CHAPTER

第5章 树和二叉树

树与二叉树都属于树(形)结构。树结构是一种比线性结构复杂的非线性数据结构,比较适合描述具有层次关系的数据,如记载“祖先—后代”的族谱、表述“上级—下级”的组织机构、表示“整体—部分”的事物构成等。

树结构在计算机领域中有着广泛的应用,特别是二叉树。例如,操作系统中用树表示文件目录的组织结构;编译系统中用语法树表示源程序的语法结构;数据挖掘中用决策树进行数据分类。

本章讨论树与二叉树的存储设计及遍历操作;二叉树的二叉链表实现;树或森林与二叉树之间的相互转换;二叉树的典型应用——最优二叉树与哈夫曼编码。

本章主要知识点

- 树的逻辑特性和存储设计。
- 树与森林的遍历方法。
- 二叉树的性质、存储设计。
- 二叉树的创建、遍历、销毁等算法。
- 树或森林与二叉树之间的相互转换。
- 最优二叉树及哈夫曼编码。

本章教学目标

- 掌握树结构的逻辑特性并用于问题建模。
- 掌握树、二叉树的存储方法并在具体应用中选择或设计合适的存储结构。
- 掌握二叉树的遍历原理与方法并能用于解决问题。
- 掌握最优二叉树与哈夫曼编码的构建并能够用于解决应用问题。

5.1 树

树是一种非线性结构，其存储与操作与线性结构完全不同，树的定义是递归的，因此，树的许多操作可用递归程序实现。


微课视频

5.1.1 树的定义与表示

树(tree)是 $n(n\geqslant 0)$ 个数据元素的有限集合。当 $n=0$ 时，称为空树；任意一棵非空树满足下列条件。

(1) 有且仅有一个没有前驱的特殊元素，即根(root)结点[①]。

(2) 当 $n>1$ 时，除根结点之外的其余数据元素被分成 $m(m>0)$ 个互不相交的集合 $T_1, T_2, \cdots, T_m$，其中每个集合又是一棵树，称为根的子树(subtree)；即数据元素 $D=\{\text{root}\}\cup T_1\cup T_2\cup\cdots\cup T_m$，且 $T_i\cap T_j=\varnothing$。

(3) 对于每棵子树，同样可看成由子树的根与子树根的子树组成。

由上可知，树的定义是递归的，例如图 5-1 是一棵树。其中，①A 为树根，T_1、T_2、T_3 是 A 的子树；B、P、C 分别为 3 棵子树的根。②T_{11}、T_{12} 分别为子树 T_1 的根 B 的子树；T_{31}、T_{32} 和 T_{33} 分别为子树 T_3 的根 C 的子树。

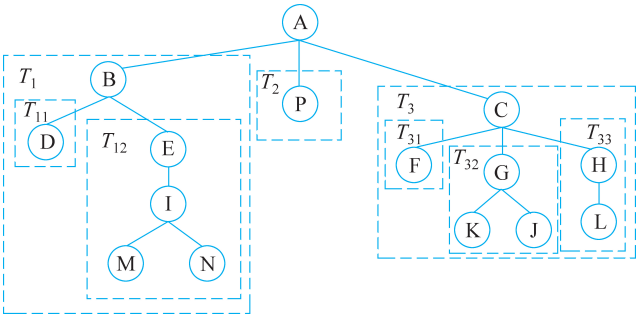
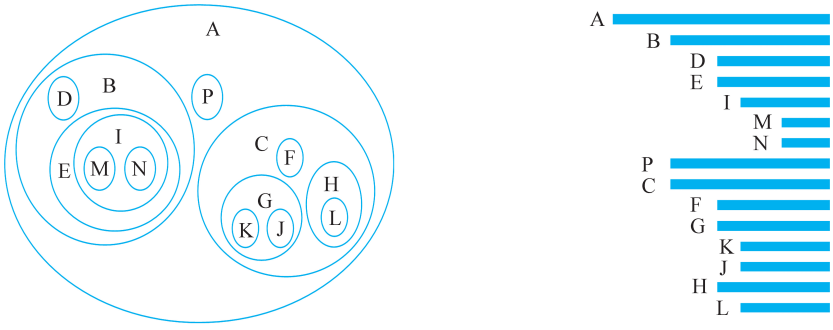


图 5-1 树 1

在树结构中，尽管用无箭头的线表示结点之间的关系，但实际上树中双亲与孩子之间是一种有向关系。

除了上述图形表示法之外，树还有嵌套集合表示、凹入表示和广义表表示等方法。**嵌套集合**是指一些集合的集体，其中任何两个集合或者不相交，或者一个包含另一个。树的嵌套集合表示是指根为一个大集合，根的子树构成这个大集合中若干互不相交的子集，如此嵌套下去，如图 5-2(a)所示。**树的凹入表示**如图 5-2(b)所示，通过不同的缩进表示层次包含关系，主要用于树的屏幕输出。**树的广义表表示**是用括号表示层次及其包含关系，如图 5-2(c)所示。

① 在树中，常常将数据元素称为结点。



(a) 树的嵌套集合表示　　(b) 树的凹入表示

(A (B (D, E (I (M, N))), P, C (F, G (K, J), H (L))))

(c) 树的广义表表示

图 5-2　树的各种表示法

5.1.2　树的术语

1. 结点的度和树的度

结点的度(degree)　结点所拥有的子树的个数称为该结点的度。例如在图 5-1 所示的“树 1”中，结点 A 和 C 的度为 3；结点 B、I、G 的度为 2；结点 E 和 H 的度为 1；其余结点的度为 0。

树的度　树内结点度的最大值称为树的度。图 5-1 所示的“树 1”的度为 3。

2. 各类结点

叶(leaf)结点　度为 0 的结点称为叶结点，也称为**终端结点**。

分支(branch)结点　度不为 0 的结点称为分支结点，也称为**非终端结点**。

孩子结点(child node)　树中某结点 X 的子树的根(即 X 的后继结点)称为 X 的孩子结点。例如“树 1”中 A 的孩子结点为 B、P、C；B 的孩子结点为 D、E；C 的孩子结点为 F、G、H。

双亲(parent)结点　如果树中某结点 Y 是另一个结点 X 的孩子，则结点 X(即 Y 的前驱)称为 Y 的双亲结点。树中除根没有双亲结点外，其余的结点均有唯一的双亲结点。例如“树 1”中 A 是 B、P、C 的双亲；B 是 D、E 的双亲；C 是 F、G、H 的双亲。

兄弟(brother)　树中具有相同双亲的结点互为兄弟结点。例如，“树 1”中 B、P、C 互为兄弟；D、E 互为兄弟；F、G、H 互为兄弟。

堂兄弟　树中双亲为兄弟的结点的孩子互为堂兄弟结点。例如，“树 1”中 D、E 与 F、G、H 互为堂兄弟。

祖先(ancestor)　从根到某结点 X 所经分支上的所有结点称为结点 X 的祖先。例如，“树 1”中 M 的祖先为 A、B、E 和 I；K 的祖先为 A、C、G。

子孙(descendant)　以某结点 X 为根的子树中任一结点都称为结点 X 的子孙结点。例如“树 1”中，除根结点 A 之外的所有结点均为 A 的子孙。F、G、H、K、J 和 L 都是 C 的

子孙。

3. 路径和路径长度

路径(path) 如果树的结点序列 $n_1,n_2,\cdots,n_k$ 满足结点 n_i 是结点 n_{i+1} 的双亲($1\leqslant i<k$),则把 $n_1,n_2,\cdots,n_k$ 称为一条由 n_1 到 n_k 的路径。

路径长度(path length) 一条路径上经过的边数称为路径长度。例如“树1”中,A、B、E、I、N是一条从根A到结点N的路径,其长度为4。

4. 层次和树的深度

层次(level) 树具有层次性,从根开始,根为第1层,根的孩子为第2层,根的孩子的孩子为第3层,以此类推。树中任一结点的层次等于其双亲结点的层次加1。例如“树1”中,第2层的结点有B、P和C;第3层的结点有D、E、F、G和H;第4层的结点有I、K、J和L;第5层的结点有M和N。

树的深度(depth) 树中结点的最大层次数称为树的深度或高度。例如,“树1”的深度为5。

5. 有序树和无序树

有序树(ordered tree) 如果一棵树中的各子树从左到右是有次序的,则称这棵树为有序树。在有序树中,如果子树位置从左到右不同,则被认为是不同的树。通常,将有序树最左边的子树称为第一棵子树,最右边的称为最后一棵子树。

无序树(unordered tree) 如果一棵树中的各子树无左、右次序之分,则称这棵树为无序树。

6. 森林

森林(forest) 森林是 $m(m\geqslant 0)$ 棵互不相交的树的集合。度大于1的树如果删去根结点,就变成了森林。例如,删去“树1”的根结点A,它就变成由3棵树构成的森林。

归纳可知,树具有以下逻辑特性。

- 树结构有明显的层次性,根为第1层,根的孩子为第2层,根的孩子的孩子为第3层,以此类推。
- 根无前驱结点,其余结点有唯一前驱(即双亲)结点。
- 叶结点无后继结点;非叶结点可以有一个或多个直接后继(即孩子)结点。
- 祖先与子孙是父子关系的延伸,它确定了树中各结点之间的纵向次序。
- 在有序树中,同一组兄弟从左到右有长幼之分,它定义了树中各结点之间的横向次序。

5.1.3 树的抽象数据类型

一棵树的数据元素属于同一个集合,数据元素之间具有一对多的关系。关于树的基本操作可分为创建与销毁类、查询访问类、编辑类。树的抽象数据类型描述如下。

```
ADT Tree
数据对象: D={a_i | a_i∈ElementSet, i=1, 2, …, n, n≥0},其中,n为树中数据元素个数,
          n=0时为空树。
数据关系: R={<a_i, a_j> | a_i, a_j∈D, 1≤i, j≤n,有且仅有一个结点没有前驱结点,其余结点有
          唯一前驱结点,任一个结点有零个、一个或多个后继结点}
```

```
基本操作:
    InitTree(&T)                        //初始化树
        操作功能: 创建一个空树 T。
        操作输出: 创建成功,返回 true;不成功,退出。
    DestroyTree(&T)                     //销毁树
        操作功能: 释放树 T 所占的存储空间。
        操作输出: 无。
    CreateTree(&T,definition)           //创建树
        操作功能: 按树的定义(definition)创建树 T。
        操作输出: 创建成功,返回 true;否则,返回 false。
    ClearTree(&T)                       //清空树
        操作功能: 把树 T 变成一棵空树。
        操作输出: 无。
    PreTree(T)                          //先根遍历树
        操作功能: 先根遍历树的各结点。
        操作输出: 遍历结果。
    PostTree(T)                         //后根遍历树
        操作功能: 后根遍历树的各结点。
        操作输出: 遍历结果。
    LevelTree(T)                        //层序遍历树
        操作功能: 层序遍历树的各结点。
        操作输出: 遍历结果。
    Root(T,&e)                          //访问树根
        操作功能: 查询树根。
        操作输出: 树非空,e 为树根元素,返回 true;否则,返回 false。
    Parent(T,e, &par_e)                 //访问双亲
        操作功能: 查询值为 e 元素的双亲。
        操作输出: 如果存在 par_e 为 e 的双亲,返回 true;否则,返回 false。
    LeftFirstChild(T,e, &lc_e)          //访问左孩子
        操作功能: 查询值为 e 的元素的左边的第一个孩子。
        操作输出: 如果存在 lc_e 为 e 左边的第一个孩子,返回 true;否则,返回 false。
    RightSibling(T, e, &rs_e)           //访问兄弟
        操作功能: 查询值为 e 的元素的右边的第一个兄弟。
        操作输出: 如果存在 rs_e 为 e 右边的第一个兄弟,返回 true;否则,返回 false。
    TreeEmpty(T)                        //测树空
        操作功能: 判断树 T 是否为空树。
        操作输出: 如果 T 是空树,返回 true;否则,返回 false。
    TreeDepth(T)                        //测树深
        操作功能: 查询树深。
        操作输出: 返回树的深度。
    InsertChild(&T, p, i, c)            //插入结点
        操作功能: p 指向树 T 的某个结点,插入结点 c 为 p 的第 i 棵子树。
        操作输出: 插入成功,返回 true;否则,返回 false。
    DeleteChild(&T,p,i)                 //删除结点
        操作功能: p 指向树 T 的某个结点,删除 p 的第 i 棵子树。
        操作输出: 删除成功,返回 true;否则,返回 false。
} ADT Tree
```

5.1.4　树的存储设计

树的存储方法有多种,如顺序存储、链式存储、顺序和链式相结合的存储方法。本节

介绍双亲表示法、孩子链表表示法、双亲孩子表示法、孩子兄弟表示法。

1. 双亲表示

树的双亲表示是用一个数组来存储树，即采用顺序存储方式。数组元素包括两个成员，即数据域和指针（游标）域，结构如图 5-3 所示。数据域用于存储数据元素，指针域用于存储双亲在数组中的位置。存储描述如下。

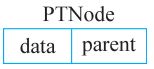
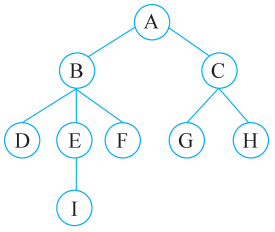


图 5-3 树的双亲表示中数组元素结构示意图

```
template<class DT>
#define MAXNODE          //树的结点个数
struct PTNode            //数组元素类型
{ DT data;               //数据域
  int parent;            //指针域，双亲在数组中的下标
};
PTNode T[MAXNODE];       //树 T
```

图 5-4(a)所示“树 2”的双亲存储示意图如图 5-4(b)所示。



(a) 树 2

PNode t2[]; //树 2 的双亲表示

0	A	−1
1	B	0
2	C	0
3	D	1
4	E	1
5	F	1
6	G	2
7	H	2
8	I	4

(b) 树 2 的双亲存储示意图

图 5-4 树的双亲表示

这种存储结构利用了树中每个结点（除根以外）有唯一双亲的性质。该方法获取双亲很方便，但求结点的孩子需要遍历整个表。

2. 孩子链表表示

树的孩子链表表示是顺序存储和链式存储相结合的一种方式，数据元素信息采用顺序存储，每个结点的所有孩子形成一个链表，双亲结点的指针指向该链表。“树 2”的孩子链表存储示意图如图 5-5 所示。

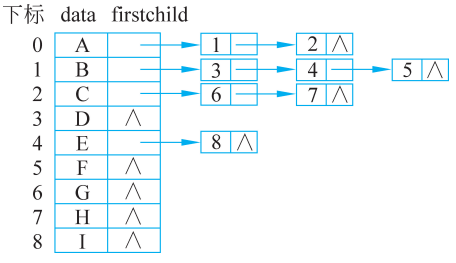


图 5-5 “树 2”的孩子链表存储示意图

存储中涉及两种结点(表头结点和孩子结点),它们的结构如图 5-6 所示。表头结点包含两个域:数据域(data)存储数据元素信息;指针域(firstchild)存储第一个孩子结点的位置信息。孩子结点也包含两个域:孩子位置(child)存储孩子结点信息在表头结点中的位置;指针域(nextchild)存储下一个孩子结点的位置信息。具体定义如下。

```
template<class DT>
struct PNode              //双亲结点
{ DT data;                //数据域
  CTNode * firstchild;    //指针域,指向孩子链的头
};
struct CTNode             //孩子结点
{ int child;              //孩子位置
  CTNode * nextchild;     //指针域,指向下一个孩子结点
};
```

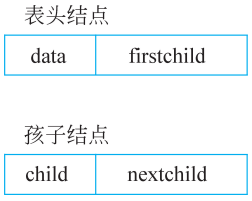


图 5-6　结点结构示意图

在孩子链表存储表示法中,结点的孩子信息很容易得到,但双亲信息需要遍历表才可获得。如果一个问题域中需要同时频繁地获取孩子信息和双亲信息,可以把双亲表示法和孩子表示法结合起来,形成双亲孩子表示法。

3. 双亲孩子表示

树的双亲孩子表示是在孩子链表表示法的表头信息中加入双亲信息形成的,这样既能直接获取双亲信息,又能直接获取孩子信息。带双亲信息的孩子链表存储示意图如图 5-7 所示。

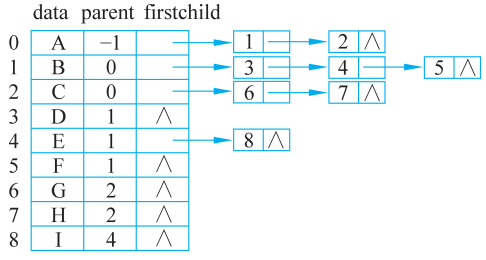


图 5-7　带双亲信息的孩子链表存储示意图

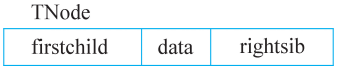


图 5-8　孩子兄弟存储的结点结构

4. 孩子兄弟表示

树的孩子兄弟表示是一种链式存储。每个结点含有一个数据域和两个指针域,结点结构如图 5-8 所示。数据域存储数据元素信息;一个指针域指向结点的第一个孩子;另一个指针域指向结点右边的第一个兄弟。结点定义如下。

```
template<class DT>
struct TNode
{
  DT data;                //数据域
  TNode * firstchild;     //指向第一个孩子
  TNode * rightsib;       //指向第一个右兄弟
};
```

“树 2”的孩子兄弟存储示意图如图 5-9 所示。

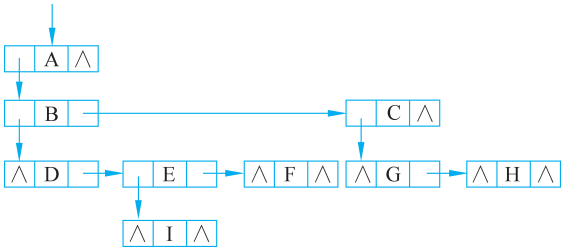


图 5-9 “树 2”的孩子兄弟存储示意图

树的孩子兄弟表示中因为每个结点有两个指针域，所以也称为树的二叉链表表示。

以上介绍了 4 种树的存储设计，但其实还有其他设计形式。具体应用中需要切合问题抽象与问题求解的需要，选择或设计合适的存储方式。


微课视频

5.1.5 树和森林的遍历

遍历是树和森林最基本的操作。树和森林为非线性结构，遍历是结点查找的有效途径。

1. 树的遍历

树的遍历(traverse)**是指从根结点出发，按照某种次序访问树的所有结点，使得每个结点被访问一次且仅被访问一次。**由树的定义可知，一棵树由根结点和 m 棵子树构成，根据先遍历根还是后遍历根，将树的遍历方法分为**先根(序)遍历**和**后根(序)遍历**。树具有层次结构，按层次对树的遍历称为**层序遍历**。关于树的遍历共有上述 3 种方法。

遍历时可以从左往右，也可以从右往左，本书仅考虑从左往右的情况。各种遍历方法如下。

(1) 先根(序)遍历(preorder traversal)。先访问根结点；然后从左往右，依次先根遍历每棵子树。例如，“树 2”的先根遍历序列为 ABDEIFCGH。

(2) 后根(序)遍历(postorder traversal)。先从左往右，依次后根遍历每棵子树；然后访问根结点。例如，“树 2”的后根遍历序列为 DIEFBGHCA。

(3) 层序遍历(level traversal)。层序遍历方法为从上往下、从左往右，依次遍历各结点。例如，“树 2”的层序遍历序列为 ABCDEFGHI。

2. 森林的遍历

森林由一棵以上的树组成。森林遍历的方法有两种：**先序遍历**和**中序遍历**。

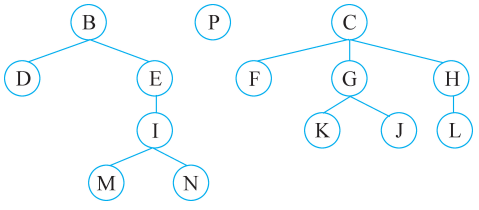


图 5-10 森林 1

(1) 先序遍历。先序遍历的方法是按树的先根遍历方法依次遍历各棵子树。

图 5-10 所示“森林 1”由 3 棵树组成。第 1 棵树的先根遍历序列为 BDEIMN；第 2 棵树的先根遍历序列为 P；第 3 棵树的先根遍历序列为 CFGKJHL。3 个序列连起来，得到森林先序遍历序列为 BDEIMNPCFGKJHL。

（2）中序遍历。中序遍历(inorder traversal)方法是按树的后根遍历方法依次遍历森林的各棵子树，即首先后序遍历第 1 棵树的根的各棵子树，接着访问第 1 棵树的根，然后后序遍历其他树。

例如“森林 1”的中序遍历：第 1 棵树的后根遍历序列为 DMNIEB；第 2 棵树的后根遍历序列为 P；第 3 棵树的后根遍历序列为 FKJGLHC。三者连起来，得到“森林 1”的中序遍历序列为 DMNIEBPFKJGLHC。

森林可以分为 3 部分：第 1 棵树的根Ⅰ、第 1 棵树根的子树Ⅱ、其他树Ⅲ。在中序遍历中，第 1 棵树的根处于遍历的中间位置，因此称为森林的中序遍历。

5.2　二叉树的定义与特性

二叉树与树一样，是一种树形结构，但它是一棵度不超过 2 的有序树。树形结构的术语同样适用于二叉树。

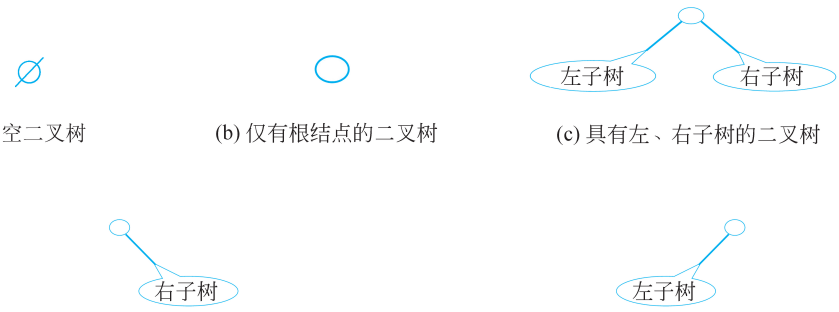
5.2.1　二叉树的定义


微课视频

二叉树(binary tree)是 $n(n\geqslant 0)$个有限结点的集合，$n=0$ 时为空二叉树。非空二叉树 T 满足下列条件。①有且仅有一个没有前驱的数据元素，即根结点。②当 $n>1$ 时，除根结点之外的其余结点被分成两个互不相交的集合 T_1 和 T_2，分别称为 T 的左子树和右子树，且 T_1 和 T_2 本身又均为二叉树；即数据元素 $D=\{\text{root}\}\cup T_1\cup T_2$，且 $T_1\cap T_2=\varnothing$。

二叉树的定义是递归的。对于每棵子树，同样可看成由子树根与子树根的左、右子树组成。

二叉树可以有 5 种基本形态，如图 5-11 所示。



(a) 空二叉树　(b) 仅有根结点的二叉树　(c) 具有左、右子树的二叉树

(d) 没有左子树而只有右子树的二叉树　(e) 只有左子树而没有右子树的二叉树

图 5-11　二叉树的 5 种基本形态

【思考】

（1）一棵有 3 个结点的二叉树有几种形态？

（2）一棵二叉树交换左、右子树后还是同一棵二叉树吗？

（3）一棵有 3 个结点的树有几种形态？

5.2.2 特殊二叉树

满二叉树、完全二叉树、斜树均为特殊形态的二叉树，如图 5-12 所示。

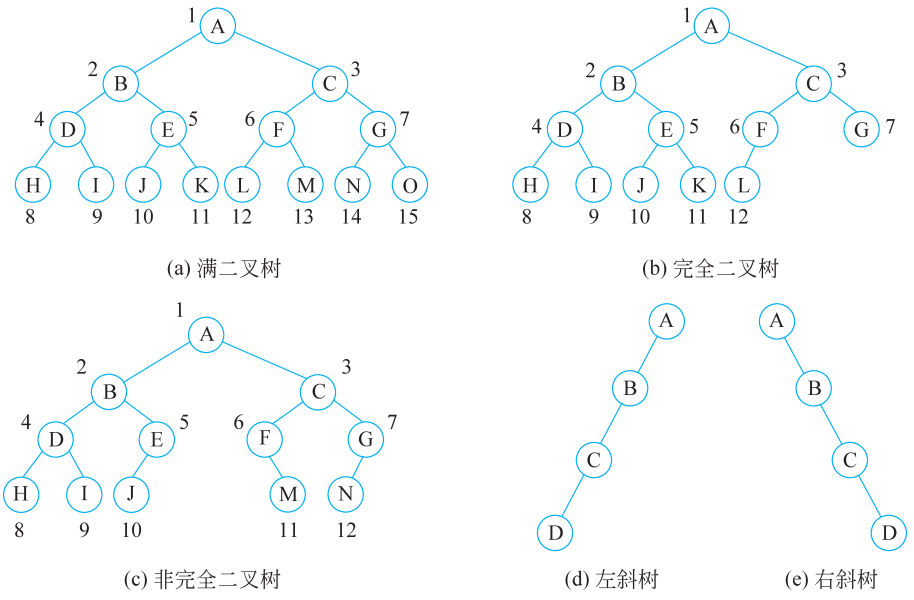


图 5-12 几种特殊二叉树

1. 满二叉树(full binary tree)

满二叉树如图 5-12(a)所示，是同样深度的二叉树中具有最多结点的二叉树。为标识结点个数，图中按层序并从左往右对结点进行了编号。

满二叉树具有下列特性：①第 i 层有 2^{i-1} 个结点；②所有叶结点均在最后一层，深度为 k 的满二叉树有 2^{k-1} 个叶结点；③深度为 k 的满二叉树的结点总数为 2^k-1；④没有度为 1 的结点；⑤具有 n 个结点的满二叉树高度为 $\log_2(n+1)$。

2. 完全二叉树(complete binary tree)

对一棵具有 n 个结点的二叉树按层序编号，如果编号为 $i(1\leqslant i\leqslant n)$ 的结点与同样深度的满二叉树中编号为 i 的结点在二叉树中的位置完全相同，则这棵二叉树称为完全二叉树。也就是说，在完全二叉树中，当树的结点小于同等深度的满二叉树时，是从满二叉树的最底层，从右到左，结点依次减少。例如，图 5-12(b)所示的二叉树是完全二叉树，图 5-12(c)所示的二叉树不是完全二叉树。

完全二叉树具有下列特性：①叶结点只出现在最下两层，且最下层的叶结点都集中在二叉树的左部；②深度为 k 的完全二叉树在 $k-1$ 层上一定是满二叉树；③在结点个数 n 一定的各棵二叉树中，完全二叉树是其中最矮的二叉树；④当结点个数为偶数时，完全二叉树只有一个度为 1 的结点；当结点个数为奇数时，没有度为 1 的结点。

3. 斜树(oblique tree)

树中结点只有左孩子(左斜树)或只有右孩子(右斜树)的二叉树称为斜树，如图 5-12(d)和图 5-12(e)所示。