# 第1章 数据结构

“数据结构”是计算机各有关专业的一门重要基础课程。计算机科学领域中，尤其是系统软件和应用软件的设计和实现中都要用到各种数据结构。学习“数据结构”既为进一步学习其他软件课程提供必要的准备知识，又有助于提高软件设计和程序编制水平。本章将介绍数据结构的基本概念和一些常用的数据结构，阐明数据结构内在的逻辑关系，讨论它们在计算机中的存储表示，以及在数据结构上进行各种运算的执行程序，并作简单的算法分析。

## 1.1 概 述

### 1.1.1 数据

数据就是对现实世界的事物采用计算机能够识别、存储和处理的方式(例如数字、字符等)进行的描述。简言之，数据就是计算机化的信息。

数据元素是数据的基本单位，即数据集合中的个体。有些情况下也把数据元素称作结点、记录、表目等。一个数据元素可由一个或多个数据项组成，数据项是有独立含义的数据最小单位。有时也把数据项称作域、字段等。例如，可以将一个学生的有关信息作为一个数据元素，它由姓名、专业、学号等数据项组成。

### 1.1.2 数据结构

被计算机加工的数据元素不是互相孤立的，它们彼此间一般存在着某些逻辑上的联系，这些联系需要在对数据进行存储和加工时反映出来。因此，数据结构概念一般包括三个方面的内容：数据之间的逻辑关系、数据在计算机中的存储方式、以及在这些数据上定义的运算的集合。

1. 数据的逻辑结构

数据的逻辑结构只抽象地反映数据元素间的逻辑关系，而不管其在计算机中的存储表示方式。

数据的逻辑结构分为线性结构和非线性结构。若各数据元素之间的逻辑关系可以用一个线性序列简单地表示出来，则称之为线性结构，否则称为非线性结构。线性表是典型的线性结构，而树、图等都是非线性结构。

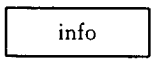
2. 数据的存储结构

数据的存储结构是逻辑结构在计算机存储器里的实现。

为全面地表示一个逻辑结构，它在存储器中的映象应包括数据元素自身值的表示和数据元素之间的关系的表示两个方面。因此，存储在计算机中的数据结构，其结点的各域按性质可分成两大类，一类是存放自身值的域，例如姓名、专业、学号等，通常称之为自身信息域，可用标识符 info 表示这些域的全体；另一类是存放该结点与其它结点的关系的域，例如一

个或多个指针，或其它形式的连接信息，通常称之为连接信息域，可用标识符 link 表示这些域的全体。一般情况下，存储结构中结点的形式为：


这里说的是一般情况，而在某些存储结构中，结点可以不包括连接信息域，这时的结点形式为：




3. 数据的运算

数据的运算是定义在数据的逻辑结构上的，但运算的具体实现要在存储结构上进行。数据的各种逻辑结构有相应的各种运算，每种逻辑结构都有一个运算的集合。常用的运算有检索、插入、删除、更新、排序等。

数据的运算是数据结构的一个重要方面，讨论任何一种数据结构时都离不开对该结构上的数据运算及实现算法的讨论。

### 1.1.3 主要的数据存储方式

有多种不同的方式来实现数据的逻辑结构到计算机存储器的映象。下面介绍两类最主要的存储方式，大多数数据结构的存储表示都采用其中一类方式，或两类方式的结合。

1. 顺序存储结构

这种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，结点之间的关系由存储单元的邻接关系来实现。

顺序存储结构的主要特点是：① 结点中只有自身信息域，没有连接信息域，因此存储密度大，存储空间利用率高；② 可以通过计算直接确定数据结构中第 i 个结点的存储地址 $L_i$，计算公式为：$L_i=L_0+(i-1)*m$，其中 $L_0$ 为第一个结点的存储地址，m 为每个结点所占用的存储单元个数；③ 插入、删除运算不便，会引起大量结点的移动，这一点在下一节还会具体讲到。

2. 链式存储结构

链式存储结构就是在每个结点中至少包括一个指针域，用指针来体现数据元素之间逻辑上的联系。这种存储结构可把人们从计算机存储单元的相继性限制中解放出来，可以把逻辑上相邻的两个元素存放在物理上不相邻的存储单元中；还可以在线性编址的计算机存储器中表示结点之间的非线性联系。

链式存储结构的主要特点是：① 结点中除自身信息外，还有表示连接信息的指针域，因此比顺序存储结构的存储密度小，存储空间利用率低；② 逻辑上相邻的结点物理上不必邻接，可用于线性表、树、图等多种逻辑结构的存储表示；③ 插入、删除操作灵活方便，不必移动结点，只要改变结点中的指针值即可，这一点在下一节还会具体讲到。

除上述两种主要存储方式外，散列法也是在线性表和集合的存储表示中常用的一种重要存储方式，我们将在 1.7.4 节中介绍。

## 1.2 线 性 表

线性表是最简单、最常用的一种数据结构。线性表的逻辑结构是 n 个数据元素的有限序列($a_1,a_2,\cdots,a_n$)。用顺序存储结构存储的线性表称作顺序表。用链式存储结构存储的线性表称作链表。对线性表的插入、删除运算可以发生的位置加以限制,则是两种特殊的线性表——栈和队列。若线性表中的元素都是单个字符,则称作串。

### 1.2.1 线性表的基本运算

线性表常用的运算分成四类,每类包含若干种运算。

1. 查找

① 查找线性表中第 i 个结点的值。

② 在线性表中查找值为 x 的结点。

2. 插入

① 把新结点插在线性表的第 i 个结点位置上。

② 把新结点插在值为 x 的结点的前面(或后面)。

3. 删除

① 在线性表中删除第 i 个结点。

② 在线性表中删除值为 x 的结点。

4. 其它运算

① 统计线性表中结点的个数。

② 遍历线性表中所有结点。

③ 把一个线性表拆成几个线性表。

④ 把几个线性表合并成一个线性表。

⑤ 根据结点的某个字段值升序(或降序)重新排列线性表。

### 1.2.2 顺序表和一维数组

各种高级语言里的一维数组就是用顺序方式存储的线性表,因此也常用一维数组来称呼顺序表。下面主要讨论顺序表的插入和删除运算。

往顺序表中插入一个新结点时,可能需要往后移动一系列结点。若顺序表中结点个数为 n,且往每个位置插入的概率相等,则插入一个结点平均需要移动的结点个数为 n/2。

类似地,从顺序表中删除一个结点可能需要往前移动一系列结点。在等概率的情况下,删除一个结点平均需要移动的结点个数也是 n/2。

### 1.2.3 链表

1. 线性链表(单链表)

线性链表就是链式存储的线性表,它的每个结点中含有一个指针域,用来指出其后继结点的位置。线性链表的最后一个结点没有后继结点,它的指针域为空(记为 NIL 或 Λ)。另外还需要设置一个指针 head,指向线性链表的第一个结点。

链表的一个重要特点是插入、删除运算灵活方便，不需移动结点，只要改变结点中指针域的值即可。图 1.1 显示了在单链表中指针 P 所指结点后插入一个新结点的指针变化情况，虚线所示为变化后的指针。

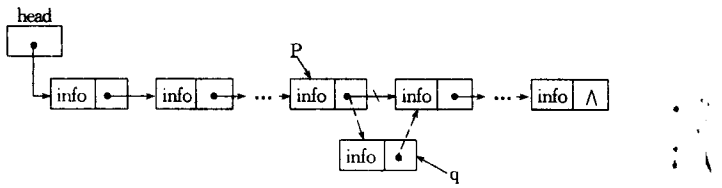



图 1.1　单链表的插入

插入运算的关键步骤为：

q↑.link := P↑.link;

P↑.link := q;

图 1.2 显示了从单链表中删除指针 P 所指结点的下一个结点的指针变化情况，虚线所示为变化后的指针。

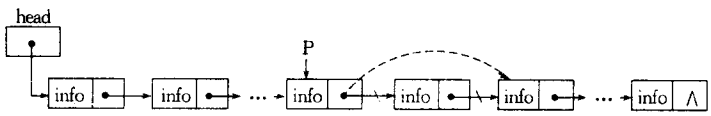



图 1.2　单链表的删除

删除运算的关键步骤为：

q := P↑.link;

P↑.link := q↑.link;

注意，做删除运算时改变的是被删结点的前一个结点中指针域的值。因此，若要求查找且删除某一结点，则应在查找被删结点的同时，记下它的前一个结点的位置。

在线性链表中，往第一个结点前面插入新结点和删除第一个结点会引起表头指针 head 值的变化。通常可以在线性链表的第一个结点之前附设一个结点，称为头结点。头结点的数据域可以不存储任何信息。头结点的指针域存储指向第一个结点的指针，如图 1.3 所示。这样，往第一个结点前面插入新结点和删除第一个结点就不影响表头指针 head 的值，而只改变头结点的指针域的值，因此就可以和其它位置的插入、删除同样处理了。

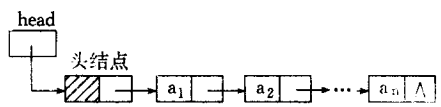



图 1.3　带头结点的线性链表

如果让线性链表的最后一个结点的指针指向第一个结点，便可得到一个环形链表。图 1.4 给出了带头结点的环形链表的结构形式。在环形链表中，可以从其中某个结点出发访问到表中所有结点。

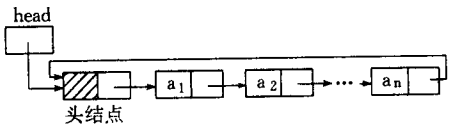



图 1.4　带头结点的环形链表

2. 双链表

在单链表中，从任何一个结点能通过 link 域找到它的后继，但不能找出它的前驱。如果在链表的每个结点中包括两个指针域，其中 rlink 指向结点的后继，llink 指向结点的前驱，就可以方便地进行向后和向前两个方向的查找了。这样的链表称作双链表，如图 1.5 所示。

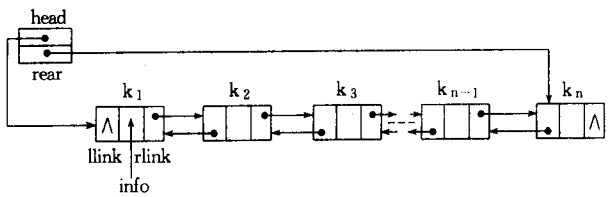



图 1.5　双链表

在双链表中，如果要删除指针变量所指的结点，只需修改该结点前驱的 rlink 字段和后继的 llink 字段，即

p↑.llink↑.rlink:=p↑.rlink;

p↑.rlink↑.llink:=p↑.llink;

如果要在 P 所指结点后插入 q 所指的新结点，只需修改 P 所指结点的 rlink 字段和原后继的 llink 字段，并置 q 所指结点的 llink 和 rlink 值，即

q↑.llink:=p;

q↑.rlink:=p↑.rlink;

p↑.rlink↑.llink:=q;

p↑.rlink:=q;

插入和删除运算前后指针的变化情况如图 1.6 和如图 1.7 所示。

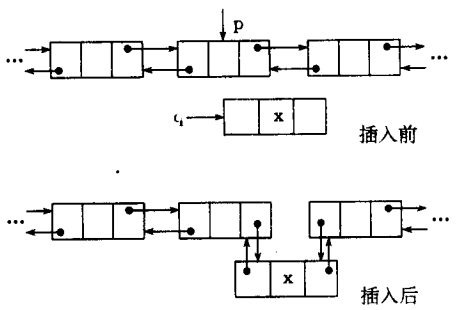



图 1.6　双链表的插入

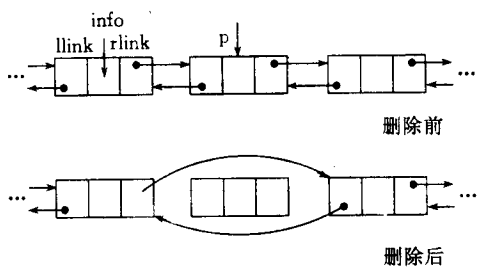



图 1.7　双链表的删除

和线性链表一样，双链表也有变形。图 1.8 给出环形的双向链表的结构形式。

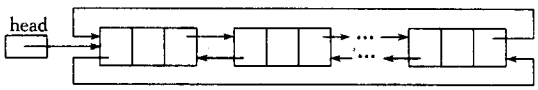



图 1.8　环形双向链表

3. 可利用空间表

要在链表上执行插入和删除操作，就有一个新的结点从何处来又回到何处去的问题。为此，设立另一种线性链表，称作可利用空间表，它与要操作的线性链表具有同样的结构，所不同的是，它的结点信息域是空的。。用一个指针 FREE 指向可利用空间表，如图 1.9 所示。

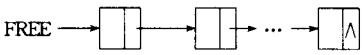



图 1.9　可利用空间表

可利用空间表的作用是管理可用于链表插入的结点。当链表插入需要一个新结点时，就从可利用空间表中删除第一个结点，用这个结点去做链表插入；当从链表中删除一个结点时，就把这个结点插入到可利用空间表的第一个结点前面。

4. 两个例子

请仔细研究下面例子，它包括了链表的检索、插入、删除运算和可利用空间表的管理。

[**例 1.1**]　阅读下列对线性表进行操作的 3 个子程序的流程图，从供选择的答案中选出应该填入 a～e 处的字句。

在主存储器中有一个如图 1.10 所示的表格结构，表格的每个元素由值(v)和指针(p)两部分组成。在表格中以链接方式存放着一个线性表 L，它的第一个元素的位置存放在 LP 中。表格中全部空闲元素也链接成一个线性表 E，它的第一个元素的位置存放在 CP 中。两个线性表的最后一个元素的指针都是 0。

(1) 子程序 enter(u,m)：在线性表 L 中位于 m(m≠0)处的元素后插入一个以 u 为值的结点，如图 1.11 所示。

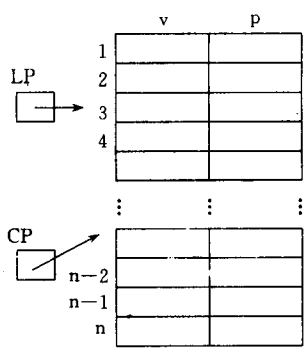



图 1.10　线性表 L 和线性表 E

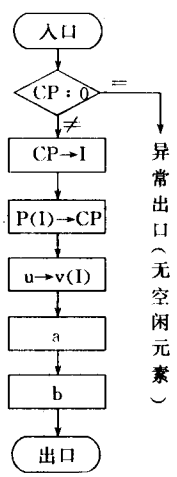



图 1.11　子程序 Enter

(2) 子程序 find(u,m)：从线性表 L 中找出第一个其值为 u 的元素，把位置送入 m。没有以 u 为值的元素时，把 m 置为 0，如图 1.12 所示。

(3) 子程序 delete(m)：从至少有两个元素的线性表 L 中删除紧接在位置 m 之后的一个元素，把删除的元素放进线性表 E 中，如图 1.13 所示。

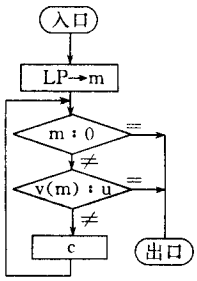



图 1.12　子程序 find

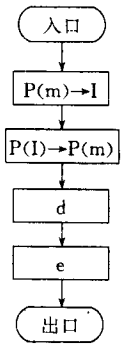



图 1.13　子程序 delete

供选择的答案

a,b,c,d,e：m→P(m)　　P(m)→m　　P(m)→P(I)
　　P(I)→P(m)　　CP→P(I)　　P(I)→CP
　　I→CP　　CP→I　　I→P(m)
　　P(m)→I

[**分析**]　插入时所用的存储单元取自可利用空间表 E，删除时释放的存储单元归还给可利用空间表 E。

子程序 Enter 对链表 L 执行插入运算。新结点所要占用的存储单元从可利用空间表 E 中已取得，其地址存放在 I 中，并且新结点的信息值 u 也已置好。于是可知余下的工作是将此新结点插入到链表 L 中位于 m 处的结点之后，即要使新结点的指针指向 m 处结点的原后继结点，而使 m 处结点的指针指向新结点。所以答案为：

a：P(m)→P(I)　　b：I→P(m)

读者也可不难分析出其它答案：

c：P(m)→m　d：CP→P(I)　e：I→CP

注意，答案中 a 和 b 的次序，d 和 e 的次序都不能交换，即有用的指针值在被使用之前不允许被破坏。

[**例 1.2**]　阅读下面关于约瑟夫(Joseph)问题的说明和流程图，如图 1.14 所示，在①～⑤处填上适当的操作语句。

[**说明**]　有 1 至 N 编号的 N 个人(N>1)按顺时针方向围坐一圈，每人持有一个密码(正整数)，一开始以正整数 M 作为报数上限值。从第一个开始顺时针方向自 1 开始顺序报数，报到 M 时停止报数，报 M 的人出列，将他的密码作为新的报数上限值。从他的顺时针方向上的下一个人开始重新报数，如此下去，直至所有的人全部出列为止。要求产生表示出列顺序的表。例如，若 N=7，每个人的密码依次是 3,1,7,2,4,8,4，M 的值为 20，则出列顺序为 6,1,4,7,2,3,5。

围成一圈的人用一个环形单链表表示，表中每个结点代表一个人。按出列次序依次将结点从环形单链表中删除，并按顺序存放在另一个单链表中。链表的每个结点包括三个字段：no是结点代表的人的编号，code代表他的密码，link是指向下一个结点的指针。若p是指向某结点的指针，则p↑.no，p↑.code，p↑.link分别表示该结点的这三个字段的值。初始时指针rear指向环形单链表的表尾，最后指针head指向表示出列序列的单链表的表头。

[**答案**]

① 1→j　　② rear↑.link→rear

③ rear→p↑.link　　④ rear↑.link→p 或 p↑.link→p

⑤ p↑.code→w

[**分析**] 此题用到链表的几种基本运算。围坐成一圈的人的依次报数用的是环形单链表的顺序查找(事实上不查找结点内容，只是对经过的结点个数计数)。流程图中用指针 rear 指向当前数到的结点。人的出列用的是环形单链表的删除运算，注意删除运算需改变被删结点的前一个结点的指针域，因此报数报到上限值减 1 就应停下来。出列的人加入到出列序列中是在单链表表尾处插入，流程图中用指针 p 指向单链表的尾，注意在表尾处插入后应令 p 指向新的表尾。此外，变量 i 是出列人数计数，变量 j 记报数值，变量 w 记当前的报数上限值。读者根据上述解答此题时，应注意链表查找、删除、插入的关键步骤，和各变量的初值，以及特殊情况(插入第一个结点，删除最后一个结点等)的处理。

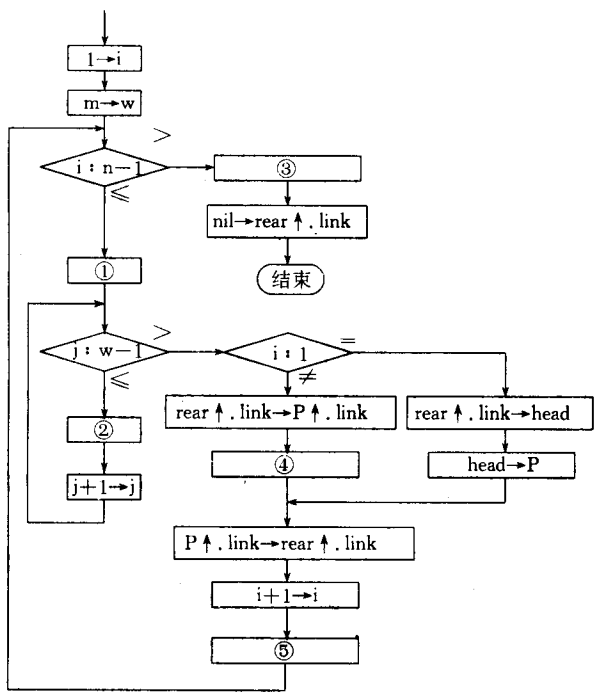



图 1.14　约瑟夫问题流程图

### 1.2.4 栈

栈是一种特殊的、限定仅在表的一端进行插入和删除运算的线性表，这一端称为栈顶(top)，另一端则叫做栈底(bottom)。表中无元素时称为空栈。栈中有元素 $a_1, a_2, \cdots, a_n$，如图 1.15 所示，称 $a_1$ 是栈底元素，$a_n$ 是栈顶元素。新元素进栈要置于 $a_n$ 之上，删除或退栈必须先对 $a_n$ 进行，这就形成了“后进先出”(LIFO)的操作原则。

栈的物理储存可以用顺序存储结构，也可用链式存储结构。

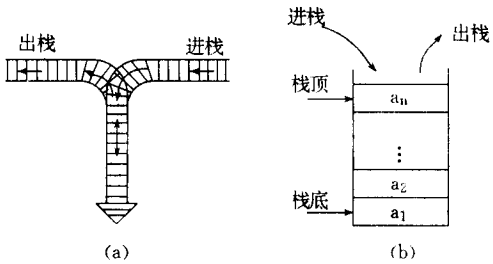



图 1.15 栈结构

栈的运算除去插入(称作推入)和删除(称作托出)外，还有取栈顶元素，检查栈是否为空，清除(置空栈)等。栈是使用最为广泛的数据结构之一，表达式求值、递归过程实现都是栈应用的典型例子。

### 1.2.5 队列

队列是一种特殊的，限定所有的插入都在表的一端进行，所有的删除都在表的另一端进行的线性表。进行删除的一端叫队列的头，进行插入的一端叫队列的尾，如图 1.16 所示。在队列中，新元素总是加入到队尾，每次删除的总是队列头上的，即当前“最老的”元素，这就形成了先进先出(FIFO)的操作原则。

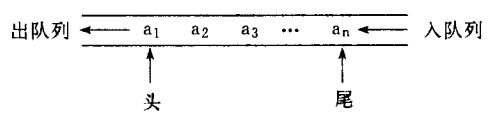



图 1.16 队列结构

队列的物理存储可以用顺序存储结构，也可以用链式存储结构。

队列的运算除了插入和删除外，还有取队头元素，检查队列是否为空，清除(置空队列)等。

在顺序方式存储的队列中实现插入、删除运算时，若采取每插入一个元素，队尾指示变量 R 的值加 1，每删除一个元素，队头指示变量 F 的值加 1 的方法，则经过若干插入，删除运算后，尽管当前队列中的元素个数少于存储空间的容量，但却可能无法再进行插入了，因为 R 已指向存储空间的末端。通常解决这个问题的方法是：把队列的存储空间从逻辑上看成

一个环，当 R 指向存储空间的末端后，就把它重新置成指向存储空间的始端，如图 1.17 所示。执行插入操作(即“进队”)时，先把值送入队尾指示变量 R 指向的结点，再把队尾指示变量的值加 1；执行删除操作(即“出队”)时，先把队头指示变量 F 指向的结点的值取出，再把队头指示变量的值加 1。当 F 和 R 的值超出队列的范围时，就把它们重新置成队列的第一个元素。因此，队头指针总是指向队列的第一个结点，而队尾指针指向队尾后的一个空白结点。

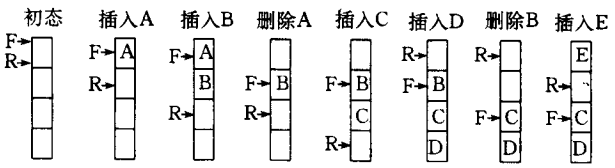



图 1.17　环状队列的插入和删除

队列在计算机中应用也十分广泛，硬设备中的各种排队器、缓冲区的循环使用技术、操作系统中的作业队列等都是队列应用的例子。

### 1.2.6　串

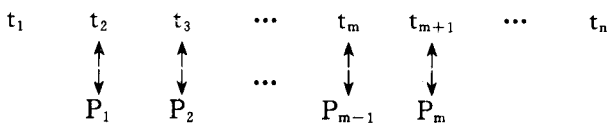
串(或字符串)是由零个或多个字符组成的有限序列，一般记为 S= ‘$a_1\ a_2 \cdots\ a_n$’，其中 S 是串的名字，用单引号括起来的字符序列是串的值。零个字符的串是空串。串中字符的数目就是串的长度。$a_i$ 是串中的字符，可以是字母、数字或其它字符。空串与空格构成的串(如：‘　　’)是不同的。

串的存储同样有顺序和链式两种。顺序存储时，既可以采用非紧缩方式，又可以采用紧缩方式。

串的基本运算有连接、赋值、求长度、全等比较、求子串、找子串位置以及替换等。有些程序设计语言以标准子程序(函数或过程)方式提供了这些运算。

在串的基本运算中，找子串位置(也称作模式匹配)是非常重要的，因为在与正文处理有关的许多应用中都需要做模式匹配，即在一串正文 T 中找一个子串 P 的出现位置。模式匹配的最简单的做法是：用 P 中字符依次与 T 中字符比较：


如果 $t_1=P_1, t_2=P_2, \cdots, t_m=P_m$，则匹配成功，找到了子串位置；否则将 P 右移一个字符，用 P 中字符从头开始与 T 中字符依次比较：




如此执行下去，直到某一步匹配成功，或者一直将 P 移到无法与 T 继续比较为止，则匹配失败。上述的匹配算法非常简单，但效率很低，在最坏的情况下，总的字符比较次数为 m * (n −m+1)。因而在实际应用中，有多种改进的模式匹配算法。

## 1.3 多维数组、稀疏矩阵和广义表

### 1.3.1 多维数组的顺序存储

多维数组是一维数组的推广，多维数组中最常用的是二维数组。

$$A_{mn}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

多维数组的所有元素并未排在一个线性序列里，要顺序存储多维数组就需要按一定次序把所有的数组元素排在一个线性序列里，常用的排列次序有行优先顺序和列优先顺序两种。二维数组在行优先顺序下元素排列次序为：$a_{11},a_{12},\cdots,a_{1n},a_{21},a_{22},\cdots,a_{2n},\cdots,a_{m1},a_{m2},\cdots,a_{mn}$；在列优先顺序下，元素排列次序为：$a_{11},a_{21},\cdots,a_{m1},a_{12},a_{22},\cdots,a_{m2},\cdots,a_{1n},a_{2n},\cdots,a_{mn}$。

给出任一数组元素的下标，可以直接计算数组元素的存放地址。二维数组元素的地址公式为：

(1) 行优先顺序下：

$$LOC(a_{ij})=LOC(a_{11})+((i-1)*n+(j-1))*l$$

(2) 列优先顺序下：

$$LOC(a_{ij})=LOC(a_{11})+((j-1)*m+(i-1))*l$$

式中，n 和 m 分别为数组每行和每列的元素个数，l 为每个数组元素占用的存储单元个数。

如果二维数组的形式为 $A[c_1..d_1,c_2..d_2]$，那么二维数组元素的地址公式变化为：

(1) 行优先顺序下：

$$LOC(a_{ij})=LOC(a_{c_1c_2})+((i-c_1)(d_2-c_2+1)+(j-c_2))*l$$

(2) 列优先顺序下：

$$LOC(a_{ij})=LOC(a_{c_1c_2})+((j-c_2)(d_1-c_1+1)+(i-c_1))*l$$

### 1.3.2 稀疏矩阵的存储

具有大量零元素的矩阵称为稀疏矩阵，对于稀疏矩阵可以进行压缩存储，只存储非零元素。下面考虑三角矩阵和一般的稀疏矩阵两种情况。

1. 三角矩阵

若非零元素的分布有规律，则可以用顺序方法存储非零元素，仍可以用公式计算数组元素的地址。例如，下三角矩阵

$$A_{nn}=\begin{bmatrix} a_{11} & 0 & \cdots & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

当 $i<j$ 时，$a_{ij}=0$。

如果按行优先顺序列出其中的非零元素，便得到如下序列：

$a_{11}\ a_{21}\ a_{22}\ a_{31}\ a_{32}\cdots\ a_{n1}\ a_{n2}\cdots\ a_{nn}$

把它顺序存储在内存中，第 1 行到第 i−1 行共有非零元素的个数为

$$\sum_{k=1}^{i-1} k = \frac{i*(i-1)}{2}$$

因此非零元素 $a_{ij}$ 的地址可用下式计算：

$$LOC(a_{ij}) = LOC(a_{11}) + \frac{i*(i-1)}{2} + (j-1), i \leqslant j \leqslant i \leqslant n$$

如果按列优先顺序列出其中的非零元素，便得如下序列：

$$a_{11}\ a_{21}\cdots\ a_{n1}\ a_{22}\ a_{32}\cdots\ a_{n2}\cdots\ a_{nn}$$

把它顺序存储在内存中，第 1 列到第 j−1 列共有非零元素的个数为：

$$\sum_{k=1}^{j-1}(n-k+1) = \frac{(2n-j+2)(j-1)}{2}$$

因此，此时非零元素 $a_{ij}$ 的地址可用下式计算：

$$LOC(a_{ij}) = LOC(a_{11}) + \frac{(2n-j+2)(j-1)}{2} + (i-j),\ 1 \leqslant j \leqslant i \leqslant n$$

对于上三角矩阵的情况，也可进行类似的计算。

2. 一般的稀疏矩阵

当非零元素的分布没有规律时，稀疏矩阵可用下列两种方法之一存储。

(1) 三元组法

这个方法用一个线性表来表示稀疏矩阵，线性表的每个结点对应稀疏矩阵的一个非零元素，其中包括 3 个域，分别为该元素的行下标、列下标和值。结点间的次序按矩阵的行优先顺序排列(跳过零元素)。这个线性表用顺序的方法存储在连续的存储区里。

[**例 1.3**] 图 1.18 表示稀疏矩阵的三元组表示形式。三元组形式中第 0 行表示稀疏矩阵有 5 行 4 列，其中非零元素个数为 7。

$$\begin{bmatrix} 12 & 15 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 36 & 46 & 0 & 52 \\ 0 & 0 & 0 & 0 \\ 0 & 72 & 0 & 68 \end{bmatrix}$$

(a) 稀疏矩阵

| | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 5 | 4 | 7 |
| 1 | 1 | 1 | 12 |
| 2 | 1 | 2 | 15 |
| 3 | 3 | 1 | 36 |
| 4 | 3 | 2 | 46 |
| 5 | 3 | 4 | 52 |
| 6 | 5 | 2 | 72 |
| 7 | 5 | 4 | 68 |

(b) 三元组形式

图 1.18 稀疏矩阵及相应的三元组表示形式

(2) 十字链表法

用三元组方法存储稀疏矩阵可以大大节省存储单元。但缺点是在非零元素增加或减少时，做插入或删除不便。链接存储则可以克服这一不足，十字链表法就是一种链接存储方法。

每个非零元素用一个结点表示，每个结点包含 5 个域，分别表示该元素的行下标 row，

列下标 col，值 val，以及指向本行下一个非零元素的指针 right，指向本列下一个非零元素的指针 down，如图 1.19 所示。

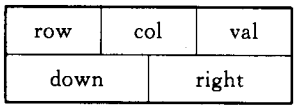



图 1.19　表示非零元素的结点的结构

稀疏矩阵的每一行用一个环形链表表示。每个行链表设置一个行表头结点，结构与表示非零元素的结点相同，其中 down 值指向下一行的行链表的表头结点，为处理方便，置 col 值为∞。并使所有的行链表的表头结点构成一个环形链表，同时设置一个总表头结点。

稀疏矩阵的每一列用一个环形链表表示。结构与行链表类似。

总表头结点的值分别表示矩阵的行数(row)、列数(col)、非零元素个数(val)、第 0 行的行链表表头位置(down)、第 0 列的列链表表头位置(right)。最后为总表头结点的位置设置一个指针 head。

[**例 1.4**]　图 1.18(a)的稀疏矩阵，按照上述方法构造的十字链表，如图 1.20 所示，图中数组的行号，列号从 0 起。

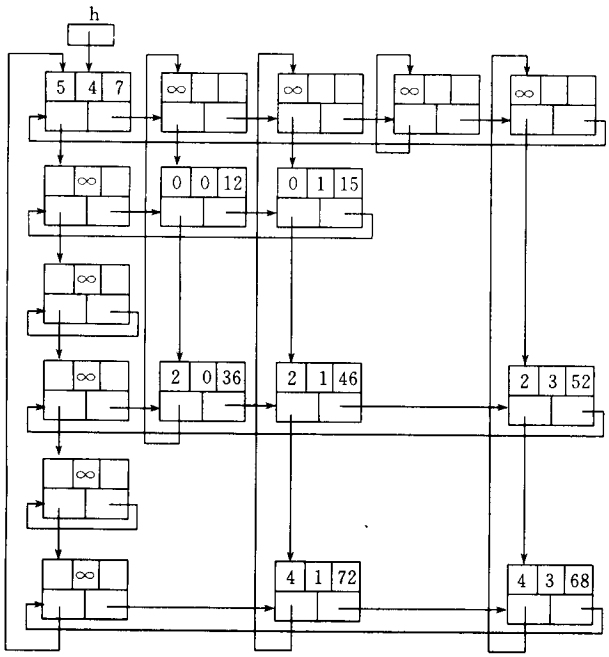



图 1.20　用十字链表表示的稀疏矩阵

十字链表的优点：经过某些运算(矩阵的转置、加法、乘法等运算)后，仍能保持十字链表的结构，非零元素出现，而零元素不出现。

## 3.3 广义表的定义和存储

广义表(又称列表)是线性表的推广,是由零个或多个单元素或子表所组成的有限序列。

广义表和线性表的区别在于:线性表的成分都是结构上不可分的单元素,而广义表的成分既可以是单元素,又可以是有结构的表。广义表一般记作

$$LS=(d_1,d_2,\cdots,d_n)$$

其中 LS 是广义表的名字,n 是其长度,$d_n$ 是广义表的成分,习惯上当一个成分为单元素时,用小写字母表示,当一个成分为子表时,用大写字母表示。当广义表 LS 非空时,称第一个元素 $d_1$ 为 LS 的表头,称其余元素组成的表$(d_2,d_3,\cdots,d_n)$为 LS 的表尾。

例如:

A=( )

B=(e)

C=(a,(b,c,d))

D=(A,B,C)

分别是长度为 0,1,2 和 3 的广义表。

E=(a,E)

是长度为 2 的广义表,但它是一递归的广义表,相当于一无限的广义表

(a,(a,(a,…)))

由此可以看出,广义表有如下特征:① 广义表的元素可以是子表,而子表的元素仍还可以是子表……;② 广义表可被其它广义表所共享(引用),如上例中,D 可通过子表 A、B、C 的名称引用它们的值,而不必列出每个子表的值;③ 广义表可以是递归的表,即广义表也可以是本身的一个子表,如上面的 E。

和线性表类似,可对广义表进行的运算有查找、插入和删除等。但广义表的两个非常重要的基本运算是取广义表表头 HEAD(LS)和取广义表表尾 TAIL(LS)。

任何一个非空广义表,其表头可能是单元素,也可能是广义表,而其表尾必定为广义表。举例如下:

HEAD(B)=e,HEAD(D)=A

TAIL(B)=( ), TAIL(D)=(B,C)

HEAD((B,C))=B,TAIL((B,C))=(C)

注意广义表( )和(())不同。前者为空表,长度为 0,后者长度为 1,可分解得到其表头和表尾均为空表( )。

广义表通常采用链接方式存储,广义表中每个元素用一个结点表示。子表结点形式为

| tag | head | link |
|---|---|---|

其中,tag=1,表示为子表结点,head 是指向子表第一个元素的指针,link 是指向该子表同一层中下一个元素的指针。单元素结点形式为

| tag | data | link |
|---|---|---|

其中，tag＝0，表示为单元素结点，data 是数据元素值，link 是指向该元素同一层中下一个元素的指针。

对于前面给出的广义表的例子，可给出它们的存储结构如图 1.21 所示。

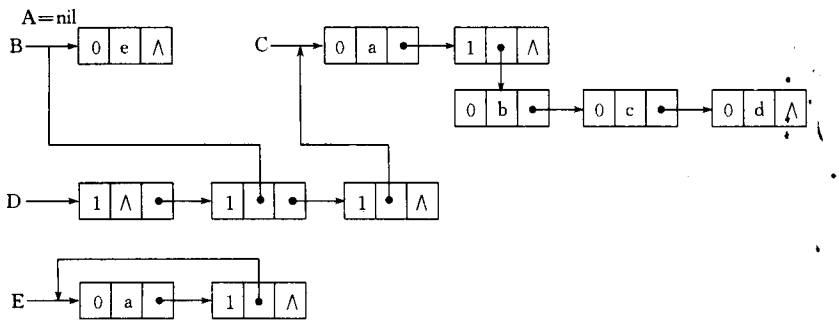



图 1.21　广义表的存储

## 1.4 集　　合

### 1.4.1 集合的概念

把一些事物汇集在一起组成一个整体，就称作集合，而这些事物就是集合的元素。

集合中的元素可以是基本元素，例如整数、字符、字符串等，也可以自身又是一个集合。当在数据结构或算法设计中使用集合作为工具时，通常一个集合的所有元素具有相同的数据类型。

我们假定可以作为一个集合的元素的所有基本元素是线性有序的，用“＜”来表示这种线性顺序，例如 3＜5，'b'＜'e'，'object'＜'subject' 等。常用的集合表示法有两种。

一种方法是列出集合的全体元素，元素之间用逗号隔开，并且有花括号括起来，例如

$$A=\{a,b,c,d\}$$

其中 A 是集合的名字，a 是集合 A 的元素，记作 $a\in A$。同样有 $b\in A$，$c\in A$，$d\in A$。e 不是集合 A 的元素，记作 $e\notin A$。

另一种方法是用谓词概括集合元素的属性，例如：

$$B=\{x \mid x \text{ 是正整数} \wedge x<10\}$$

集合中的元素各不相同，即任何元素在同一集合中都不会出现两次。集合仅由它所包含的不同元素决定，与各元素的出现顺序无关。因此{a，c，b，d}与{a，b，c，d}是同样的集合。

如果集合 B 中的每个元素都是集合 A 的元素，则称 B 为 A 的子集，这时我们说 B 被 A 包含，或 A 包含 B，记作 $B\subseteq A$。

如果 $A\subseteq B$，且 $B\subseteq A$，则称 A 与 B 相等，记作 $A=B$。如果 A 和 B 不相等，则记作 $A\neq B$。

如果 $B\subseteq A$，且 $B\neq A$，则称 B 是 A 的真子集，记作 $B\subset A$。

不含任何元素的集合叫做空集，记作 $\phi$。，空集是一切集合的子集。

### 1.4.2 集合的运算

在集合结构上可以定义许多运算，下面前三种是最基本的运算。

1. UNION(A,B,C)

求集合 A 和集合 B 的并，将结果放到集合变量 C 中。集合 A 和集合 B 的并 A∪B 定义为

$$A \cup B = \{X \mid X \in A \vee X \in B\}$$

2. INTERSECTION(A,B,C)

求集合 A 和集合 B 的交，将结果放到集合变量 C 中。集合 A 和集合 B 的交 A∩B 定义为

$$A \cap B = \{X \mid X \in A \wedge X \in B\}$$

3. DIFFERENCE(A,B,C)

求集合 A 和集合 B 的差，将结果放到集合变量 C 中。集合 A 和集合 B 的差 A−B 定义为

$$A - B = \{X \mid X \in A \wedge X \notin B\}$$

4. MEMBER(X,A)

这是一个函数，当 X∈A 时，返回值为 'true'，否则返回值为 'false'。

5. MAKENULL(A)

将集合变量 A 的值置为空集合。

6. INSERT(X,A)

把元素 X 加入到集合 A 中。注意若 X 本来已是集合 A 中的元素，则 INSERT(X,A)不改变集合 A。

7. DELETE(X,A)

将元素 X 从集中 A 中去掉。注意若 X 本来不是集合 A 中的元素，则 DELETE(X,A)不改变集合 A。

8. ASSIGN(A,B)

将集合 B 的值赋给集合变量 A。

9. MIN(A)

这是一个函数，其返回值为集合 A 的所有元素中按线性顺序最小的那个元素。例如

$$MIN\{2,3,1\} = 1$$

10. MAX(A)

这是一个函数，其返回值为集合 A 的所有元素中按线性顺序最大的那个元素。

11. EQUAL(A,B)

这是一个函数，若集合 A 与集合 B 相等，则其返回值为 'true'，否则返回值为 'false'。

12. FIND(X)

这是一个函数，它只在各个集合的交都是空集的情况下运行，返回值是元素 X 所在的那个集合的名字。

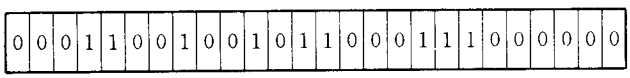
### 1.4.3 集合的存储表示

集合的存储表示方法有多种，根据集合的大小以及集合上要实现的主要运算来选择具体的表示方法。

1. 位向量存储

当要表示的每一个集合都是某一元素个数有限的“全集”的子集时，位向量存储表示法非常适用。将每一个集合表示成一个位向量（一维数组，其元素个数为“全集”的元素个数，元素的取值为 0 或 1），若“全集”的第 i 个元素是该集合的元素，则位向量的第 i 位取值为 1，否则第 i 位取值为 0。

例如，当集合元素只能是单个英文字母时，可用长度为 26 的位向量来表示集合。集合{d,h,e,m,t,s,k,n,r}表示为




集合的位向量存储表示的主要优点是：MEMBER，INSERT，DELETE 运算可以在常数时间内完成。而 UNION，INTERSECTION 和 DIFFERENCE 运算所需的时间和“全集”的大小成正比。

2. 链式存储

在这种存储表示方法里，我们用单链表来表示一个集合，集合的每一个元素对应于单链表的一个结点。

前面讲过，集合中各元素的出现顺序是无关紧要的。为提高运算的速度，在用链式存储方法表示集合时通常按比较集合元素大小的线性顺序将集合元素链接起来。这样，对于包含几个元素的集合，UNION，INTERSECTION，DIFFERENCE 等运算可在 O(n)的时间内完成；MIN 运算只要简单地取链表的第一个结点即可；INSERT 运算的实现也不复杂，但要注意把新的元素插入到链表中适当的位置上。

3. 顺序存储

用一个定长的一维数组来存储一个集合，每个数组元素代表一个集合元素，数组元素的数据类型与集合元素的数据类型相同，另外还需要一个整数来指明当前集合中元素的个数。

这种表示法很简单，但缺点较多：集合的大小不能超过数组的长度任意增长，DELETE 运算会引起大量数组元素的移动，定长数组中常会有一部分空间未被利用，等等。因此这是一种较少使用的集合表示法。

4. 散列存储

在 1.7.4 节中将要介绍的散列法也是存储集合结构的一个重要方法。

### 1.4.4 典型的集合结构

我们在 1.4.2 节中给出了集合上的一个比较完全的运算集，而在实际应用中往往不必定义所有这些运算。

1. 字典