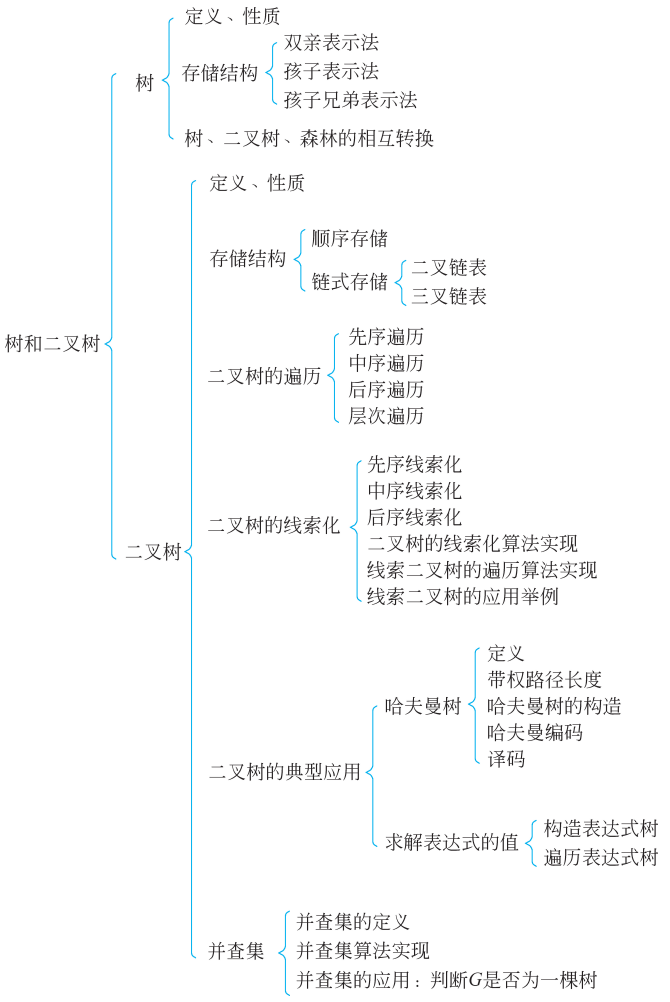
# 第5章 树和二叉树

前面介绍了几种常见的线性结构，本章介绍的树和二叉树与第 6 章介绍的图属于非线性数据结构。线性结构中的每个元素有唯一的前驱元素和唯一的后继元素，而非线性结构中元素间前驱和后继的关系并不具有唯一性。其中，树中结点间的关系是前驱结点唯一而后继结点不唯一，即结点间是一对多的关系；图中结点的前驱结点和后继结点都不唯一，即结点间是多对多的关系。树的应用非常广泛，特别是在需要进行大量数据处理的时候，如在文件系统、编译系统、目录组织等方面，显得更加突出。

**本章学习重难点：**



# 5.1　树

树是一种非线性的数据结构，树中的元素之间具有一对多的层次关系。

## 5.1.1　树的定义

树(tree)是 $n(n\geqslant 0)$个结点的有限集合。其中，当 $n=0$ 时，称为空树；当 $n>0$ 时，称为非空树。树满足以下条件：

(1) 有且只有一个称为根(root)的结点。

(2) 当 $n>1$ 时，其余 $n-1$ 个结点可以划分为 $m$ 个有限集合 $T_1,T_2,\cdots,T_m$，且这 $m$ 个有限集合互不相交，其中 $T_i(1\leqslant i\leqslant m)$又是一棵树，称为根的子树。

图 5-1 给出了一棵树的逻辑结构，它像一棵倒立的树。

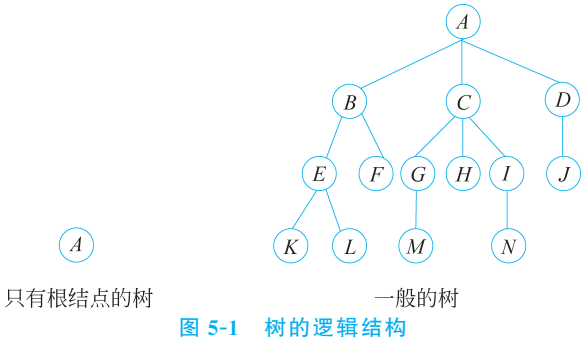



图 5-1　树的逻辑结构

在图 5-1 中，$A$ 为根结点。左边的树只有根结点。右边的树有 14 个结点，除了根结点，其余的 13 个结点分为 3 个不相交的子集：$T_1=\{B,E,F,K,L\}$、$T_2=\{C,G,H,I,M,N\}$和 $T_3=\{D,J\}$。其中，$T_1$、$T_2$ 和 $T_3$ 是根结点 $A$ 的子树，并且它们本身也是一棵树。例如，$T_2$ 的根结点是 $C$，其余的 5 个结点又分为 3 个不相交的子集：$T_{21}=\{G,M\}$、$T_{22}=\{H\}$和 $T_{23}=\{I,N\}$。$T_{21}$、$T_{22}$和 $T_{23}$是 $T_2$ 的子树。$G$ 是 $T_{21}$的根结点，$\{M\}$是 $G$ 的子树；$I$ 是 $T_{23}$的根结点，$\{N\}$是 $I$ 的子树。

表 5-1 是树的基本概念。

表 5-1　树的基本概念

| 概念 | 定　义 | 举　例 |
|---|---|---|
| 树的结点 | 包含一个数据元素及若干指向子树分支的信息 | $A$、$B$、$C$ 等都是结点 |
| 结点的度 | 一个结点拥有子树的个数 | 结点 $C$ 有 3 个子树，度为 3 |
| 叶子结点 | 也称为终端结点，是没有子树的结点，它的度为 0 | $K$、$L$、$F$、$M$、$H$、$N$ 和 $J$ 都是叶子结点 |
| 分支结点 | 也称为非终端结点，是度不为 0 的结点 | $B$、$C$、$D$、$E$ 等都是分支结点 |
| 孩子结点 | 一个结点的子树的根结点 | $B$ 是 $A$ 的孩子结点，$E$ 是 $B$ 的孩子结点，$H$ 是 $C$ 的孩子结点 |
| 双亲结点 | 也称父结点，如果一个结点存在孩子结点，则该结点就称为孩子结点的双亲结点 | $A$ 是 $B$ 的双亲结点，$B$ 是 $E$ 的双亲结点，$I$ 是 $N$ 的双亲结点 |

续表

| 概念 | 定　义 | 举　例 |
| --- | --- | --- |
| 子孙结点 | 在一个根结点的子树中的任何一个结点都称为该根结点的子孙结点 | {$G$,$H$,$I$,$M$,$N$}是 $C$ 的子树,子树中的结点 $G$、$H$、$I$、$M$ 和 $N$ 都是 $C$ 的子孙结点 |
| 祖先结点 | 从根结点开始到达一个结点经过的所有分支结点都称为该结点的祖先结点 | $N$ 的祖先结点为 $A$、$C$ 和 $I$ |
| 兄弟结点 | 一个双亲结点的所有孩子结点之间互为兄弟结点 | $E$ 和 $F$ 是 $B$ 的孩子结点,因此,$E$ 和 $F$ 互为兄弟结点 |
| 树的度 | 树中所有结点的度的最大值 | 结点 $C$ 的度为 3,结点 $A$ 的度为 3,这两个结点的度是树中度最大的结点,因此树的度为 3 |
| 结点的层次 | 从根结点开始,根结点为第一层,根结点的孩子结点为第二层,以此类推,如果某一个结点是第 $L$ 层,则其孩子结点位于第 $L+1$ 层 | 在图 5-1 所示的树中,$A$ 的层次为 1,$B$ 的层次为 2,$G$ 的层次为 3,$M$ 的层次为 4 |
| 树的深度 | 也称为树的高度,树中所有结点的层次最大值 | 图 5-1 所示的树的深度为 4 |
| 有序树 | 如果树中各个子树有先后次序,则称之为有序树 | |
| 无序树 | 如果树中各个子树没有先后次序,则称之为无序树 | |
| 森林 | $m$ 棵互不相交的树构成一个森林。如果把一棵非空树的根结点删除,则该树就变成了一个森林,森林中的树由原来的根结点和各个子树构成。如果给一个森林加上一个根结点,将该森林中的树变成该根结点的子树,则该森林就转换成一棵树 | |

## 5.1.2　树的逻辑表示

树的逻辑表示可分为 4 种:树形表示法、文氏图表示法、广义表表示法和凹入表示法。

(1) 树形表示法。图 5-1 就是树形表示法。树形表示法是最常用的一种逻辑表示,它能直观、形象地表示出树的逻辑结构,能够清晰地反映出树中结点之间的逻辑关系。树中的结点使用圆圈表示,结点间的关系使用直线表示,位于直线上方的结点是双亲结点,位于直线下方的结点是孩子结点。

(2) 文氏图表示法。文氏图是利用数学中的集合描述树的逻辑关系的图。图 5-1 的树采用文氏图表示如图 5-2 所示。

(3) 广义表表示法。可以采用广义表的形式表示树的逻辑结构,广义表的子表表示结点的子树。图 5-1 的树利用广义表表示如下所示:

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

(4) 凹入表示法。图 5-1 的树采用凹入表示法如图 5-3 所示。

在这 4 种树的表示法中,树形表示法最为常用。

## 5.1.3　树的抽象数据类型

树的抽象数据类型定义了树中的数据对象、数据关系及基本操作。树的抽象数据类型描述如表 5-2 所示。

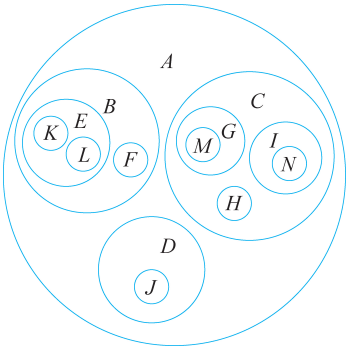



图 5-2　树的文氏图表示法

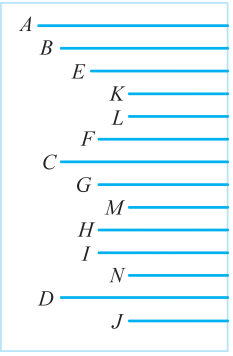



图 5-3　树的凹入表示法

表 5-2　树的抽象数据类型描述

| | | |
|---|---|---|
| 数据对象 | $D$ 是具有相同特性的数据元素的集合 | |
| 数据关系 | 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：<br>(1) 在 $D$ 中存在唯一的称为根的数据元素 root，它在关系 $H$ 下无前驱结点。<br>(2) 若 $D-\{\text{root}\}\neq\varnothing$，则存在 $D-\{\text{root}\}$ 的一个划分 $D_1,D_2,\cdots,D_m(m>0)$，对任意的 $j\neq k(1\leqslant j,k\leqslant m)$ 有 $D_j\cap D_k=\varnothing$，且对任意的 $i(1\leqslant i\leqslant m)$，唯一存在数据元素 $x_i\in D_i$，有 $<\text{root},x_i>\in H$。<br>(3) 对应于 $D-\{\text{root}\}$ 的划分，$H-\{<\text{root},x_1>\},<\text{root},x_2>,\cdots,<\text{root},x_n>\}$ 有唯一的一个划分 $H_1,H_2,\cdots,H_m(m>0)$，对任意的 $j\neq k(1\leqslant j,k\leqslant m)$ 有 $D_j\cap D_k=\varnothing$，且对任意的 $i(1\leqslant i\leqslant m)$，$H_i$ 是 $D_i$ 上的二元关系，$(D_i,\{H_i\})$ 是一棵符合本定义的树，称为 root 的子树 | |
| 基本操作 | InitTree(&T) | 初始条件：树 $T$ 不存在。<br>操作结果：构造空树 $T$ |
| | DestroyTree(&T) | 初始条件：树 $T$ 存在。<br>操作结果：销毁树 $T$ |
| | CreateTree(&T) | 初始条件：树 $T$ 不存在。<br>操作结果：根据给定条件构造树 $T$ |
| | TreeEmpty(T) | 初始条件：树 $T$ 存在。<br>操作结果：若树 $T$ 为空树，则返回 True；否则返回 False |
| | Root(T) | 初始条件：树 $T$ 存在。<br>操作结果：若树 $T$ 非空，则返回树的根结点；否则返回 None |
| | Parent(T,e) | 初始条件：树 $T$ 存在，$e$ 是 $T$ 中的某个结点。<br>操作结果：若 $e$ 不是根结点，则返回该结点的双亲；否则返回 None |
| | FirstChild(T,e) | 初始条件：树 $T$ 存在，$e$ 是 $T$ 中的某个结点。<br>操作结果：若 $e$ 是树 $T$ 的非叶子结点，则返回该结点的第一个孩子结点；否则返回 None |
| | NextSibling(T,e) | 初始条件：树 $T$ 存在，$e$ 是 $T$ 中的某个结点。<br>操作结果：若 $e$ 不是其双亲结点的最后一个孩子结点，则返回它的下一个兄弟结点；否则返回 None |
| | InsertChild(&T,p,i,Child) | 初始条件：树 $T$ 存在，p 指向 $T$ 中的某个结点，非空树 Child 与 $T$ 不相交。<br>操作结果：将非空树 Child 插入到 $T$ 中，使 Child 成为 p 指向的结点的第 $i$ 棵子树 |

续表

| | | |
|---|---|---|
| 基本操作 | DeleteChild(&T,p,i) | 初始条件：树 $T$ 存在，p 指向 $T$ 中的某个结点，$1 \leqslant i \leqslant d$，$d$ 为 p 所指向结点的度。<br>操作结果：将 p 所指向的结点的第 $i$ 棵子树删除。如果删除成功，返回 True，否则返回 False |
| | TraverseTree(T) | 初始条件：树 $T$ 存在。<br>操作结果：按照某种次序对 $T$ 的每个结点访问且仅访问一次 |
| | TreeDepth(T) | 初始条件：树 $T$ 存在。<br>操作结果：若树 $T$ 非空，返回树的深度；否则返回 0 |

## 5.2 二叉树

在深入学习树之前，先介绍一种比较简单的树——二叉树。

### 5.2.1 二叉树的定义

**二叉树**(binary tree)是另一种树结构，它的特点是每个结点最多只有两棵子树。在二叉树中，每个结点的度只可能是 0、1 和 2。每个结点的孩子结点有左右之分，位于左边的孩子结点称为左孩子结点或左孩子，位于右边的孩子结点称为右孩子结点或右孩子。如果 $n=0$，则称该二叉树为空二叉树。

下面给出二叉树的 5 种基本形态，如图 5-4 所示。

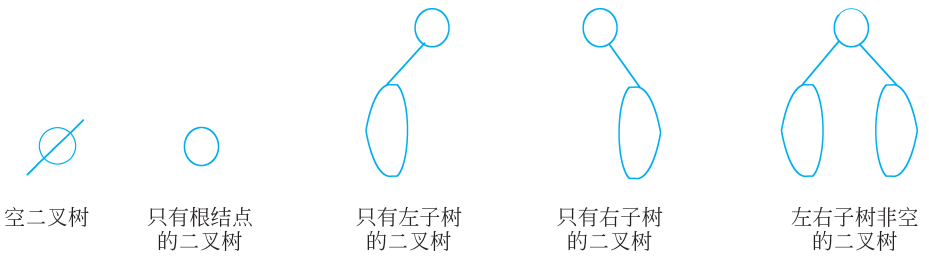



图 5-4 二叉树的 5 种基本形态

一个由 12 个结点构成的二叉树如图 5-5 所示。$F$ 是 $C$ 的左孩子结点，$G$ 是 $C$ 的右孩子结点，$L$ 是 $G$ 的右孩子结点，$G$ 的左孩子结点不存在。

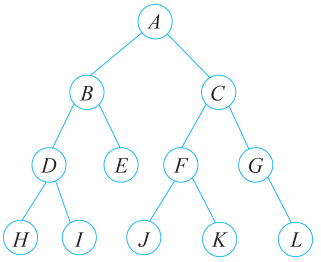



图 5-5 一棵二叉树

对于深度为 $k$ 的二叉树，若结点数为 $2^k-1$，即除了叶子结点外，其他结点都有两个孩子结点，这样的二叉树称为满二叉树。在满二叉树中，每一层的结点都具有最大的结点个数，每个结点的度或者为 2，或者为 0(即叶子结点)，不存在度为 1 的结点。从根结点出发，从上到下，从左到右，依次对每个结点进行连续编号，一棵深度为 4 的满二叉树及其结点编号如图 5-6 所示。

如果一棵二叉树有 $n$ 个结点，并且这 $n$ 个结点的结构与满二叉树的前 $n$ 个结点的结构完全相同，则称这样的二叉树为完全二叉树。完全二叉树及其结点编号如图 5-7 所示。而

图 5-8 所示就不是一棵完全二叉树。

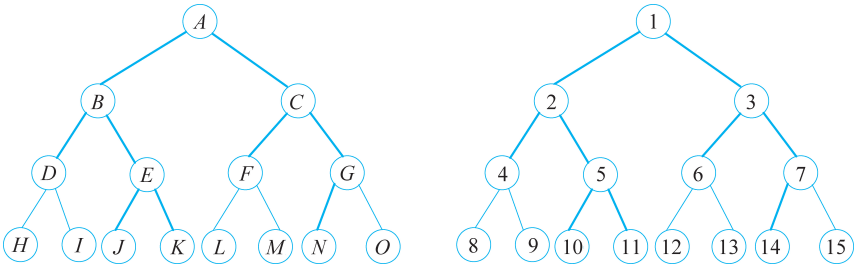



图 5-6　一棵深度为 4 的满二叉树及其结点编号

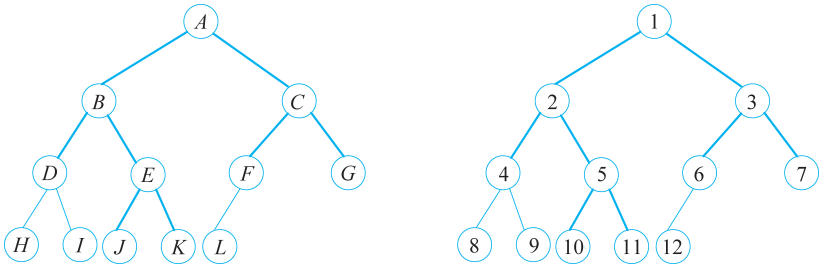



图 5-7　一棵完全二叉树及其结点编号

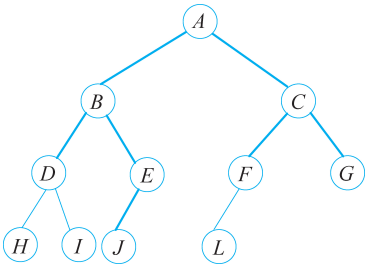



图 5-8　一棵非完全二叉树

由此可以看出，如果二叉树的层数为 $k$，则满二叉树的叶子结点一定在第 $k$ 层，而完全二叉树的叶子结点一定在第 $k$ 层或者第 $k-1$ 层。满二叉树一定是完全二叉树，而完全二叉树却不一定是满二叉树。

## 5.2.2　二叉树的性质

二叉树具有以下重要的性质。

**性质 1**　在二叉树中，第 $m(m\geqslant 1)$ 层上至多有 $2^{m-1}$ 个结点（规定根结点为第一层）。

证明：利用数学归纳法证明。

当 $m=1$ 时，即根结点所在的层次，有 $2^{m-1}=2^{1-1}=2^0=1$，命题成立。

假设当 $m=k$ 时命题成立，即第 $k$ 层至多有 $2^{k-1}$ 个结点。因为在二叉树中每个结点的度最大为 2，则在第 $k+1$ 层，结点的个数最多是第 $k$ 层的 2 倍，即 $2\times 2^{k-1}=2^{k-1+1}=2^k$。因此，当 $m=k+1$ 时，命题成立。

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

证明：第 $i$ 层结点的最多个数 $2^{i-1}$，将深度为 $k$ 的二叉树中每一层结点的最大值相加，就得到二叉树中结点的最大值，因此深度为 $k$ 的二叉树的结点总数至多为

$$\sum_{i=1}^{k}(\text{第 } i \text{ 层的结点最大个数})=\sum_{i=1}^{k}2^{i-1}=2^0+2^1+\cdots+2^{k-1}=\frac{2^0(2^k-1)}{2-1}=2^k-1$$

命题成立。

**性质 3** 对任何一棵二叉树 $T$，如果叶子结点总数为 $n_0$，度为 2 的结点总数为 $n_2$，则有 $n_0=n_2+1$。

证明：假设在二叉树中，结点总数为 $n$，度为 1 的结点总数为 $n_1$。二叉树中结点的总数 $n$ 等于度为 0、度为 1 和度为 2 的结点总数的和，即 $n=n_0+n_1+n_2$。

假设二叉树的分支数为 $Y$。在二叉树中，除了根结点外，每个结点都存在一个进入的分支，所以有 $n=Y+1$。

又因为二叉树的所有分支都是由度为 1 和度为 2 的结点发出，所以分支数 $Y=n_1+2n_2$。故 $n=Y+1=n_1+2n_2+1$。

联合 $n=n_0+n_1+n_2$ 和 $n=n_1+2n_2+1$ 两式，得到 $n_0+n_1+n_2=n_1+2n_2+1$，即 $n_0=n_2+1$。命题成立。

**性质 4** 如果完全二叉树有 $n$ 个结点，则深度为 $\lfloor \log_2 n \rfloor+1$。符号 $\lfloor x \rfloor$ 表示不大于 $x$ 的最大整数，而 $\lceil x \rceil$ 表示不小于 $x$ 的最小整数。

证明：假设具有 $n$ 个结点的完全二叉树的深度为 $k$。$k$ 层完全二叉树的结点个数介于 $k-1$ 层满二叉树与 $k$ 层满二叉树的结点个数之间。根据性质 2，$k-1$ 层满二叉树的结点总数为 $n_1=2^{k-1}-1$，$k$ 层满二叉树的结点总数为 $n_2=2^k-1$。因此有 $n_1<n\leqslant n_2$，即 $n_1+1\leqslant n<n_2+1$，又 $n_1=2^{k-1}-1$ 和 $n_2=2^k-1$，故得到 $2^{k-1}-1\leqslant n<2^k-1$，同时对不等式两边取对数，有 $k-1\leqslant\log_2 n<k$。因为 $k$ 是整数，$k-1$ 也是整数，所以 $k-1=\lfloor \log_2 n \rfloor$，即 $k=\lfloor \log_2 n \rfloor+1$。命题成立。

**性质 5** 如果完全二叉树有 $n$ 个结点，按照从上到下、从左到右的顺序对二叉树中的每个结点从 1 到 $n$ 进行编号，则对于任意结点 $i$ 有以下性质：

(1) 如果 $i=1$，则序号 $i$ 对应的结点就是根结点，该结点没有双亲结点。如果 $i>1$，则序号为 $i$ 的结点的双亲结点的序号为 $\lfloor i/2 \rfloor$。

(2) 如果 $2i>n$，则序号为 $i$ 的结点没有左孩子结点。如果 $2i\leqslant n$，则序号为 $i$ 的结点的左孩子结点的序号为 $2i$。

(3) 如果 $2i+1>n$，则序号为 $i$ 的结点没有右孩子结点。如果 $2i+1\leqslant n$，则序号为 $i$ 的结点的右孩子结点序号为 $2i+1$。

证明：

(1)利用性质(2)和(3)证明。当 $i=1$ 时，该结点一定是根结点，根结点没有双亲结点。当 $i>1$ 时，假设序号为 $m$ 的结点是序号为 $i$ 的结点的双亲结点。如果序号为 $i$ 的结点是序号为 $m$ 的结点的左孩子结点，则根据性质(2)有 $2m=i$，即 $m=i/2$；如果序号为 $i$ 的结点是序号为 $m$ 的结点的右孩子结点，则根据性质(3)有 $2m+1=i$，即 $m=(i-1)/2=i/2-1/2$。综上以上两种情况，当 $i>1$ 时，序号为 $i$ 的结点的双亲结点序号为 $\lfloor i/2 \rfloor$。结论成立。

(2) 利用数学归纳法证明。当 $i=1$ 时，有 $2i=2$。如果 $2>n$，则该二叉树中不存在序号为 2 的结点，也就不存在序号为 $i$ 的左孩子结点；如果 $2\leqslant n$，则该二叉树中存在两个结

点，序号为 2 的结点是序号为 $i$ 的结点的左孩子结点。

假设当 $i=k$ 时，如果 $2k \leqslant n$，序号为 $k$ 的结点的左孩子结点存在且序号为 $2k$；如果 $2k > n$，序号为 $k$ 的结点的左孩子结点不存在。

当 $i=k+1$ 时，在完全二叉树中，如果序号为 $k+1$ 的结点的左孩子结点存在（$2i \leqslant n$），则其左孩子结点的序号为序号为 $k$ 的结点的右孩子结点的序号加 1，即序号为 $k+1$ 的结点的左孩子结点的序号为 $(2k+1)+1=2(k+1)=2i$。因此，当 $2i > n$ 时，序号为 $i$ 的结点的左孩子不存在。结论成立。

（3）同理，利用数学归纳法证明。当 $i=1$ 时，如果 $2i+1=3>n$，则该二叉树中不存在序号为 3 的结点，即序号为 $i$ 的结点的右孩子不存在；如果 $2i+1=3 \leqslant n$，则该二叉树存在序号为 3 的结点，且序号为 3 的结点是序号为 $i$ 的结点的右孩子结点。

假设当 $i=k$ 时，如果 $2k+1 \leqslant n$，序号为 $k$ 的结点的右孩子结点存在且序号为 $2k+1$，当 $2k+1>n$ 时，序号为 $k$ 的结点的右孩子结点不存在。

当 $i=k+1$ 时，在完全二叉树中，如果序号为 $k+1$ 的结点的右孩子结点存在，其序号为 $2i+1 \leqslant n$，则其右孩子结点的序号为序号为 $k$ 的结点的右孩子结点的序号加 2，即序号为 $k+1$ 的结点的右孩子结点的序号为 $(2k+1)+2=2(k+1)+1=2i+1$。因此，当 $2i+1>n$ 时，序号为 $i$ 的结点的右孩子不存在。结论成立。

### 5.2.3　二叉树的抽象数据类型

二叉树的抽象数据类型定义了二叉树中的数据对象、数据关系及基本操作。二叉树的抽象数据类型描述如表 5-3 所示。

表 5-3　二叉树的抽象数据类型描述

| | | |
|---|---|---|
| 数据对象 | $D$ 是具有相同特性的数据元素的集合 | |
| 数据关系 | 若 $D=\varnothing$，则称之为空二叉树。<br>若 $D \neq \varnothing$，则 $R=\{H\}$，$H$ 是如下二元关系：<br>（1）在 $D$ 中存在唯一的称为根的数据元素 root，它在关系 $H$ 下无前驱结点。<br>（2）若 $D-\{\text{root}\} \neq \varnothing$，则存在 $D-\{\text{root}\}=\{D_l, D_r\}$，且 $D_l \cap D_r=\varnothing$。<br>（3）若 $D_l \neq \varnothing$，则 $D_l$ 中存在唯一的元素 $x_l$，$<\text{root}, x_l> \in H$，且存在 $D_l$ 上的关系 $H_l \subset H$；若 $D_r \neq \varnothing$，则 $D_r$ 中存在唯一的元素 $x_r$，$<\text{root}, x_r> \in H$，且存在 $D_r$ 上的关系 $H_r \subset H$；$H=\{<\text{root}, x_l>, <\text{root}, x_r>, H_l, H_r\}$。<br>（4）$(D_l, \{H_l\})$是一棵符合本定义的二叉树，称为根的左子树；$(D_r, \{H_r\})$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) | 初始条件：二叉树 $T$ 不存在。<br>操作结果：构造空二叉树 $T$ |
| | CreateBiTree(&T) | 初始条件：给出了二叉树 $T$ 的定义。<br>操作结果：创建一棵非空的二叉树 $T$ |
| | DestroyBiTree(&T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：销毁二叉树 $T$ |
| | InsertLeftChild(p,c) | 初始条件：二叉树 $c$ 存在且非空 $c$ 的右子树为空<br>操作结果：将 $c$ 插入到 p 所指向的左子树，使 p 所指结点的左子树成为 $c$ 的右子树 |

续表

| | | |
|---|---|---|
| 基本操作 | InsertRightChild(p,c) | 初始条件：二叉树 $c$ 存在且非空，$c$ 的右子树为空。<br>操作结果：将 $c$ 插入到 p 所指向的右子树，使 p 所指结点的右子树成为 $c$ 的右子树 |
| | LeftChild(&T,e) | 初始条件：二叉树 $T$ 存在，$e$ 是 $T$ 中的某个结点。<br>操作结果：若结点 $e$ 存在左孩子结点，则将 $e$ 的左孩子结点返回；否则返回空 |
| | RightChild(&T,e) | 初始条件：二叉树 $T$ 存在，$e$ 是 $T$ 的某个结点。<br>操作结果：若结点 $e$ 存在右孩子结点，则将 $e$ 的右孩子结点返回；否则返回空 |
| | DeleteLeftChild(&T,p) | 初始条件：二叉树 $T$ 存在，p 指向 $T$ 中的某个结点。<br>操作结果：将 p 所指向的结点的左子树删除。如果删除成功，返回 True；否则返回 False |
| | DeleteRightChild(&T,p) | 初始条件：二叉树 $T$ 存在，p 指向 $T$ 中的某个结点。<br>操作结果：将 p 所指向的结点的右子树删除。如果删除成功，返回 True；否则返回 False |
| | PreOrderTraverse(T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：先序遍历二叉树 $T$，即先访问根结点，再访问左子树，最后访问右子树，对二叉树中的每个结点访问且仅访问一次 |
| | InOrderTraverse(T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：中序遍历二叉树 $T$，即先访问左子树，再访问根结点，最后访问右子树，对二叉树中的每个结点访问且仅访问一次 |
| | PostOrderTraverse(T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：后序遍历二叉树 $T$，即先访问左子树，再访问右子树，最后访问根结点，对二叉树中的每个结点访问且仅访问一次 |
| | LevelTraverse(T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：对二叉树 $T$ 进行层次遍历，即按照从上到下、从左到右的顺序依次对二叉树中的每个结点进行访问 |
| | BiTreeDepth(T) | 初始条件：二叉树 $T$ 存在。<br>操作结果：若二叉树 $T$ 非空，返回它的深度；若 $T$ 是空二叉树，返回 0 |

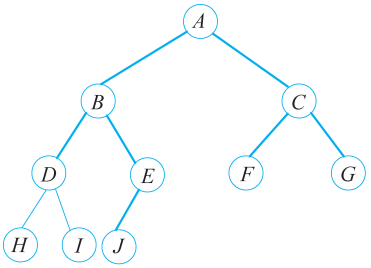
## 5.2.4 二叉树的存储表示

二叉树的存储表示方式有两种：顺序存储结构和链式存储结构。

### 1. 二叉树的顺序存储结构

我们已经知道，完全二叉树中每个结点的序号可以通过公式计算得到，因此，完全二叉树可以按照从上到下、从左到右的顺序依次存储在一维数组或列表中。完全二叉树及其顺序存储结构如图 5-9 所示。

如果按照从上到下、从左到右的顺序把非完全二叉树的结点依次存放在一维数组或列表



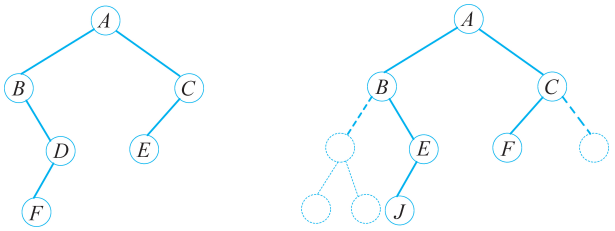

完全二叉树

| 结点序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 数组 | A | B | C | D | E | F | G | H | I | J |

完全二叉树的顺序存储结构

图 5-9　完全二叉树及其顺序存储结构

中。为了能够正确反映二叉树中结点之间的逻辑关系，需要在一维数组（列表）中将二叉树中不存在的结点位置空出，并用∧填充。非完全二叉树及其顺序存储结构如图 5-10 所示。




非完全二叉树　　非完全二叉树对应的完全二叉树形式

| 结点序号 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 数组 | A | B | C | ∧ | D | E | ∧ | ∧ | ∧ | F |

非完全二叉树的顺序存储结构

图 5-10　非完全二叉树及其顺序存储结构

顺序存储结构对于完全二叉树来说是比较适合的，因为采用顺序存储结构能够节省存储单元，并能够利用公式得到每个结点的存储位置。但是，对于非完全二叉树来说，这种存储方式会浪费存储空间。在最坏的情况下，如果每个结点只有右孩子结点，而没有左孩子结点，则需要占用 $2^k-1$ 个存储单元，而实际上该二叉树只有 $k$ 个结点。

### 2. 二叉树的链式存储结构

在二叉树中，每个结点有一个双亲结点和两个孩子结点。从一棵二叉树的根结点开始，通过结点的左右孩子地址就可以找到二叉树的每一个结点。因此二叉树的链式存储结构包括三个域：数据域、左孩子指针域和右孩子指针域。其中，数据域存放结点的值，左孩子指针域指向左孩子结点，右孩子指针域指向右孩子结点。这种链式存储结构称为二叉链表，其结点结构如图 5-11 所示。

| lchild | data | rchild |
|---|---|---|
| 左孩子指针域 | 数据域 | 右孩子指针域 |

图 5-11　二叉链表的结点结构

二叉树的二叉链表存储表示如图 5-12 所示。

有时为了方便找到结点的双亲结点，在二叉链表的存储结构中增加一个指向双亲结点的指针域 parent，这种存储结构称为三叉链表，其结点结构如图 5-13 所示。

通常情况下，二叉树采用二叉链表表示。二叉链表存储结构的类型定义如下：

```
class BiTreeNode():                              #二叉树中的结点
    def __init__(self,data,lchild=None,rchild=None):
        self.data=data                           #二叉树的结点值
        self.lchild=lchild                       #左孩子指针
        self.rchild=rchild                       #右孩子指针
```

定义了二叉树的结点后，为了实现二叉树的插入、删除、遍历和线索化，必须先创建二叉