

第5章

树和二叉树

CHAPTER 5

本章概述	前面讨论的数据结构都属于线性结构,线性结构主要描述具有单一的前驱和后继关系的数据。树结构是一种比线性结构更复杂的数据结构,适合描述具有层次关系的数据,如祖先—后代、上级—下属、整体—部分以及其他类似的关系。树结构在计算机领域有着广泛的应用,例如,在编译程序中用语法树来表示源程序的语法结构,在数据挖掘中用决策树来进行数据分类等。 本章的内容分为树和二叉树两部分。由实际问题引出树结构,介绍树的定义和基本术语,给出树的抽象数据类型定义,讨论树的存储结构;给出二叉树的定义和基本性质,讨论二叉树的存储结构,实现二叉链表存储的二叉树的遍历操作,最后讨论二叉树的经典应用——哈夫曼树。				
教学重点	二叉树的性质;二叉树和树的存储表示;二叉树的遍历及算法实现;树与二叉树的转换关系;哈夫曼树				
教学难点	二叉树的层序遍历算法;二叉树的建立算法;哈夫曼算法				
教学内容和 教学目标	知 识 点	教 学 要 求			
		了解	理解	掌握	熟练掌握
	树的定义和基本术语			√	
	树和二叉树的抽象数据类型定义		√		
	树和二叉树的遍历				√
	树的存储结构			√	
	二叉树的定义				√
	二叉树的基本性质				√
	二叉树的顺序存储结构		√		
	二叉链表				√
	三叉链表	√			
	二叉树遍历的递归算法				√
	二叉树遍历的非递归算法		√		
	二叉树的层序遍历算法			√	
	二叉树的建立算法			√	
	树、森林和二叉树之间的转换			√	
	哈夫曼树及哈夫曼编码			√	

5.1 引 言



电子课件

树结构比较适合描述具有层次关系的数据,如祖先—后代、上级—下属、整体—部分以及其他类似的关系。很多实际问题抽象的数据模型是树结构,请看下面两个例子。

【例 5-1】 文件统计。Windows 操作系统的文件目录结构如图 5-1(a)所示,其中,“/”表示文件夹,括号内的数字表示文件的大小,单位是 KB。假设文件夹本身的大小是 1KB,请统计每个文件和文件夹的大小。

【想法——数据模型】 文件目录结构具有层次特点,每个文件夹均可包含多个子文件夹和文件,将每个文件或文件夹抽象为一个结点,文件夹与文件之间的关系抽象为结点之间的边,从而将文件目录结构抽象为一个树结构,如图 5-1(b)所示。可以对树进行某种遍历,即对树中所有结点进行没有重复没有遗漏的访问,在遍历过程中统计每个文件和文件夹的大小。那么,应该如何存储文件目录结构并在遍历过程中统计大小呢?

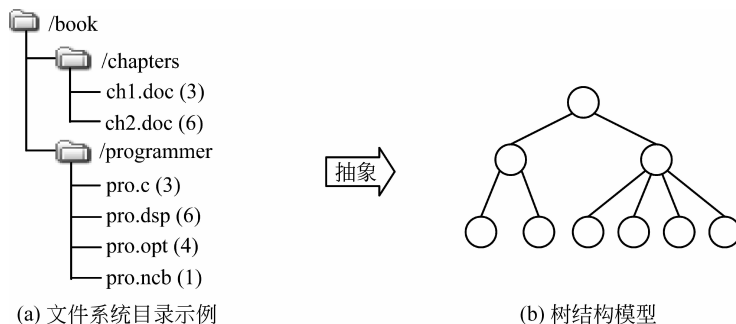


图 5-1 文件系统目录及其数据模型

【例 5-2】 二叉表示树。编译系统在处理算术表达式时,通常将表达式转换为一棵二叉表示树,通过二叉表示树可以判断算术表达式是否存在语法错误。请将给定的算术表达式转换为二叉表示树。

【想法——数据模型】 二叉表示树是对应一个算术表达式的二叉树,并具有以下特点:①叶子结点一定是操作数;②分支结点一定是运算符。将一个算术表达式转化为二叉表示树基于如下规则:

- ① 根据运算符的优先顺序,将表达式结合成(左操作数 运算符 右操作数)的形式;
- ② 由外层括号开始,运算符作为二叉表示树的根结点,左操作数作为根结点的左子树,右操作数作为根结点的右子树;
- ③ 如果某子树对应的操作数为一个表达式,则重复第②步的转换,直到该子树对应的操作数不能再分解。

例如,将表达式 $(A+B) * (C+D * E)$ 结合成 $((A+B) * (C+(D * E)))$,则二叉表示树的根结点是运算符 $*$,左操作数即左子树是 $(A+B)$,右操作数即右子树是 $(C+(D * E))$ 。

E)。对于左子树 $(A+B)$,其根结点是运算符 $+$,左子树是 A ,右子树是 B 。对于右子树 $(C+(D * E))$,其根结点是运算符 $+$,左子树是 C ,右子树是 $(D * E)$ 。依此类推,构造过程如图 5-2 所示。

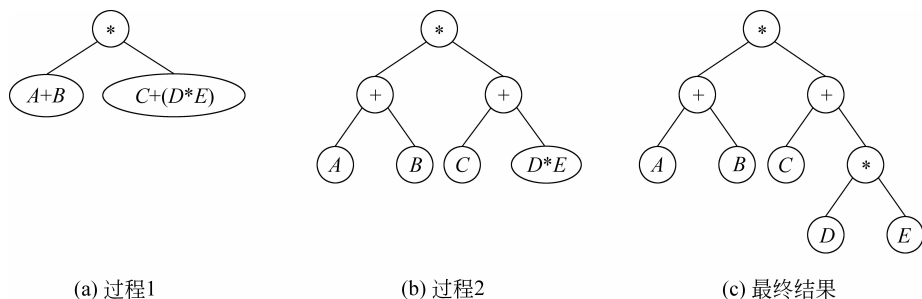


图 5-2 二叉表示树的构造过程

5.2 树的逻辑结构

5.2.1 树的定义和基本术语

1. 树的定义

在树中通常将数据元素称为**结点**(node)。

树(tree)是 $n(n \geq 0)$ 个结点的有限集合^①。当 $n=0$ 时,称为空树;任意一棵非空树满足以下条件:

- (1) 有且仅有一个特定的称为**根**(root)的结点;
- (2) 当 $n > 1$ 时,除根结点之外的其余结点被分成 $m(m > 0)$ 个互不相交的有限集合 $T_1, T_2, \dots, T_m$,其中每个集合又是一棵树^②,并称为这个根结点的**子树**(subtree)。

图 5-3(a)是一棵具有 9 个结点的树, $T = \{A, B, C, D, E, F, G, H, I\}$,结点 A 为树 T 的根结点,除根结点 A 之外的其余结点分为两个不相交的集合: $T_1 = \{B, D, E, F, I\}$ 和 $T_2 = \{C, G, H\}$, T_1 和 T_2 构成了根结点 A 的两棵子树。子树 T_1 的根结点为 B ,其余结点又分为三个不相交的集合: $T_{11} = \{D\}$, $T_{12} = \{E, I\}$ 和 $T_{13} = \{F\}$, T_{11} 、 T_{12} 和 T_{13} 构成了根结点 B 的三棵子树。以此类推,直到每棵子树只有一个根结点为止。

需要强调的是,树中根结点的子树之间是互不相交的。例如,图 5-3(b)由于根结点 A 的两个子树之间存在交集,结点 E 既属于集合 T_1 又属于集合 T_2 ,所以不是树;图 5-3(c)中根结点 A 的两个子树之间也存在交集,边 (B, C) 依附的两个结点属于根结点 A 的两个子集 T_1 和 T_2 ,所以也不是树。



电子课件

^① 从数学角度讲,树结构是图的特例,无回路的连通图就是树,称为**自由树**(free tree),也是有**根树**(rooted tree)。本章讨论的是有序树,所谓有序树是指根结点的子树从左到右是有顺序的。

^② 显然,树的定义是递归的。由于树结构本身具有递归特性,因此,对树的操作通常采用递归方法。

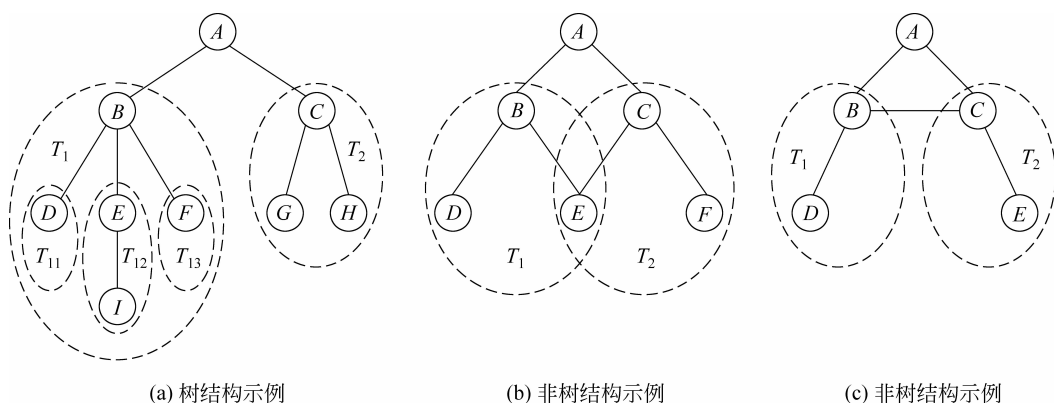


图 5-3 树结构和非树结构的示意图

2. 树的基本术语

(1) 结点的度、树的度

某结点所拥有的子树的个数称为该结点的**度**(degree); 树中各结点度的最大值称为该树的度。如图 5-3(a)所示的树中, 结点 A 的度为 2, 结点 B 的度为 3, 该树的度为 3。

(2) 叶子结点、分支结点

度为 0 的结点称为**叶子结点**(leaf node), 也称为终端结点; 度不为 0 的结点称为**分支结点**(branch node), 也称为非终端结点。如图 5-3(a)所示的树中, 结点 D、I、F、G 和 H 是叶子结点, 其余结点都是分支结点。

(3) 孩子结点、双亲结点、兄弟结点

某结点的子树的根结点称为该结点的**孩子结点**(child node); 反之, 该结点称为其孩子结点的**双亲结点**(parent node); 具有同一个双亲的孩子结点互称为**兄弟结点**(brother node)。如图 5-3(a)所示的树中, 结点 B 是结点 A 的孩子结点, 结点 A 是结点 B 的双亲结点, 结点 B 和 C 互为兄弟结点, 结点 I 没有兄弟结点。

(4) 路径、路径长度

如果树的结点序列 $n_1 n_2 \cdots n_k$ 满足如下关系: 结点 n_i 是结点 n_{i+1} 的双亲 ($1 \leq i < k$), 则 $n_1 n_2 \cdots n_k$ 称为一条由 n_1 至 n_k 的**路径**(path); 路径上经过的边数称为**路径长度**(path length)。显然, 在树中路径是唯一的。如图 5-3(a)所示的树中, 从结点 A 到结点 I 的路径是 ABEI, 路径长度为 3。

(5) 祖先、子孙

如果从结点 x 到结点 y 有一条路径, 那么 x 就称为 y 的**祖先**(ancestor), y 称为 x 的**子孙**(descendant)。显然, 以某结点为根的子树中的任一结点都是该结点的子孙。如图 5-3(a)所示的树中, 结点 A、B、E 均为结点 I 的祖先, 结点 B 的子孙有 D、E、F、I。

(6) 结点的层数、树的深度(高度)、树的宽度

规定根结点的**层数**(level)为 1, 对其余任何结点, 若某结点在第 k 层, 则其孩子结点在第 $k+1$ 层; 树中所有结点的最大层数称为树的**深度**(depth), 也称为树的高度; 树中每一层结点个数的最大值称为树的**宽度**(breadth)。如图 5-3(a)所示的树中, 结点 D 的层数

为3,树的深度为4,树的宽度为5。

5.2.2 树的抽象数据类型定义

树的应用很广泛,在不同的实际应用中,树的基本操作不尽相同。下面给出一个树的抽象数据类型定义的例子,简单起见,基本操作只包含树的遍历。



电子课件

```
ADT Tree
DataModel
    树由一个根结点和若干棵子树构成,树中结点具有层次关系
Operation
    InitTree
        输入: 无
        功能: 初始化一棵树
        输出: 一个空树
    DestroyTree
        输入: 无
        功能: 销毁一棵树
        输出: 释放该树占用的存储空间
    PreOrder
        输入: 无
        功能: 前序遍历树
        输出: 树的前序遍历序列
    PostOrder
        输入: 无
        功能: 后序遍历树
        输出: 树的后序遍历序列
    LeverOrder
        输入: 无
        功能: 层序遍历树
        输出: 树的层序遍历序列
endADT
```

5.2.3 树的遍历操作

树的**遍历**(traverse)是指从根结点出发,按照某种次序访问树中所有结点,使得每个结点被访问一次且仅被访问一次。访问是一种抽象操作,在实际应用中,可以是对结点进行的各种处理,比如输出结点的信息、修改结点的某些数据等,对应到算法上,访问可以是一条简单语句,可以是一个复合语句,也可以是一个模块。不失一般性,在此将访问定义为输出结点的数据信息。树的遍历次序通常有前序(根)遍历和后序(根)遍历两种方式,此外,如果按树的层序依次访问各结点,则可得到另一种遍历次序:层序遍历。

树的前序遍历操作定义为:若树为空,则空操作返回;否则执行以下操作。

- ① 访问根结点;

② 按照从左到右的顺序前序遍历根结点的每一棵子树。

树的后序遍历操作定义为: 若树为空, 则空操作返回; 否则执行以下操作。

① 按照从左到右的顺序后序遍历根结点的每一棵子树;

② 访问根结点。

树的层序遍历也称树的广度遍历, 其操作定义为: 从树的根结点开始, 自上而下逐层遍历, 在同一层中, 按从左到右的顺序对结点逐个访问。

例如, 图 5-3(a) 所示树的前序遍历序列为: $ABDEIFCGH$, 后序遍历序列为: $DEIFBGHCA$, 层序遍历序列为: $ABCDEFGHI$ 。

5.3 树的存储结构

在大量的实际应用中, 人们使用多种存储方法来表示树。无论采用何种存储方法, 都要求存储结构不仅能存储树中各结点的数据信息, 还要表示结点之间的逻辑关系——父子关系。下面介绍几种基本的存储方法。

5.3.1 双亲表示法

由树的定义可知, 除根结点外, 树中每个结点都有且仅有一个双亲结点, 根据这一特性, 可以用一维数组来存储树的各个结点(一般按层序存储), 数组中的一个元素对应树中的一个结点, 数组元素包括树中结点的

data	parent
------	--------

图 5-4 双亲表示法的数组元素

数据信息以及该结点的双亲在数组中的下标。树的这种存储方法称为 **双亲表示法** (parent expression), 数组元素的结构如图 5-4 所示。其中, data

存储树中结点的数据信息; parent 存储该结点的双亲在数组中的下标。数组元素的结构体类型定义如下。

```
template <typename DataType>
struct PNode
{
    DataType data;
    int parent;
};
```

例如, 图 5-3(a) 所示树采用双亲表示法的存储示意图

如图 5-5 所示, 图中 parent 域的值为一 1 表示该结点无双亲, 即该结点是根结点。

5.3.2 孩子表示法

树的 **孩子表示法** (child express) 是一种基于链表的存储方法, 即把每个结点的孩子排列起来, 看成是一个线性表, 且以单链表存储, 称为该结点的孩子链表, 则 n 个结点共有 n 个孩子链表(叶子结点的孩子链表为空



电子课件

下标	data	parent
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

图 5-5 双亲表示法的存储示意图



电子课件

表)。n 个孩子链表共有 n 个头指针,这 n 个头指针又构成了一个线性表,为了便于进行查找操作,可采用顺序存储。最后,将存放 n 个头指针的数组和存放 n 个结点数据信息的数组结合起来,构成孩子链表的表头数组。所以在孩子表示法中,存在两类结点: 孩子结点和表头结点,其结点结构如图 5-6 所示,结点的结构体类型定义如下。

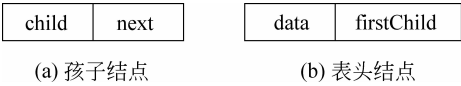


图 5-6 孩子表示法的结点结构

```
struct CTNode //孩子结点
{
    int child;
    CTNode *next;
};
template <typename DataType>
struct CBNode //表头结点
{
    DataType data;
    CTNode *firstChild; //指向孩子链表的头指针
};
```

例如,图 5-7 是图 5-3(a)所示树采用孩子表示法的存储示意图。孩子表示法不仅表示了孩子结点的信息,而且链在同一个孩子链表中的结点具有兄弟关系。

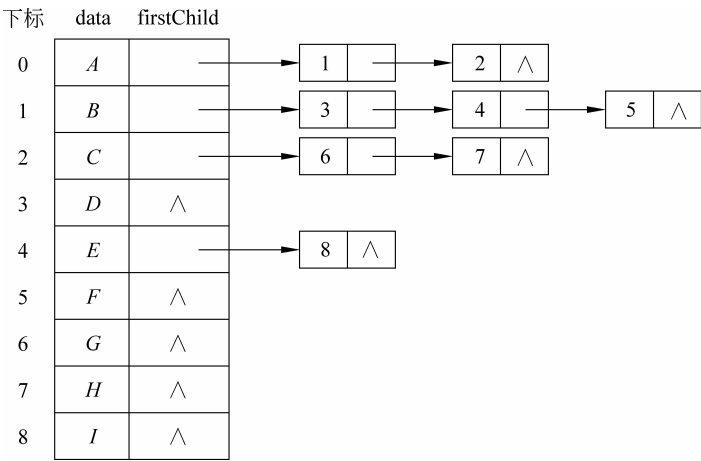


图 5-7 孩子表示法的存储示意图

5.3.3 孩子兄弟表示法

树的孩子兄弟表示法(children brother expression)又称为二叉链表表示法,其方法是链表中的每个结点除数据域外,还设置了两个指针分别指向该结点的第一个孩子和右兄弟,链表的结点结构如图 5-8 所示。其中,data 存储该结点的数据信息;firstchild 存储该结点的第一个孩子结点



电子课件

的存储地址;rightsib 存储该结点的右兄弟结点的存储地址。孩子兄弟表示法的结点定义如下:

firstChild	data	rightSib
------------	------	----------

图 5-8 孩子兄弟表示法的结点结构

```
template <typename DataType>
struct TNode
{
    DataType data;
    TNode<DataType> * firstChild, * rightSib;
};
```

例如,图 5-9 是图 5-3(a)所示树采用孩子兄弟表示法的存储示意图,这种存储方法便于实现树的各种操作。例如,若要访问某结点 x 的第 i 个孩子,只需从该结点的第一个孩子指针找到第 1 个孩子后,沿着孩子结点的右兄弟域连续走 $i-1$ 步,便可找到结点 x 的第 i 个孩子。

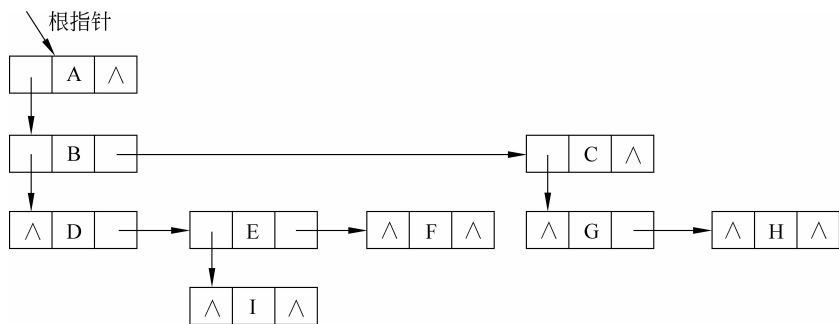


图 5-9 树的孩子兄弟表示法的存储示意图

5.4 二叉树的逻辑结构

树结构有很多变种,二叉树是一种最简单的树结构,而且任何树都可以简单地转换为对应的二叉树。所以,二叉树是本章的重点。

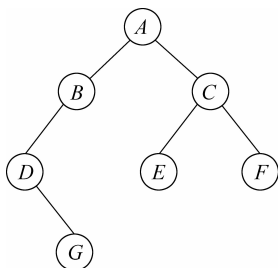


图 5-10 一棵二叉树

5.4.1 二叉树的定义

二叉树(binary tree)是 $n(n \geq 0)$ 个结点的有限集合,该集合或者为空集(称为空二叉树),或者由一个根结点和两棵互不相交的、分别称为根结点的左子树(left subtree)和右子树(right subtree)的二叉树组成。观察图 5-10 所示的一棵二叉树,可以发现二叉树具有如下特点:



电子课件

① 每个结点最多有两棵子树,所以二叉树中不存在度大于2的结点;

② 二叉树的左右子树不能任意颠倒,如果某结点只有一棵子树,一定要指明它是左子树还是右子树。

需要强调的是,二叉树和树是两种不同的树结构。首先,二叉树不是度为2的树。例如图5-11(a)所示是一棵二叉树,但这棵二叉树的度是1;假设图5-11(b)是一棵度为2的树,则结点B是结点A的第一个孩子,结点C是结点A的第二个孩子,并且可以为结点A再增加孩子。其次,树的孩子只有有序的关系,即第1个孩子,第2个孩子,……,第*i*个孩子,但二叉树的孩子却有左右之分,即使二叉树中某结点只有一个孩子,也要区分它是左孩子还是右孩子。例如,假设图5-11(c)所示是二叉树,则它们是两棵不同的二叉树,假设图5-11(d)所示是树,则它们是同一棵树。

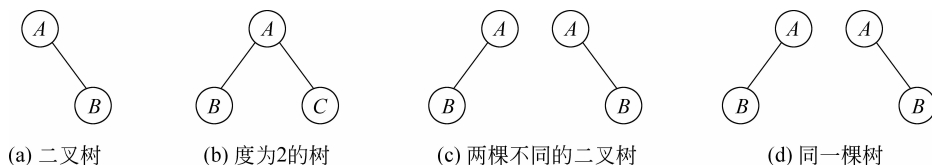


图 5-11 二叉树和树是两种树结构

实际应用中,经常用到如下几种特殊的二叉树。

(1) 斜树

所有结点都只有左子树的二叉树称为**左斜树**(left oblique tree);所有结点都只有右子树的二叉树称为**右斜树**(right oblique tree);左斜树和右斜树统称为**斜树**(oblique tree),如图5-12所示。斜树的特点是:① 每一层只有一个结点;② 斜树的结点数与其深度相同。

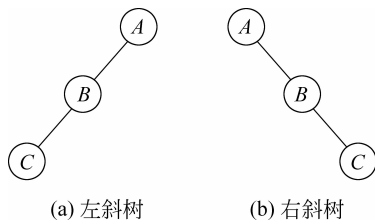


图 5-12 斜树示例

(2) 满二叉树

在一棵二叉树中,如果所有分支结点都存在左子树和右子树,并且所有叶子都在同一层上,这样的二叉树称为**满二叉树**(full binary tree)。图5-13(a)是一棵满二叉树,(b)不是满二叉树^①,因为,虽然所有分支结点都存在左右子树,但叶子不在同一层上。满二叉树的特点是:① 叶子只能出现在最下一层;② 只有度为0和度为2的结点。

(3) 完全二叉树

对一棵具有*n*个结点的二叉树按层序编号,如果编号为*i*($1 \leq i \leq n$)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**(complete binary tree)。显然,一棵满二叉树必定是一棵完全二叉树。完全二叉树的特点是:① 深度为*k*的完全二叉树在*k*-1层是满二叉树;② 叶子结点只能出现在最下两层,且最下层的叶子结点都集中在左侧连续的位置;③ 如果有度为1的结点,只可能

^① 所有分支结点都存在左右子树的二叉树称为**正则二叉树**(regular binary tree),也称为严格二叉树。在正则二叉树中只有度为0和度为2的结点。

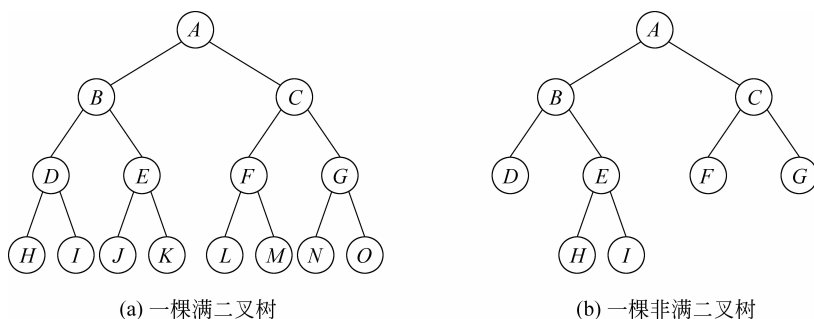


图 5-13 满二叉树和非满二叉树示例

有一个,且该结点只有左孩子。图 5-14 是完全二叉树和非完全二叉树的示例。

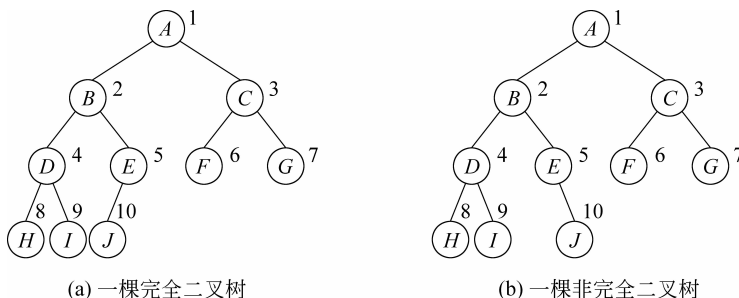


图 5-14 完全二叉树和非完全二叉树示例

5.4.2 二叉树的基本性质

性质 5-1 在一棵二叉树中,如果叶子结点的个数为 n_0 ,度为 2 的结点个数为 n_2 ,则 $n_0 = n_2 + 1$ 。

证明: 设 n 为二叉树的结点总数, n_1 为二叉树中度为 1 的结点个数,因为二叉树中所有结点的度均小于或等于 2,则有:

$$n = n_0 + n_1 + n_2 \quad (5-1)$$

考虑二叉树中的分枝数。除了根结点外,其余结点都有唯一的分枝进入,因此,对于有 n 个结点的二叉树,其分枝数为 $n-1$ 。这些分枝是由度为 1 和度为 2 的结点射出的,一个度为 1 的结点射出一个分枝,一个度为 2 的结点射出两个分枝,所以有:

$$n-1 = n_1 + 2n_2 \quad (5-2)$$

由式(5-1)和式(5-2)可以得到:

$$n_0 = n_2 + 1$$

性质 5-2 二叉树的第 i 层上最多有 2^{i-1} 个结点($i \geq 1$)。

证明: 采用归纳法证明。

当 $i=1$ 时,只有一个根结点,而 $2^{i-1} = 2^0 = 1$,结论显然成立。

假设 $i=k$ 时结论成立,即第 k 层上最多有 2^{k-1} 个结点。

考虑 $i=k+1$ 时的情形。由于第 $k+1$ 层上的结点是第 k 层上结点的孩子,而二叉树



电子课件