数 据 结 构

清 华 大 学

严蔚敏 沈佩娟等编

国 防 工 业 出 版 社

内 容 简 介

《数据结构》是电子计算机专业的一门重要课程。全书共分十章。本书阐述了数据结构的基本概念、基本类型及其存贮结构。此外，还详细介绍了常用的查找及排序方法，并作了定性的分析与比较。本书的着重点在于应用。书中各章都附有数量不等的习题可供参考。

本书是高等院校电子计算机硬件专业及软件专业的试用教材，也可供从事计算机工程与科学的科技工作者参考。

前　言

随着电子计算机技术的飞速发展，电子计算机得到了越来越广泛的应用。在应用中，需要处理大量的非数值数据。近年来，虽然科学计算程序设计已经有了迅速提高，但难以解决非数值问题。因此，需要研究新的数据结构以及相应的算法。在国外，自1968年开始就有了《数据结构》这门课程。在国内，从1978年秋季起，也有若干院校先后开出了学时不等的《数据结构》课。由于大多数院校将从1981年春开始陆续开设此课程，迫切需要有关教材，因此，根据四机部召开的高等院校电子计算机硬件专业教材会议提出的要求，我们编写了这本教材。

本书主要介绍了线性表、串、树和图等几种基本类型的数据结构，以及使用这些数据结构进行程序设计时经常遇到的查找及排序问题。第一章综述了数据、数据类型及数据结构等基本概念，并对书中所用程序设计语言和算法分析作了粗略的说明；第二章至第六章分别介绍了上述几种数据结构的特性、基本运算及其在计算机中的存贮结构，并且对每一种结构都举了一些例子；第七至第九章，在上述基本数据结构的基础上介绍了常用的查找和排序方法，这里，除了给出相应的算法外，还着重从时间和空间两方面作了定性的分析和比较；第十章介绍文件。这一章的重点是文件的组织及其查找、修改的方法。《数据结构》可作为《编译构造》、《操作系统》、《数据库》、《信息检索》与《人工智能》等课程的基础。但从内容上来说，又很难与其绝然分开。由于我们考虑存贮管理部分放在《操作系统》等课程中更为合适，故本书无这部分内容。此外，由于《数据结构》是一门专业技术基础课，重点在于实践，所以在每一章（除第十章外）的最后都附有一定数量的习题。

本书可作为电子计算机硬件专业和软件专业学生的试用教材。讲授学时为60～80。它的教学要求是：学会分析研究计算机加工的数据对象的特性，以便选择合适的数据结构和存贮结构以及相应的算法；掌握各种算法在时间和空间上的分析技巧。另一方面，学习本课程的过程也是进行复杂程序设计的训练过程，因此，要求学员能写清晰、正确的算法，以便为进行软件工程设计及实现打好基础。

全书的内容可视学时的多少进行取舍，在学时较少的情况下，可以舍去诸如迷宫问题、判定树与文件等内容。

对个别较复杂的算法，我们用PASCAL语言写出其过程，作为附录，供大家参考。

只需要掌握编制程序的基本技术便可学习本书。具有离散数学和概率论知识的读者学习本书就更容易些。

参加本书初稿编写工作的有清华大学严蔚敏、沈佩娟、蒋国南、郑启华、吴家勋和张素琴等同志。全稿由严蔚敏和沈佩娟修改整理。本书承中国人民解放军国防科技大学陈火旺、王广芳和曹兰斌等同志主审。在审稿会上，中国科技大学、华南工学院、大连工学院、成都电讯工程学院、西北电讯工程学院、长沙铁道学院和湖南大学等兄弟院校的代表认真审阅了初稿，提出了许多宝贵意见。此外，在编写过程中还得到了我校本教研组的俞盘祥等同志的

帮助，金慧芬同志作了抄写工作。在此，一并表示衷心感谢。

由于编者业务水平及教学经验所限，加之编写时间非常仓促，因此，书中错误及缺点在所难免，恳切希望读者批评指正。

编　者

1980.8

目　录

第一章 绪 论

1.1 引 言

近廿年来，电子数字计算机一直在飞速发展。这种发展不仅是指计算机本身运算速度的不断提高，信息存贮容量的日益增大，而且也指计算机应用范围的逐渐扩大。早期的电子计算机几乎只是用于科技计算，六十年代以后，计算机则更多地用于数据处理和实时控制。与此相应，计算机加工处理的对象也从简单的数值发展到字符，进而发展到更复杂的具有一定结构的数据。从程序设计的观点出发，为了设计出效率高、可靠性强的程序，人们必须对程序设计的方法进行系统的研究。这就要求程序设计人员不但要掌握一般的程序设计技巧，而且还要研究计算机程序加工的对象，即研究数据的特性以及数据之间存在的关系。这样一来，便促进了《数据结构》这门学科的发展。

在国外，《数据结构》作为一门独立的课程是在1968年才开始的。在这之前，它的某些内容曾在其它课程（例如“表处理语言”）中有所讲述。1968年，在美国一些大学的计算机系的教学计划中，虽然把《数据结构》规定为一门课程，但对课程的范围仍没有作明确规定。当时，数据结构几乎和图论，特别和表、树的理论是同义语。随后，数据结构这个概念被扩充到包括网络、集合代数论、格、关系等方面，从而变成了现在称之为《离散结构》的内容。然而，由于数据必须在计算机中进行处理，因此，不仅要考虑数据本身的数学性质，而且还必须考虑数据的存贮结构。这样一来，就要求进一步扩大数据结构的内容。近年来，随着数据库系统的不断发展，在数据结构课程中又增加了文件管理（特别是大型文件的组织等）的内容。

由此可见，《数据结构》在计算机科学中是一门综合性的专业基础课[1]。数据结构的研究不仅涉及到计算机硬件（特别是编码理论、存贮装置和存取方法等）的研究范围，而且和计算机软件的研究有更密切的关系。无论是编译程序还是操作系统，都涉及到数据元素在存贮器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以使查找和存取数据元素更为方便。有人在描述数据结构这门课程所研究的内容时，曾将其归纳为数学、计算机硬件和计算机软件这三个学科范围互相重叠的区域（图1.1）。

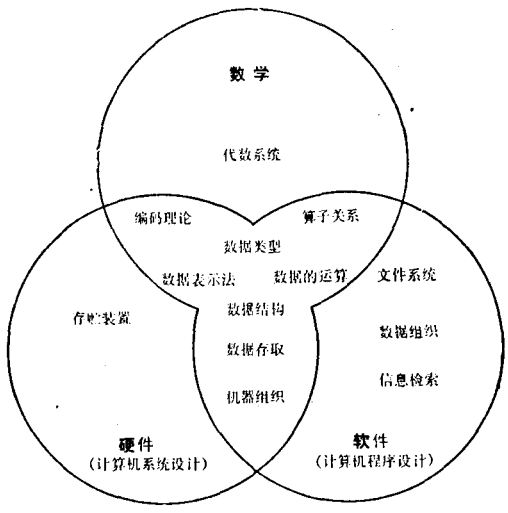

图 1.1 数据结构的研究对象

在我国，最近几年也已将《数据结构》作为计算机专业教学计划中的核心课程之一。这是因为，在计算机科学中，数据结构这一门课的内容不仅是一般程序设计（特别是非数值性程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序的重要基础。

1.2 什么是数据结构

什么是数据结构？这是一个难于直接回答的问题。然而，我们可以把问题分割成两个部分，先回答什么是数据，然后再回答什么是数据结构的问题。

直观地说，数据是描述客观事物的数、字符，以及所有能输入到计算机中并被计算机程序处理的符号的集合。它是计算机程序使用和加工的“原料”。例如，一个简单的计算程序所使用的数据只是一些实数和整数，而对于一个编译程序来说，它所使用和加工的数据则是程序员写的源程序（即字符的集合）。

数据的基本单位是数据元素① (data element)。性质相同的数据元素的集合就叫做数据对象 (data object)。例如，整数的数据对象是集合 $D=\{0, \pm1, \pm2, \cdots\}$，按字母顺序排列的字母字符的数据对象是集合 $D=\{\text{'A'}, \text{'B'}, \cdots, \text{'Z'}\}$。因此，数据对象是数据的一个子集，它可以是有限的，也可以是无限的。

除了最简单的数据对象外，通常，数据对象中的数据元素不是孤立的，而是彼此相关的。这种彼此之间存在的相互关系就叫做结构。数据结构是指数据元素之间抽象化的相互关系，并不涉及数据元素的具体内容。

数据结构和数据对象不同，描述一个数据结构不仅要描述数据对象，而且要描述数据元素之间的相互关系，这个相互关系可以用一组能对数据对象诸元素进行的运算及运算规则来描述。例如，可以把复数定义为一个简单数据结构，它的数据对象是一对有序的实数（即实部和虚部，它们之间存在着次序关系）的集合。允许对这种数据元素（一对有序的实数）进行的算术运算，可以有加、减、乘、除等。于是，一对有序的实数的集合，再加上如何进行加、减、乘、除等运算的描述，这两方面便构成了复数这个数据结构的定义。对计算机程序来说，把复数看成一种数据结构时，所需研究的不是复数的值，而是复数的特性、存贮，以及复数的各种基本运算。

数据结构有时是很复杂的，某个数据结构又可以是另一个数据结构的数据元素。

严格地说，数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构仅考虑数据元素之间的逻辑关系；而数据的物理结构，则是指数据元素在计算机存贮器中的表示及其配置。为了使其不至混淆，我们姑且把前者统称为数据结构，后者统称为存贮结构。一种数据结构② 可以通过映象得到与它相应的存贮结构。本书中将要讨论的数据结构有线性结构（包括字符串、栈、队列和数组）和非线性结构（包括树和图）。它们分别可以通过顺序映象和非顺序映象得到不同的存贮结构。

---

① 在某些数据结构中我们又称元素为顶点、结点、记录。

② 指数据的逻辑结构，以下同。

在本节的最后，我们还要引进一个后面将要用到的术语，这就是数据类型(data type)。数据类型指的是程序设计语言中允许的变量种类。例如数组、文件等。每种程序设计语言都有一组内部数据类型。例如：在ALGOL语言中，数据类型有整型、实型和布尔型；在PASCAL语言中，除了有标准的整型、实型、布尔型、字符型，以及由用户自己定义说明的纯量类型之外，还允许构造用户定义的构造类型。

一个数据类型不仅定义了一个变量可以设定的值的集合（程序里所出现的每个变量，必须与一个、而且仅仅与一个数据类型相联系），而且还规定了允许对变量进行的一组运算和运算规则。例如，ALGOL语言中的整型变量的值是某个有定义的整数子集的元素（子集的大小由具体的机器决定），允许对整型变量进行的算术运算有加、减、乘、除、整除和乘幂等（其中，除了除和乘幂之外，其运算结果仍为整数）。因此，我们可以把数据类型看成是程序设计语言中已经实现了的数据结构。

1.3 关于算法语言和算法分析的说明

由于研究数据结构的目的在于更好地进行程序设计（主要是非数值性程序设计），因此，本书在讨论各种数据结构的基本运算时，都将给出相应的算法。在算法描述上，除了用文字进行必要的叙述外，还将给出一个用程序设计语言描述的算法。为了使这些算法简单、明确、一目了然，本书未采用现存的语言（因为这些语言的语句都有一定的局限性），而采用一种假想的便于教学的程序设计语言。

这种语言类似于PASCAL语言，但不同于PASCAL语言。我们对它作如下说明：

(1) 用这种语言写算法时不需要写变量类型的说明，但允许算法中出现的变量可能是一个简单变量（实型、整型、布尔型、字符型），也可能是数组、记录、集合、文件和指针型的变量。允许对变量作相应的算术运算和布尔运算。

(2) 所有的算法都写成如下过程的形式：

```
PROCEDURE  过程名（参量表）;
  BEGIN
      语句组
  END;
```

其中，参量表可以含有若干个参量；语句组由一个或一个以上的语句组成，用“；”号作语句的结束符，但最后一个语句结束时不带分号。在正常的情况下，过程结束于END，在特殊情况下，允许过程有非正常出口。

(3) 基本语句有下述这些：

赋值语句：

```
变量名:=表达式;
```

条件语句：

```
IF  条件  THEN  语句1;
```

或

```
IF  条件  THEN  语句1
```

ELSE 语句 2;

其中，语句 1 和语句 2 可以是任意的语句组。当语句组包含多于一个语句时，则用方括号将这些语句括起来。

循环语句：

WHILE 条件 DO 语句;

或

REPEAT 语句组 UNTIL 条件;

或

FOR 变量:=初值 TO 终值 DO 语句;

或

FOR 变量:=初值 DOWNTO 终值 DO 语句;

在后两个循环语句中，前者的终值大于初值，增量为 1；后者的终值小于初值，增量为 - 1。这里的语句都和前面的语句 1 类同

情况语句

```
CASE
    条件 1：语句 1;
    条件 2：语句 2;
     ⋮      ⋮
    条件 n：语句 n
END;
```

或

```
CASE
    条件 1：语句 1;
    条件 2：语句 2;
     ⋮      ⋮
    条件 n：语句 n;
    ELSE 语句 n+1
END;
```

其中，语句 i (1≤i≤n+1) 与前面的语句类同。

调用过程语句：

CALL 过程名 (参量表);

过程允许嵌套调用，亦允许递归调用。

非正常出口语句：

EXIT 或 EXIT (变量)

用以控制跳出过程。

(4) 对于输入和输出，假定存在两个标准过程：

read (变量表); write (变量表);

其中变量表可以由若干个变量名或字符串组成，中间用逗号隔开。

(5) 在程序的任何地方，都可以插入用一对花括号括起来的中文注解。

书中，将对大多数的算法作性能上的评价。判定一个算法好坏的准则很多。在此，我们以随着问题规模（大小）的扩大，解决问题所用的算法所需时间（计算量）和空间增长的速度为标准来衡量算法的好坏。假设问题的规模（例如：矩阵的阶；线性表的长度；图中顶点数；等等）为 n，那么算法所需时间和空间是 n 的什么函数呢？假若算法的计算量为 $O(n^k)$（其中 k 是和 n 无关的常数），则称该算法为多项式阶的算法；假若算法的计算量为 $O(2^n)$，则称该算法为指数阶的算法。显然，前者是我们所渴望的，后者是应该尽量避免的。这里，我们引用了“O”这个记号[2]（对“O”，读者并不陌生，在学习微积分时曾经遇到过）。“O”表示这样一个数量级的概念，即：如果 f(n) 是正整数 n 的一个函数，则 $x_n = O(f(n))$ 表示存在一个正的常数M，使得当 $n \geqslant n_0$ 时都满足 $|x_n| \leqslant M|f(n)|$。至于M和 n_0，我们不必、也无法说出它们的确切数值。

然而，要事先对一个算法的计算量作仔细的分析，这是一件复杂的事情，而且这不是本课程的主要内容。因此，书中只是根据算法中语句执行的最大次数（频度）来估算一个算法执行时间的数量级。例如，对于下列三个简单的程序段：

```
(a) x:=x+1;
(b) FOR i:=1 TO n DO
        x:=x+1;
(c) FOR i:=1 TO n DO
      FOR j:=1 TO n DO
          x:=x+1;
```

假设在(a)中，语句“x:=x+1”，不在任何一个循环内，则它的频度为 1，执行时间是个常量。若程序段(b)中的同一语句被执行 n 次，则其执行时间和 n 成正比；若程序段(c)中同一语句的频度为 n^2，则执行时间和 n^2 成正比。因此，上述三个程序段的执行时间的数量级分别为 O(1)、O(n) 和 $O(n^2)$。对于程序执行时所占的存贮空间，也可作类似的分析。

第二章　线性表和向量

计算机的程序，通常都是对一些信息表进行操作。在大多数情况下，这些表并非无组织的数据集合，而是含有数据元素间的重要结构关系。本章我们讨论信息表中最简单的一种形式——线性表。

2.1　线性表及其存贮结构

2.1.1　线性表

线性表（linear list）是计算机程序中最常用、最简单的一种数据结构。一般来说，线性表是由一组数据元素组成的。至于一个数据元素的具体含义，则在不同的具体情况下可以不同，它可以是一个单一的符号（例如：'A'），或者一个数；也可以是一页书，甚至其它更复杂的信息。

例如，英文字母表

(A, B, C, …, Z)

是一个线性表，表中的数据元素是一个字母。又如，一个星期七天

(星期日，星期一，星期二，星期三，星期四，星期五，星期六)

也是一个线性表，表中的数据元素是星期中一天的名称。

在稍复杂的线性表中，一个数据元素可以由若干个数据项（item）组成。在这种情况下，通常把数据元素称为记录（record），把含有大量记录的线性表称为文件（file）。例如：一个班的学生健康情况登记表构成一个文件（图 2.1），表中每一个学生的情况就是一个记录（数据元素），一个记录由姓名、学号、性别、年龄、健康状况五个数据项组成。

姓　名	学　　号	性　　别	年　　龄	健康状况
王小林	770631	男	18	健　康
陈　红	770632	女	20	一　般
刘建平	770633	男	21	健　康
张立立	770634	男	24	神经衰弱
·	·	·	·	·
·	·	·	·	·
·	·	·	·	·

图 2.1　学生健康情况表

综合上述三个例子，我们可以如下描述线性表：

一个线性表是 $n \geqslant 0$ 个数据元素 a_1，a_2，…，a_n 的有限序列，表中每个数据元素，除第

一个和最后一个外，有且仅有一个直接前趋，有且仅有一个直接后继。换句话说，线性表或者是一个空表，或者可以写成

$$(a_1, a_2, \cdots, a_i, \cdots, a_n) \qquad (2\text{-}1)$$

其中，a_i 是属于某个数据对象的元素。

从上述描述可见线性表的结构特性如下：

(1) 线性表是数据元素的一个有限序列。线性表的长度定义为线性表中数据元素的个数 n。当 n = 0 时为空表。

(2) 数据元素在线性表中的位置只取决于它们自己的序号，数据元素之间的相对位置是线性的。例如，在线性表 (2-1) 中，a_1 是第一个数据元素，a_n 是最后一个数据元素；当 $1<i<n$ 时，a_i 的前一个数据元素（直接前趋）是 a_{i-1}，后一个数据元素（直接后继）是 a_{i+1}。因此，线性表是一个线性结构。

2.1.2 对线性表的运算

对线性表可能进行的运算有如下几种：

(1) 确定线性表的长度 n；

(2) 存取线性表的第 i 个数据元素，检验或改变某个数据项值；

(3) 在第 i－1 个和第 i 个数据元素之间，插入一个新的数据元素；

(4) 删除第 i 个数据元素；

(5) 将两个或两个以上的线性表合并成一个线性表；

(6) 把一个线性表拆成两个或两个以上的线性表；

(7) 重新复制一个线性表；

(8) 按某个特定的值查找线性表；

(9) 对线性表中的数据元素按其某个数据项值递增（或递减）的顺序进行重新排列。

并非每个表都必须进行所有这些运算，很多情况下只需要完成其中的一部分运算即可。由于运算和运算规则是构成一个数据结构的不可缺少的部分，因此，在讨论研究一种数据结构时，考虑对数据元素进行哪些运算是很重要的。本章主要讨论上述的运算 (3) 和 (4)。此外，第七、八、九章还要分别对运算(8)和(9)进行详细讨论。

2.1.3 线性表的存贮结构

在电子数字计算机上处理信息的最小单位是一位二进制数，叫做位 (bit)。在计算机中，我们可以用一个由若干位组合起来形成的一个位串来表示一个数据元素。当数据元素由若干个数据项组成时，每个数据项可用一个子位串来表示。称这个位串为元素[①] (element) 或结点 (node)，称其中的子位串为域 (field)

在计算机内，可以用不同的方式来表示线性表，这一章先介绍一种最简单、最普通的方式——用一组连续的存贮单元依次存贮线性表的元素。

假设表的每个元素需占用 l 个存贮单元，则表的第 i＋1 个元素的存贮地址 $LOC(a_{i+1})$

① 以后我们不严格区分数据元素和元素，而是一律统称为元素。

和第 i 个元素的存贮地址 LOC（a_i）满足下面的关系：

$$LOC(a_{i+1}) = LOC(a_i) + l$$

一般来说，线性表的第 i 个元素的存贮地址为

$$LOC(a_i) = LOC(a_1) + (i-1) * l \qquad (2\text{-}2)$$

式中，$LOC(a_1)$是表的第一个元素的存贮地址。

线性表的这种机内表示法叫做线性表的顺序分配（sequential allocation）或顺序映象（sequential mapping）。每一个元素在计算机内的存贮地址都与第一个元素的地址相差一个与序号成正比的常数（见图 2.2）。

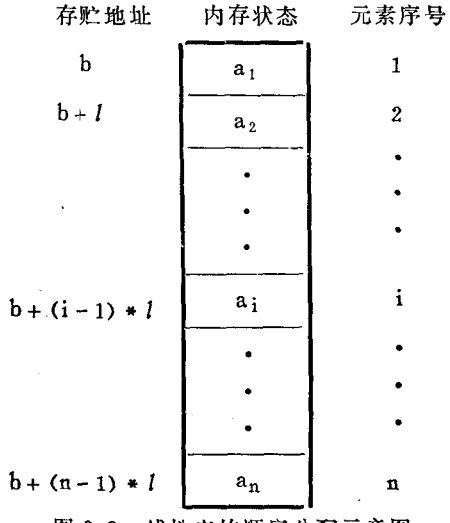

图 2.2 线性表的顺序分配示意图

线性表的这种存贮结构叫做向量（我们用 v 表示向量，用 v[i]表示向量的第 i 个分量）。向量的上界是线性表的长度 n。换言之，在顺序分配时，我们用上界为 n 的向量来表示长度为 n 的线性表。向量的第 i 个分量 v[i]是线性表的第 i 个元素 a_i 在计算机存贮器中的映象；向量的下标 i 代替了 a_i 的存贮地址。由于计算机内存是随机存取的装置，所以向量是一个随机存取的结构。表中任意一个元素的存取时间都相等。

也可以用非顺序映象存贮线性表。例如，链表和哈希（Hash）表就是两种线性表的非顺序分配的存贮结构。对此，我们将分别在第三章和第七章中作详细讨论。

在所有的程序语言中都已建立了向量①的形式定义并实现了对向量的运算。因此，在用语言编制程序时只要按该语言的规定加以引用即可。例如在 PASCAL 语言中，前述英文字母表的例子可用一维数组变量

```
VAR v: ARRAY[1..26] OF letter
```

来说明。其中，letter 是字符的子界类型

```
TYPE letter = 'A'..'Z'
```

v[10]表示一维数组的第 10 个分量，即字母表中第 10 个字母 'J' 在存贮器中的映象，'J' 的存贮地址可用下标'10'代替。

2.1.4 线性表的插入和删除

线性表的插入是指在线性表的第 i－1 个元素之后和第 i 个元素之前插入一个新的元素。换句话说，使长度为 n 的线性表

$$(a_1, a_2, \cdots, a_i, \cdots, a_n)$$

① 通常在程序语言中称向量为一维数组。

变成长度为 n+1 的线性表

$$(a'_1, a'_2, \cdots, a'_i, a'_{i+1}, \cdots, a'_{n+1})$$

其中，a'_i 为新插入的元素，且

当 $1 \leqslant j \leqslant i-1$ 时，$a'_j = a_j$

当 $i+1 \leqslant j \leqslant n+1$ 时，$a'_j = a_{j-1}$

线性表的删除是指删除线性表的第 i 个元素。换句话说，使长度为 n 的线性表

$$(a_1, a_2, \cdots, a_i, \cdots, a_n)$$

变成长度为 n-1 的线性表

$$(a'_1, a'_2, \cdots, a'_{i-1}, a'_i, \cdots, a'_{n-1})$$

其中

当 $1 \leqslant j \leqslant i-1$ 时，$a'_j = a_j$

当 $i \leqslant j \leqslant n-1$ 时，$a'_j = a_{j+1}$

对于线性表的这两种运算，在不同的存贮结构中，实现的方法也不同。

在线性表为顺序分配的情况下，若插入或删除运算只在表的末尾进行（即在第 n 个元素之后插入一个元素或删除第 n 个元素），那么很简单，只要在表的末尾增加或删除一个元素即可。但是，若在第 i（$1 \leqslant i \leqslant n$）个元素之前进行插入，或删除第i（$1 \leqslant i \leqslant n-1$）个数据元素，则必须移动表的元素。

例如，图 2.3 所示为一个有 8 个元素的有序的线性表（表中元素按元素值由小至大顺序排列），它们存贮在 8 个连续的内存单元中（假设一个元素占一个单元），现要插入一个值为 24 的元素，并且要求插入后该表仍是有序的（若 $i<j$，则 $a_i \leqslant a_j$）。为此，24 必须插在 24 和 28 之间，同时，把值为 28 到 77 的元素都依次往后移动一个位置。

又如，若要删除表中值为 24 的元素，则必须把值为 28 到 77 的元素都依次往前移动一个位置（见图 2.4）。

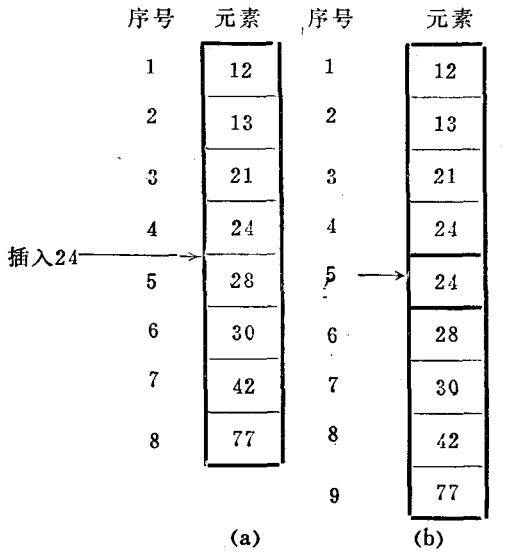

图 2.3 插入前后的线性表示例
(a)插入前 n=8；(b)插入后 n=9。

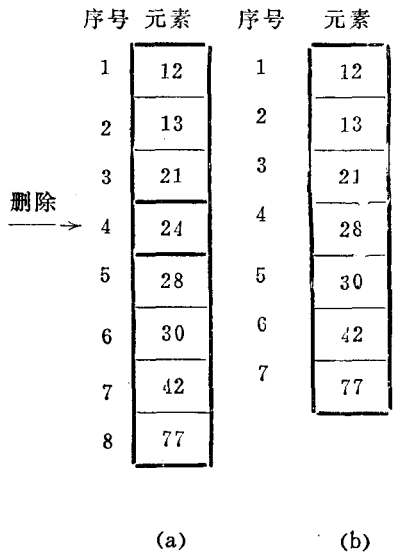

图 2.4 删除前后的线性表示例
(a)删除前 n=8；(b)删除后 n=7。

一般情况下，在第 i(1≤i≤n) 个元素之前插入一个元素时，需将从第 i 到第 n（共 n−i+1）个元素向后移动一个位置；删除第 i (1≤i≤n−1) 个元素时，需将从第 i+1 到第 n（共 n−i）个元素向前移动一个位置。

其插入算法如下：

```
PROCEDURE insertsqlist(v,n,i,x);
{在长度为 n 的线性表第 i 个元素之前插入一个元素 x，v 为存贮线性表的向量，v 的上界 m>n① }
BEGIN
  IF i≤n
    THEN FOR j:=n DOWNTO i DO
                  v[j+1]:=v[j];
  v[i]:=x; n:=n+1
END;
```

其删除算法如下：

```
PROCEDURE deletesqlist(v,n,i);
{在长度为 n 的线性表中删除第 i 个元素}
BEGIN
  IF i>n
    THEN write ('NO THIS EIEMENT')
    ELSE [FOR j:=i TO n-1 DO
                  v[j]:=v[j+1];
          n:=n-1]
END;
```

若设 p_j 是在第 j 个元素之前插入元素的概率，则在长度为 n 的表中插入一个元素时所需移动元素的期望值（平均次数）由式② (2-3) 确定：

$$E_{is}=\sum_{j=1}^{n+1}p_j\ (n-j+1) \tag{2-3}$$

同理，若设 q_j 是删除第 j 个元素的概率，则在长度为 n 的表中删除一个元素时所需移动元素的期望值由式 (2-4) 确定：

$$E_{de}=\sum_{j=1}^{n}q_j\ (n-j) \tag{2-4}$$

不失一般性，我们可以假定在表的任何位置插入或删除元素是等概率的，即

$$p_j=\frac{1}{n+1},\qquad q_j=\frac{1}{n}$$

① 在本章的所有算法中，v 的含义都与此相同。

② 包括在线性表的末尾（第 n 个元素之后）插入一个新的元素。