第5章

数 据 思 维

在我们生活的世界中，万事万物每天都在不断发生变化，而这种事物的变化即对应我们获得的信息，这些信息往往需要通过数据作为载体，将它们记录下来，然后利用计算机将数据通过二进制编码进行存储和处理。那么这些数据是如何在计算机中进行存储的？

本章首先讲述数据结构，数据在计算机中通常包含哪些类型的存储方式；接着分析如何将客观事物抽象成信息世界的实体，如何用概念模型来表示实体及其之间的联系；最后介绍大数据的概念及其应用、数据挖掘的概念及其应用。

5.1 导　　学

本章结构导图如图 5-0 所示。

图 5-0　第 5 章结构导图

5.2 数据的表示

5.2.1 数据结构

对数据进行处理时，需要将大量数据存放在计算机内，那么大量数据在计算机内是如何存储的呢？下面首先来介绍什么是数据结构。

1. 数据结构的基本概念

数据结构是计算机存储、组织数据的方式，是相互之间存在一种或多种特定关系的数据元素的集合。数据结构主要研究以下 3 个方面的内容。

(1) 数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构。

(2) 对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构。

(3) 对各种数据结构进行的运算。

2. 逻辑结构和存储结构

数据的逻辑结构是对数据元素之间的逻辑关系的描述，它可以用一个数据元素的集合和定义在此集合中的若干关系来表示。数据的逻辑结构有两个要素：一是数据元素的集合，通常记为 D；二是 D 上的关系，它反映了数据元素之间的前后件关系，通常记为 R。一个数据结构可以表示成 $B=(D,R)$，其中，B 表示数据结构。为了反映 D 中各数据元素之间的前后件关系，一般用二元组来表示。

例如，如果把一年四季看作一个数据结构，则可表示成 $B=(D,R)$。其中，

$$D=\{\text{春季},\text{夏季},\text{秋季},\text{冬季}\}$$

$$R=\{(\text{春季},\text{夏季}),(\text{夏季},\text{秋季}),(\text{秋季},\text{冬季})\}$$

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构(也称为数据的物理结构)。

由于数据元素在计算机存储空间中的位置关系可能与逻辑关系不同，因此，为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系(即前后件关系)，在数据的存储结构中，不仅要存放各数据元素的信息，还需要存放各数据元素之间的前后件关系的信息。

一种数据的逻辑结构根据需要可以表示成多种存储结构，常用的存储结构有顺序、链式等存储结构。

顺序存储方式主要用于线性的数据结构，它把逻辑上相邻的数据元素存储在物理上相邻的存储单元里，节点之间的关系由存储单元的邻接关系来体现。如线性表(K_1，K_2，…，K_{10})，假定每个节点占一个存储单元，K_1 放在 10 号单元中，则顺序存储的实现如图 5-1(a)所示。链式存储结构就是在每个节点中至少包含一个指针域，用指针来体现数据元素之间逻辑上的联系，如图 5-1(b)所示。

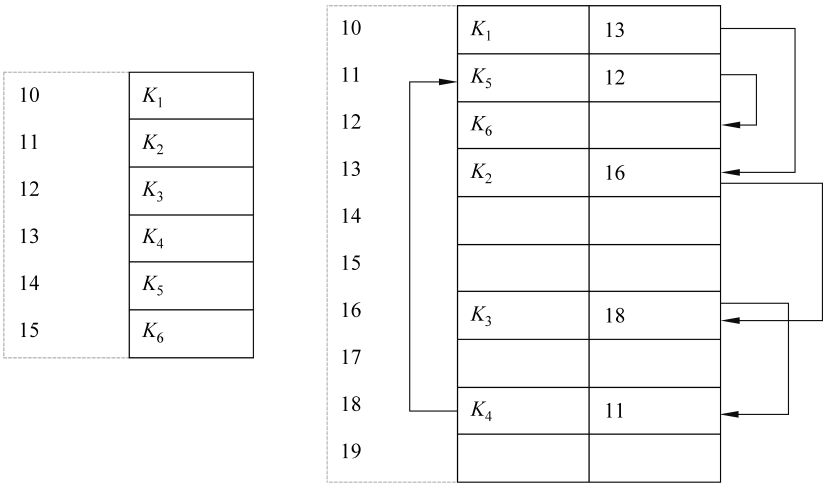

图 5-1　线性表

3. 线性结构与非线性结构

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

线性结构满足：①有且仅有一个根节点；②每一个节点最多有一个直接前趋节点，也最多有一个直接后继节点；③在一个线性结构中插入或删除任何一个节点后，还是线性结构。线性结构元素之间是一对一的联系。典型的线性结构有线性表、栈、队列、字符串等。

如果一个数据结构不是线性结构，则称之为非线性结构。数组、广义表、树和图等数据结构都是非线性结构。

5.2.2　线性表

1. 线性表的概念

线性表是由 $n(n>=0)$ 个数据元素组成的一个有限序列，如图 5-2 所示，表中的每一个数据元素，除了第一个（首元素）外，有且只有一个前趋，除了最后一个（尾元素）外，有且仅有一个后继。即线性表或是一个空表，或可表示为 $(a_1,a_2,\cdots,a_i,\cdots,a_n)$，其中，$a_i$ 是性质相同的数据元素，也称为线性表中的一个节点。线性表的长度，即表中的元素个数 n，当 $n=0$ 时称为空表。例如，英文字母表就是一个长度为 26 的线性表。

图 5-2　线性表

2. 线性表的顺序存储结构(顺序表)

线性表的顺序存储结构的基本特点如下。

(1) 线性表中所有元素所占的存储空间是连续的(一般用数组实现)。

(2) 线性表中每个元素在存储空间中是按逻辑顺序存放的,即用物理上的相邻关系来体现逻辑上的相邻关系。

线性表的随机存取地址计算公式为:

$$\mathrm{ADD}(a_i)=\mathrm{ADD}(a_1)+(i-1)\times k$$

这里 $\mathrm{ADD}(a_i)$是第 i 个元素的地址,k 是每个元素占用空间字节数。

如在顺序表中存储数据(20,34,52,43,74,30),每个数据元素占 2 个存储单元,第 1 个数据元素的存储地址是 30,则第 4 个数据元素 43 的存储地址是 36。

3. 线性表的插入运算

在长度为 n 的线性表$(a_1,a_2,\cdots,a_i,\cdots,a_n)$的第 i 个元素 a_i 之前插入一个新元素 x 后得到的长度为 $n+1$ 的线性表为$(a_1,a_2,\cdots,x,a_i,\cdots,a_n)$。

实现方法:要在第 $i(1\leqslant i\leqslant n)$个元素之前插入一个新元素 x,首先要从最后一个(即第 n 个)元素开始,直到第 i 个元素之间的 $n-i+1$ 个元素依次向后移动一个位置,移动结束后,在第 i 个位置写入新元素 x。插入结束后,表长度增加 1。

插入操作算法的时间复杂度分析:如果插入的位置在第 n 个元素之前,则只需将第 n 个元素后移,这是最好的情况;如果插入的位置在第 1 个元素之前,则 n 个元素都要后移,这是最坏的情况;在等概率情况下(即在任何位置插入的概率相同)平均移动元素个数为 $n/2$。

4. 线性表的删除运算

在长度为 n 的线性表$(a_1,a_2,\cdots,a_i,\cdots,a_n)$中删除第 i 个元素 a_i 后,变为长度为 $n-1$的线性表$(a_1,a_2,\cdots,a_{i-1},a_{i+1}\cdots,a_n)$。

实现方法:要删除第 $i(1\leqslant i<n)$个元素,首先要从第 $i+1$ 个元素开始,直到最后一个(即第 n 个)元素之间的 $n-i$ 个元素依次向前移动一个位置。删除结束后,表长度减 1。

删除操作算法的时间复杂度分析:与插入操作类似,平均情况下需要移动表中一半的元素。两种操作的算法时间复杂度都可记作 $O(n)$。

例如,图 5-3(a)给出一个存储空间为 8,长度为 6 的线性表,进行如下操作。

(1) 在图 5-3(a)中第 4 个元素之前插入一个元素 5,则需要将第 4 个及其后的元素往后移动 1 位,此时线性表的长度变为 7,如图 5-3(b)所示。

(2) 将图 5-3(a)中第 3 个元素 52 删除,则需要将第 4 个及其后的元素往前移动 1 位,此时线性表的长度变为 5,如图 5-3(c)所示。

5. 线性表顺序存储结构的适用场合

线性表的顺序存储结构对于小线性表或者表中元素不常变动的线性表来说是合适

20	34	52	43	74	30		

(a) 原线性表

20	34	52	5	43	74	30	

(b) 插入元素 5 后的线性表

20	34	43	74	30			

(c) 删除元素 52 后的线性表

图 5-3　线性表的插入与删除运算

的，因为顺序存储结构比较简单且易于随机读取；但这种顺序存储方式对于元素需要变动的大线性表就不太适合，因为插入和删除操作的效率比较低。

5.2.3　栈和队列

本节主要讲述栈和队列的基本概念、特点及其运算；同时介绍循环队列及其运算。

1. 栈的定义及其基本运算

1）栈的基本概念

栈(Stack)是一种特殊的线性表，是限定只在一端进行插入与删除的线性表。

在栈中，一端是封闭的，既不允许进行插入元素，也不允许删除元素；另一端是开口的，允许插入和删除元素。通常称插入、删除的这一端为栈顶，另一端为栈底。当表中没有元素时称为空栈。栈顶元素总是最后被插入的元素，从而也是最先被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素。

栈是按照“先进后出”或“后进先出”的原则组织数据的。例如，枪械的子弹匣就可以用来形象地表示栈结构。子弹匣的一端是完全封闭的，最后被压入弹匣的子弹总是最先被弹出，而最先被压入的子弹最后才能被弹出。

2）栈的顺序存储及其运算

栈的基本运算有 3 种：入栈、退栈与读栈顶元素，如图 5-4 所示。

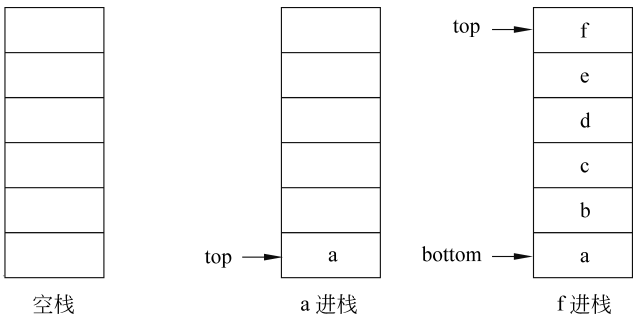

图 5-4　栈的运算

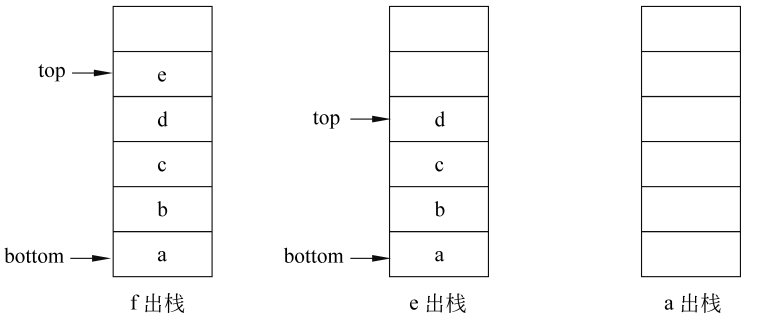

图 5-4 （续）

（1）入栈运算：在栈顶位置插入一个新元素。具体地，将栈顶指针加 1，将新元素插入到栈顶指针指向的位置。当栈顶指针指向存储空间的最后一个位置时，表明栈空间已满，不能再进行入栈操作，此时称为栈的“上溢”。

（2）退栈运算：取出栈顶元素并赋给一个指定的变量。具体地，将栈顶元素赋给一个指定的变量，然后将栈顶指针减 1。

（3）读栈顶元素：将栈顶元素赋给一个指定的变量。

2. 队列的定义及运算

1）队列的基本概念

队列是只允许在一端进行删除，在另一端进行插入的顺序表。通常将允许插入的这一端称为队尾，通常用一个尾指针(rear)指向队尾元素。允许删除的这一端称为队首，通常用一个头指针(front)指向队列中第一个元素的前一个位置。当表中没有元素时称为空队列。

队列的修改是依照先进先出的原则进行的，因此队列也称为先进先出的线性表，或者后进后出的线性表。例如，火车进遂道，最先进遂道的是火车头，最后进遂道的是火车尾，而火车出遂道的时候也是火车头先出，最后出的是火车尾。

若有队列 $Q=(q_1,q_2,\cdots,q_n)$，那么 q_1 为队首元素，q_n 为队尾元素。队列中的元素是按照 $q_1,q_2,\cdots,q_n$ 的顺序进入的，退出队列也只能按照这个次序依次退出，即只有在 $q_1,\cdots,q_{n-1}$ 都退队之后，q_n 才能退出队列。因最先进入队列的元素将最先出队，所以队列具有先进先出的特性，体现“先来先服务”的原则。

队首元素 q_1 是最先被插入的元素，也是最先被删除的元素。队尾元素 q_n 是最后被插入的元素，也是最后被删除的元素。因此，与栈相反，队列又称为“先进先出”或“后进后出”的线性表。

2）队列运算

入队运算是往队列队尾插入一个数据元素；退队运算是从队列的队首删除一个数据元素。图 5-5 演示了一个队列及其入队和退队运算。

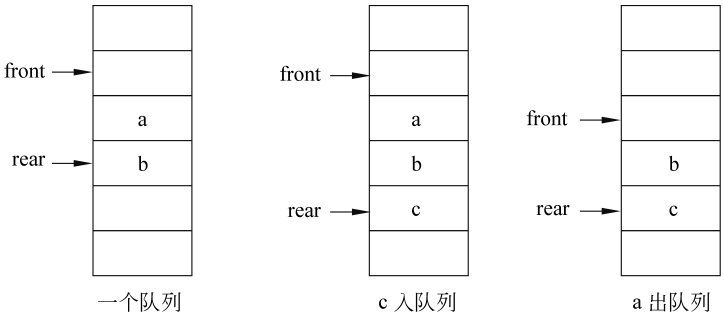

图 5-5　队列运算示意图

3）循环队列及其运算

在实际应用中，队列的顺序存储结构一般采用循环队列的形式。所谓循环队列，即队列存储空间的最后一个位置绕到第一个位置，形成逻辑上的环状空间。循环队列主要包括两种运算：入队运算与退队运算。

入队运算步骤：首先将队尾指针 rear 加 1，接着在 rear 指针指向的位置插入新元素。如果循环队列最大的存储空间为 m，则当尾指针 rear 为 $m+1$ 时，令 rear＝1，即在最后一个位置插入元素后，接着在第一个位置插入元素。

退队运算步骤：首指针 front＋1，然后删除 front 指针指向的位置上的元素。如果循环队列最大的存储空间为 m，则当头指针 front 为 $m+1$ 时，令 front＝1，即在最后一个位置删除元素后，接着在第一个位置删除元素。

例：在图 5-6 中，图 5-6(a)为空的循环队列；图 5-6(b)为 a 入队后的循环队列；图 5-6(c)为 b～h 入队后的循环队列；图 5-6(d)为 a 退队后的循环队列。首先，在图 5-6(a)中进行入队运算，队尾指针加 1，即 rear＝m＋1，令 rear＝1，再在第一个位置插入数据 a，如图 5-6(b)；依次插入 b～h，得到图 5-6(c)；对图 5-6(c)进行退队运算，首指针 front＝m＋1，令 front＝1，删除该位置数据 a。

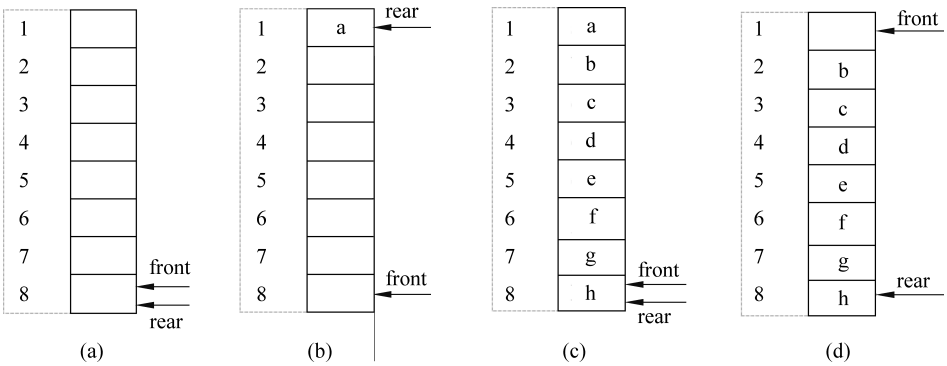

图 5-6　循环队列运算示意图

从图 5-6 可以看出，循环队列在队列满和空时，都有 front＝rear。在实际中使用循环队列时，为区分队列是满还是空，通常增加一个标志 S，定义如下。

$$S=\begin{cases}0, & \text{循环队列为空}\\ 1, & \text{循环队列为非空}\end{cases}$$

由此可以判断队列空还是满两种情况。

(1) 当 $S=0$,循环队列为空,此时不能进行退队运算,这种情况称为“下溢”。

(2) 当 $S=1$,循环队列为满,此时不能进行入队运算,这种情况称为“上溢”。

5.2.4 链表

线性表的顺序存储方式具有结构简单、可以随机存取等优点,但也存在两大缺点:①插入或删除操作时,需要移动大量元素,效率低;②对于长度可变的线性表,要预先分配(静态分配)足够的空间,这是很困难的。分配太大存储空间可能使部分空间长期闲置不用,分配太小空间会造成表的容量难以扩充。

为克服以上缺点,本节介绍线性表的链式存储方式。线性表的链式存储称为线性链表。线性链表中各个元素(节点)的存储空间可以连续也可以不连续,根据表的使用情况可以随时申请和撤销元素占用空间。

1. 线性链表

线性表的链式存储结构称为线性链表。

在链式存储方式中,要求每个节点由两部分组成:一部分用于存放数据元素值,称为数据域;另一部分用于存放指针,称为指针域。其中,指针用于指向该节点的前一个或后一个节点(即前件或后件),如图 5-7 所示。

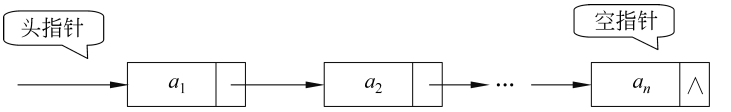

图 5-7 单链表

在线性链表中,各数据元素节点的存储空间可以是不连续的,且各数据元素的存储顺序与逻辑顺序可以不一致。在线性链表中进行插入与删除,不需要移动链表中的元素。

线性单链表中,HEAD 称为头指针,HEAD=NULL(或 0)称为空表。

如果是双向链表的两指针:左指针(Llink)指向前件节点,右指针(Rlink)指向后件节点。

线性链表的基本运算有:查找,插入,删除。

(1) 查找节点:线性链表的查找过程是从头指针(或链式栈的栈顶指针、或链式队列的队首指针)指向的节点开始,沿指针进行扫描,直到找到下一节点的数据域为查找值 x 或已无后继节点为止。

(2) 插入节点:先向系统申请一个新节点(由指针 p 指向),并赋值;然后利用查找算法找到待插入位置的前一节点的指针 q;修改两个指针即可:先将 p 指向 q 的后继节点,再将 p 挂接在 q 的后面。

与顺序存储结构的插入操作的最大区别：一是可随时申请新节点空间；二是只修改两个指针，无须移动节点。

(3) 删除节点：先判空，对非空表利用查找算法找到待删除位置的前一节点的指针 q，用另一指针 p 暂时保存 q 的后继节点(即待删除节点)，然后把 p 节点的后继链直接挂接在 q 的后面。最后释放 p 节点所分配的内存空间。

与顺序存储结构的删除操作的最大区别是，只修改一个指针，无须移动节点。

2. 双向链表

链表中每个节点设两个指针域(可称为左指针 Llink 和右指针 Rlink)，分别指向前趋节点和后继节点，表头节点的左指针指向尾节点，表尾节点的右指针指向头节点。

在某些应用中，对线性链表中的每个节点设置两个指针，一个称为左指针，用以指向其前件节点；另一个称为右指针，用以指向其后件节点。这样的表称为双向链表，如图 5-8 所示。

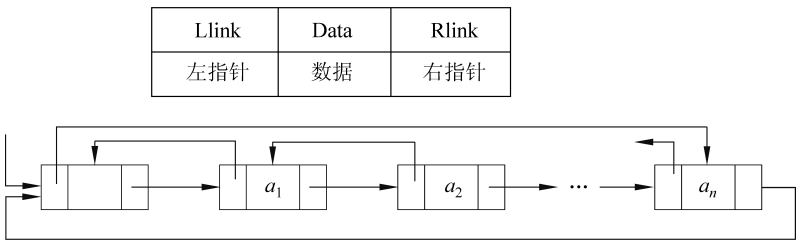

图 5-8　双向链表

3. 链式栈

栈也是线性表，也可以采用链式存储结构，栈的链式结构和单链表存储结构基本相同，单链表的头指针含义为栈的栈顶指针。插入和删除节点只能通过栈顶指针在栈顶端进行。带链的栈可以用来收集计算机存储空间中所有空闲的存储节点，这种带链的栈称为可利用栈。

4. 链式队列

队列的链式结构也和单链表的存储结构基本相同，不过链式队列有两个指针，一个队首指针，一个队尾指针。

5.2.5　树和二叉树

前面提到的栈、队列等都是线性结构。本节主要讲解数据结构中的非线性结构部分，包括树与二叉树。其中，树的部分介绍树的基本概念；二叉树部分主要给出了满二叉树和完全二叉树及其性质。最后给出了二叉树的前序、中序和后序遍历。

1. 树的基本概念

树是一种简单的非线性结构，由于它呈现与自然界中树类似的结构，所以称它为树。在客观世界中，树结构大量存在。比如一个家族的族谱关系：A 有后代 B，C，D；B 有后代 E，F；C 有后代 G；D 有后代 H，I，J；E 有后代 K；I 有后代 L，如图 5-9 所示。下面以图 5-9 为例，给出树结构的基本术语。

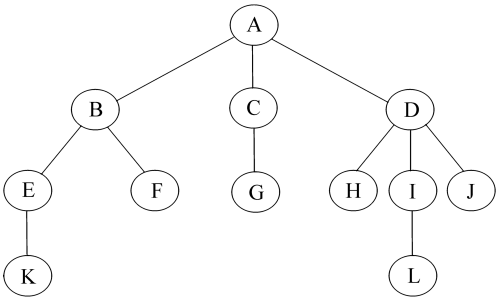

图 5-9　树结构

树是包含 $n(n\geqslant 0)$ 个节点的有限集合。当 n 为 0 时称为空树。对于非空树的结构中，每一个节点只有一个前件，称为父节点，没有前件的节点只有一个，称为树的根节点，简称为树的根。在图 5-9 中，A 是根节点。

在树结构中，每一个节点可以有多个后件，它们都称为该节点的子节点。没有后件的节点称为叶子节点。在图 5-9 中，节点 K，F，G，H，L，J 均为叶子节点。

在树结构中，一个节点所拥有的后件的个数称为该节点的度，所有节点中最大的称为树的度。在图 5-9 中，根节点 A 和节点 D 的度为 3，节点 C 的度为 1，叶子节点 K，F，G 的度为 0。所以，该树的度为 3。

在图 5-9 中，树共有 12 个节点，该树又可再分为若干不相交的子树，如 $T_1=\{B,E,F,K\}$，$T_2=\{C,G\}$，$T_3=\{D,H,I,J,L\}$ 等子树。

树具有明显的层次结构。定义一棵树的根节点所在的层次为 1，其他节点所在的层次等于它的父节点所在的层次加 1。树的最大层次称为树的深度。例如，在图 5-9 中，根节点 A 在第 1 层，节点 B，C，D 在第 2 层，节点 E，F，G，H，I，J 在第 3 层，节点 K，L 在第 4 层，该树的深度为 4。

2. 二叉树概念及性质

1）二叉树基本概念

二叉树是一种很有用的非线性结构，具有以下两个特点。

（1）二叉树可以为空；若非空，则二叉树只有一个根节点。

（2）每一个节点最多有两棵子树，且分别称为该节点的左子树和右子树。

在二叉树中，每一个节点的度最大为 2，即所有子树（左子树或右子树）也均为二叉树。二叉树中的每个节点的子树被明显地分为左子树和右子树。