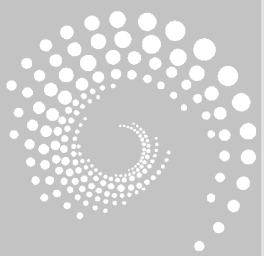
第5章

树和二叉树

CHAPTER 5



树和二叉树属于非线性数据结构，是一种层次结构，结点间的关系是前驱唯一而后继不唯一，即结点间是一对多的关系。树结构在实际生活中有非常广泛的应用，特别是在大数据处理方面，如文件系统、编译系统和目录组织等。

本章重难点

- 树和二叉树的相关概念
- 二叉树的性质
- 二叉树的存储表示
- 二叉树的先序、中序和后序遍历
- 二叉树的线索化
- 树、森林、二叉树的相互转换
- 并查集
- 哈夫曼树和表达式求值

5.1　树和二叉树的基本应用

树和二叉树可用于数据通信过程中的数据编码、算术表达式求值和堆排序等。

5.1.1　数据编码

在数据通信过程中，为了确保数据传输的安全性，需要先将数据编码后再进行传输。数据编码既要尽可能地短，还要使接收方能正确翻译。为此，需要根据给出的数据建立带权路径最短的二叉树——哈夫曼树，实现数据编码。例如，要传输的数据为 success，包含 s、u、c、e 共 4 种字符，分别出现 3、1、2、1 次，将这些次数作为权值构造哈夫曼树，如图 5.1 所示。

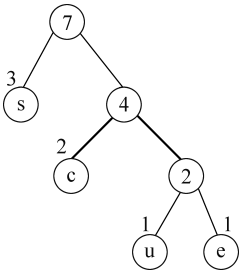


图 5.1　哈夫曼树

根据构造的哈夫曼树，按照左右分支分别为 0、1 进行编码，即可得到 s、u、c、e 的编码，s：0，c：10，u：110，e：111，因此 success 的编码为 0110101011100。

5.1.2　算术表达式求值

对于算术表达式“6＋(7－1)×3＋9/2”，除了利用栈的“后进先出”思想将其转换为后缀表达式再求值，也可以创建二叉树后对其进行后序遍历实现求值。根据 6＋(7－1)×3＋9/2 创建的二叉树如图 5.2 所示。

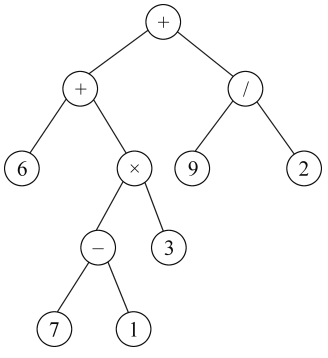


图 5.2　6＋(7－1)×3＋9/2 的二叉树

5.1.3 堆排序

堆排序是一种树形选择排序,其基本思想是将待排序元素序列看成一个树结构(按照完全二叉树的结构进行编码),通过创建堆输出堆顶元素,调整堆后再输出堆顶元素,以此类推,直到所有元素都排好序。例如,待排序元素序列 22、46、37、53、35、51、49、62 的堆排序过程如图 5.3 所示。

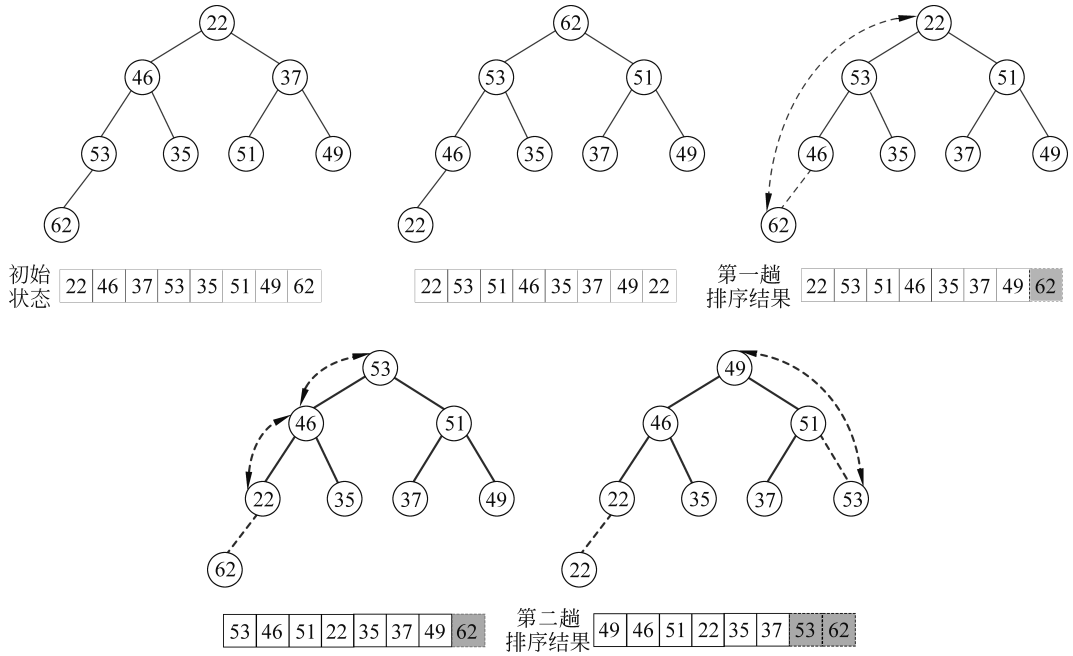


图 5.3 堆排序过程

5.2 树的定义和抽象数据类型

树是一种非线性的数据结构,树中的元素之间是一对多的层次关系。

5.2.1 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) 有且仅有一个称为根(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 个有限集合不相交。其中,$T_i(1\leqslant i\leqslant m)$也是一棵树,称为根的子树(subtree)。

图 5.4 是树的逻辑结构,形似一棵倒立的树。

在图 5.4 中,A 为根结点,左边树只有根结点,右边树有 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 $T_3=$

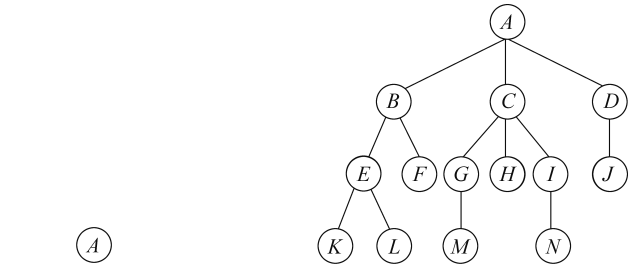


图 5.4　树的逻辑结构

$\{D,J\}$。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 个结点又分为三个不相交的子集：$T_{21}=\{G,M\}$、$T_{22}=\{H\}$和 $T_{23}=\{I,N\}$。其中，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。

下面介绍关于树的一些基本概念。

树的结点：包含一个数据元素及若干指向子树分支的信息。

结点的度：一个结点拥有子树的个数称为结点的度。例如，结点 C 有 3 棵子树，度为 3。

叶子结点：没有子树(即度为零)的结点称为叶子结点，也称终端结点。例如，结点 K 和 L 不存在子树，度为 0，称为叶子结点，F、M、H、N 和 J 也是叶子结点。

分支结点：度不为零的结点称为分支结点，也称非终端结点。例如，B、C、D、E 等都是分支结点。

孩子结点：对于任何一个结点，其子树的根结点为该结点的孩子结点。例如，$\{E,K,L\}$是根结点 B 的子树，而 E 又是这棵子树的根结点，因此 E 是 B 的孩子结点。

双亲结点：如果一个结点存在孩子结点，则该结点称为孩子结点的双亲结点，也称父结点。例如，E 是 B 的孩子结点，则 B 是 E 的双亲结点。

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。例如，$\{G,H,I,M,N\}$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，所经过的所有分支结点都称为该结点的祖先结点。例如，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。例如，E 和 F 是 B 的孩子结点，则 E 和 F 互为兄弟结点。

树的度：树中所有结点的度的最大值。例如，图 5.4 中右边树的度为 3，因为结点 C 的度为 3，该结点的度是树中所有结点最大的度。

树的层次：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。

树的深度：树中所有结点的层次的最大值称为树的深度，也称树的高度。例如，图 5.4 中的右边树的深度为 4。

有序树：如果树中各个子树是有先后次序的，则称该树为有序树。

无序树：如果树中各个子树是没有先后次序，则称该树为无序树。

森林：m 棵互不相交的树构成一个森林。如果将一棵非空树的根结点删除，则该树就

变为一个森林,森林中的树由原来的根结点的各个子树构成。如果将一个森林加上一个根结点,将森林中的树变成该根结点的子树,则该森林就变为一棵树。

5.2.2 树的逻辑表示

树的逻辑表示可分为4种形式: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5.4所示为树形表示法。树形表示法是最常用的一种表示法,它能直观、形象地表示出树的逻辑结构,并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。文氏图表示是利用数学中的集合来图形化描述树的逻辑关系。将图5.4的树用文氏图表示,如图5.5所示。

(3) 广义表表示法。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。将图5.4的树用广义表表示为

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

(4) 凹入表示法。将图5.4的树用凹入法表示,如图5.6所示。

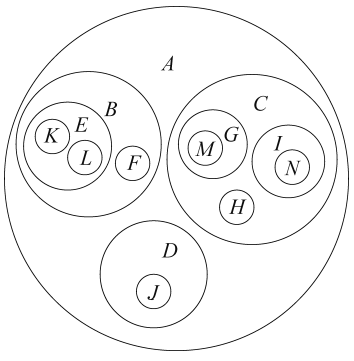


图5.5 树的文氏图表示法

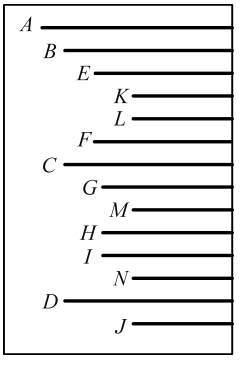


图5.6 树的凹入表示法

5.2.3 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,如表5.1所示。

表5.1 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 不为空集,则 D 与 H 的关系如下。 (1) 在 D 中存在唯一的根数据元素 root,它在关系 H 下无前驱。 (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 $D_j\cap D_k=\varnothing$;对任意的 $i(1\leqslant i\leqslant m)$,唯一存在数据元素 $x_i\in D_i$,有$<\text{root},x_i>\in H$。 (3) 对应 $D-\{\text{root}\}$的划分,$H-\{<\text{root},x_1>\},\cdots,<\text{root},x_n>\}$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 $D_j\cap D_k=\varnothing$;对任意的 $i(1\leqslant i\leqslant m)$,$H_i$ 是 D_i 上的二元关系。 (4) $(D_i,\{H_i\})$是一棵符合该定义的树,称为 root 的子树

续表

基本操作	InitTree(&T)	初始条件：树 T 不存在。 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)	初始条件：树 T 存在。 操作结果：根据给定条件构造树 T
	TreeEmpty(T)	初始条件：树 T 存在。 操作结果：若树 T 为空树，则返回 1；否则，返回 0
	Root(T)	初始条件：树 T 存在。 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。 操作结果：若 e 不是根结点，则返回该结点的双亲；否则，返回空
	FirstChild(T,e)	初始条件：树 T 存在，e 是 T 中的某个结点。 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。 操作结果：若 e 不是其双亲结点的最后一个孩子结点，则返回其下一个兄弟结点；否则，返回 None
	InsertChild(&T,p,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 操作结果：将非空树 Child 插入 T 中，使 Child 成为 p 指向的结点的子树
	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。 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，则返回 1；否则，返回 0
	TraverseTree(T)	初始条件：树 T 存在。 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。 操作结果：若树 T 非空，则返回树的深度；否则，返回 0

5.3　二叉树

在深入学习树之前，需要先认识一种比较简单的树——二叉树。

5.3.1　二叉树的定义

二叉树(binary tree)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 5.7 所示。

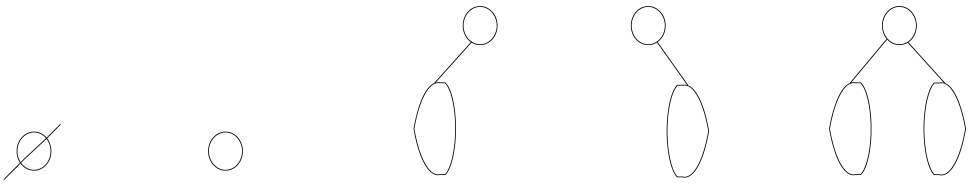


图 5.7 二叉树的 5 种基本形态

一个由 12 个结点构成的二叉树如图 5.8 所示。其中 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。

每层结点都满的二叉树称为满二叉树。在满二叉树中，每层的结点都具有最大的结点个数。图 5.9 为一棵满二叉树，其中每个结点的编号从根结点开始，从上到下、从左到右地依次进行连续编号。在满二叉树中，每个结点的度为 2 或 0(即叶子结点)，不存在度为 1 的结点。

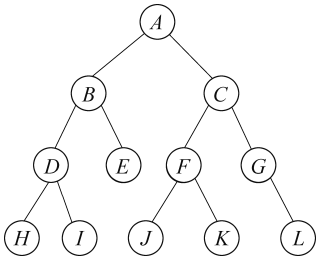


图 5.8 由 12 个结点构成的二叉树

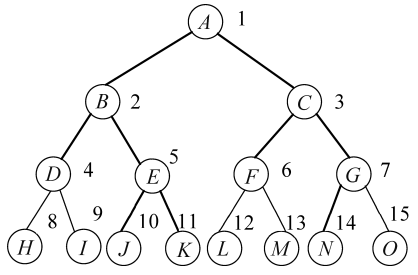


图 5.9 满二叉树

如果一棵二叉树有 n 个结点且其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。完全二叉树及结点对应编号如图 5.10 所示。图 5.11 所示的树是一棵非完全二叉树。

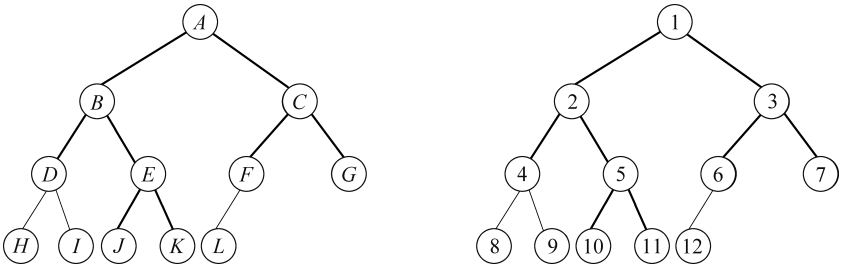


图 5.10 完全二叉树及结点对应编号

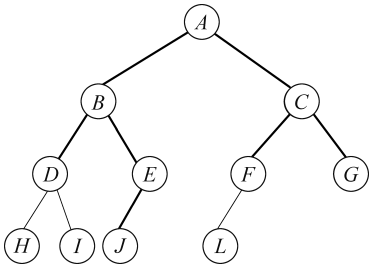


图 5.11 非完全二叉树

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是在第 k 层，而完全二叉树的叶子结点一定在第 k 层或第 $k-1$ 层。满二叉树一定是完全二叉树，而完全二叉树不一定是满二叉树。

5.3.2　二叉树的性质

二叉树具有以下重要性质。

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，在根结点所在的层次，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。因为二叉树中每个结点的度最大为 2，则第 $k+1$ 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 层结点的个数最多为 2^{i-1}，将深度为 k 的二叉树的每层结点的最大值相加，即可得到二叉树中结点个数的最大值，因此深度为 k 的二叉树的结点总数最多为

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

由此可得，命题成立。

性质 3　对于任何一棵二叉树 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，则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 $Y=n_1+2\times n_2$。因此，$n=Y+1=n_1+2\times n_2+1$。

联合 $n=n_0+n_1+n_2$ 和 $n=n_1+2\times n_2+1$ 两式，得到 $n_0+n_1+n_2=n_1+2\times n_2+1$，即 $n_0=n_2+1$。由此可知，命题成立。

性质 4　如果完全二叉树有 n 个结点，则其深度为$\lfloor\log_2 n\rfloor+1$。其中，符号$\lfloor x\rfloor$表示不大于 x 的最大整数。

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，$n_1<n\leqslant n_2$，即 $n_1+1\leqslant n<n_2+1$，又因 $n_1=2^{k-1}-1$ 和 $n_2=2^k-1$，则 $2^{k-1}-1\leqslant n<2^k-1$。同时对不等式两边取对数，可得 $k-1\leqslant\log_2 n<k$。因为 k 是整数且 $k-1$ 也是整数，所以 $k-1=\lfloor\log_2 n\rfloor$，即 $k=\lfloor\log_2 n\rfloor+1$。由此可知，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进行编号(1～n)，则对于任意结点 i 有以下性质：

性质 5.1　如果 $i=1$，则序号 i 对应的结点为根结点，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$\lfloor i/2 \rfloor$。

性质 5.2 如果 $2\times i>n$,则序号为 i 的结点没有左孩子结点,如果 $2\times i\leqslant n$,则序号为 i 的结点的左孩子结点的序号为 $2\times i$。

性质 5.3 如果 $2\times i+1>n$,则序号为 i 的结点没有右孩子结点,如果 $2\times i+1\leqslant n$,则序号为 i 的结点的右孩子结点序号为 $2\times i+1$。

证明:(1)利用性质 5.2 和性质 5.3 证明性质 5.1。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 5.2 有 $2\times m=i$,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 5.3 有 $2\times m+1=i$,即 $m=(i-1)/2=i/2-1/2$。综合以上两种情况,当 $i>1$ 时,序号为 i 的结点的双亲结点序号为$\lfloor i/2 \rfloor$。由此可知,性质 5.1 结论成立。

(2) 利用数学归纳法证明。当 $i=1$ 时,有 $2\times i=2$,如果 $2>n$,则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 2 是序号为 i 的结点的左孩子结点的序号。

假设当序号 $i=k$ 且 $2\times k\leqslant n$ 时,序号为 k 的结点的左孩子结点存在且序号为 $2\times k$;当 $2\times k>n$ 时,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($2\times i\leqslant n$),则其左孩子结点的序号为 k 的结点的右孩子结点序号加 1,即序号为 $k+1$ 的结点的左孩子结点序号为$(2\times k+1)+1=2\times(k+1)=2\times i$。因此,当 $2\times i>n$ 时,序号为 i 的结点的左孩子不存在。由此可知,性质 5.2 结论成立。

(3) 利用数学归纳法证明。当 $i=1$ 时,如果 $2\times i+1=3>n$,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 $2\times i+1=3\leqslant n$,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 i 的结点的右孩子结点。

假设当序号 $i=k$ 且 $2\times k+1\leqslant n$ 时,序号为 k 的结点的右孩子结点存在且序号为 $2\times k+1$;当 $2\times k+1>n$ 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($2\times i+1\leqslant n$),则其右孩子结点的序号为 k 的结点的右孩子结点序号加 2,即序号为 $k+1$ 的结点的右孩子结点序号为$(2\times k+1)+2=2\times(k+1)+1=2\times i+1$。因此,当 $2\times i+1>n$ 时,序号为 i 的结点的右孩子不存在。由此可知,性质 5.3 结论成立。

5.3.3 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,如表 5.2 所示。

表 5.2 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 $D=\varnothing$,则称为空二叉树。 若 $D\neq\varnothing$,则 D,H 的关系如下。 (1) 在 D 中存在唯一的根数据元素 root,它在关系 H 下无前驱。 (2) 若 $D-\{\text{root}\}\neq\varnothing$,则存在 $D-\{\text{root}\}=\{D_l,D_r\}$,且 $D_l\cap D_r=\varnothing$。

续表

数据关系	(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 $H_r \subset H$；其中，$H=\{<\text{root},x_l>,<\text{root},x_r>,H_l,H_r\}$。 (4) $(D_l,\{H_l\})$是一棵符合该定义的二叉树，称为根的左子树；$(D_r,\{H_r\})$是一棵符合该定义的二叉树，称为根的右子树	
基本操作	InitBiTree(&T)	初始条件：二叉树 T 不存在。 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。 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。 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。 操作结果：将 c 插入 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
	InsertRightChild(p,c)	初始条件：二叉树 c 存在且非空。 操作结果：将 c 插入 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。 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
	RigthChild(&T,e)	初始条件：二叉树 T 存在，e 是 T 的某个结点。 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。 操作结果：将 p 所指向的结点的左子树删除。如果删除成功，则返回 1；否则，返回 0
	DeleteRightChild(&T,p)	初始条件：二叉树 T 存在，p 指向 T 中的某个结点。 操作结果：将 p 所指向的结点的右子树删除。如果删除成功，则返回 1；否则，返回 0
	PreOrderTraverse(T)	初始条件：二叉树 T 存在。 操作结果：先序遍历二叉树 T，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进行访问且仅访问一次
	InOrderTraverse(T)	初始条件：二叉树 T 存在。 操作结果：中序遍历二叉树 T，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进行访问且仅访问一次
	PostOrderTraverse(T)	初始条件：二叉树 T 存在。 操作结果：后序遍历二叉树 T，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进行访问且仅访问一次
	LevelTraverse(T)	初始条件：二叉树 T 存在。 操作结果：对二叉树进行层次遍历，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。 操作结果：若二叉树非空，则返回二叉树的深度；若二叉树为空，则返回 0