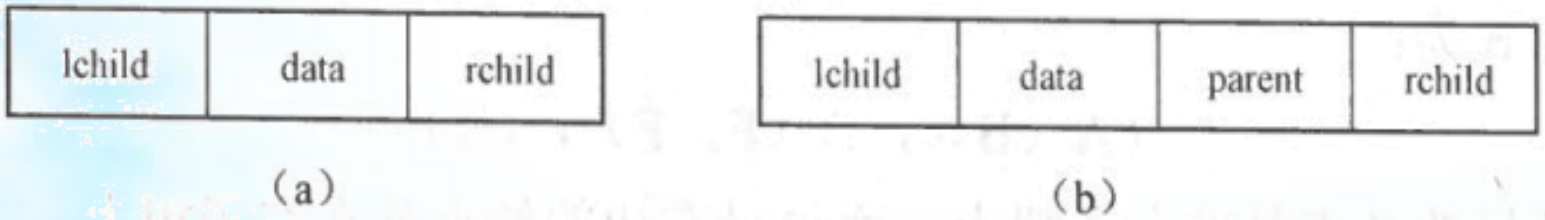


图 5.7　二叉树的链式存储结点结构

相应的类型说明为如下：

```
typedef struct node {
    DataType data;
    struct node * lchild, * rchild;
} BinTNode;
typedef BinTNode * BinTree;
```

在一棵二叉树中，设有一个指向其根结点（即开始结点）的BinTree型头指针bt及所有类型为BinTNode的结点，就构成了二叉树的链式存储结构，并称其为二叉链表。例如，图5.8(a)所示二叉树的二叉链表如图5.8(b)所示。

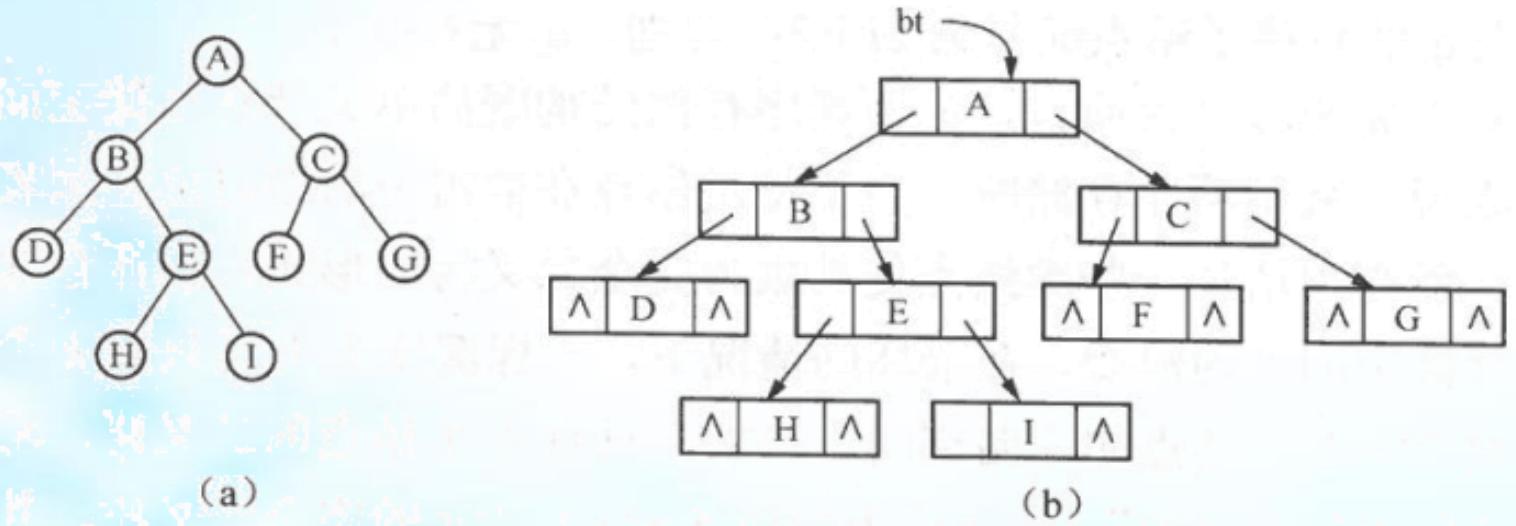


图 5.8　二叉树的链式存储结构示意图

5.2.2 二叉树的存储结构

二叉链表是一种常用的二叉树存储结构，在二叉树上的有关运算一般都是采用这种链式存储结构，但有时为了便于查找结点的双亲，还可以在结点结构中增加一个指向其双亲的指针parent，其结点结构如图5.7(b)所示。

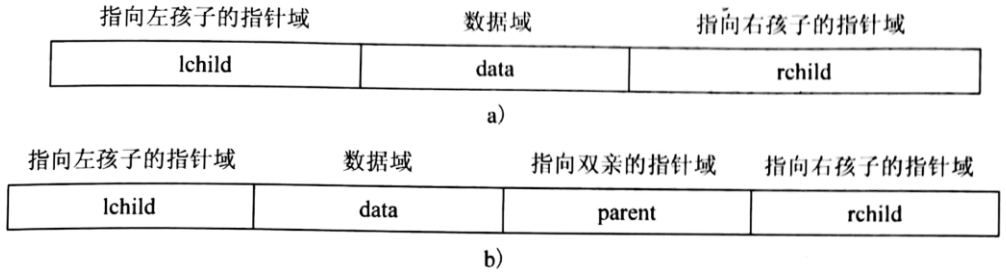
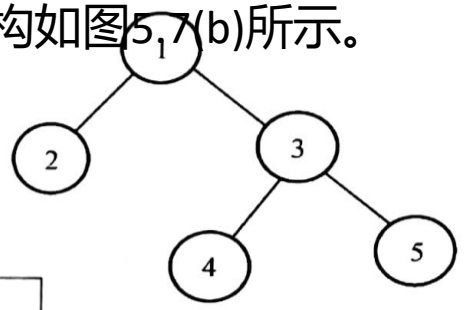
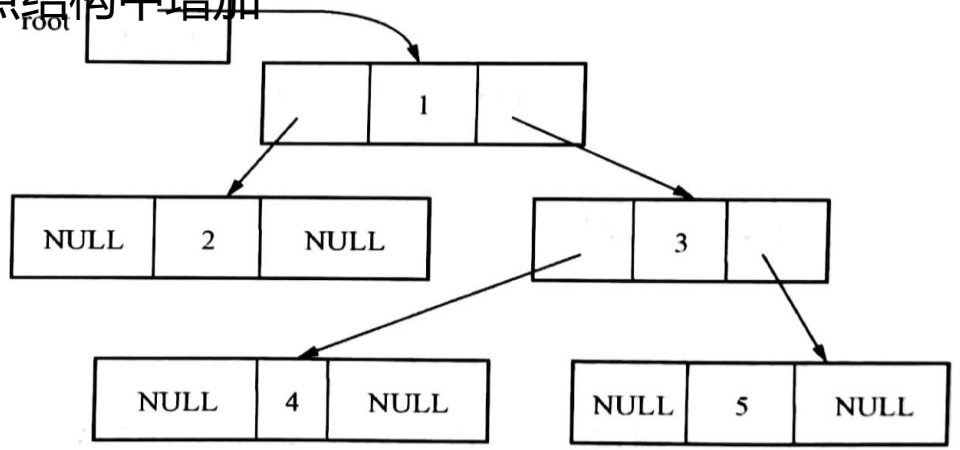


图 4-9　二叉链表和三叉链表结点结构

a) 二叉链表的结点　b) 三叉链表的结点





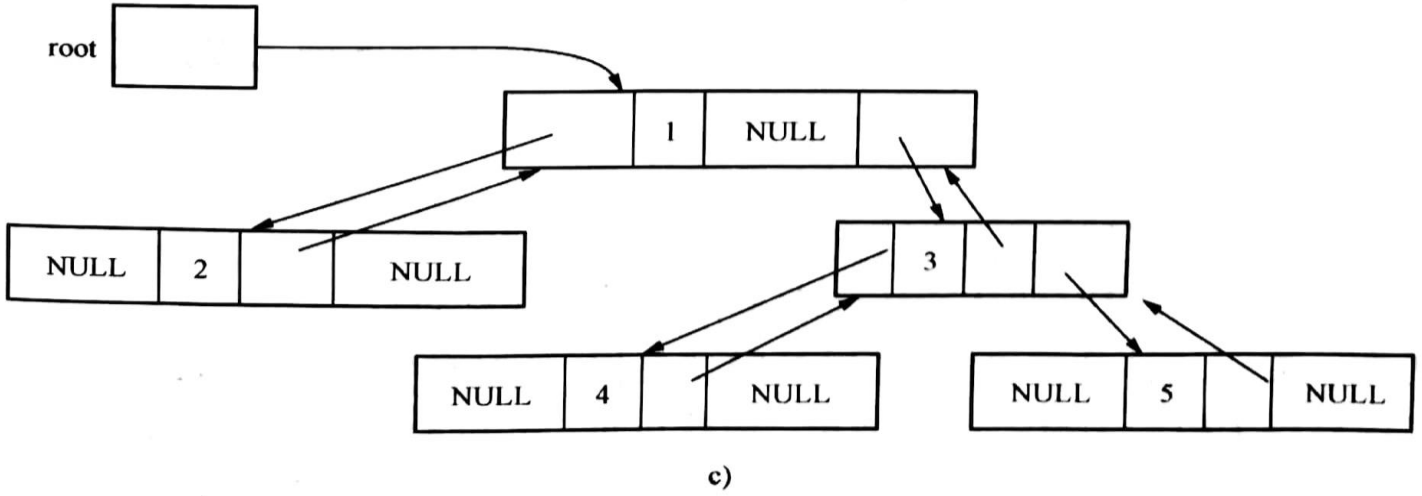
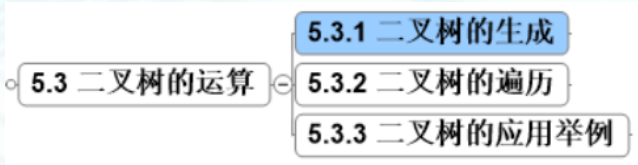


图 4-10　二叉树的链式存储结构



1. 按广义表表示二叉树结构生成二叉链表的算法

该算法相对简单一些，但要求对用广义表表示二叉树的理解要深刻。它与普通的表示树的广义表形式有所不同，因为它有左右子树之分。例如，如图5.9所示的二叉树的广义表表示形式为：

(A (B (D (E, F)), C))

靠近左括号的结点是在左子树上，而逗号右边的结点是在右子树上。

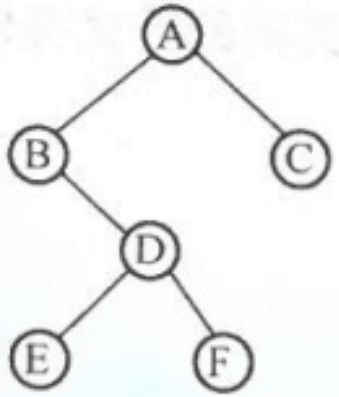
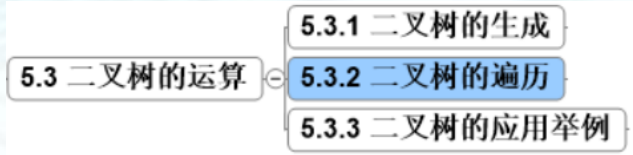


图 5.9　一棵二叉树

2. 按完全二叉树的层次顺序依次输入结点信息建立二叉链表的算法

该算法的基本思想是：首先对一般的二叉树添加若干个虚结点，使其成为完全二叉树，然后依次输入结点信息，若输入的结点不是虚结点则建立一个新结点，若是第一个结点，则令其为根结点，否则将新结点作为左孩子或右孩子链接到它的双亲结点上。如此重复下去，直到输入结束符号“#”时为止（假设结点数据域为字符型）。

5.3.2二、二叉树的遍历



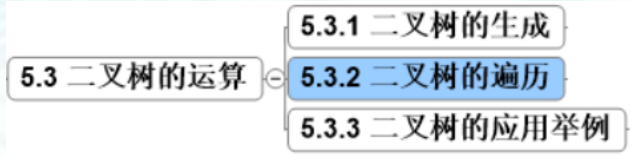
在二叉树的应用中，遍历二叉树是一种最重要的运算，是二叉树中所有其他运算的基础。所谓遍历，是指沿着某条搜索路径（线）周游二叉树，依次对树中每个结点访问且仅访问一次。遍历对于一般的线性结构来说是一个很容易解决的问题，只需要从开始结点出发，依次访问当前结点的直接后继，直到终端结点为止。

令：L —— 遍历左子树
D —— 访问根结点
R —— 遍历右子树

组合 → LDR、LRD、DLR、RDL、RLD、DRL

先左后右

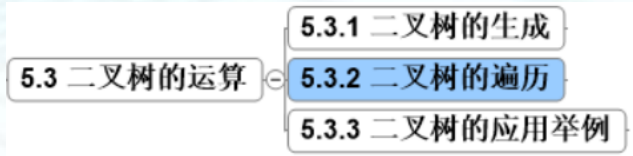
DLR——先（根）序遍历，
LDR——中（根）序遍历，
LRD——后（根）序遍历。



1、先序遍历DLR ——首先访问根结点，其次遍历根的左子树，最后遍历根右子树，对每棵子树同样按这三步（先根、后左、再右）进行。

2、中序遍历LDR ——首先遍历根的左子树，其次访问根结点，最后遍历根右子树，对每棵子树同样按这三步（先左、后根、再右）进行。

3、后序遍历LRD ——首先遍历根的左子树，其次遍历根的右子树，最后访问根结点，对每棵子树同样按这三步（先左、后右、最后根）进行。



三种遍历的递归算法定义：

(1) 前序遍历二叉树的递归定义

若二叉树非空，则依次进行操作：

①访问根结点；②前序遍历左子树；③前序遍历右子树。

(2) 中序遍历二叉树的递归定义

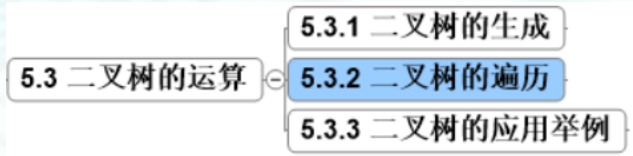
若二叉树非空，则依次进行操作：

①中序遍历左子树；②访问根结点；③中序遍历右子树。

(3) 后序遍历二叉树的递归定义

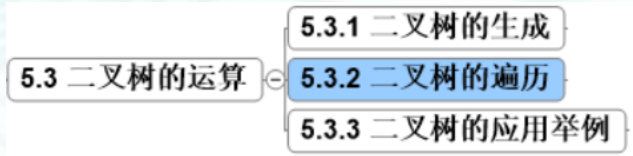
若二叉树非空，则依次进行操作：

①后序遍历左子树；②后序遍历右子树；③访问根结点。



用C语言描述的前序遍历的递归算法如下：

```
void Preorder ( BinTree bt )
{   //采用二叉链表存储结构，并设结点值为字符型
    if(bt!=NULL) {
        printf("%c", bt->data);      //访问根结点
        Preorder ( bt->lchild );     //前序遍历左子树
        Preorder ( bt->rchild );     //前序遍历右子树，r表示返回点
    }
}
```



类似地，中序遍历和后序遍历二叉链表的递归算法如下:

```
void Inorder (BinTreebt)
{  //中序遍历二叉链表算法
   if(bt!=NULL) {
      Inorder(bt->lchild);              //中序遍历左子树
      printf("%c", bt->data)            //访问根结点
      Inorder(bt->rchild);              //中序遍历右子树
   }
}

void Postorder(BinTree bt)
{  //后序遍历二叉链表算法
   if(bt!=NULL) {
      Postorder(bt->lchild);            //后序遍历左子树
      Postorder(bt->rchild);            //后序遍历右子树
      printf("%c", bt->data);           //访问根结点
   }
}
```

5.3.2 二叉树的遍历

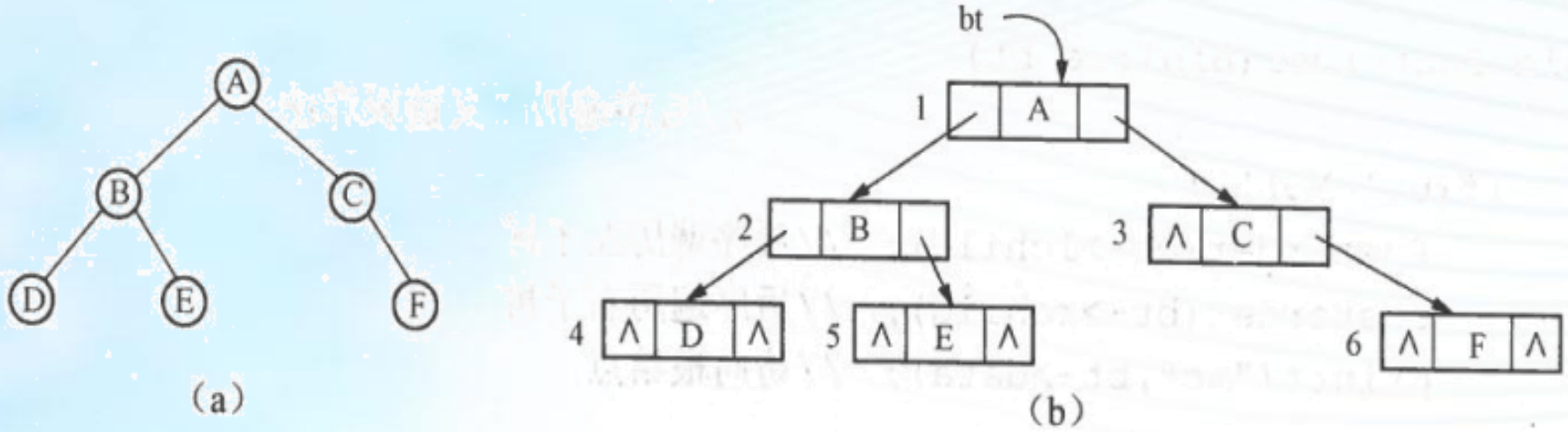


图 5.10　二叉树和二叉链表示意图

前序遍历图5.10所示的二叉树，访问结点次序为ABDECF。

中序序列和后序序列分别为DBEACF和DEBFCA。

【例5.1】分别写出如图5.12所示的二叉树的前、中、后序遍历序列。

解：按照前面介绍的三种递归或非递归的遍历二叉树算法，很容易给出遍历序列。其中，前序序列为ABDHEICFG，中序序列为DHBEIAFCG，后序序列为HDIEBFGCA。

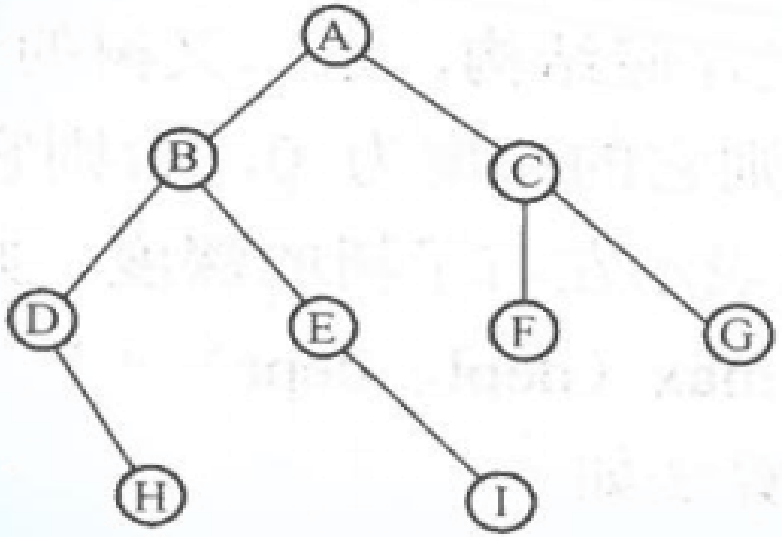
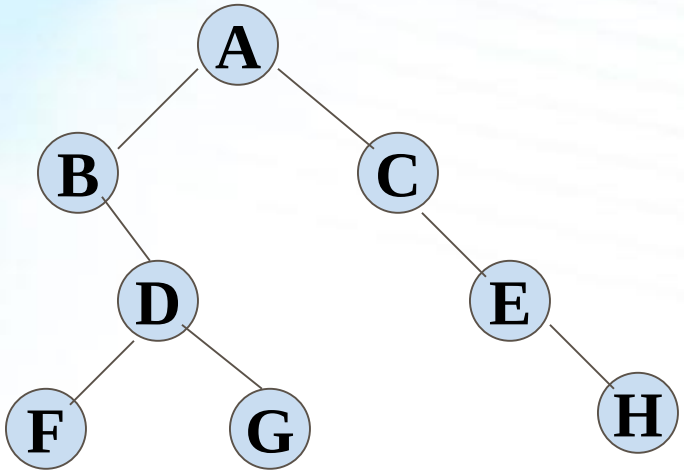


图 5.12　一棵二叉树

对于如下图的二叉树，其先序、中序、后序遍历的序列为：



先序遍历： A、B、D、F、G、C、E、H。
中序遍历： B、F、D、G、A、C、E、H。
后序遍历： F、G、D、B、H、E、C、A。

5.3.3三、二叉树的应用举例

【例5.2】已知二叉树的**前序和中序**遍历序列或**中序和后序**遍历序列，求其二叉树。分析：根据二叉树的三种遍历算法可以得出这样一个结论：已知一棵二叉树的前序和中序遍历序列或中序和后序遍历序列，可唯一地确定一棵二叉树。具体方法如下：

（1）根据前序或后序遍历序列确定二叉树的各子树的根；

（2）根据中序遍历序列确定各子树根的左、右子树。

例如，一棵二叉树的前序和中序遍历序列分别为ABDEGHCFI和DBGEHACIF，要求出其后序遍历序列，就必须求出其二叉树。

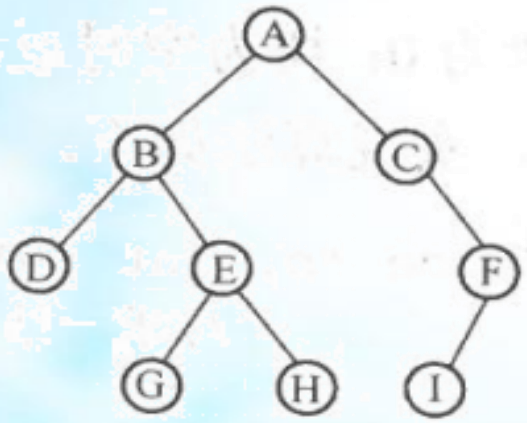
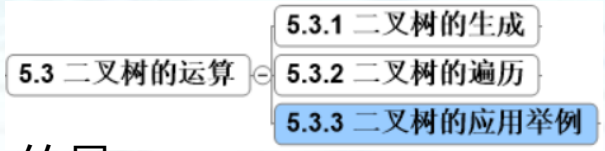


图 5.13　二叉树

二叉树的后序遍历序列为：DGHEBIFCA。

5.3.3 二叉树的应用举例



【例5.3】已知二叉树的链式存储结构，求二叉树的深度。

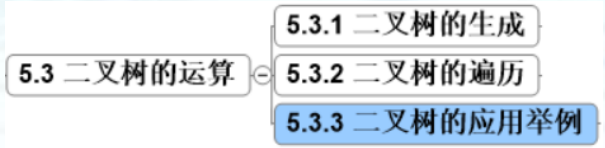
分析：若一棵二叉树为空，则它的深度为0，否则它的深度等于其左右子树中的最大深度加1。设depl和depr分别表示左右子树的深度，则二叉树的深度为：

max(depl, depr)+1

因此，求二叉树深度的递归算法如下：

```
int BinTreeDepth (BinTree bt)
{   //求由bt指向的二叉树的深度
    int depl, depr;
    if(bt==NULL)
        return 0;                          //对于空树，返回0值，结束递归
    else {
        depl=BinTreeDepth(bt->lchild);     //计算左子树的深度
        depr=BinTreeDepth(bt->rchild);     //计算右子树的深度
        if(depl>depr)
            return depl+1;
        else
            return depr+1;
    }
}
```

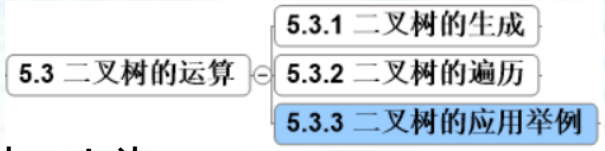
5.3.3 二叉树的应用举例



【例5.4】以二叉链表为存储结构，试编写在二叉树中查找值为x的结点及求x所在结点在树中层数的算法。

分析：（1）按值查找。该算法是比较简单的，无论是利用三种遍历算法的哪一种，都很容易实现，不妨用前序遍历算法。

```
int found=0;                          //用found来作为是否查找到的标志
BinTNode *p;
void FindBT(BinTree bt, DataType x)
{   if((bt!=NULL) && (!found))
      if(bt->data==x) {
         p=bt; found=1;
      }
      else {
         FindBT(bt->lchild, x);       //遍历查找左子树
         FindBT(bt->rchild, x);       //遍历查找右子树
      }
}
```



（2）求结点的层次。依照题意，仍然采用递归算法。设p为指向待查找的结点，h为返回p所指结点的所在层数，初值为0；树为空时返回0。lh指示二叉树bt的层数（即高度），调用时置初值为为1。因此，实现算法如下：

```
int Level(BinTree bt, BinTNode *p, int lh)
{  //求一结点在二叉树中的层次
    static int h=0;
    if(bt==NULL) h=0;
        else if(bt==p)
            h=lh;
        else {
            Level(bt->lchild, p, lh+1);
            if(h==0)                              //表示左子树已查完
                Level(bt->rchild, p, lh+1);
        }
        return h;
}
```

真题演练

若一棵具有n(n>0)个结点的二叉树的先序序列与后序序列正好相反，则该二叉树一定是(　)。

A: 结点均无左孩子的二叉树

B: 结点均无右孩子的二叉树

C: 高度为n的二叉树

D: 存在度为2的结点的二叉树

真题演练

若一棵具有n(n>0)个结点的二叉树的先序序列与后序序列正好相反，则该二叉树一定是(　)。

A: 结点均无左孩子的二叉树

B: 结点均无右孩子的二叉树

C: 高度为n的二叉树

D: 存在度为2的结点的二叉树

答案：C

真题演练

下列数据结构中，不属于二叉树的是(　)。

A:B树

B:AVL树

C: 二叉排序树

D: 哈夫曼树

真题演练

下列数据结构中，不属于二叉树的是(　)。

A:B树

B:AVL树

C: 二叉排序树

D: 哈夫曼树

答案：A

真题演练

在一棵非空二叉树的后序遍历序列中，所有列在根结点前面的是(　)。

A: 左子树中的部分结点

B: 右子树中的全部结点

C: 左右子树中的部分结点

D: 左右子树中的全部结点

真题演练

在一棵非空二叉树的后序遍历序列中，所有列在根结点前面的是(　)。

A: 左子树中的部分结点

B: 右子树中的全部结点

C: 左右子树中的部分结点

D: 左右子树中的全部结点

答案：D

真题演练

已知二叉树T的前序遍历序列为a，b，c，e，d，中序遍历序列为c，e，b，d，a，则T的后序遍历序列为(　)。

A: c，e，d，b，a

B: d，e，c，b，a

C: e，c，d，b，a

D: e，c，b，a，d

真题演练

已知二叉树T的前序遍历序列为a，b，c，e，d，中序遍历序列为c，e，b，d，a，则T的后序遍历序列为(　)。

A: c，e，d，b，a

B: d，e，c，b，a

C: e，c，d，b，a

D: e，c，b，a，d

答案：C

真题演练

若一棵二叉树的前序遍历序列与后序遍历序列相同，则该二叉树可能的形状是(　)。

A:树中没有度为2的结点

B:树中只有一个根结点

C:树中非叶结点均只有左子树

D:树中非叶结点均只有右子树

真题演练

若一棵二叉树的前序遍历序列与后序遍历序列相同，则该二叉树可能的形状是(　)。

A:树中没有度为2的结点

B:树中只有一个根结点

C:树中非叶结点均只有左子树

D:树中非叶结点均只有右子树

答案：B

真题演练

一棵非空二叉树T的前序遍历和后序遍历序列正好相反，则T一定满足(　)。

A: 所有结点均无左孩子

B: 所有结点均无右孩子

C: 只有一个叶子结点

D: 是一棵满二叉树

真题演练

一棵非空二叉树T的前序遍历和后序遍历序列正好相反，则T一定满足(　)。

A: 所有结点均无左孩子

B: 所有结点均无右孩子

C: 只有一个叶子结点

D: 是一棵满二叉树

答案：C

真题演练

在一棵非空二叉树的中序遍历序列中，所有列在根结点前面的是(　)。

A: 左子树中的部分结点

B: 左子树中的全部结点

C: 右子树中的部分结点

D: 右子树中的全部结点

真题演练

在一棵非空二叉树的中序遍历序列中，所有列在根结点前面的是(　)。

A: 左子树中的部分结点

B: 左子树中的全部结点

C: 右子树中的部分结点

D: 右子树中的全部结点

答案：B

当用二叉链表作为二叉树的存储结构时，因为每个结点中只有指向其左、右孩子结点的指针域，所以从任一结点出发只能直接找到该结点左、右孩子，而一般情况下无法直接找到该结点在某种遍历序列中的前趋和后继结点。若在每个结点中增加两个指针域来存放遍历时得到的前趋和后继信息，就可以通过该指针直接或间接访问其前趋和后继结点，但是这将大大降低存储空间的利用率。另一方面，在有n个结点的二叉链表中必定存在n+1个空指针域，因此可以利用这些空指针域存放指向结点在某种遍历次序下的前趋和后继结点的指针，这种指向前趋和后继结点的指针称为“线索”，加上线索的二叉链表称为线索链表，相应的二叉树称为线索二叉树。

在一个线索二叉树中，为了区分一个结点的左、右孩子指针域是指向其孩子的指针还是指向其前趋或后继的线索，可在结点结构中增加两个线索标志域，一个是左线索标志域，用ltag表示，另一个是右线索标志域，用rtag表示。ltag和rtag只能取值0和1。增加线索标志域后的结点结构如下：

lchild	ltag	Data	rtag	rchild

$$ltag=\begin{cases}0 & \text{lchild域指向结点的左孩子}\\ 1 & \text{lchild域指向结点的前趋}\end{cases}$$

$$rtag=\begin{cases}0 & \text{rchild域指向结点的右孩子}\\ 1 & \text{rchild域指向结点的后继}\end{cases}$$

5.4第四节 线索二叉树 5.4.1一、二叉树的线索化

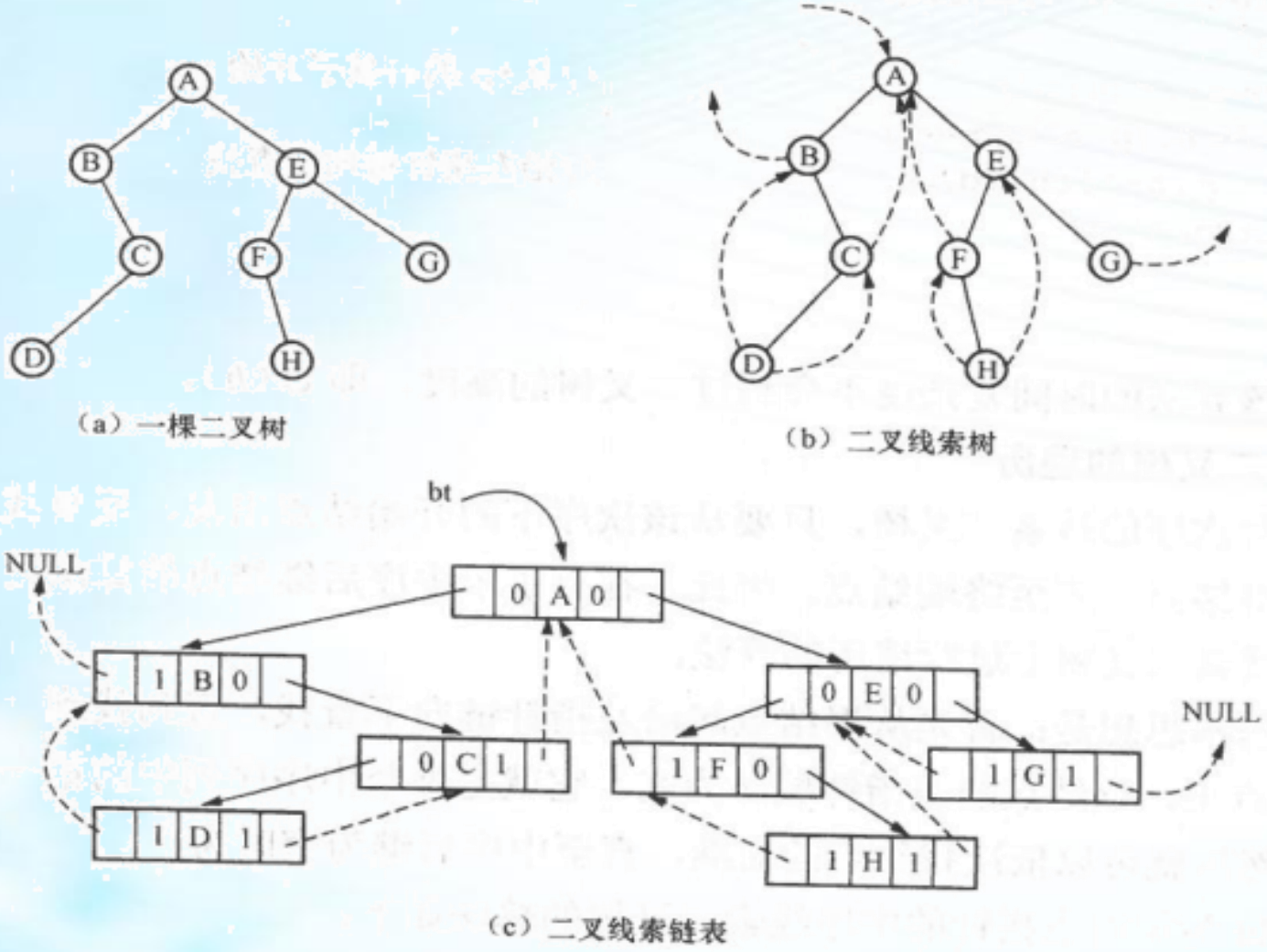
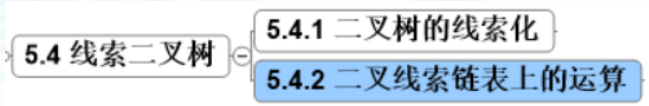


图 5.16 线索二叉树及二叉线索链表示意图

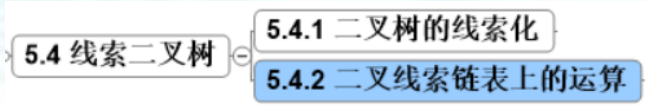


1. 查找某结点*p的后继结点

在中序线索二叉树上求结点*p的中序后继结点的算法：

```
BinThrNode * InorderNext(BinThrNode * p)
{       //在中序线索二叉树上求结点*p的中序后继结点
    if(p->rtag==1)                                  //rchild域为右线索
        return p->rchild;                           //返回中序后继结点指针
    else {
        p=p->rchild;                                //从*p的右孩子开始
        while(p->ltag==0)
          p=p->lchild;                              //沿左指针链向下查找
        return p;
    }
}
```

显然，该算法的时间复杂度不会超过二叉树的高度，即O(h)。



2. 线索二叉树的遍历

遍历以bt为根结点指针的中序线索二叉树的算法如下：

```
void TinorderThrTree(BinThrTree bt)
{   BinThrNode *p;
    if(bt!=NULL) {                                  //二叉树不空
        P=bt;                                       //使p指向根结点
        while(p->ltag==0)
            p=p->lchild;                            //查找出中序遍历的第一个结点
        do {
            printf("%c", p->data);          //输出访问结点值
            p=InorderNext(p);                       //查找结点*p的中序后继
        } while(p!=NULL);                           //当p为空时算法结束
    }
}
```

在上述的算法中，while循环是找遍历的第一个结点，次数是一个很小的常数，而do循环是以右线索为空为终结条件，所以该算法的时间复杂度为O(n)。

真题演练

以二叉链表作为二叉树的存储结构，在有n(n>0)个结点的二叉链表中，空指针域的个数是(　)。

A: n-1

B: n+1

C: 2n-1

D: 2n+1

真题演练

以二叉链表作为二叉树的存储结构，在有n(n>0)个结点的二叉链表中，空指针域的个数是(　)。

A: n-1

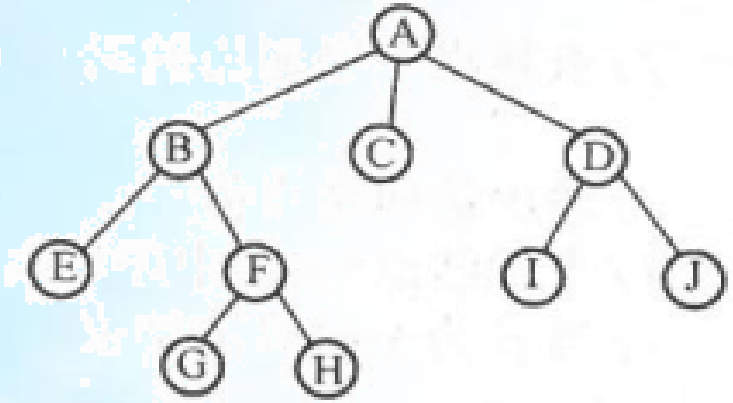
B: n+1

C: 2n-1

D: 2n+1

答案：B

1. 双亲表示法



(a)

下标	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	5
7	H	5
8	I	3
9	J	3

(b)

图 5.17　树的双亲表示法示意图

2. 孩子链表法

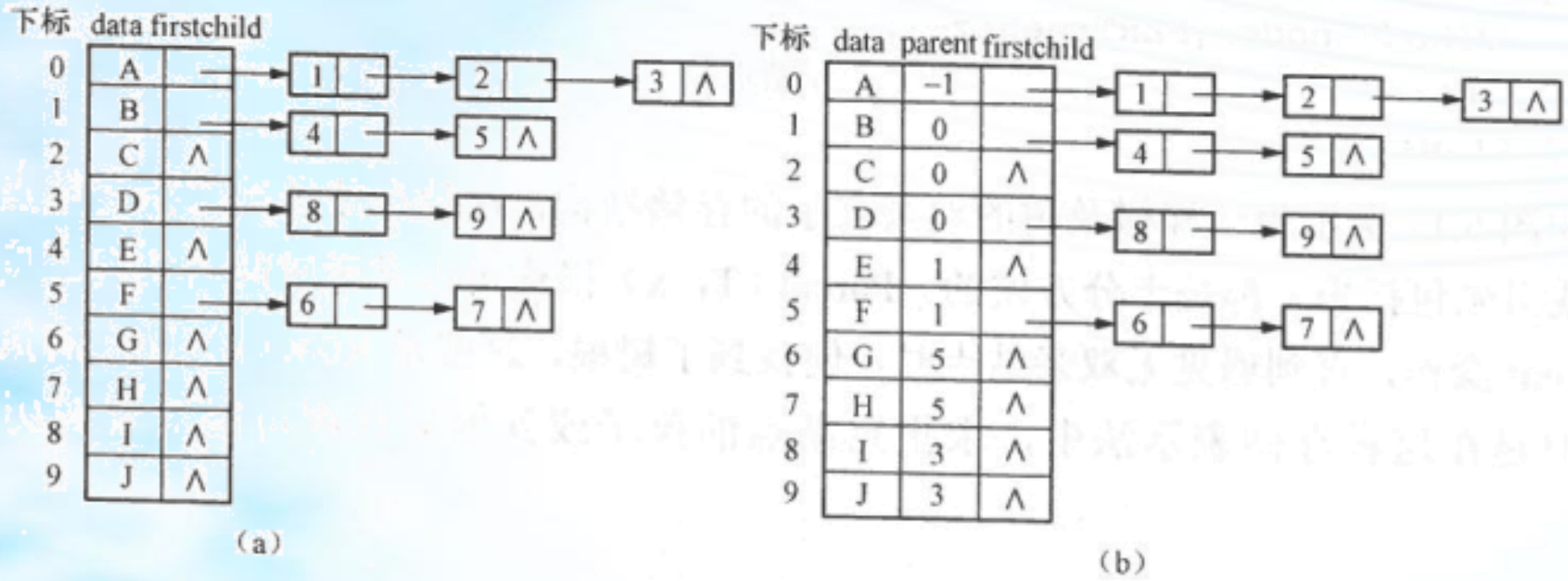


图 5.18　孩子链表和带双亲的孩子链表

3. 孩子兄弟表示法

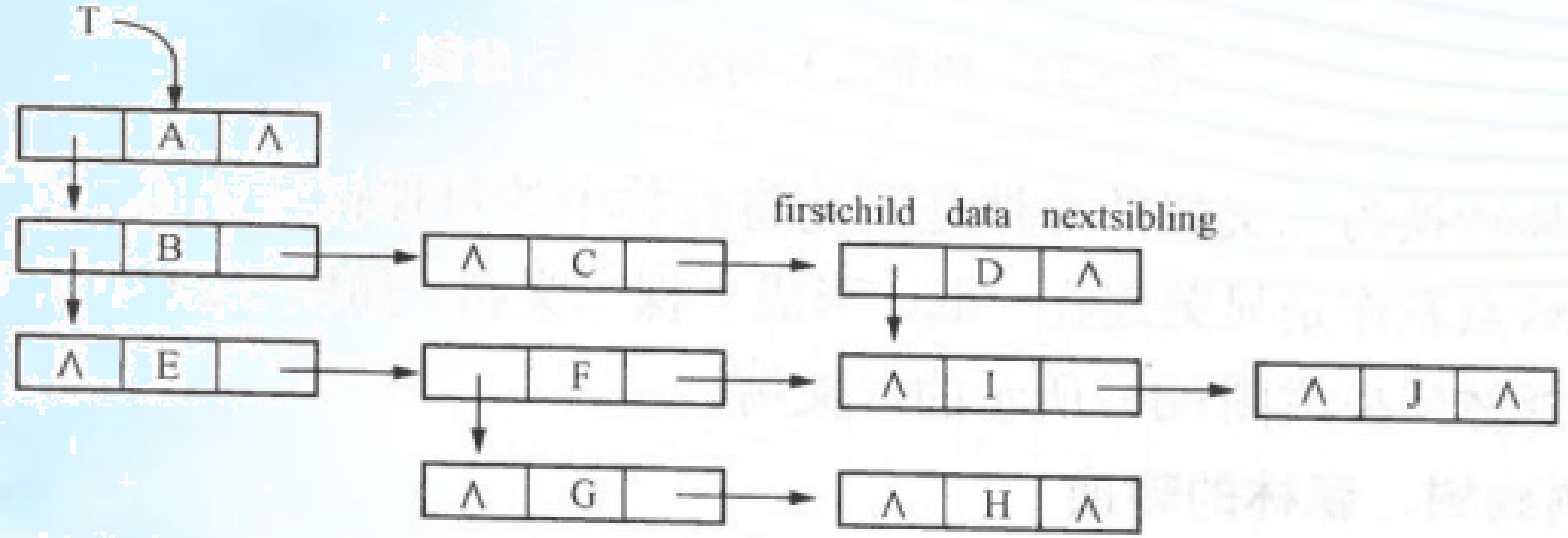


图 5.19　图 5.17（a）所示树的二叉链表（孩子兄弟表示）

5.5.2二、树、森林与二叉树的转换

树、森林与二叉树之间有一个自然的对应关系，即任何一棵树或一个森林都可唯一地对应于一棵二叉树，而任何一棵二叉树也能唯一地对应于一个森林或一棵树。

1. 树、森林到二叉树的转换

以二叉链表作为媒介，可导出树与二叉树之间的一个对应关系。也就是说，给定一棵树，可以找到唯一的一棵二叉树与之对应，从物理结构上来看，它们的结构是相同的，只是解释不同而已。图5.20直观地展示了树与二叉树之间的对应关系。

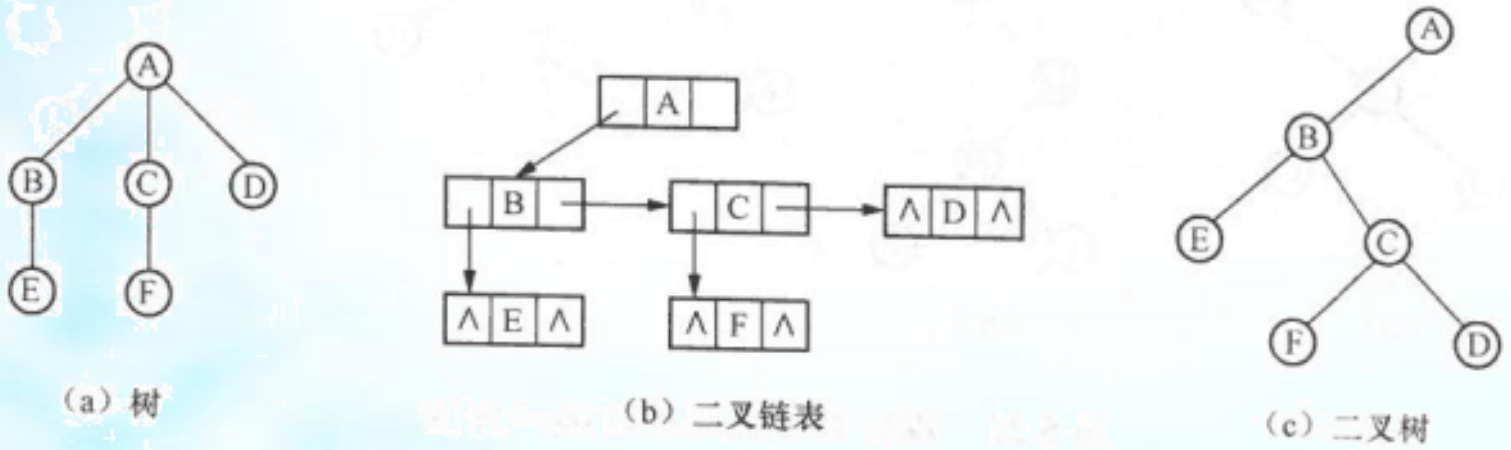


图 5.20　树与二叉树的对应关系示例

首先在所有兄弟结点之间加一道连线，然后再对每个结点保留长子的连线，去掉该结点与其他孩子的连线。由于树根没有兄弟，所以转换后的二叉树，其根结点的右子树必为空。使用上述转换方法可以将图5.21(a)所示的树转换成图5.21(b)所示的形式。它实际上就是一棵二叉树，若将所有兄弟之间的连线按顺时针方向旋转45°就看得更清楚，如图5.21(c)所示。

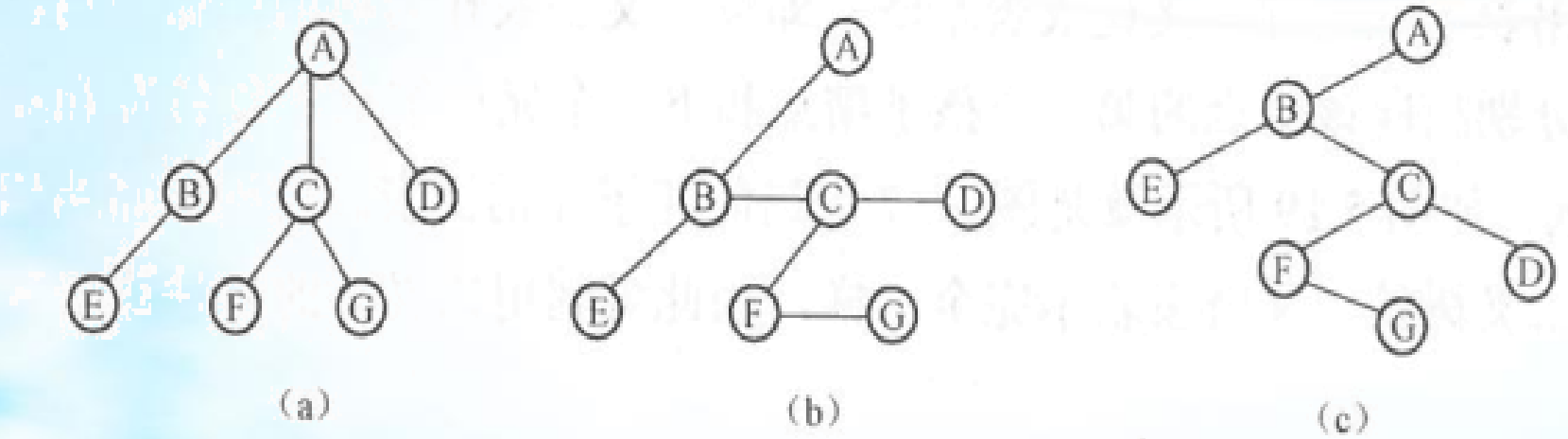


图 5.21　树到二叉树的转换示意图

5.5.2 树、森林与二叉树的转换

将一个森林转换为二叉树的方法是：先将森林中的每棵树转化成二叉树，然后再将各二叉树的根结点看作是兄弟连在一起，形成一棵二叉树。如图5.22中图（a）、（b）、（c）三棵树构成的森林转换成图（d）所示的二叉树。

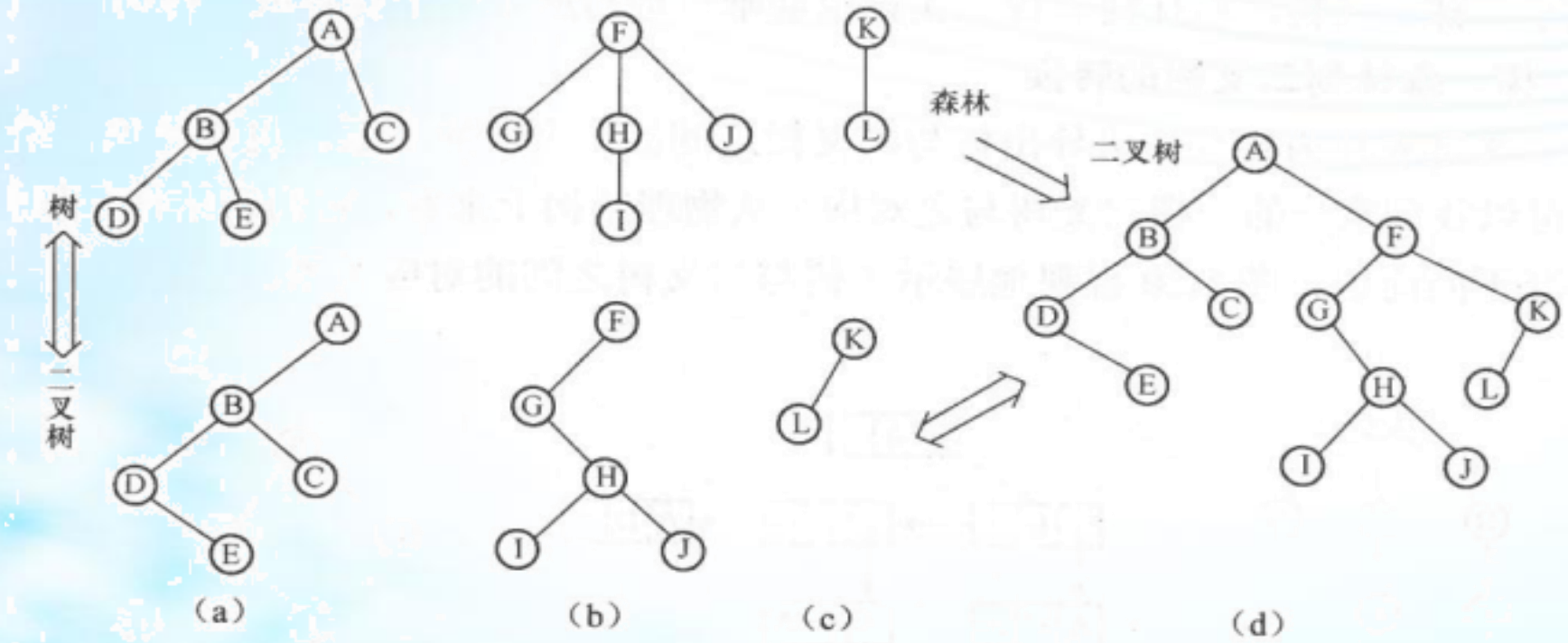


图 5.22　森林到二叉树的转换示意图

5.5.2 树、森林与二叉树的转换

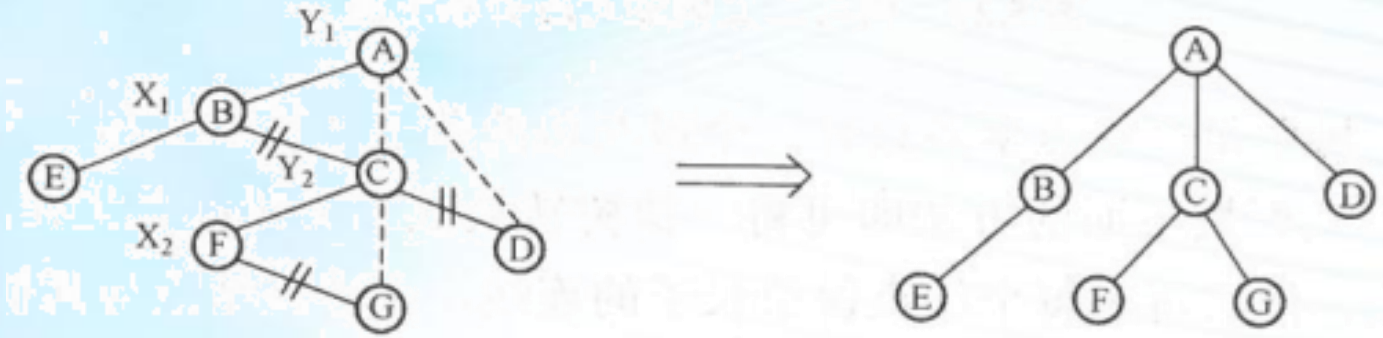
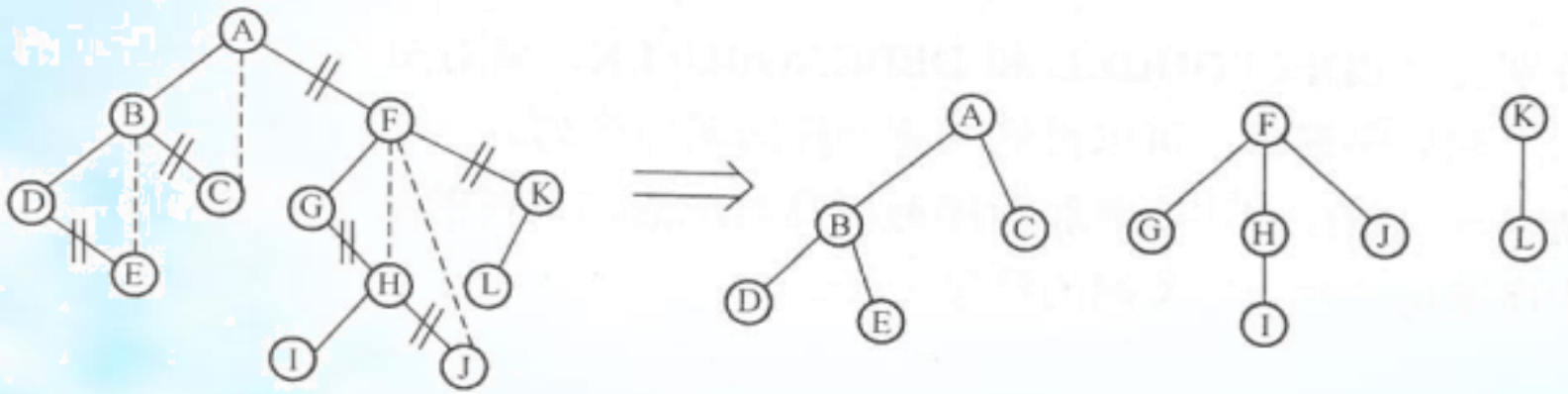


图 5.23　二叉树到树的转换



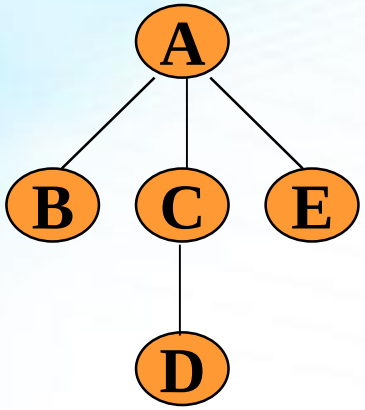
（a）二叉树　　（b）森林

图 5.24　二叉树到森林的转换

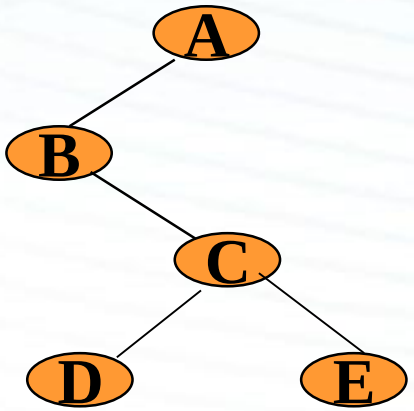
一棵树唯一对应一棵二叉树

例：

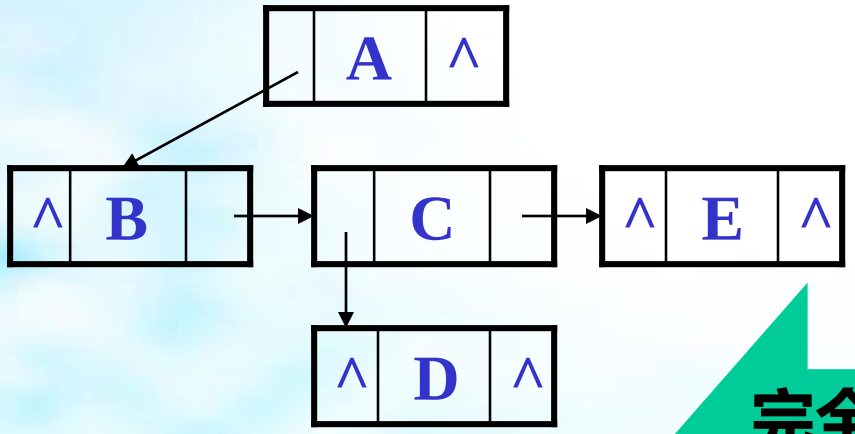
树



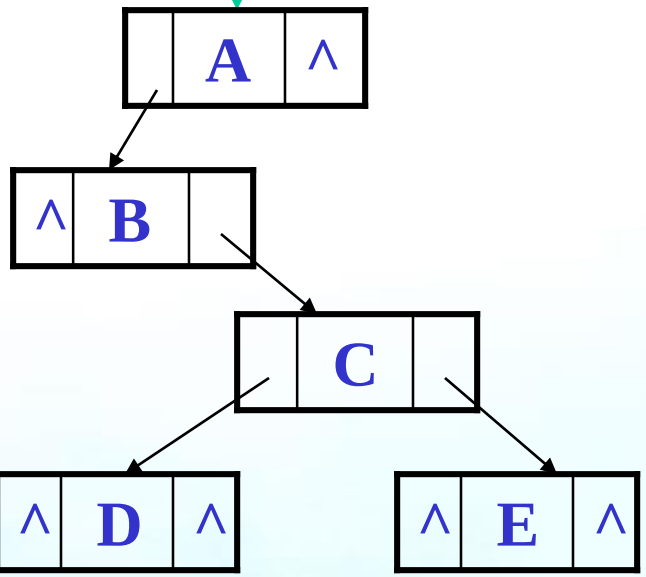
二叉树



孩子兄弟表示



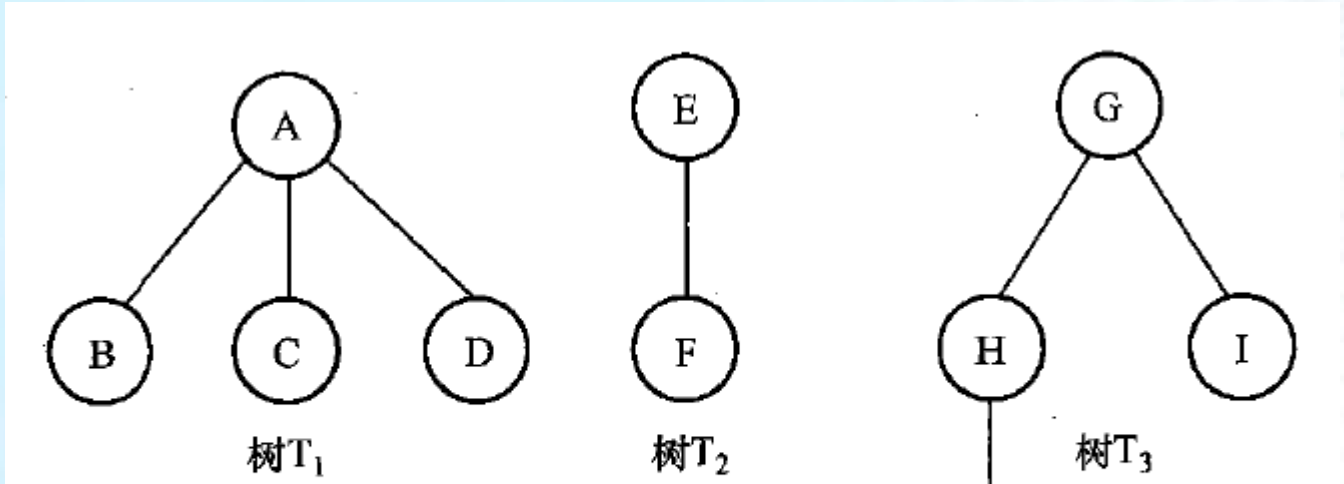
二叉链表表示

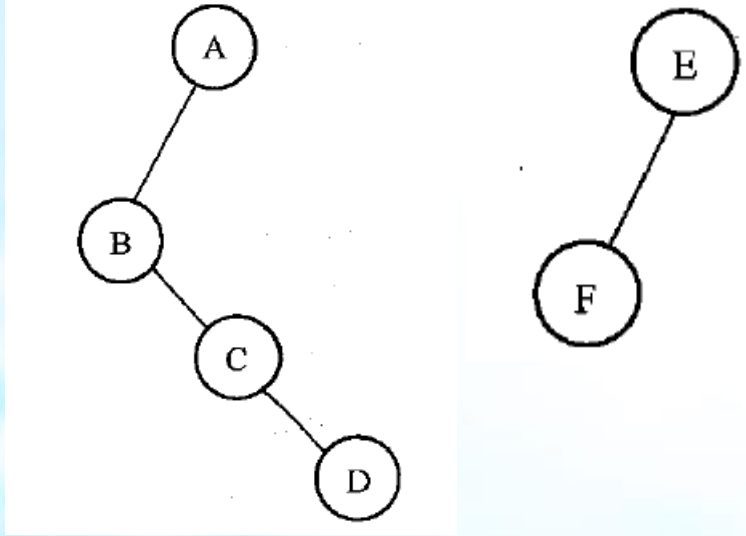


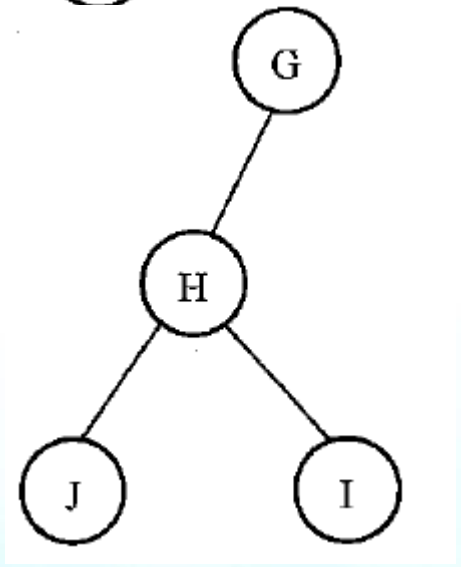
完全一样

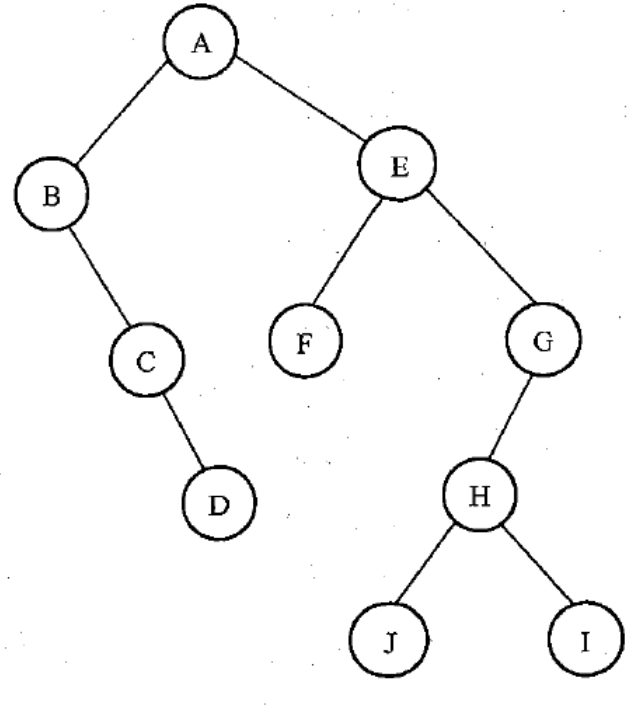
孩子兄弟链表的结构形式与二叉链表完全相同，但结点中指针的含义不同。

5.5.2 树、森林与二叉树的转换

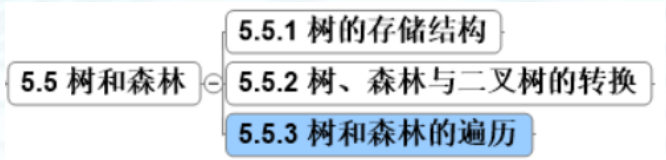








5.5.3三、树和森林的遍历



按照森林和树相互递归的定义，可以推出访问森林的两种遍历方法：前序遍历和后序遍历。

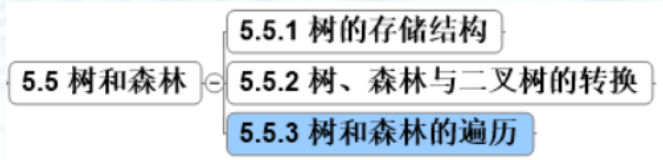
1. 前序遍历森林

若森林为非空，则可按下述规则遍历：

①访问森林中的第一棵树的根结点。

②前序遍历第一棵树中的根结点的子树森林。

③前序遍历除去第一棵树之后剩余的树构成的森林。



2. 后序遍历森林

若森林为非空，则可按下述规则遍历：

①后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后序遍历除去第一棵树之后剩余的树构成的森林。

简而言之，前序遍历森林是从左到右依次按前序（先根）次序遍历森林中的每一棵树，而后序遍历森林则是从左到右依次按后序（后根）次序遍历森林中的每一棵树。

若对如图5.22所示的森林进行前序遍历和后序遍历，则得到该森林的前序序列和后序序列分别为ABDECFGHLIKL和DEBCAGIHJFLK；而对图5.22(d)所示的二叉树进行前序遍历和后序遍历，可以得到同前一样的遍历序列。

也就是说，前序遍历森林和前序遍历其对应的二叉树的结果是相同的，而后序遍历森林和中序遍历其对应二叉树的结果一样。

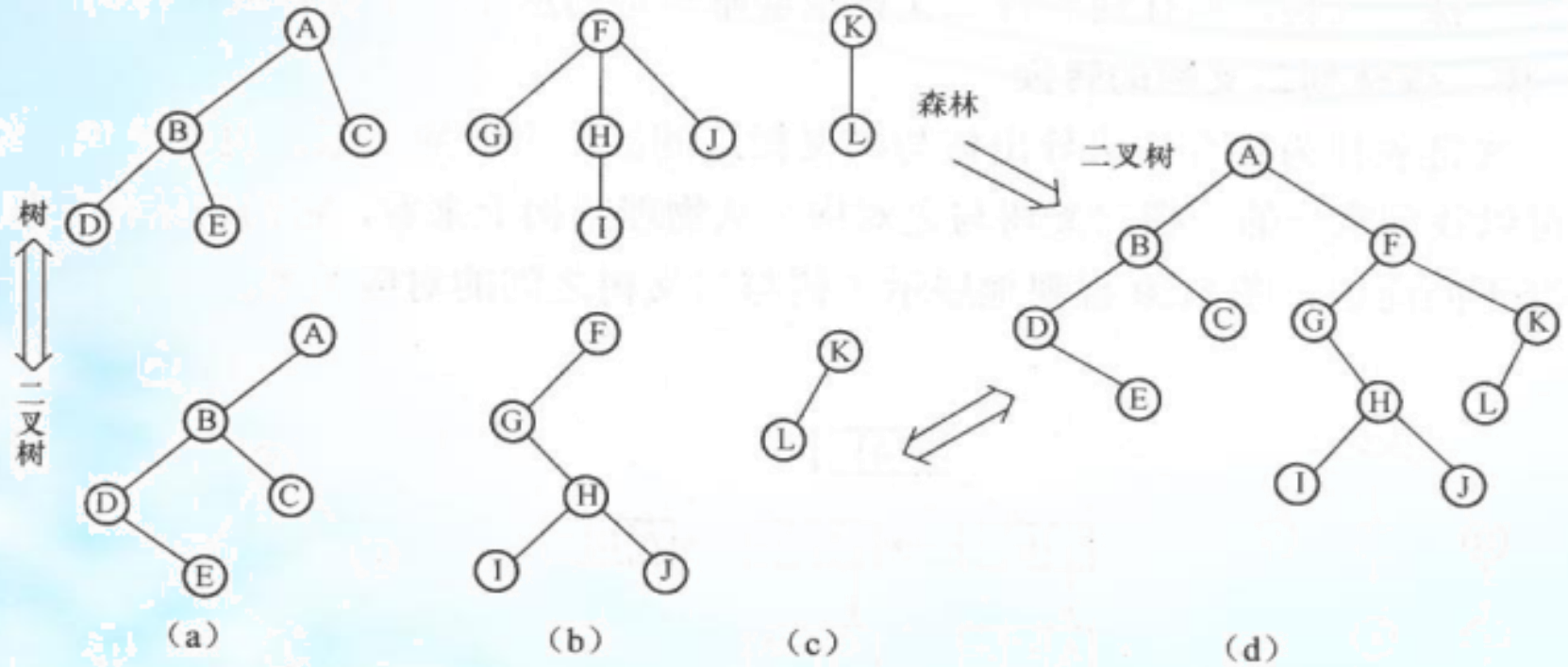


图 5.22　森林到二叉树的转换示意图

5.5.3 树和森林的遍历

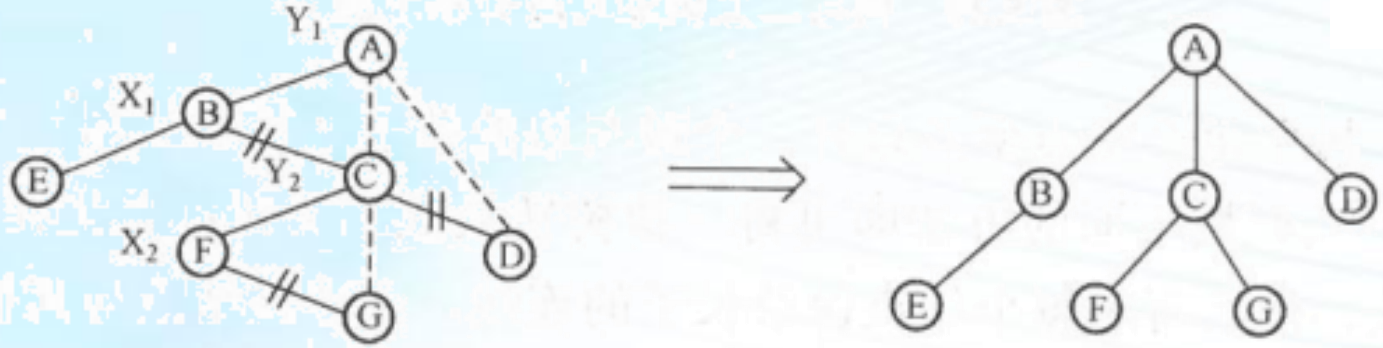
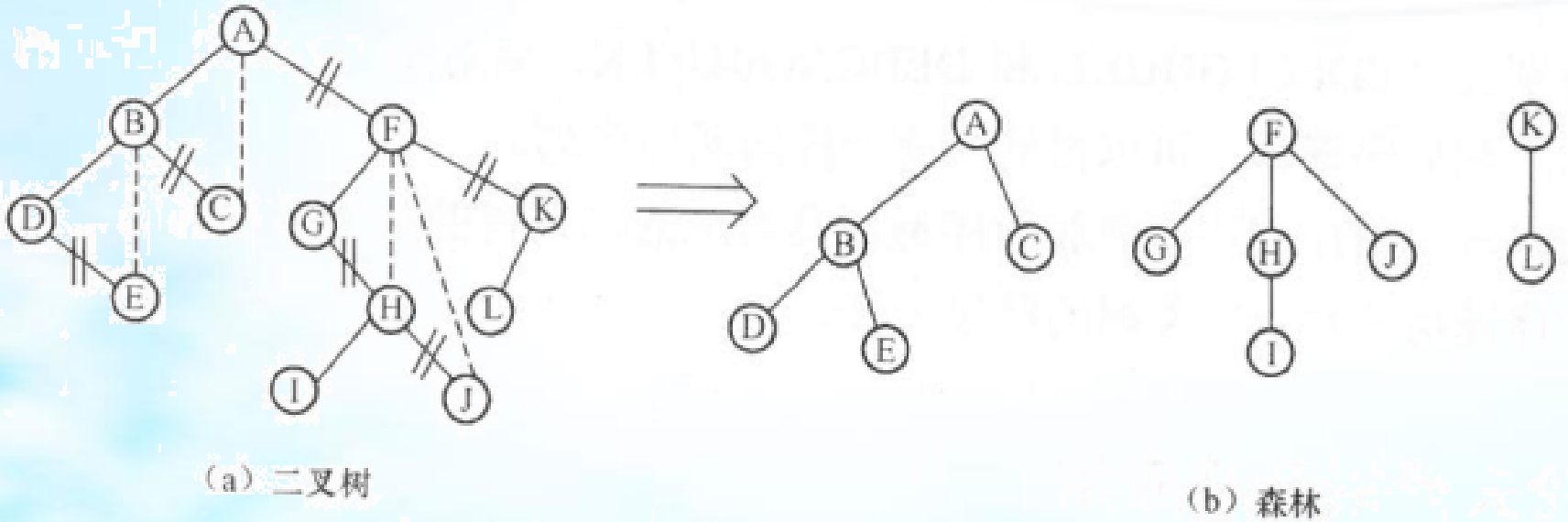
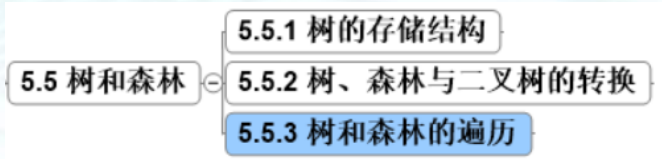


图 5.23　二叉树到树的转换



（a）二叉树　　（b）森林

图 5.24　二叉树到森林的转换



访问森林的两种遍历方法：前序遍历和后序遍历。

1. 前序遍历森林

若森林为非空，则可按下述规则遍历：

①访问森林中的第一棵树的根结点。

②前序遍历第一棵树中的根结点的子树森林。

③前序遍历除去第一棵树之后剩余的树构成的森林。

2. 后序遍历森林

若森林为非空，则可按下述规则遍历：

①后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后序遍历除去第一棵树之后剩余的树构成的森林。

简而言之，前序遍历森林是从左到右依次按前序（先根）次序遍历森林中的每一棵树，而后序遍历森林则是从左到右依次按后序（后根）次序遍历森林中的每一棵树。

若对如图5.22所示的森林进行前序遍历和后序遍历，则得到该森林的前序序列和后序序列分别为ABDECFGHLIKL和DEBCAGIHJFLK；而对图5.22(d)所示的二叉树进行前序遍历和后序遍历，可以得到同前一样的遍历序列。

也就是说，前序遍历森林和前序遍历其对应的二叉树的结果是相同的，而后序遍历森林和中序遍历其对应二叉树的结果一样。

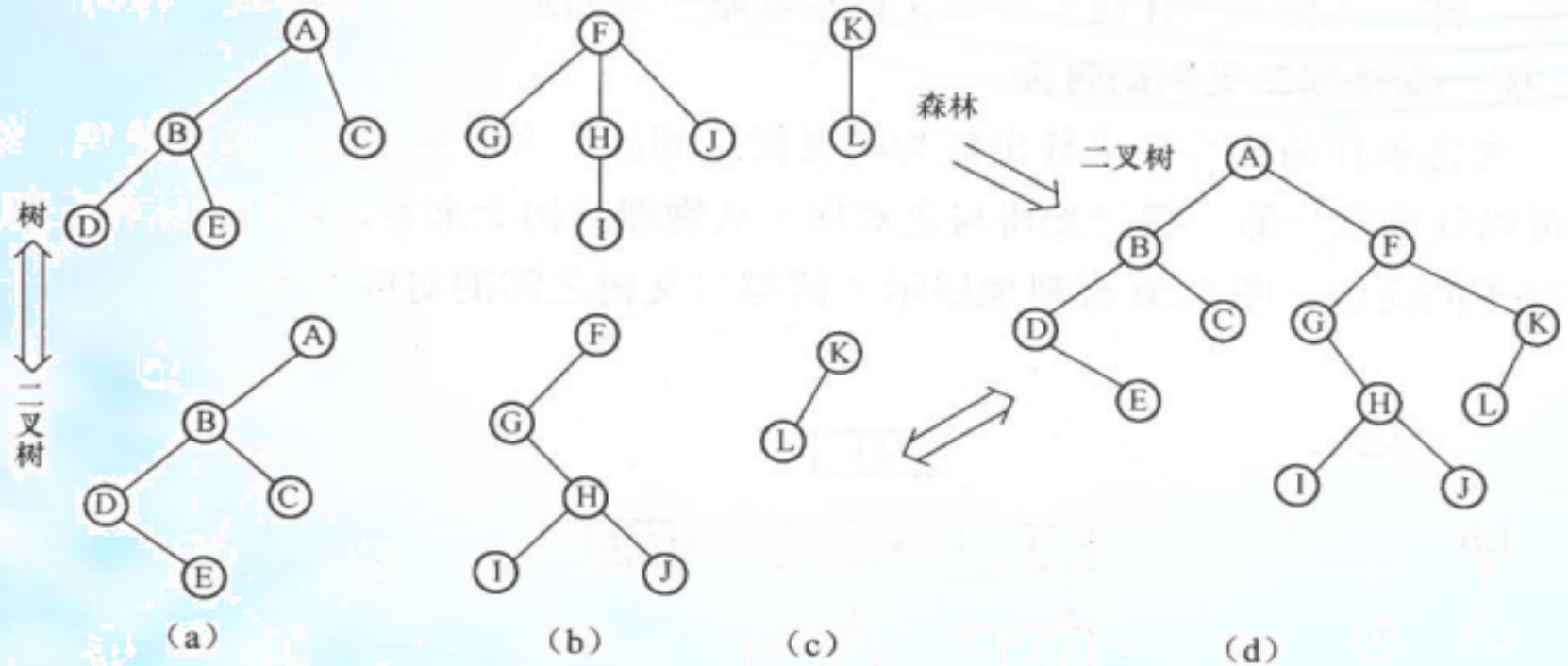
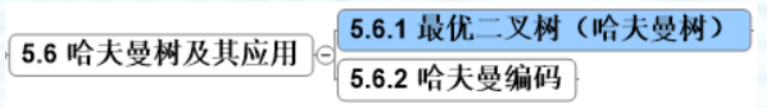


图 5.22　森林到二叉树的转换示意图



哈夫曼树又称最优树，是一类带权路径长度最短的树。

从树根结点到某结点之间的路径长度与该结点上权的乘积称为该结点的带权路径长度，树中所有叶子结点的带权路径长度之和称为树的带权路径长度，通常记为

$$WPL=\sum_{i=1}^{n} w_i l_i$$

其中，n 表示叶子结点个数，w_i 和 l_i 分别表示叶子结点 k_i 的权值和根到 k_i 之间的路径长度。

在权值为州 $w_1, w_2, \ldots, w_n$ 的 n 个叶结点构成的所有二叉树中，带权路径长度 WPL 最小的二叉树称为哈夫曼树或最优二叉树。

5.6.1一、最优二叉树（哈夫曼树）

例如，假设给定4个叶结点的权值分别为8、5、2和4，可以构造出如图5.25所示的三棵二叉树（当然还可以构造更多的二叉树），它们的带权路径长度分别为：

图5.25(a):WPL=8X3+5X3+2X1+4X2=49

图5.25(b):WPL=8X3+5X2+2X3+4X1=44

图5.25(c):WPL=8X1+5X2+2X3+4X3=36

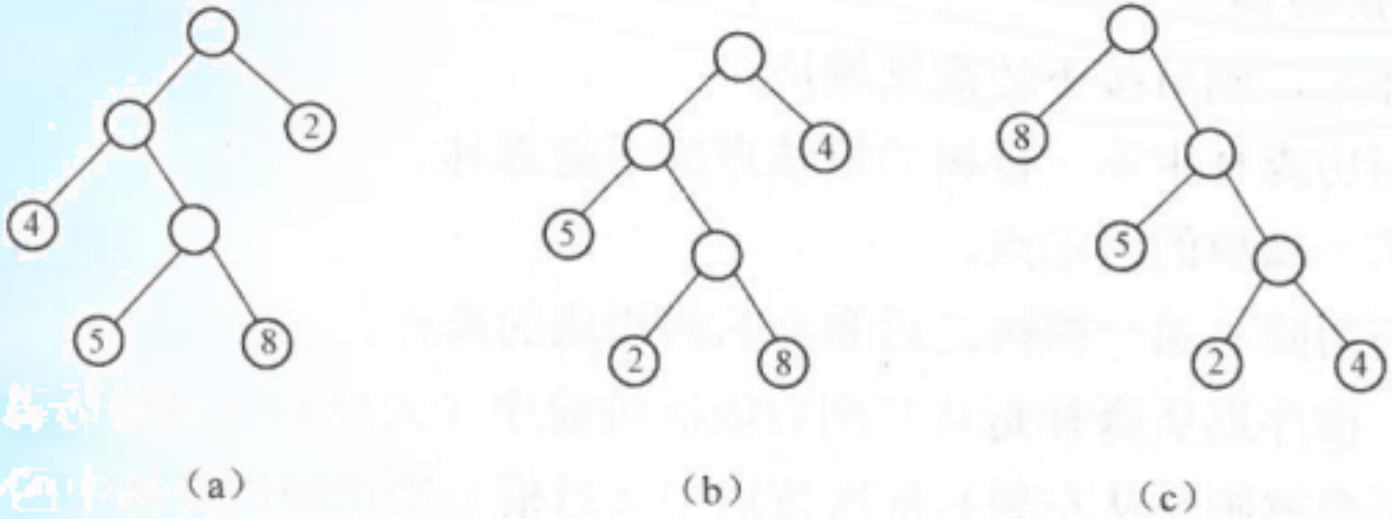


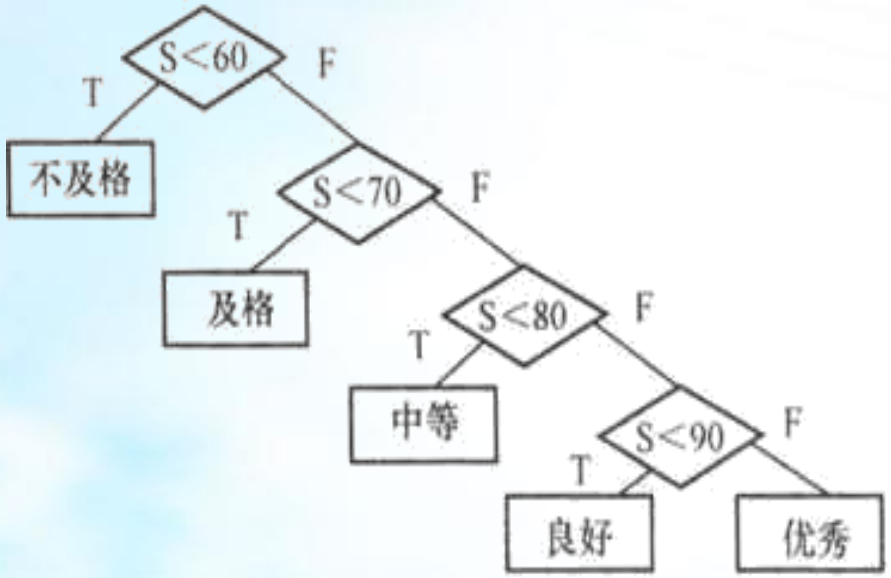
图 5.25　具有不同带权路径长度的二叉树

其中，图5.25(c)所示树的WPL最小，可以验证它就是一棵哈夫曼树，即它的带权路经长度在所有带权值为8、5、2、4的四个叶结点的二叉树中为最小。若叶结点上的权值均相同，其中完全二叉树一定是最优二叉树，否则不一定是最优二叉树。

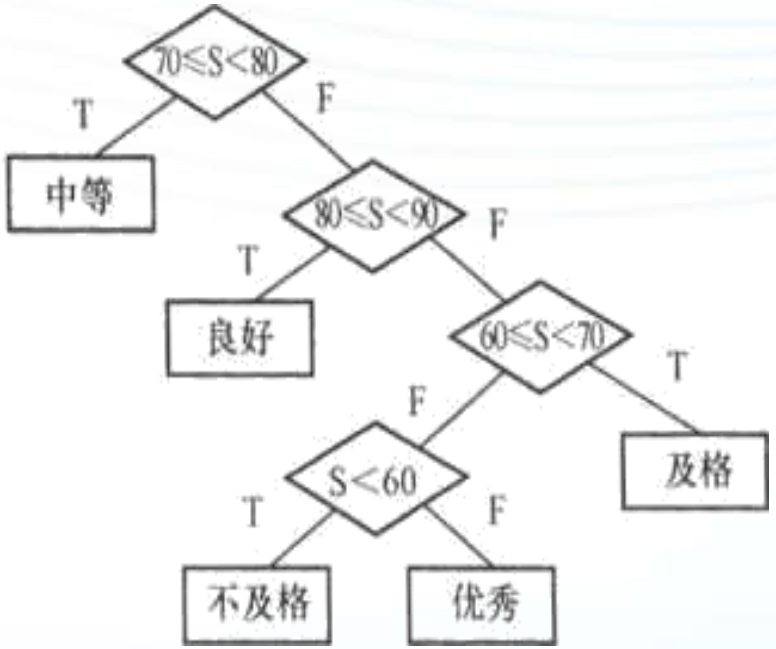
5.6.1 最优二叉树（哈夫曼树）

表5.1 成绩分布概率

成绩	0～59	60～69	70～79	80～89	90～100
占百分数	5	15	40	30	10



31500次比较



20500次比较

1. 哈夫曼算法

那么，如何构造一棵哈夫曼树呢？哈夫曼首先提出了构造最优二叉树的方法，所以称其为哈夫曼算法，其基本思想如下：

（1）根据与 n 个权值 $\{w_1, w_2, \ldots, w_n\}$ 对应的 n 个结点构成 n 棵二叉树的森林 $F = \{T_1, T_2, \ldots, T_n\}$

其中每棵二叉树 T_i 都只有一个权值为 w_i 的根结点，其左、右树均为空。

（2）在森林F中选出两棵根结点的权值最小的树作为一棵新树的左、右子树，且置新树的附加根结点的权值为其左、右子树上根结点的权值之和。

（3）从F中删除这两棵树，同时把新树加入到F中；

（4）重复步骤（2）和（3），直到F中只有一棵树为止，此树便是哈夫曼树。

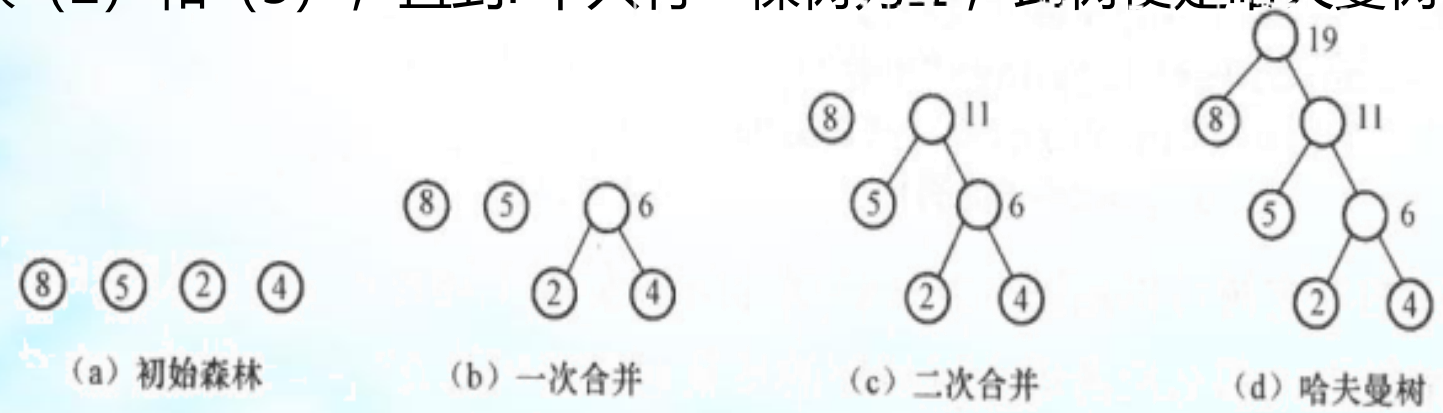


图 5.27　哈夫曼树的构造过程示例

【例5.5】某种系统在通信中只可能出现8种字符：a，b，c，d，e，f，g，h。它们在电文出现的频率分别为：0.06，0.15，0.07，0.09，0.16，0.27，0.08，0.12。现以其频率的百分数作为权值，即6，15，7，9，16，27，8，12建立哈夫曼树。

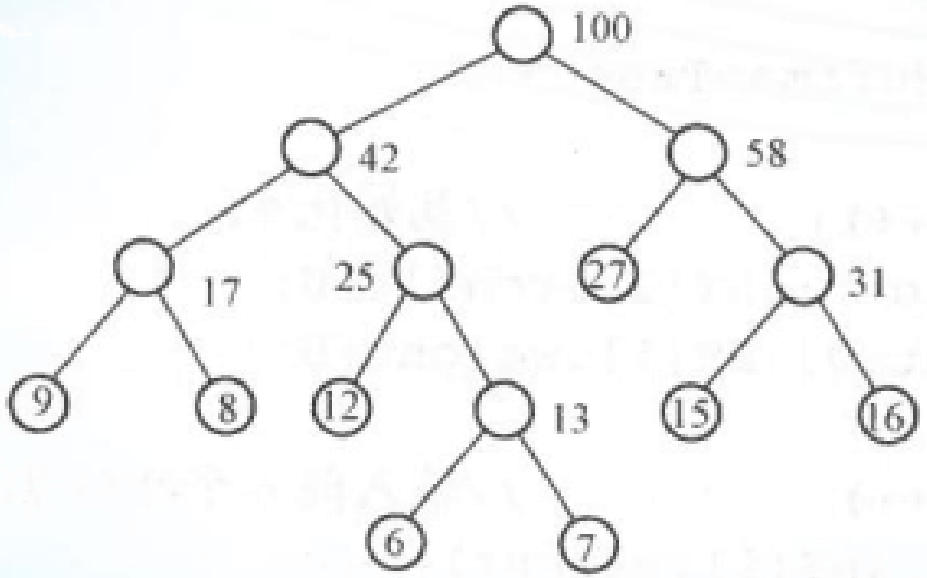


图 5.28　例 5.5 构造的哈夫曼树

5.6.2二、哈夫曼编码

利用哈夫曼树求得的用于通信的二进制编码称为哈夫曼编码。树中从根到每个叶子都有一条路径，对路径上的各分支约定指向左子树的分支表示“0”码，指向右子树的分支表示“1”码，取每条路径上的“0”或“1”的序列作为各个叶子结点对应的字符编码，这就是哈夫曼编码。

可以利用二叉树来设计二进制的前缀编码。如图5.29(a)所示的哈夫曼树，其左分支表示字符“0”，右分支表示字符“1”，则以根结点到叶结点路径上的分支字符组成的串作为该叶结点的字符编码，可得到字符a、b、c、d的二进制前缀编码分别为：0、10、110、111。

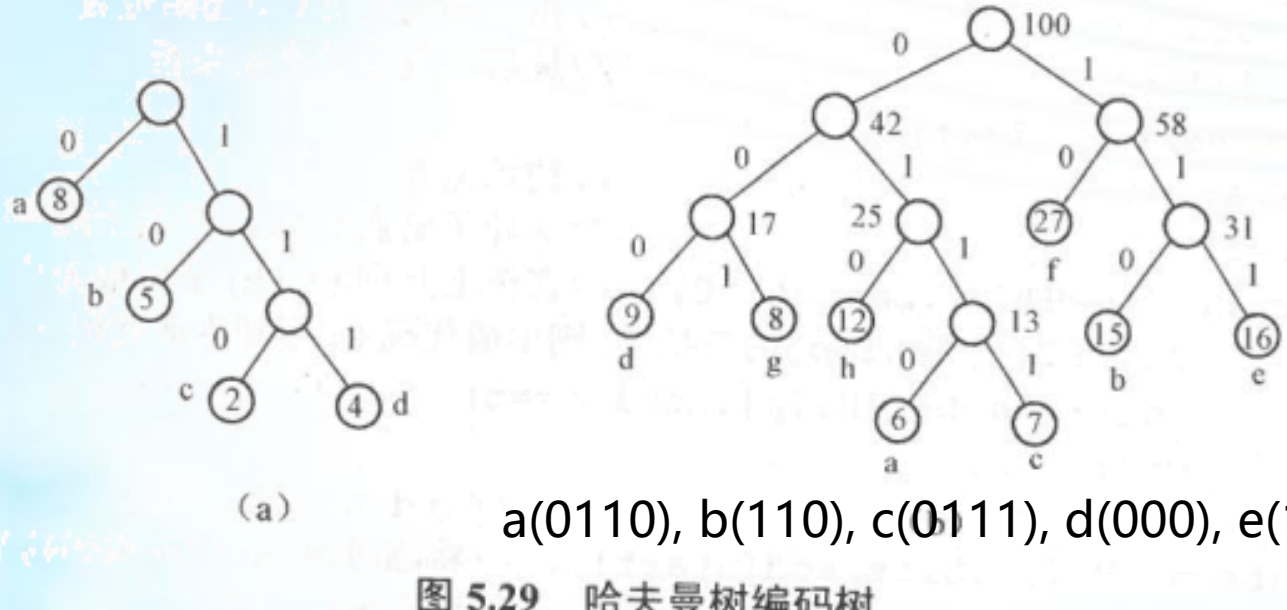
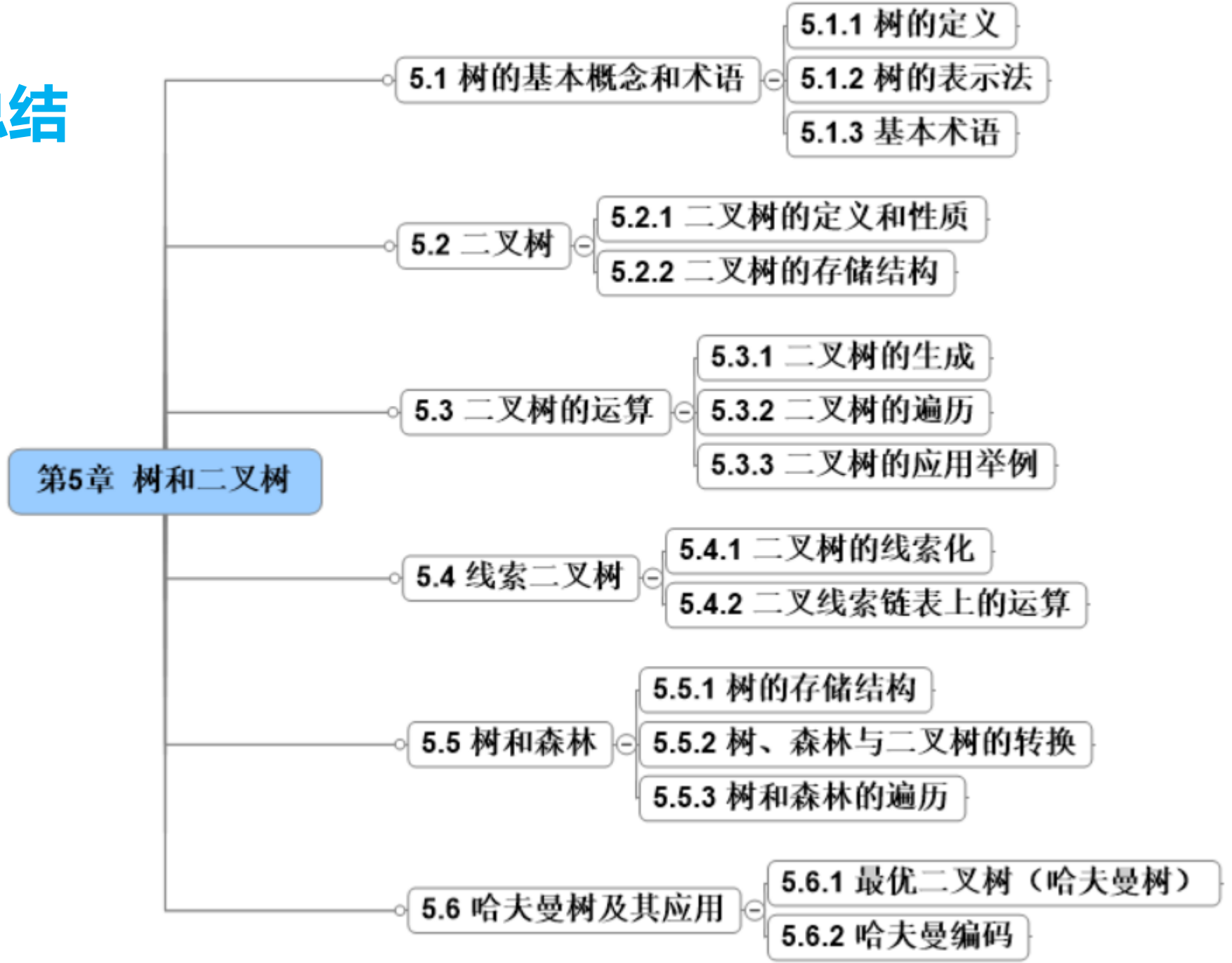


图 5.29　哈夫曼树编码树

a(0110), b(110), c(0111), d(000), e(111), f(10), g(001), h(010)。

假设每种字符在电文中出现的次数为W_i，编码长度为L_i，电文中有 n 种字符，则电文编码总长为$\sum W_i L_i$。若将此对应到二叉树上，W_i为叶结点的权，L_i为根结点到叶结点的路径长度。那么，$\sum W_i L_i$恰好为二叉树上带权路径长度。

本章总结


第5章 树和二叉树
5.1 树的基本概念和术语
5.1.1 树的定义
5.1.2 树的表示法
5.1.3 基本术语
5.2 二叉树
5.2.1 二叉树的定义和性质
5.2.2 二叉树的存储结构
5.3 二叉树的运算
5.3.1 二叉树的生成
5.3.2 二叉树的遍历
5.3.3 二叉树的应用举例
5.4 线索二叉树
5.4.1 二叉树的线索化
5.4.2 二叉线索链表上的运算
5.5 树和森林
5.5.1 树的存储结构
5.5.2 树、森林与二叉树的转换
5.5.3 树和森林的遍历
5.6 哈夫曼树及其应用
5.6.1 最优二叉树（哈夫曼树）
5.6.2 哈夫曼编码

祝大家顺利通过考试！