高等院校非计算机专业教材

新编计算机基础教程

（下册）

教材编写组　编

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
URL:http://www.phei.com.cn

高等院校非计算机专业教材

新编计算机基础教程

（下　册）

教材编写组　　编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书是参照九八年教育部高等教育司为提高高等院校计算机基础教育水平，加大计算机基础课的改革力度所制定的《普通高等学校计算机基础教育教学基本要求(本科)的意见》(征求意见稿)通知精神，以及部分省、市计算机应用基础课的教学大纲，为非计算机专业本科学生编写的。

全书共分为上、下两册。

下册共分十章。第一、第二章讲述计算机的软件基础，包括数据结构和操作系统的基础知识。第三章讲述软件工程基础。第四章至第十章较详细地讲述了面向对象的编程语言 Visual Basic 6.0 的基本知识及其特点、编程基础、高级应用和编程案例。

图书在版编目(CIP)数据

新编计算机基础教程　下册/赖邦惠等编. -北京：电子工业出版社，1999.12

高等院校非计算机专业教材

ISBN　7-5053-5742-5

Ⅰ.新…　Ⅱ.赖…　Ⅲ.电子计算机-基本知识-高等学校-教材　Ⅳ.TP3

中国版本图书馆 CIP 数据核字(1999)第 70450 号

丛　书　名：高等院校非计算机专业教材

书　　名：新编计算机基础教程(下册)

编　　者：教材编写组

责任编辑：王昌铭

排版制作：北京博顿新技术开发公司

印　刷　者：北京金特印刷厂

出版发行：电子工业出版社　URL：http://www.phei.com.cn

北京市海淀区万寿路 173 信箱　邮编 100036

经　　销：各地新华书店

开　　本：787×1092　1/16　印张：15　字数：394 千字

版　　次：1999 年 12 月第 1 版　1999 年 12 月第 1 次印刷

书　　号：ISBN 7-5053-5742-5 / TP·2962

定　　价：20.00 元

前　言

《新编计算机基础教程》是参照九八年教育部高等教育司为提高高等院校计算机基础教育水平，加大计算机基础课的改革力度所制定的《普通高等学校计算机基础教育教学基本要求（本科）的意见》（征求意见稿）通知精神，以及部分省、市计算机应用基础课的教学大纲，为非计算机专业本科学生编写的。

随着21世纪信息时代的到来，计算机科学和技术知识将在社会经济、科技、文化诸多领域发挥越来越大的作用。掌握必要的计算机科学和技术知识已成为高等院校广大学生必备的基础知识和基本技能，同时对提高大学生的素质乃至对经济的发展和社会的进步都将产生深远的影响。所以我们编写的这本教材力求反映《普通高等学校计算机基础教育教学基本要求（本科）的意见》的精神，并使之重点突出、内容新颖、实用性强、概念清晰、表述准确。

《新编计算机基础教程》分上、下两册。下册共分十章。

第一、第二章讲述计算机的软件基础，包括数据结构知识和操作系统理论。

第三章讲述计算机软件工程基础，包括软件开发过程、开发模型及软件开发方法（结构化方法和面向对象方法）。

第四章至第十章讲述面向对象的编程语言 Visual Basic 6.0 的基本知识、编程基础、高级应用和编程案例。其中第四章是 VB 的特点和集成开发环境；第五章是 VB 的语言结构和编程基础；第六章是 VB 过程和函数的使用；第七章是 VB 中面向对象的编程方法；第八章是 VB 的窗体和内部控件的使用；第九章是图像图形的使用及文件管理；第十章是 VB 高级应用，包括数据库编程和 API 函数使用等。

参加教程下册编写的有：赖邦蕙（第七章、第八章、第九章）、周启海（第一章）、甘嵘静（第十章）、杨祥茂（第二章、第三章）、缪春池（第四章、第五章、第六章）。

在教程的策划与编写过程中得到了西南财经大学经济信息管理系和电子工业出版社的大力支持，使该书能在较短时间内出版。在此特向对该书出版给予大力支持的同志致以诚挚的谢意。

由于作者水平所限，书中难免有不足乃至错误之处，恳请读者批评指正。

编　者

于西南财大

第一章　数据结构基础

数据(Data),是计算机处理的对象与基础。数据所采用的数据结构(Data Structure),对计算机信息处理的速度、效率、费用、效益,关系重大。

一般说来,在计算机的各种应用(特别是数据处理)中,除了极少数情形下尚可允许其数据呈现杂乱无章的自然组织状态外,绝大多数情形下都要求所论数据按其性质、特点和需要,采用相应的数据结构——数据的组织方式和结构形式(即结构关系)。为此,本书第一章将简要讲述有关数据结构的最基本知识。

1.1　数据结构及其运算

按照数据结构组织起来的数据,只有采用该数据结构下所允许的运算(Operation),方能始终确保这些数据所使用的数据结构的一致性和有效性。

1.1.1　数据与数据结构

数据,是一切能被计算机所接受并处理的对象(例如:数值、字符、文件、图形、图象、声音、……)之总称。它是信息处理(包括数据处理)中最基本的重要概念。

数据元素(简称元素,或称结点、节点),是数据的基本单位。它常随所论数据性质不同而有不同的表现形态:简单时,它仅由一个基本项(即不可再分解的、最基本的数据构成成分)单独构成;复杂时,它由若干个基本项、组合项(即尚可分解为多个基本项的数据构成成分)共同构成。

数据对象,是具有相同性质的数据元素的集合。通常,数据对象中各元素往往不是彼此无关的孤立元素,而是互相关联的群体元素。这种数据对象中各元素彼此间的相互关系(包括其逻辑关系与存储关系),称为数据对象的结构关系。

数据结构,既指数据对象的结构关系,也指研究和处理数据对象间结构关系的理论、方法与技术。常用的数据结构有:线性表(包括数组、栈、队、链表、串、文件)、树、图、数据库,等等。实际上,数据结构是指数据在两方面的结构——逻辑结构与存储结构。

数据的逻辑结构,只研究数据元素之间的逻辑联系,而不涉及它们在计算机内外(即内存储器和外存储器中)的表示(或称存放、存储)方式,通常分为线性结构与非线性结构。数据的存储结构(亦称物理结构),是指数据元素在计算机内外的表示方式,通常分为静态结构、半动态结构与动态结构。事实上,数据的逻辑结构与存储结构,正是构成数据结构的两个方面,其关系是:一种逻辑结构可以通过映象(即经某种映射变换后的所得结果)而得到它相应的存储结构,而且这种映象通常是非唯一的(即同一逻辑结构一般都可以映射成若干不同的存储结构)。因此,只要需要,同一数据对象完全可以用不同的数据结构进行描述和处理;反之,同一数据结构也可用于处理和描述不同的数据对象。

例 1.1.1　在现实社会经济生活中,一批前后相继、彼此相关的数据(例如:逐年国民生产总值 GNP 增长状况研究中的各年 GNP 值),通常可用对应的序列(例如:时间序列 $a_1, a_2, \cdots, a_n$)

来描述；而一个序列 a_1、a_2、…、a_n，在数学中则可用一个向量 $\mathbf{A}(a_1, a_2, \cdots, a_n)$ 来表示。显而易见，该向量的各分量 a_i（$1 \leqslant i \leqslant n$）彼此间然有别，相互由其下标（或称脚标）联系。要描述和处理向量，只需用最简单、最基本的数据结构——“数组”（例如一维数组）即可。

例 1.1.2 在数据处理中，报表是最常见的原始数据形态（例如各种基础数据的统计报表），也是最常用的信息输出形态（例如各种分析汇总报表）。无论何种报表总可以抽象为统一的数据处理模式——矩阵数据处理模式。例如，设一组数据可以组成矩阵 $\mathbf{A} = (a_{ij})$，其中 $i = 1, 2, \cdots, m$，$j = 1, 2, \cdots, n$。通常，可用“二维数组”作为描述和处理矩阵的数据结构工具：若对矩阵中每个数据都赋予一对相应行号 i 和列号 j，则这组数据就可按唯一的确定顺序组织起来，从而使每一个数据均只用其行号 i 和列号 j 便可唯一确定。事实上，这些数据的一种排列顺序可表示如下：

$$\begin{array}{cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{array}$$

显然，数据“a_{ij}”与其“行号 i 和列号 j”完全一一对应。

1.1.2 数据处理与数据结构

数据处理是计算机应用中用途最广、应用最频、影响最大、使用最多的领域。从根本上说，数据处理就是设计和应用相应计算机程序，以实现用计算机处理给定数据对象并解决给定问题。在计算机科学中，计算机程序无非是计算机程序设计的最终产物，而计算机程序设计的基础和核心则是计算机算法设计，并且它们遵从如下的基本法则：

计算机算法设计 = 算法 + 数据结构 + 程序设计方法学；

计算机程序设计 =（算法 + 数据结构 + 程序设计方法学）+ 计算机语言。

由此足见，数据结构在数据处理中有着何等重要的地位。

诚然在数据处理中，同一数据可用不同的数据结构进行描述和处理；但是，只有与所论数据本身性质相称的数据结构，才能更有效地处理这些数据。一般说来，数据处理必须首先考虑数据结构问题，而数据结构的设计又务必服务于数据处理的客观要求。因此，能否依据所给问题的目标、特性和要求，是否根据所论数据的性质、特点和需要，来正确选择、精心构造适当的数据结构，是提高数据处理算法效率的关键环节之一。

例 1.1.3 设某市为了加速实现通讯现代化，拟建立一个自动电话查号管理系统。该系统要求将本市电话号码簿（它记录了 n 个用户的姓名和相应电话号码）存储在计算机中，并能对任何一个人的姓名自动进行查找：若有此人，则输出其姓名与电话号码；否则，输出查无此人的信息。当然，要解决这样一个常见的数据处理问题并不难。但是，倘若对其数据结构选择不当，就会事倍而功半。请看：

如果设计者盲目采用该市原有的杂乱无章的电话号码簿自然顺序作为其数据结构（即不加任何组织而只按照用户申请安装电话时的先后顺序，来排列各用户姓名及其电话号码）；那么，以此散乱无序的平凡数据结构（即自然状态数据结构）为基础的自动电话查号管理系统，要查找某人的电话号码，就势必只能进行顺序查找：从头至尾、逐人查找。不难想象，这将是多么麻烦而费工、耗时、低效！并且，当该城市规模扩大，电话用户激增时，这样的“自动电话查号管理系统”就只能被淘汰，因为用它来查找一个人的电话号码所需时间之长，已远远超过当今信

息化社会人们所能忍受的等待时间限度。

与此相反，如果采用井然有序、组织有方的数据结构(例如：有序化的数组结构或树结构等)，并采用字典顺序(字母顺序或区位码顺序)来排列各用户姓名及其电话号码，则以此为数据结构的自动电话查号管理系统，就可施行高效率查找(例如二分查找)；可根据所给某人姓名的第一个字母(或汉字)，直接去查找以该字母(或汉字)开头的诸用户姓名，从而无需如前一种方式那样进行"大海捞针"似的低效率查找。显然，以后一种合理数据结构方式下的快速查找，是相当方便而合理的。

1.1.3 数据结构的基本运算

研究数据结构的一个重要方面，就是研究如何将抽象的数据结构映射为相应的存储结构，并使该数据结构的运算能在其不同存储结构下有效地施行。在数据处理中，选择恰当的数据结构的目的，不单是为了提高数据处理速度，更主要的是为了便于用户在此结构上对其数据施行各种基本操作。显然，这些基本操作不应也不准破坏所用数据结构，否则就破坏了作为数据处理工具和基础的数据结构本身。一般地讲，凡施加于数据结构之上并经处理后，仍保持原数据结构不变的有关操作，统称为该数据结构的运算。

正如整数的运算(加、减、乘、整指数乘方)并不同于实数的运算(加、减、乘、除、乘方，开方)一样，不同的数据结构的运算通常是有所区别的。事实上，数据结构的运算往往随问题要求、结构性质、存储方式不同而有所不同。通常，数据结构一般可施行如下的主要基本运算：

①插入——在数据结构中某处新增一个元素。

②查找(或称检索)——根据用户所给定的查找特征要求，在数据结构中查找(即找出)满足所给查找条件的相关元素。

③访问(即只取用而不删除)——察看(显示或打印)或者赋给数据结构中某处元素的有关数据。

④修改——对数据结构中符合用户修改条件的某元素，修改其指定内容。

⑤删除——剔除数据结构中某处的一个元素。

⑥排序(或称分类)——根据用户指定的排序要求和数据结构的结构特征，对具备这些排序要求和结构特征的元素进行分类(例如：将例 1.1.3 中姓王的所有用户都挑选出来)，并按某种顺序重新进行排列(例如：例 1.1.3 中将姓王的所有用户按电话号码大小顺序排列)。

⑦合并——把两个或两个以上的同类数据结构，合并成一个更大的同类数据结构。

⑧分解——把一个较大的数据结构，分解成两个或两个以上的较小的同类数据结构。

⑨复制——把一个数据结构复制为一个或多个其他同类数据结构。

例 1.1.4 对于例 1.1.3 中的自动电话查号管理系统中的数据结构，自然需要施行如下的基本操作：随时在它上面进行查找；有时，需要按某种要求进行排序(例如姓氏)；而当用户出现增加或减少时，还要作相应的插入或删除。这些查找、排序、插入和删除，显然不会改变原数据结构，所以它们是该数据结构的运算。

1.2 线性表基础知识

在数据处理中，最常用、最简单的一类数据结构是线性表(Linear list)。常用的数组、栈、队、链表、串、文件等基本数据结构，实际上都是线性表不同表现形式的各种特例。

1.2.1 线性表的基本概念、性质与运算

线性表，是 $n(\geqslant 0)$ 个同类数据元素的有限序列，且本质上其结构只有元素间的线性关系。线性表中数据元素的个数，称为线性表的长度。这表明线性表有两个基本性质：一是其各元素必须同类（指各元素的数据类型必须相同），二是其各元素的结构关系为线性化关系（简称线性关系），而后者比前者更为重要。

线性关系是线性表的最根本性质：在线性表（元素个数 $\geqslant 2$）中，除首元素有且仅有一个直接后继（元素），尾元素有且仅有一个直接前趋（元素）而外，其余元素均各自有且仅有一个直接前趋与直接后继。例如，大写英文字母表中"A"只有一个直接后继"B"，而"Z"则只有一个直接前趋"Y"，但是，除"A"和"Z"外，其余各字母（如"H"）均各有一个直接前趋（如"G"）和直接后继（如"I"）。显然，正是由于这种线性关系，才使线性表中各数据元素的确定，只取决于它们在表中的各自位置（即序号或地址）。

例 1.2.1 英文大写字母表""A","B","C",…,"Z""是线性表，因为它的数据元素结构关系是线性的，且可用字符型来统一描述。前 100 个自然数所成数列"1,2,…,100"也是线性表，因为其数据元素结构关系为线性，且可统一用数值型描述。但把二者合而为一所得""A","B","C",…,"Z",1,2,…,100"就不再是线性表了，因为两者各有不同的数据类型。家族成员血缘表不是线性表，因其结构关系是非线性的——家族各成员的血缘关系都来自其父母，也传给其子女，从而使他或她至少有多个直接前趋。

例 1.2.2 具有多个记录（即由多个数据项构成的数据元素）的线性表称为数据文件（简称文件）。例如某厂全体职工的档案表（见表 1.1 所示），就可以作为一个职工档案文件。显然，该档案表的数据元素就是其职工个人的档案材料记录，其特点是：各个记录依次排列，且每一记录均由各自的姓名、职工编号、性别、出生年、出生月、职务、职称、文化程度和工资等 8 个数据项共同构成，缺一不可。

表 1.1 职工档案简况一览表

姓 名	职工编号	性别	出 生		职 务	职 称	文化程度	工资
			年	月				
赵人中	0001	男	1945	10	厂 长	工程师	大专	1250
钱之华	0002	女	1955	1	副厂长	经济师	大学	900
孙初民	0003	女	1960	3	副厂长	会计师	大学	1100
李性族	0004	男	1973	9	工 人	技术员	中专	510
…	…	…	…	…	…	…	…	…

通常，线性表的基本运算有：长度测定（即求出线性表中元素个数），插入，察看，修改，删除，合并，分解，复制，查找，排序等。

1.2.2 线性表的存储结构及其运算实现

线性表的存储结构，既可以采用顺序存储结构，也可采用链式存储结构。但是，线性表（实际上，各种数据结构亦然）的运算，在不同存储结构中，其实现方法往往不同。

1. 线性表的顺序存储结构及其运算实现

线性表的线性有序化特性，使得顺序存储结构（即一维数组）是线性表最简单、最自然的存储结构。这里，所谓数组是一种常用基本数据结构，是一组有序化同类变量——下标变量的合

称(详见本章第 1.2.3 节);所谓变量,是计算机科学中用以存放数据的某个或某些内存单元及其名称,它存放数据的基本特性是"新的不来、旧的不去,新的一来、旧的必去"。一维数组的存储特点是:用若干个顺序连续的内存区域依次逐个存放线性表的各元素,而每个内存区域只存放一个元素,且各元素所占用的内存空间大小相同。

若用一维数组 a 表示一个线性表"$a_1, a_2, \cdots, a_n$"(设其长度为 n),a[i]表示一维数组 a 的第 i 个下标变量(也称数组分量),则该线性表的第 i 个元素 a_i(即原象)在计算机存储器中的映象为 a[i],且 a[i]与 a_i 是一一对应的。

设线性表的第一个元素的顺序存储物理位置(即所占存储区域起始地址,简称首地址)是 $LOC(a_1) = b$,每个元素需占用 d 个存储单元;则该表第 i 个元素的顺序存储首地址 $LOC(a_i)$,可用如下寻址(或称定位)公式直接求得:

$$LOC(a_i) = LOC(a_1) + (i-1) \times d = b + (i-1) \times d$$

线性表在顺序存储结构下,其原象(即数据元素)与映象(一维数组)间的对应关系示意图,如图 1-1 所示。

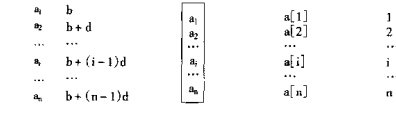

图 1-1 线性表顺序存储结构下原象与映象间的对应关系

现仅以插入与删除为例,简要说明如何实现顺序存储结构下线性表的基本运算。

例 1.2.3 设线性表"$a_1, a_2, \cdots, a_n$"采用顺序存储结构(一维数组 a)。现需先将新元素 x 插入该表并作为其新表第 j 个元素 a_j,然后将新表第 m 个元素 a_m 从新线性表中删除;其中,$1 \leqslant j \leqslant n \leqslant 100, j \leqslant m \leqslant n+1$。

问题分析:

①顺序存储结构下插入新元素 x 的算法设计要点如下:

显然,原线性表中 a_j 到 a_{j-1} 各元素不影响 x 的插入,故无需动它们;而 a_j 到 a_n 共 n-j+1 个元素则必须先顺次整体向后移一个位置,才可在所空出的原 a_j 处插入 x;由于新增一个元素 x,故线性表原长度 n 应改为新长度 n+1。

②顺序存储结构下删除其旧元素 a_m 的算法设计要点如下:

首先注意此时的线性表长度已为 n+1。与插入相似,只需将新表中 a_{m+1} 到 a_n 共 n-m 个元素整体向前移动一个位置,就能删除(实为挤掉)指定元素 a_m;同时,还必须修改其当前长度(即让其长度从 n+1 变为 n)即可。

据此,可得其算法 Eg01203,并可用 NS 周图描述如图 1-2 所示.

算法 Eg01203	{线性表顺序存储结构的插入、删除运算算法(算法名称描述行)}
{{变量: i,j,n,x}}	{变量定义描述(行尾注释结构框)}
{{数组: a[101]}}	{数组定义描述(其具体定义形式从简)}
> > >	{算法开始标志框}
{注释结构框:线性表生成　(即原始数据输入并生成一维数组)处理部分}	
输入 n	{顺序结构框:使用输入操作,以输入线性表的长度}
对 i←1,n	{步长型循环结构框:用一维数组 a 存放线性表各元素}
输入 a[i]	{循环体:分别用下标变量 a[i]存放线性表第 i 个元素}
{线性表的新元素插入处理部分}	
输入 x	{输入待插入新元素 x 之值}
输入 j	{输入待插入新元素 x 所在位置序号(即下标)}
对 i←n+1,j+1,-1	{使数组分量 a[j]到 a[n]整体后移一位(其步长 -1)}
a[i]←a[i-1]	{赋值操作:从后往前逐个后移原 a[i],以空出待插入位置}
a[j]←x	{使 a[j]取得 x 之值:把 x 插入原 a[j]处,让新 a[j]为新元素 x}
{线性表的旧元素删除处理部分}	
输入 m	{输入待删除元素 a_m 所在位置序号(即下标)}
对 i←m+1,n+1	{使数组分量 a[m+1]到 a[n+1]整体前移一位}
a[i-1]←a[i]	{从前往后逐个前移原 a[i],以挤掉应删除元素}
行输出 "OK!"	{换行式输出操作:输出字符串"OK",以示意其任务已完成}
!!!	{算法结束标志框}

图 1-2　线性表顺序存储结构的插入、删除运算算法之 NS 周图

算法 Eg01203 的执行效果示意图,如图 1-3 所示。

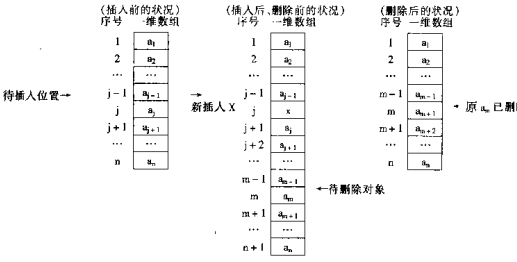

图 1-3　线性表顺序存储结构的插入与删除运算执行效果示意图

顺便指出:

①关于同构化(包括结构化)算法设计描述先进工具—NS 周图(即:Nissa – Shniederman – 周启海图)详细论述,可详见《计算机同构化程序设计原理及其应用导论》(周启海著,清华大学出版社 1993 年版)。

②为了便于读者查阅,本书中凡形如"算法 Eg01203"者,均表示"该算法为本书第 1 章第 2 节第 3 例的算法"。余类推。

2. 线性表的链式存储结构及其运算实现

线性表的顺序存储结构的主要优点是：可用其下标对表中各元素直接定位；但是，它同时有不便插入和删除的主要缺点（它通常要先移动相关元素）。所以，对需要进行大量插入和删除运算的线性表，就不宜采用顺序存储结构，而应选用便于直接插入和删除运算的链式存储结构。

链式存储结构的存储特点是：用若干个未必连续的内存区域依次逐个存放线性表的各元素，而各元素所占用的内存空间大小相同，但每个内存区域除存放其元素自身数据（称为数据域，或数据字段）外，还必须另用指针（它是一种用于存放内存地址的特殊变量）存放该元素的直接前趋或直接后继所在内存区域首地址（称为指针域，或指针字段），以便把这些原本未必连续的各个不同内存区域（称为指针对象，即其指针所指首地址开始存放的某元素自身及其直接前趋或直接后继所在首地址），彼此"链接"起来，构成一个如同链条那样的线性表（称为链表）。换言之，在链式存储方式下的线性表中，每个结点都由两部分结点信息构成：数据域所存放的其元素的数据信息，与指针域存放的该元素所链接对象（即周边邻元）的地址信息。线性表链式存储结构的原象与映象间的对应关系，如图 1-4 所示。图中，各映象的数据域、指针域分别描述各自结点的数据、直接后继指针，"/"表示空指针（表示该结点已无直接后继，程序设计中常记为 NIL）。

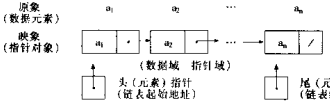

图 1-4　线性表链式存储结构的原象与映象间的对应关系示意图

现仅以插入为例，简要说明如何实现链式存储结构下线性表的基本运算。

例 1.2.4　设线性表"$a_1, a_2, \cdots, a_n$"采用链式存储结构。请将新元素 x 插入该表中间某处并作为其新表第 j 个元素 a_j，其中 $1 < j < n$。

问题分析　链式存储结构下从头部施行插入中间（注意：不是首尾！）新元素 x 的算法设计要点如下：

首先，在原线性表中找到元素 a_{j-1}，并用赋值操作标记其当前地址（即它所对应的指针值）。

其次，另生成一个新指针及其指针对象，并用此新指针对象的数据域存放新元素 x 之值。

然后，用此新指针对象的指针域存放原链表中元素 a_{j-1} 所在指针对象的指针域之值（注意它所存放的是元素 a_j 指针值），以建立起新元素 x 与其直接后继（即原元素 a_j）的链接关系。

最后，把原链表中元素 a_{j-1} 所在指针对象的指针域存放新指针值，以建立起新元素 x 与其直接前趋（即原链表中元素 a_{j-1}）的链接关系即可。

据此，可得其示意算法 Eg01204，并可用 NS 周图描述，如图 1-5 所示。

算法 Eg01204	线性表链式存储结构的插入运算示意算法
{变量: i,j,n,x}	整型变量定义
{指针: ……}	指针定义(因其具体定义形式较繁,故从略)
> > >	算法开始
原始数据输入并创建初始(单向)链表处理部分	
输入 n	输入线性表长度
输入 x	用简单变量 x 暂时存放线性表第 1 个元素 a_1 之值
线性表(采用单向链表存储结构)的创建处理	
生成新指针;头指针←新指针;尾指针←新指针	创建初始线性表处理
头指针对象的数据域←x	加载数据 a_1,以生成起始(头)结点
线性表的其他结点生成处理部分	
对 i←2,n	第 2 个到第 n 个元素所对应的各结点生成处理
输入 x	用简单变量 x 暂时存放线性表第 i 个元素 a_i 之值
生成新指针;新指针对象的数据域←x	加载数据 a_i,生成第 i 个新结点
新指针对象的指针域←头指针	从头部施行,使原头结点链接新结点
头指针←新指针	生成线性表新头指针
线性表的新结点插入处理部分	
输入 x	输入待插入新结点数据 x 之值
输入 j	输入待插入新结点 x 所在位置序号(即下标)
定位指针←头指针	定位指针初值(准备找出第 j-1 个元素 a_{j-1} 所在位置)
对 i←1,j-1	从头指针开始查找并确定应插入结点位置
定位指针←定位指针的指针域	链式顺序查找当前结点的直接后继
生成新指针;新指针对象的数据域←x	新结点装载数据 x
新指针对象指针域←最后所得定位指针对象指针域	新结点链接直接后继
最后所得定位指针对象指针域←新指针	直接前趋链接新结点
行输出 "OK!"	输出操作:示意用户其任务已完成
!!!	算法结束

图 1-5 线性表链式存储结构的插入运算示意算法之 NS 周图

算法 Eg01204 执行效果示意图,如图 1-6 所示。

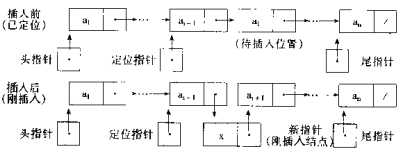

图 1-6 线性表链式存储结构的插入运算执行效果示意图

1.2.3 数组及其基本运算与存储结构

数组(Mathematical Array)是数据处理中应用最广泛的一种基本数据结构,应当予以足够重视。它是一种静态数据结构,是线性表最简单的一种特例。

1. 线性表与数组

如前所述，线性表的基本特征是“一个元素有一个与它一一对应的下标”（例如，线性表“$a_1,a_2,\cdots,a_i,\cdots,a_n$”中，$a_i$ 有且仅有一个下标 i 与它一一对应，$i=1,2,\cdots,n$）。如果将线性表的这一特征推广为“一个元素与一组下标一一对应”，则可得结构特性（即其元素间只有线性关系）颇类似于线性表的一种重要基本数据结构——数组，即一组个数固定的有序化下标变量（即数组元素，或称数组分量）的合称。因此，数组实际上是线性表的自然推广。

数组的下标个数，称为数组的维数。例如，由 n 个下标变量“a[1]，a[2]，…，a[n]”所构成的“数组 a”是一维数组，因为它只有一个下标；而由 $m\times n$ 个下标变量“b[1,1]，b[1,2]，…，b[1,n]；b[2,1]，b[2,2]，…，b[2,n]；…；b[m,1]，b[m,2]，…，b[1,n]”构成的“数组 b”是二维数组，因为它共有两个下标。

例 1.2.5 设矩阵 $\mathbf{A}=(a_{ij})_{m\times n}$，则 A 中第 i 行和第 j 列的元素 a_{ij} 与其下标 i，j 是一一对应的，故矩阵 A 就可以用一个二维数组 a[i,j]来表征（其中，$1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$）。

鉴于前面在讲述线性表的顺序存储结构时已初步简述了一维数组，而二维数组又具有应用上的广泛性和学习上的典型性，故下面特以二维数组为代表来简介数组。

由于数组的元素个数是固定的，故一般不便在数组上执行插入和删除运算。通常，数组宜施行如下基本运算：

①查找——给定一组下标（或一个元素值），查找其对应元素（或其对应某组下标）；

②修改——给定一组下标，修改其对应元素中某数据项的值；

③存放——给定一组下标，存放其对应元素中某数据项的值；

④取用——给定一组下标，取用其对应元素中某数据项的值。

2. 数组的存储结构

数组的存储结构是顺序存储结构。在具体按顺序存储二维数组各元素时，通常按行（必要时，也可按列）为序，逐一存储在一组顺序连续的存储区域中。例如高级语言 BASIC、COBOL、PASCAL、C 等采用按行存储方式，而高级语言 FORTRAN 等则采取按列存储方式。必须强调指出，若采用的存储方式不同，则将使施加于数组上的算法有时会大有不同，这一点必须引起足够重视。

例 1.2.6 已知矩阵

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

描述矩阵 A 的二维数组 a，如果采用图 1-7 中图 A 所示的逻辑结构，则其存储结构可能是图 1-7 中的图 B 或图 C。

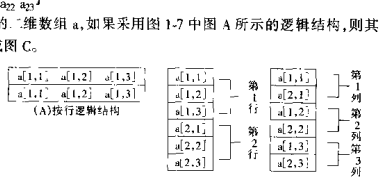

图 1-7 二维数组逻辑结构与其存储结构关系的示意图

设数组的每个元素占 d 个单元，每行元素个数为 n，每列元素个数为 m，第一个元素 a[1,

i]的首地址为 LOC[1,1]=b，则元素 a[i,j]的首地址可用如下寻址函数 LOC[i,j]之一来计算（1≤i≤m，1≤j≤n）：

(1)当存储结构为按行存储方式时，其寻址函数为

$$LOC[i,j]=LOC[1,1]+(n\times(i-1)+(j-1))\times d=b+(n\times(i-1)+(j-1))\times d$$

(2)当存储结构为按列存储方式时，其寻址函数为

$$LOC[i,j]=LOC[1,1]+((i-1)+m\times(j-1))\times d=b+((i-1)+m\times(j-1))\times d$$

3. 特殊矩阵的压缩存储及其特殊运算

值得注意的是：对于非零元素较多的一般矩阵，采用上述整体性顺序存储结构是适当的。然而，对特殊矩阵（例如：对角阵、三角阵、稀疏阵、……），倘若也盲目地同样采取这种整存式的顺序存储结构，就势必大大浪费存储空间宝贵资源，有时甚至会因内存空间占用过大而使本来可以解决的问题变得无法用计算机来处理。比如：三角阵就需用其一半的内存空间，来存储它的 n(n-1)/2 个零元素"0"；而单位矩阵或一般对角阵就更要花其绝大部分内存空间，来存储它的 n^2-n 个非对角线零元素。因此，对于诸如三角阵、对称**阵**、对角阵、三对角阵和稀疏阵等特殊矩阵，都应根据其数学特点而采用相应的特殊存储结构——压缩存储结构，其存储特点是"压缩掉无益的各零元素，只存储有益的各元素"。

例 1.2.7 主对角阵 $\mathbf{A}=(a_{ij})$的特点是：除主对角线元素 a_{ii}外均为零元素（其中，$1\leqslant i\leqslant n$，$1\leqslant j\leqslant n$），即当 $i\neq j$ 时，恒有 $a_{ij}=0$。故主对角阵 **A** 的最佳压缩存储结构，是只用仅有 n 个元素的一维数组来存储其各对角线元素"$a_{11},a_{22},\cdots,a_{nn}$"。显然，如果记 a_{11} 的首地址为 LOC(1,1)=b，每个元素占 d 个单元，则主对角矩阵的对角元素 a_{ii}的寻址公式为

$$LOC(i,i)=LOC[1,1]+(i-1)\times d=b+(i-1)\times d$$

例 1.2.8 上三角形矩阵(a_{ij})的特点是：其主对角线以下各元素均为零元素（其中，$1\leqslant i\leqslant n$，$1\leqslant j\leqslant n$），即

$$a_{ij}\begin{cases}\neq 0 & \text{当 } i\leqslant j\\ =0 & \text{当 } i>j\end{cases}$$

故只需压缩存储其上半部分元素（即 $i\leqslant j$ 者）即可。设该上三角矩阵元素 a_{11}的首地址为 LOC[1,1]=b，每个元素占用 d 个单元，则其上半部分元素 a_{ij}可采用按照行顺序存储结构（其中，$1\leqslant i\leqslant j\leqslant n$），且其寻址公式为

$$\begin{aligned}LOC[i,j]&=LOC[1,1]+(((i-1)\times n-i\times(i-1)/2)+(j-1))\times d\\&=b+(((i-1)\times n-i\times(i-1)/2)+(j-1))\times d\end{aligned}$$

在工程、力学、管理、……等科学计算与数据处理中，不时会遇到一类除少数非零元素外，大多数（甚至绝大多数）元素均为零的特殊矩阵——稀疏阵。由于这些非零元素所在位置是随机的，故稀疏阵的存储结构，应考虑到可同时描述其各非零元素的位置与数值两方面。为此，稀疏阵较好的一种压缩存储结构是以三元组(i,j,a_{ij})的形式只存储稀疏阵的各非零元素 a_{ij}，以构成一个所需存储空间小得多的压缩存储矩阵（其中，$1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$）。实际上，它是一个若干行、共三列的二维数组；它除第 1 行描述的是稀疏矩阵的总行数、总列数和非零元素个数外，其余每行各顺序对应于一个非零元素所决定的三元组。

例 1.2.9 试设计如下稀疏阵 $\mathbf{A}_{6\times8}$的压缩存储结构。

$$\mathbf{A}=\begin{bmatrix}3&0&0&7&0&2&0&0\\0&0&0&0&0&0&0&0\\0&2&0&0&3&1&0&0\\0&0&0&0&4&0&0&0\\0&12&21&0&0&0&0&0\\91&0&0&0&0&0&0&0\end{bmatrix}\begin{matrix}\text{第 1 行}\\\text{第 2 行}\\\text{第 3 行}\\\text{第 4 行}\\\text{第 5 行}\\\text{第 6 行}\end{matrix}$$

首先，将稀疏阵 **A** 中 10 个非零元素 a_{ij} 全部挑出并构成一新的压缩存储矩阵 $\mathbf{B}_{11\times3}$ 如下所示。该压缩存储矩阵 **B** 除第 1 行外，其他每一行的特点是：行号表示稀疏阵 **A** 按行扫描的非零元素序号 +1，第 1、2、3 列分别表示各自非零元素的行号、列号、非零值。

$$\mathbf{B}=\begin{bmatrix}6&8&10\\1&1&3\\1&4&7\\1&6&2\\3&2&2\\3&5&3\\3&6&1\\4&5&4\\5&2&12\\5&3&21\\6&1&91\end{bmatrix}\begin{matrix}\text{第 1 行：稀疏阵 A 整体信息(总行数、总列数和非零元素个数)}\\\text{第 2 行：稀疏阵 A 第 1 行、第 1 列是按行扫描的第 1 个非零元素 3}\\\text{第 3 行：稀疏阵 A 第 1 行、第 4 列是按行扫描的第 2 个非零元素 7}\\\cdots\\\cdots\\\cdots\\\cdots\\\cdots\\\cdots\\\text{第 10 行：稀疏阵 A 第 5 行、第 3 列是按行扫描的第 9 个非零元素 21}\\\text{第 11 行：稀疏阵 A 第 6 行、第 1 列是按行扫描的第 10 个非零元素 91}\end{matrix}$$

然后，将压缩存储矩阵 B 用二维数组 b 施行一般的顺序存储即可。

必须指出：采用压缩存储的特殊矩阵，其运算已与通常的整体存储方式大有不同，故绝不能简单地移用一般矩阵运算的相应算法。限于篇幅，下面仅以压缩存储的稀疏阵的存放运算为例，简要说明特殊矩阵有关运算的实现方法。

例 1.2.10 试给出实现例 1.2.9 稀疏矩阵 **A** 压缩存储的存放运算之算法。

问题分析 在例 1.2.9 的问题分析基础上，还应特别注意如下算法设计要点：

首先，按行逐列取出并统计稀疏矩阵 **A** 中非零元素的个数；

其次，决定描述压缩存储矩阵 **B** 的二维数组 b 的容量，并用数组 b 第 1 行存放稀疏矩阵 **A** 的整体数据；

然后，取出稀疏矩阵 **A** 中各非零元素，并用数组 b 对应行分别顺次压缩存放之。

据此，可得其压缩存储算法 Eg01210，并可用 NS 周图描述如图 1-8 所示。

1.2.4 栈及其基本运算与存储结构

栈(Stack)是数据处理中应用较广泛的一种基本数据结构。它是一种半动态数据结构，是线性表的一种简单特例。

栈，是只能固定在线性表的某一端进行压入(也称插入，进栈)和弹出(也称删除，出栈)其元素的特殊线性表。如同冲锋枪弹夹及其子弹的压入与弹出那样，栈及其元素的压入与弹出只能在栈的顶端施行。栈中允许压入和弹出的那一端称为栈顶，而不允许进行任何操作(包括压入、弹出、插入、删除等)的另一端称为栈底。若记栈为 $S=(a_1,a_2,\cdots,a_n)$，则其栈底元素为

a_1,栈顶元素为 a_n,栈的容量(即栈所能容纳元素的最大个数)为 n(其上限,在顺序、链式存储结构下分别不得超过数组元素、指针的上限个数)。因此,栈实质上是其运算受单端化限制的线性表。

```
算法 Eg01210                          {压缩存储的稀疏矩阵之存放运算算法}
{{变量: aij,i,j,k,m,n}}               {变量定义}
> > >                                 {算法开始}
{按行逐列取出并统计稀疏矩阵 A 中非零元素的个数处理部分}
m←6; n←7; k←0                         {取得稀疏矩阵 A 的行数、列数,非零元素个数计数器初值}
对 i←1,m                              {外层循环结构框:用外循环变量 i 控制稀疏矩阵 A 的行数}
  对 j←1,n                            {内层循环结构框:用内循环变量 j 控制稀疏矩阵 A 的列数}
    行输出 "请输入稀疏矩阵 A 的第",i,"行、第",j,"列元素:"   {提示输入}
    输入 aij                          {用简单变量 aij 存放所输入的稀疏矩阵 A 的元素 a_ij}
    如果 aij < > 0                    {选择结构框:选择条件是"矩阵元素 a_ij 是非零元素吗?"}
      真  k←k + 1                     {选择条件为真(即成立)时:统计非零元素个数}
      假                              {选择条件为假(即不成立)时:不作任何操作(空操作)}
{定义压缩存储稀疏矩阵 A 的二维数组 b 处理部分}
{{数组: b[k + 1,3]}}                  {注意:只有此时才可最后确定数组 b 的实际大小(行数)!}
{生成压缩存储的二维数组 b 处理部分}
b[1,1]←m; b[1,2]←n; b[1,3]←k          {矩阵 B 首行存放矩阵 A 行、列数,非零元素个数}
k←0                                   {重新取得非零元素个数计数器初值}
对 i←1,m                              {外层循环结构框:用循环变量 i 控制稀疏矩阵 A 的行数}
  对 j←1,n                            {内层循环结构框:用循环变量 j 控制稀疏矩阵 A 的列数}
    行输出 "请输入稀疏矩阵 A 的第",i,"行、第",j,"列元素:"   {提示输入}
    输入 aij                          {再次重新输入稀疏矩阵 A 的元素 a_ij}
    如果 aij < > 0                    {稀疏矩阵元素 aij 是非零元素吗?}
      真  k←k + 1                     {再次重新统计稀疏矩阵 A 中非零元素个数}
          b[k + 1,1]←i; b[k + 1,2]←j  {存放第 k 个非零元素所在行数、列数}
          b[k + 1,3]←aij              {存放第 k 个非零元素之值}
      假
行输出 "OK!"
!!!                                   {算法结束}
```

图 1-8　压缩存储的稀疏矩阵之存放运算算法之 NS 周图

栈的存储结构,可采用顺序存储结构(如图 1-9 中图(a)所示),也可采用链式存储结构(如图 1-9 中图(b)所示),且前者运算的实现比后者更为简便。图中,为了动态描述栈中经压入或弹出后的栈顶元素是谁,必须使用栈顶指示变量 Top(常简称栈顶指针)来指明。其中:栈顶指针 Top,当采用顺序存储时为存放栈顶元素序号(即下标)值的简单变量,当采用链式存储时为存放栈顶元素首地址值的指针变量。

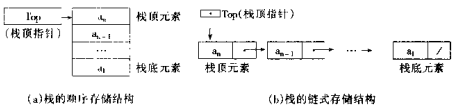

图 1-9　栈的存储结构示意图

由于栈中元素只能从栈顶进出，故其元素的特点是“先进后出，后进先出”。通常，栈可施行如下基本运算：

①创建——新建一个空栈。

②压入——在栈顶加入一个新元素。

③察看——察看栈中当前栈顶元素。

④弹出——取出(删除)栈中当前栈顶元素。

现只以顺序存储结构下栈的压入、察看、弹出运算为例，简要说明栈的运算实现方法。

例 1.2.11　设某栈的存储结构采用顺序存储结构，其容量为 100。试将任给 m($1 \leqslant m \leqslant 100$)个自然数压入该栈，然后弹出并察看该栈所存放的这些自然数。

问题分析　显然，若用一维数组 s 表示栈 S，n 表示栈的容量，Top 表示栈顶指针，则数组元素 a[1]为栈中的栈底元素(即最先压入的元素)a_1，而 a[Top]为栈 S 中的栈顶元素(即最后压入的元素)a_n。自然，栈指针 Top 之值将随压入和弹出而动态变化。当 Top = n 时，表示此栈已满，再不能往栈中压入新元素；当 Top = 0 时，表示此栈为空栈，已不能再从栈中弹出元素。因此，本题算法设计要点如下：

①压入运算设计要点——欲将新元素压入栈，首先要看栈是否未满，即其栈顶指针是否满足 Top < n：若是，则可压入，否则因栈已满而不删压入新元素，并应及时输出“本栈已上溢!”的警告信息。压入时，必须先用累加器“Top←Top + 1”向上移动栈顶指针，然后才可用“s[Top]←x”压入新元素。

②弹出运算设计要点——欲从栈中弹出其栈顶元素 a[Top]，首先要看栈顶指针 Top 是否为 0：若 Top = 0，则表明栈已空而无元素可供弹出，故应退栈(即及时输出“本栈已下溢!”的警告信息)；否则(即 Top≠0)，才可弹出当前栈顶元素 a[Top]并逆序(即与原输入时相反的顺序)输出之，然后立即用累减器“Top← Top - 1”向下移动栈顶指针。显而易见，当输入时的第一个自然数被最后弹出后，此栈已为空栈!

据此，可得其算法 Eg01211，并可用 NS 周图描述如图 1-10 所示。

1.2.5　队及其基本运算与存储结构

队列(Queus，简称队)是数据处理中应用较广泛的一种常用基本数据结构。它是一种动态数据结构，是线性表的一种较复杂特例。

队，是加入(也称进队，即插入)固定在表的一端，而删除(也称出队)固定在表的另一端的线性表。有如列车各车箱进出单行隧道一样，队的元素只能从一端进、另一端出(如图 1-11 所示)。队中，允许加入的一端称为队尾(Rear 或 Tail)或称队顶(Top)，并称队的尾端元素(即位于队尾者)为排尾；允许删除的一端称为队首(Front 或 Head)或称队底(Bottom)，并称队的首端元素(即位于队首者)为排头。若记队为 $Q = (a_1, a_2, \cdots, a_n)$，则其排头为 a_1，排尾为 a_n，n 为队的