程序员级软件知识

软件人员水平考试辅导教材

张　然
马学强
顾若平

复旦大学出版社

软件人员水平考试辅导教材

程序员级软件知识

张　然　马学强　顾若平　编著

复旦大学出版社

内 容 提 要

本书由复旦大学计算机系编写。主要内容有，数据结构、程序结构、程序语言、文件系统等四大部分的基础知识，在例题选解中，大部分例题是85、86、87年水平考试试题的分析。并配有大量习题及答案。

由于作者辅导过历次水平考试 经验丰富、把握方向正确，所以，本书不失为水平考试的好教材，也能供一般科技人员、大、中学师生、干部等有关计算机人员学习使用。

程序员级软件知识

张 然 马学强 顾若平 编著

复旦大学出版社出版

（上海国权路579号）

新华书店上海发行所发行 复旦大学印刷厂印刷

开本 850×1168 1/23 印张 8 插页 0 字数 227,000

1988年11月第1版 1992年 3 月第3次印刷

印数 10,001—20,000

ISBN7—309—00098—6/T·02

定价：3.00元

前　言

现代化的社会离不开计算机。计算机与社会各方面的紧密联系，促进了整个社会的发展。为使我国社会主义建设事业能稳步向前发展，社会需要多层次结构的计算机应用人才。计算机应用软件人员水平考试（以下简称水平考试）就是一种衡量计算机应用方面设计能力、编程能力及综合性知识水平的标准。

自1985年上海市首次举行水平考试以来，水平考试每年举行一次。1987年起已普及到全国大多数省、市、自治区。水平考试设有“程序员级”和“高级程序员级”两级。从1989年起将增设“系统分析员级”。水平考试得到了国家政府领导部门的高度肯定和热情支持，通过考试者将发给证书。

我们在历年水平考试辅导讲课时，学员们普遍反映需要一套内容齐全，紧扣大纲的水平考试辅导书籍。为此，我们在水平考试辅导讲课的基础上，深入分析了考试大纲、历年考题及考试动向，参考了大量国内外资料，编写了这套水平考试教材。即：程序员级硬件知识、程序员级软件知识、高级程序员级硬件知识、高级程序员级软件知识、综合知识、系统分析员级硬件知识、系统分析员级软件知识。在编写中，力求做到突出重点，强调概念，条理清晰，内容广而不杂。每一章都有例题分析，希望读者能细心揣摩，举一反三。每章后还附有大量习题供读者练习巩固。

本丛书为参加水平考试者的必备资料。此外，本丛书也适合于大、中学生，工程技术人员深入学习计算机科学之用，本丛书还可用作报考计算机科学系研究生的复习资料。

本丛书在编写过程中，得到了复旦大学计算机科学系有关教师、研

究生及复旦大学出版社的鼎力相助，在此一并致谢。

编　者

1987年12月于复旦大学

目　录

第一章　数据结构基础知识

计算机程序的加工对象是数据。“数据结构”(data structure) 就是关于数据的特性以及各类数据之间关系的学科。这里我们主要介绍数据的各种逻辑结构(线性表,栈,队列,树,图等)以及用于对这些逻辑结构进行操作的各类算法。在本章的例题选解中，我们都采用流程图或 PASCAL 语言描述算法。

1.1 表(List)

我们先介绍最简单的数据结构——线性表及其两种特殊形式——栈和队列，然后介绍线性表的一种存贮方式——链接表及其各种具体形式。数组是多维表。串是一种字符型的线性表。

1.1.1 线性表(Linear List)

线性表 L 是由 n(n≥0) 个结点组成的有限序列 $(k_1,k_2,\cdots,k_n)$。其中 k_1 和 k_n 分别称为开始结点(或始结点) 和终端结点(或终结点)。当 $2\leqslant i\leqslant n-1$ 时,称 k_{i-1} 为 k_i 的直接前趋，k_{i+1} 为 k_i 的直接后继。当 $n=0$ 时，L 称为空表。

在线性表 $L=(k_1, k_2, \cdots, k_n)$ 上实行的常规操作有：

1) 查找 L 中给定值的结点的位置。

2) 在 L 中的指定位置上插入新的结点。

3) 删除 L 中的某个结点。

4) 按某种方式重新排列线性表中的结点(如分类)。

在计算机内，线性表有各种不同的存贮方式。最常见的是用数组形式存贮——顺序存贮。其他存贮方式还有链接存贮、压缩存贮、索引存贮和散列存贮等。

1.1.2 栈和队列 (Stack and Queue)

栈是一种特殊的线性表 $S=(a_1, a_2, \cdots, a_n)$。它的开始结点 a_1 称为栈底结点，终端结点 a_n 称为栈顶结点。对 S 的所有插入或删除操作都只能在栈顶进行。栈也叫作后进先出 (LIFO) 表或先进后出 (FILO)表，意即最后进栈的结点最先出栈，最先进栈的结点最后出栈。

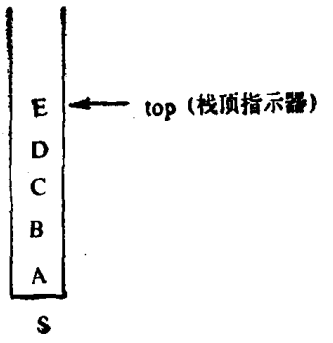


图 1.1 栈

栈的顺序存贮方式可以用一个线性表外加一个栈顶指示器 (top) 来实现(图 1.1)。

用一维数组S[1..m]表示线性表时，对栈 S 进行的四种基本操作可叙述如下：

1) empty(S)　判别栈是否为空，即判别 top>0 是否成立。当栈为空时称为下溢。

2) full(S)　判别栈是否已满，即判别top>m是否成立，栈已满时称为上溢。

3) push(x)　将值为x的结点送入栈S内(压入)。

4) pop(x)　从栈S内取出栈顶结点，把值放入x(弹出)。

当有 n 个栈时，也可用一个一维数组 S[1 ··m]来存放多个栈实现空间共享。

在程序设计中栈是一种非常有用的数据结构。子程序的调用与返回处理大多是用栈来实现的。递归算法也可以用栈结构来实现。

队列也是一种特殊的线性表 $Q=(b_1, b_2, \cdots, b_n)$，其中的 b_1 称为首部结点，b_n 称为尾部结点。队列只能在始端删除结点，在终端插入结点。队列

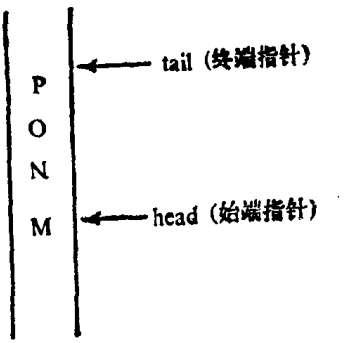


图1.2 队列

也叫做先进先出(FIFO)表(图1.2)。

双向队列是可以在首尾两端进行删除和插入操作的线性表。

队列Q的顺序存贮可用一个数组q[1..n]及首、尾指针：head和tail来实现。一般作法是把q[1..n]看成是首尾相接的环形结构（图1.3)。即把q[1]看成是q[n]的后继。这时的队列称为环形队列。

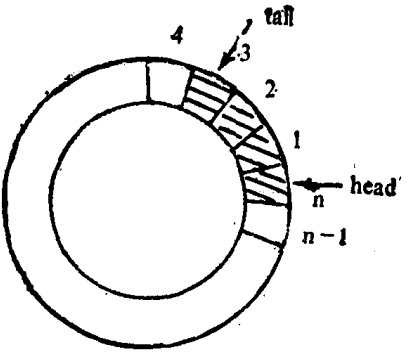


图1.3 环形队列

对队列来说也有判空,判满,插入,删除等四种基本操作。

1.1.3 链接表 (Chained List)

对顺序存贮的线性表作插入或删除操作时需移动原有的结点。同时，由于用来存贮线性表的数组的大小是预先给定的，表的结点的个数受到数组大小的限制，又往往会造成存贮空间的浪费。为克服这些缺点，提出了线性表的链接存贮方式。以链接存贮方式存贮的线性表叫作链接表(或链表)。链表中每个结点都是一个记录，其中除了有自己本身的一些信息外,还应有一个指示其后继结点存放地址的指针(图1.4)。链表的最后一个结点的指针值应该取一个不是地址的特殊值

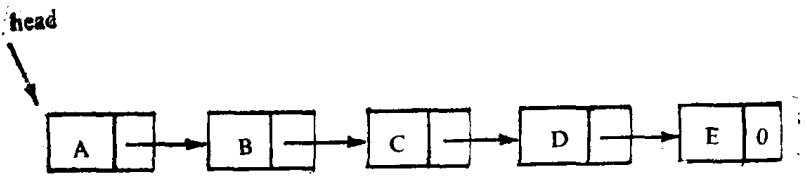


图1.4 链接表

(例如0),用来表示表的结束。另外,对链表来讲还应有一个指示第一个结点位置的首指针(head)。

应该注意的是，此时线性表中的相邻结点不一定是连续存放的。各结点的存贮地址是动态分配的。

由于栈和队列是特殊的线性表，它们自然也可以用链接存贮方式存贮,称为链接栈和链接队列(图1.5)。此时，不必预先规定栈和队列的大小，也不必为它们分配一个连续的存贮区域。

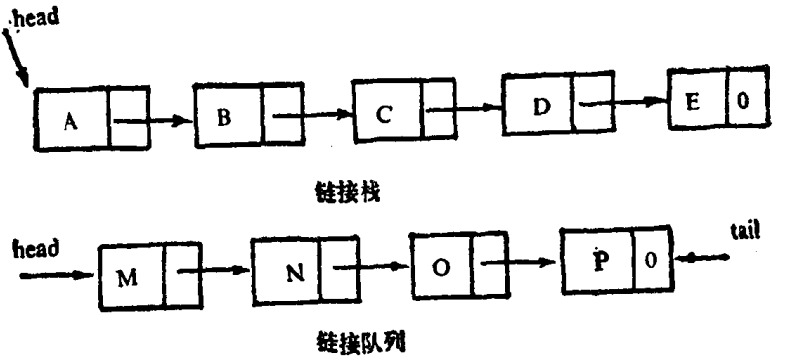


图1.5 链接栈和链接队列

PASCAL 语言中的标准过程 new(p)和dispose(p) 就是将存贮器中的空闲存贮区域组成一个空闲链接栈来实现的。

下面介绍一下环形链表和双向链表。

环形链表就是使最后一个结点的指针指向第一个结点的链接表。环形链表的优点是能从表中任一结点出发依次访问表中所有结点。

作为一种技巧，环形链表常引入一个称为“哨兵”的附加结点作为表头结点，从而使空表(仅有表头结点)与非空表的操作得以统一 。

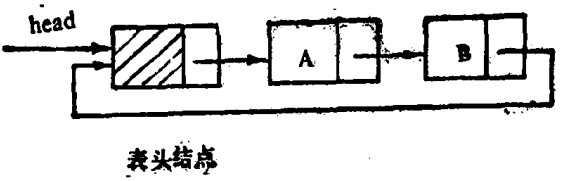


图 1.6 环形链表

当链表中每个结点都有两个指针，左(右)指针指向其前趋结点，右(左)指针指向其后继结点时，称作双向链表。

把双向链表和环形链表结合起来，可得到带表头结点的双向环形链表(图1.7)。

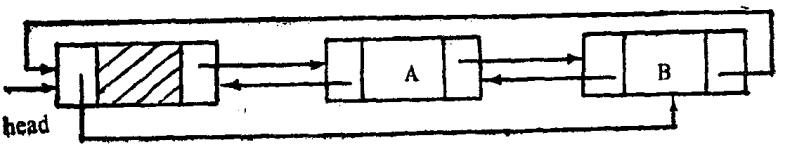


图1.7 双向环形链表

1.1.4 数组(Array)

数组是线性表的推广。数组的顺序分配就是用一个连续的主存区域存贮数组。两维数组的顺序存贮又分为按行顺序存贮和按列顺序存贮两种。

n维数组(n≥2)可看成是每个结点都是一个(n－1)维数组的线性表。

假设A是一个n维数组：

A：ARRAY [$l_1..u_1$, $l_2..u_2$, …, $l_n..u_n$]OF integer；

为了顺序存贮数组A需 $N(A)=\prod_{i=1}^{n}(u_i-l_i+1)$个存贮单元(假设每个元素只占一个存贮单元)。

如果按行存贮方法顺序存贮，并用$\alpha A[i_1,i_2,\cdots,i_n]$表示元素$A[i_1,i_2,\cdots,i_n]$的地址，则有：

$$\alpha A[i_1,i_2,\cdots,i_n]=\alpha A[l_1,l_2,\cdots,l_n]+\sum_{j=1}^{n}[(i_j-l_j)\prod_{k=j+1}^{n}(u_k-l_k+1)]$$

当n＝2时就是二维数组的情形。二维数组可用一个矩阵来表示。例如对数组 A

A：ARRAY[1..m, 1..n] OF integer；

我们可用

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

来直观地表示A，其中a_{ij} = A[i, j]。显然当按行顺序存贮时我们有

$$\alpha a_{ij} = \alpha a_{11} + n(i-1) + (j-1)$$

而按列顺序存贮时，则有

$$\alpha a_{ij} = \alpha a_{11} + n(j-1) + (i-1)$$

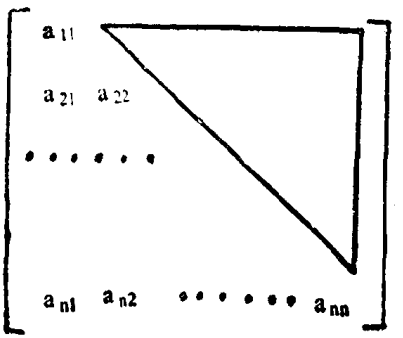
下面我们着重介绍几种特殊二维数组的存贮方式。

1）三角矩阵和对称矩阵

设有方阵A

A:ARRAY[1..n,1..n]OF integer;

若A的对角线以上(或以下)的元素均为零，则A称作三角矩阵。



若A满足$a_{ij} = a_{ji}(1 \leqslant i,j \leqslant n)$，则把A叫作对称矩阵。三角矩阵和对称矩阵只需存贮对角线以下(或以上)的元素即可。

作为例子，我们只考虑按行顺序存贮如下左三角矩阵的情况。设有矩阵A为(见左图)。

此时，我们有

$$\alpha a_{ij} = \alpha a_{11} + i(i-1)/2 + (j-1)$$

所占用的存贮单元个数为

$$N(A) = n(n+1)/2$$

2） 带状矩阵

全部非零元素都在一个以主对角线为中心的对称带状区域内的方阵称为带状矩阵(如图 1.8)。

设带状区域中在主对角线一侧的非主对角线的个数为b，则带宽为 $2b+1$。当 $|i-j|>b$ 时 $a_{ij}=0$，对于按行顺序存贮，我们有

$$\alpha a_{ij}=\alpha a_{11}+2b(i-1)+(j-1)$$

$$(1\leqslant i,j\leqslant n,|i-j|\leqslant b)$$

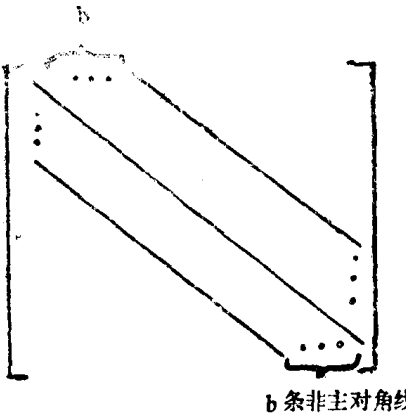


图1.8 带状矩阵

3）稀疏矩阵

稀疏矩阵是其中有很多元素为零的二维数组。为节约存贮空间起见，我们一般以压缩存贮方式（例如三元组表示法）或链接存贮方式（例如十字链表）存贮稀疏矩阵。

三元组表示法是用三元组(i, j,value)表示稀疏矩阵A的每个非零元素A[i,j]=value≠0。然后按行的递增次序存贮。图1.9是稀疏矩阵及其三元组表示的一个例子。

$$A=\begin{pmatrix}15&0&0&22&0&-15\\0&11&3&0&0&0\\0&0&0&-6&0&0\\0&0&0&0&0&0\\91&0&0&0&0&0\\0&0&28&0&0&0\end{pmatrix}$$

稀疏矩阵A

T：

	1	2	3
0	6	6	8
1	1	1	15
2	1	4	22
3	1	6	−15
4	2	2	11
5	2	3	3
6	3	4	−6
7	5	1	91
8	6	3	28

A的三元组表示 T

图1.9 稀疏矩阵的三元组表示

其中的T[0,1]，T[0,2]，T[0,3]分别表示稀疏矩阵的行数、列数及非零元素的个数。

稀疏矩阵A的十字链表表示法，是将A的每一列用一个带表头结点的环形链表表示，每一行也用一个带表头结点的环形链表表示。链表中的每个结点除包含所表示的A中非零元素的行号、列号及值等信息外，还有两个指针，分别用来指向同一行中的下一个非零元素和同一列中下一个非零元素。

1.1.5 串(string)

串是字符的有限序列，通常记作 $A='a_1a_2\cdots a_n'$。A是串名，$a_1a_2a_3\cdots a_n$为A的值。$a_i(1\leqslant i\leqslant n)$是字符型常量，我们用$\Phi$表示空白符，在对串作删除操作时我们常用$\Phi$来代替被删字符。

串可用一个字符型数组来顺序存贮，也可用链表存贮。用链表存贮时，先将A分成若干块，然后把各块依次链接起来，成为一个链表。链表表示法又可按各块长度是否相等分为等量分块表示法和不等量分块表示法。

串的操作比较复杂，常见的有：在串中查找一个子串，向串中插入或从串中删除一个子串等。

1.2 排序(Sorting)

排序就是按照关键码的上升顺序或下降顺序重新排列文件中的记录的过程。排序也称作分类。

设有文件F由记录$R_1,R_2,\cdots,R_n$组成，记录的关键码为k_i，排序即是把F重排成

$$F'=(R_{p(1)},R_{p(2)},\cdots,R_{p(n)})(1\leqslant p(i)\leqslant n,1\leqslant i\leqslant n)$$

使得满足下式：

$$K_{p(1)}\leqslant K_{p(2)}\leqslant\cdots\leqslant K_{p(n)}。$$

当在文件F中有两个记录R_i和R_j其关键码相等，即：$K_i=K_j$，且$i<j$时，若用某排序方法得到的已排序文件F'中R_i仍排在R_j前面，则

称此排序方法是稳定的。

排序可分为内部排序和外部排序两大类。

内部排序是对存放在主存里的文件进行排序。外部排序是指主存中只存放文件的一部分记录，记录需要在主存和辅存之间移动的排序。外部排序用于对大型文件进行排序。

下面我们着重介绍内部排序的几种方法。以下设文件长度为n，并只考虑按上升顺序的排序。

1.2.1 选择排序法

选择排序法的步骤是把下述挑选步骤执行n－1次：

1）在第i次挑选时，我们从记录R_i，R_{i+1}，…，R_n中选出具有最小关键码的记录r。

2）把r和R_i进行交换。

例如对于具有记录8,4,3,6,9,2,7的文件，选择排序法的排序过程如下：(标有下划线的两个元素进行交换，下划线的元素只有一个时不必进行交换)

8, 4, 3, 6, 9, 2, 7
2, 4, 3, 6, 9, 8, 7
2, 3, 4, 6, 9, 8, 7
2, 3, 4, 6, 9, 8, 7
2, 3, 4, 6, 7, 8, 9

算法所需的平均比较次数约为$n^2/2$次，记录的最大移动次数为3n次。

选择排序法是不稳定的。适宜于小型文件的排序。

1.2.2 插入排序法

插入排序法的基本步骤是把一个记录r插入到一已排序序列R_1，R_2，…，R_i中，使长度为i＋1的结果序列也为已排序序列。因此应先假设一个关键值为K_0的记录R_0，且设$R_0=r$，然后在已排序的序列R_1，R_2…，R_i中找出r的插入位置，即把r依次与R_i，…，R_1相比较，把比它