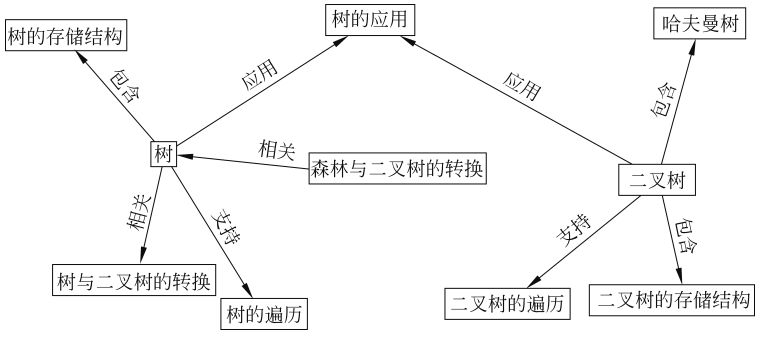
第5章 树

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，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。树结构在计算机领域中有着广泛的应用，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
学习重点	树的基本概念； 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； 树与二叉树之间的转换； 哈夫曼树及应用
知识图谱	

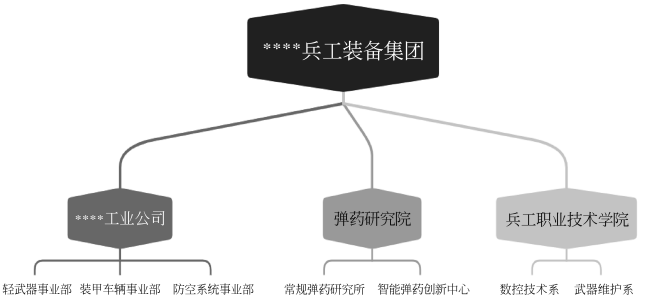
5.1 树的定义和基本术语

工程引例

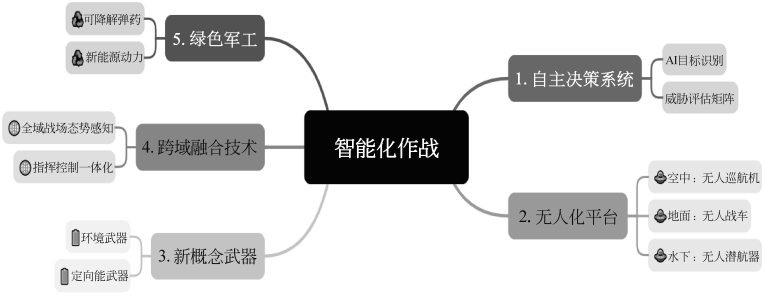
【应用举例】 在计算机的文件系统中，目录结构通常被表示为一棵树。根目录作为树的根结点，文件和子目录作为结点，文件之间的关系通过树结构来表示。

【详细说明】 树结构为文件系统提供了高效的数据组织方式。通过树结构，用户可以方便地浏览和访问文件，系统也能够快速地定位和管理文件。

树结构形象来看，犹如一棵现实中倒长的树，是以分支形式表现的层次结构，树结构中结点间关系是一对多的关系，是非线性结构。树结构特别是二叉树应用非常广泛，在文件系统、编译系统、数据库系统等方面尤为突出。图 5-1(a)所示的组织架构图、图 5-1(b)所示的思维导图均是树结构的例子。



(a) 组织架构图



(b) 思维导图

图 5-1 树的结构

5.1.1 树的定义

树是 $n(n\geqslant 0)$个结点的有限集。当 $n=0$ 时，称为空树。

在任意一棵非空树中：

(1) 有且仅有一个特定结点，该结点没有前驱，称为根(root)结点；

(2) 当 $n>1$ 时，除根结点的其余结点可分为 $m(m>0)$个互不相交的有限集 $T_1,T_2,\cdots,$

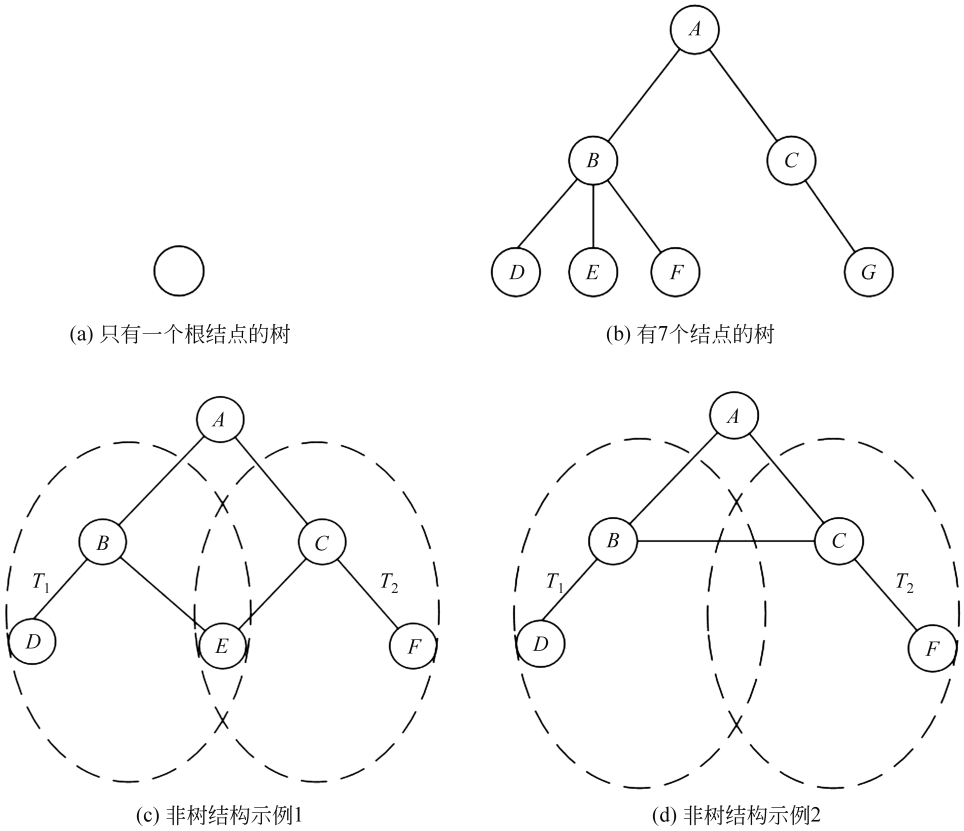
T_m，其中每一个集合本身又是一棵树，称为根的子树(subtree)。显然，以上对树的描述是递归的。

例如，在图 5-2(a)中，是只有一个根结点的树；

图 5-2(b)是有 7 个结点的树，其中 A 是根，其余的结点分成两个互不相交的子集：$T_1=\{B,D,E,F\}$，$T_2=\{C,G\}$；T_1 和 T_2 都是根 A 的子树，且本身也是一棵树；

图 5-2(c)中由于根结点 A 的两个集合之间存在交集，结点 E 既属于集合 T_1 又属于集合 T_2，所以不是树；

图 5-2(d)中根结点 A 的两个集合之间也存在交集，边(B,C)的两个结点分属于根结点 A 的两个集合 T_1 和 T_2，所以也不是树。



(a) 只有一个根结点的树　(b) 有7个结点的树

(c) 非树结构示例1　(d) 非树结构示例2

图 5-2　树的结构

5.1.2　树相关术语

树的结点：树的结点包含一个数据元素和若干指向其子树的分支。

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)；在树中各个结点度的最大值称为该树的度。如图 5-2(b)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。

叶子结点、分支结点：度为 0 的结点称为叶子结点(leaf node)，即没有子树的结点，

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 5-2(b)所示的树中，结点 D、E、F 和 G 是叶子结点，其余结点都是分支结点。除根结点之外，分支结点也称为内部结点。

孩子结点、双亲结点、兄弟结点、祖先、子孙：某结点的子树的根结点称为该结点的孩子结点(children node)；相应地，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 5-2(b)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 G 没有兄弟结点。将这些关系进一步推广，可认为 A 是 D 的祖先。结点的祖先是从根到该结点所经分支上的所有结点，例如，D 的祖先为 A 和 B；反之，以某结点为根的子树中的任一结点都称为该结点的子孙，如 B 的子孙为 D、E 和 F。

结点的层数、树的深度(高度)：规定根结点的层数为 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(level)；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。如图 5-2(b)所示的树中，结点 D 的层数为 3，树的深度为 3。

有序树、无序树：如果一棵树中结点的各子树从左到右是有次序的，即若交换了结点各子树的相对位置，则构成不同的树，称这棵树为有序树(ordered tree)；反之，称为无序树(unordered tree)。在有序树中，最左边子树的根为第一个孩子，最右边子树的根为最后一个孩子。除特殊说明，在数据结构中讨论的树一般都是有序树。

森林(forest)是 $m(m\geqslant 0)$ 棵互不相交的树的集合。对树中每个结点而言，其子树的集合即为森林。由此，也可以用森林和树相互递归的定义来描述树。

就逻辑结构而言，任何一棵树是一个二元组 Tree=(root,F)，其中，root 是数据元素，称作树的根结点；F 是 $m(m\geqslant 0)$ 棵树的森林，$F=(T_1,T_2,\cdots,T_m)$，其中，$T=(r_i,F_i)$ 称作根 root 的第 i 棵子树；当 $m\neq 0$ 时，在树根和其子树森林之间存在下列关系：

$$RF=\{<root,r_i>\mid i=1,2,\cdots,m,m>0\}$$

这个定义将有助于得到森林和树与二叉树之间转换的递归定义。树的应用广泛，在不同的软件系统中树的基本操作集不尽相同。

5.1.3 树的抽象数据类型定义

在 5.1.1 节中对树的定义中加入基本操作就构成了树的抽象数据类型定义。

```
ADT Tree {
    数据对象 D:D 是具有相同特性的数据元素的集合。
    数据关系 R:若 D 为空集,则称为空树;
    若 D 仅含一个数据元素,则 R 为空集,否则 R={H},H 是如下二元关系:
    (1)在 D 中存在唯一的称为根的数据元素 root,它在关系 H 下无前驱;
    (2)若 D-{root}≠∅,则存在 D-{root}的一个划分 D1,D2,…,Dm(m>0),对任意 j≠k(1≤
j,k≤m)有 Dj∩Dk=∅,且对任意的 i(1≤i≤m),唯一存在数据元素 xi∈Di,有<root,xi>∈H;
    (3)对应于 D-{root}的划分,H-{<root,x1>,…,<root,xm>}有唯一的一个划分 H1,
H2,…,Hm(m>0),对任意 j≠k(1≤j,k≤m)有 Hj∩Hk=∅,且对任意 i(1≤1≤m),Hi 是 Di 上的二
元关系,(Di,{Hi})是一棵符合本定义的树,称为根 root 的子树。
    基本操作:
```

```
    InitTree(*T);
    操作结果:构造空树 T
    DestroyTree(*T);
    初始条件:树 T 存在
    操作结果:销毁树 T
    CreateTree(*T, definition);
    初始条件:definition 给出树 T 的定义
    操作结果:按 definition 构造树 T
    ClearTree(*T);
    初始条件:树 T 存在
    操作结果:将树 T 清为空树
    TreeEmpty(T);
    初始条件:树 T 存在
    操作结果:若 T 为空树,则返回 TRUE,否则返回 FALSE
    Root(T);
    初始条件:树 T 存在
    操作结果:返回 T 的根
    InsertChild(*T,*p,i,c);
    初始条件:树 T 存在,p 指向 T 中某个结点,1≤i≤p 所指结点的度+1,非空树 c 与 T 不相交
    操作结果:插入 c 为 T 中 p 所指结点的第 i 棵子树
    DeleteChild(*T,*p.i);
    初始条件:树 T 存在,p 指向 T 中某个结点,1≤i≤p 指结点的度
    操作结果:删除 T 中 p 所指结点的第 i 棵子树
    TraverseTree(T, Visit());
    初始条件:树 T 存在,Visit()是对结点操作的应用函数
    操作结果:按某种次序对 T 的每个结点调用函数 Visit()一次且至多 1 次。一旦 Visit()失
败,则操作失败
}ADT Tree
```

5.2 二 叉 树

二叉树是一种非常重要的数据结构,它在搜索引擎、文件系统、数据库和网络路由等领域都有广泛的应用,学习二叉树的相关理论和算法对于我们理解这些领域的工作原理和技术实现都非常有帮助。另外,二叉树是一种最简单的树结构,特别适合计算机处理,而且任何树都可以简单地转换为二叉树。因此,二叉树是本章学习和研究的重点。

5.2.1 二叉树的定义

二叉树(binary tree)是一种特殊类型的树结构,有如下两个重要的特点:

(1) 任何结点至多有两棵子树(即二叉树中不存在度大于 2 的结点);

(2) 二叉树的子树有左右之分,其次序不能任意颠倒(即二叉树是有序树)。

二叉树具有 5 种基本形态(如图 5-3 所示):①空二叉树;②只有一个根结点;③根结点只有左子树;④根结点只有右子树;⑤根结点既有左子树又有右子树。

5.2.2 特殊形态的二叉树

实际应用中,经常用到如下几种特殊的二叉树。

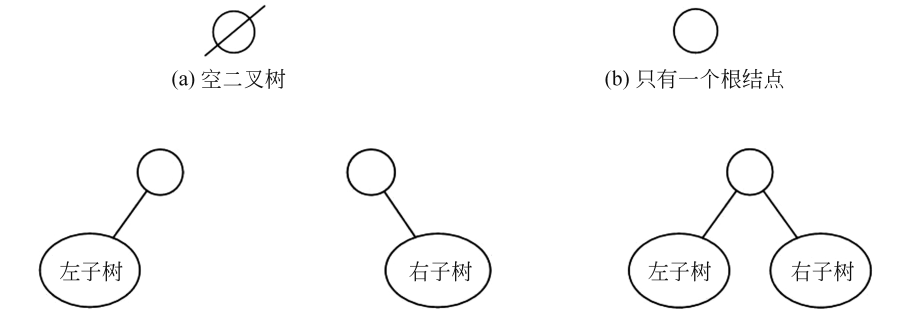


图 5-3　二叉树的 5 种基本形态

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 5-4 所示。

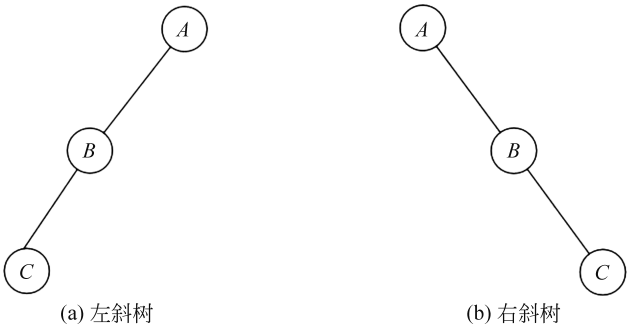


图 5-4　斜树示例

在斜树中,每一层只有一个结点,所以斜树的结点个数与其深度相同。

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 5-5(a)所示是一棵满二叉树,图 5-5(b)所示不是满二叉树,因为,虽然所有分支结点都存在左、右子树,但叶子未在同一层上。

满二叉树的特点如下。

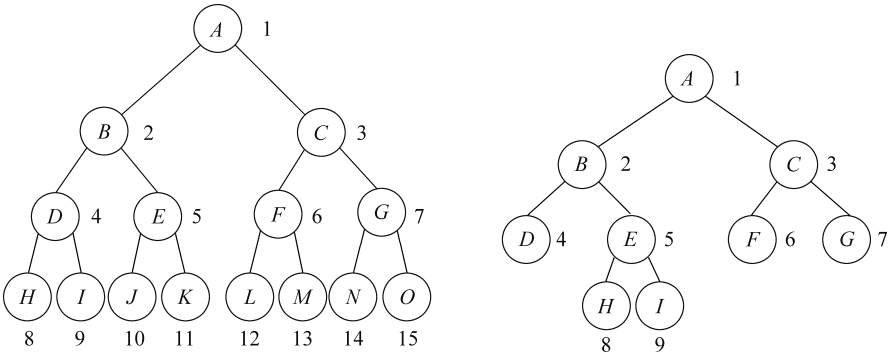
(1) 叶子只能出现在最下一层;

(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 $i(1\leqslant i\leqslant n)$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**(complete binary tree)。显然,一棵满二叉树必定是一棵完全二叉树。

完全二叉树的特点如下。

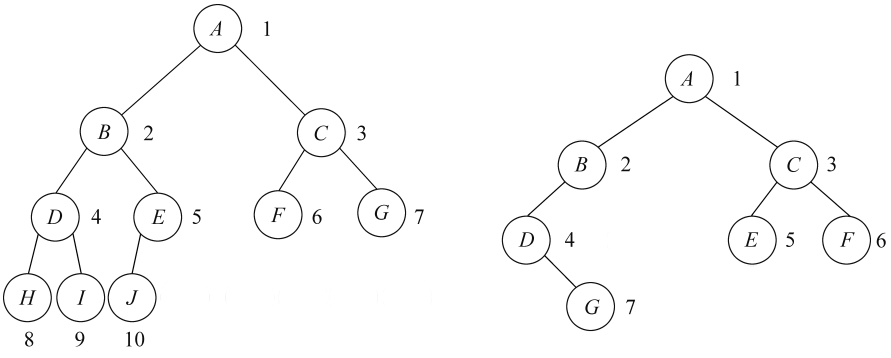


(a) 一棵满二叉树　　(b) 一棵非满二叉树

图 5-5　满二叉树和非满二叉树示例

(1) 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二叉树左侧连续的位置；

(2) 如果有度为 1 的结点，只可能有一个，且该结点只有左孩子。图 5-6 是完全二叉树和非完全二叉树示例。



(a) 一棵完全二叉树　　(b) 一棵非完全二叉树

图 5-6　完全二叉树和非完全二叉树示例

5.2.3　二叉树常用性质

性质 1　在二叉树的第 i 层至多有 2^{i-1} 个结点($i \geqslant 1$)。

下面用数学归纳法证明这一性质。

证明：当 $i=1$ 时，只有根结点，$2^{i-1}=2^0=1$。

假设对所有 j，$1 \leqslant j < i$，命题成立，即第 j 层上至多有 2^{j-1} 个结点。

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，故在第 i 层上的最大结点数为第 $i-1$ 层上的最大结点数的 2 倍，即 $2 \times 2^{i-2}=2^{i-1}$。

证毕。

性质 2　深度为 k 的二叉树至多有 2^k-1 个结点($k \geqslant 1$)。

证明：由性质 1 可见，深度为 k 的二叉树的最大结点数为

$$\sum_{i=1}^{k} \text{第 } i \text{ 层结点的最大个数} = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

性质 3 对任何一棵二叉树 T，如果其叶结点数为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$。

证明：$n = n_0 + n_1 + n_2$，$e = 2n_2 + n_1 = n - 1$（e 为分支数。除了根结点，每个结点对应 1 个分支。）

因此，有

$$2n_2 + n_1 = n_0 + n_1 + n_2 - 1$$

整理得

$$n_2 = n_0 - 1$$

即

$$n_0 = n_2 + 1$$

性质 4 具有 $n(n \geqslant 0)$ 个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$。

证明：设完全二叉树的深度为 h，则根据性质 2 和完全二叉树的定义有

$$2^{h-1} - 1 < n \leqslant 2^h - 1 \quad \text{或} \quad 2^{h-1} \leqslant n < 2^h$$

对上式取对数得

$$h - 1 \leqslant \log_2 n < h$$

又 h 是整数，因此有 $h = \lfloor \log_2 n \rfloor + 1$。

性质 5 如将一棵有 n 个结点的完全二叉树自顶向下，同层自左向右连续为结点编号 $0, 1, \cdots, n-1$，则有：

(1) 若 $i = 0$，则 i 无双亲，即编号为 i 的结点为根结点；若 $i > 0$，则 i 的双亲为 $\left\lfloor \frac{i}{2} \right\rfloor$。

(2) 如果 $2i \leqslant n$，则结点 i 的左孩子的编号为 $2i$；否则结点 i 无左孩子。

(3) 如果 $2i + 1 \leqslant n$，则结点 i 的右孩子的编号为 $2i + 1$；否则结点 i 无右孩子。

证明：在证明过程中，可以从(2)和(3)推出(1)，所以先证明(2)和(3)。采用归纳法证明。

对于 $i = 1$，结点 i 就是根结点，因此无双亲。由完全二叉树的定义，其左孩子是结点 2，若 $2 > n$，即不存在结点 2，此时，结点 i 无左孩子。结点 i 的右孩子也只能是结点 3，若结点 3 不存在，即 $3 > n$，此时结点 i 无右孩子。

对于 $i > 1$，可分为如下两种情况讨论：

① 设第 $j(1 \leqslant j \leqslant \lfloor \log_2 n \rfloor)$ 层第一个结点的编号为 i，由二叉树的性质 2 可知，$i = 2^{j-1}$，则结点 i 的左孩子必定为第 $j+1$ 层的第一个结点，其编号为 $2^j = 2 \times 2^{j-1} = 2i$，如果 $2i > n$，则无左孩子；其右孩子必定为第 $j+1$ 层的第二个结点，编号为 $2^j + 1 = 2i + 1$，若 $2i + 1 > n$，则结点 i 无右孩子，如图 5-7 所示。

② 假设第 $j(1 \leqslant j \leqslant \lfloor \log_2 n \rfloor)$ 层上某个结点编号为 $i(2^{j-1} \leqslant i \leqslant 2^j - 1)$，其左孩子为 $2i$，右孩子为 $2i+1$，则结点 $i+1$ 是结点 i 的右兄弟或堂兄弟（结点 i 不是第 j 层最后一个结点），或结点 $i+1$ 是第 $j+1$ 层的第一个结点（结点 i 是第 j 层最后一个结点），若结点 $i+1$ 有左孩子，则左孩子的编号必定为 $2i + 2 = 2 \times (i+1)$；若它有右孩子，则右孩子的

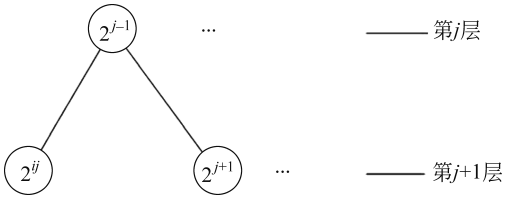
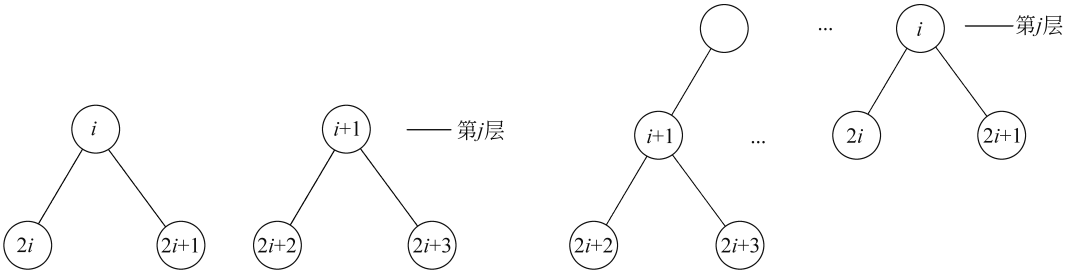


图 5-7　结点 i 是第 j 层第一个结点的情况

编号必定为 $2i+3=2\times(i+1)+1$，如图 5-8 所示。



(a) 结点 i 和 i+1 在同一层上　　(b) 结点 i 和 i+1 不在同一层上

图 5-8　归纳情况的证明

当 $i>1$ 时，如果 i 为左孩子，即 $2\times(i/2)=i$，则 $i/2$ 是 i 的双亲 p；如果 i 为右孩子，则 $i=2p+1$，即结点 i 的双亲 $p=(i-1)/2=\left\lfloor\frac{i}{2}\right\rfloor$。

5.2.4　二叉树的抽象数据类型

下面给出二叉树的抽象数据类型定义，在抽象数据类型定义中仅给出几种简单的基本操作接口，在实际应用中需要根据问题具体定义并实现相应的基本操作。

二叉树的抽象数据类型定义如下：

```
ADT BinaryTree(
    数据对象 D:D 是具有相同特性的数据元素的集合。
    数据关系 R:若 D=∅,则 R=∅,称 BinaryTree 为空二叉树;若 D≠∅,则 R={H},H 是如下二元关系:
    (1)在 D 中存在唯一的称为根的数据元素 root,它在关系 H 下无前驱;
    (2)若 D-{root}≠∅,则存在 D-{root}={Dl,Dr},且 Dl∩Dr=∅;
    (3)若 Dl≠∅,则 Dl 中存在唯一的元素 xl,<root,xl>∈H,且存在 Dl 上的关系 Hl∈H;若 Dr≠∅,则 Dr 中存在唯一的元素 xr,<root,xr>∈H,且存在 Dr 上的关系 Hr∈H;H={<root,xl>,<root,xr>,Hl,Hr};
    (4)(Dl,{Hl})是一棵符合本定义的二叉树,称为根的左子树,(Dr,{Hr})是一棵符合本定义的二叉树,称为根的右子树。
    基本操作:
    InitBiTree(&T);
    操作结果:构造空二叉树 T
    DestroyBiTree(&T);
    初始条件:二叉树 T 存在
    操作结果:销毁二叉树 T
```

CreateBiTree(&T, definition);
初始条件:definition 给出二叉树 T 的定义
操作结果:按 definition 构造二叉树 T
ClearBiTree(&T);
初始条件:二叉树 T 存在
操作结果:将二叉树 T 清为空树
BiTreeEmpty(T);
初始条件:二叉树 T 存在
操作结果:若 T 为空二叉树,则返回 TRUE,否则返回 FALSE
Root(T);
初始条件:二叉树 T 存在
操作结果:返回 T 的根
InsertChild(T,p,LR,c);
初始条件:二叉树 T 存在,p 指向 T 中某个结点,LR 为 0 或 1,非空二叉树 c 与 T 不相交且右子树为空
操作结果:根据 LR 为 0 或 1,插入 c 为 T 中 p 所指结点的左或右子树。p 所指结点的原有左或右子树则成为 c 的右子树
DeleteChild(T,p,LR);
初始条件:二叉树 T 存在,p 指向 T 中某个结点,LR 为 0 或 1
操作结果:根据 LR 为 0 或 1,删除 T 中 p 所指结点的左或右子树
PreOrderTraverse(T, Visit());
初始条件:二叉树 T 存在,Visit()是对结点操作的应用函数
操作结果:先序遍历 T,对每个结点调用函数 Visit()一次且仅一次。一旦 Visit()失败,则操作失败
InOrderTraverse(T, Visit());
初始条件:二叉树 T 存在,Visit()是对结点操作的应用函数
操作结果:中序遍历 T,对每个结点调用函数 Visit()一次且仅一次。一旦 Visit()失败,则操作失败
PostOrderTraverse(T. Visit());
初始条件:二叉树 T 存在,Visit()是对结点操作的应用函数
操作结果:后序遍历 T,对每个结点调用函数 Visit()一次且仅一次。一旦 Visit()失败,则操作失败
LevelOrderTraverse(T, Visit());
初始条件:二叉树 T 存在,Visit()是对结点操作的应用函数
操作结果:层序遍历 T,对每个结点调用函数 Visit()一次且仅一次。一旦 Visit()失败,则操作失败
}ADT BinaryTree

5.2.5 二叉树的存储结构

二叉树的结构是非线性的,每一结点最多可有两个后继。有两种存储结构来存储二叉树,分别是顺序存储结构和链式存储结构。

1. 顺序存储结构

```
//二叉树的顺序存储表示
#define MAX_TREE_SIZE 100                          //二叉树的最大结点数
typedef TElemType SqBiTree[MAX_TREE_SIZE];         //0 号单元存储根结点
SqBiTree bt;
```