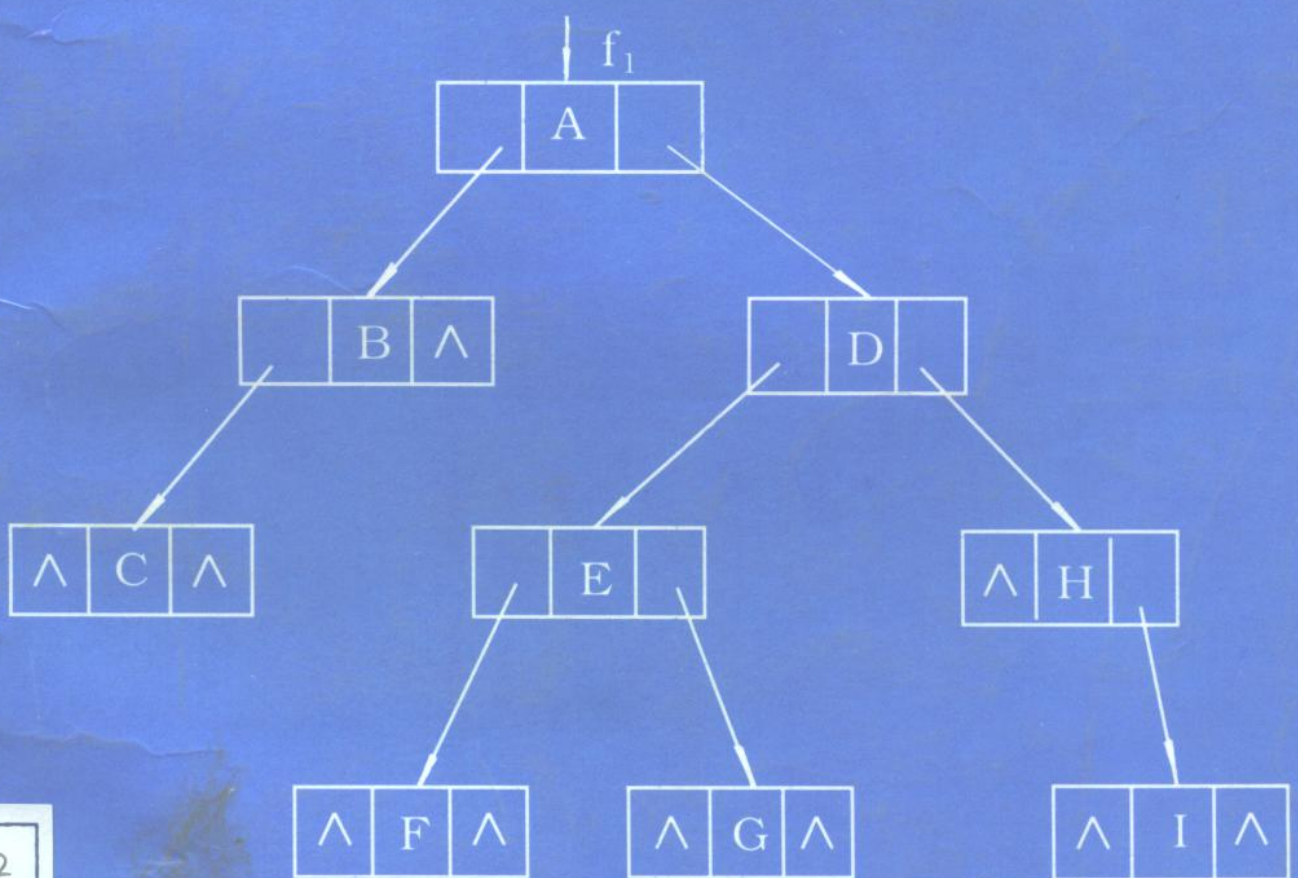
数据结构简明教程

徐 孝 凯 编著



清华大学出版社

数据结构简明教程
徐孝凯 编著
清华大学出版社

数据结构简明教程

徐孝凯　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书是为大专类和本科少学时类编写的“数据结构”课程的教材。全书共分八章，分别为：绪论、线性表、链接表、树、图、查找、排序和文件。本书以数据的三大逻辑结构——线性结构、树结构和图结构为主线，以类 pascal 语言为描述语言，详细分析了每一种逻辑结构并讨论了其对应的各种存储结构以及相应算法，每章均有习题，书后附有部分习题解答。

图书在版编目(CIP)数据

数据结构简明教程/徐孝凯编著. —北京：清华大学出版社，1994
ISBN 7-302-01663-1

Ⅰ.数… Ⅱ.徐… Ⅲ.数据结构-教材 Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字（94）第 15100 号

TS38/12

出 版 者：清华大学出版社(北京清华大学校内，邮编 100084)
责任编辑：徐培忠
印 刷 者：北京市海淀区清华园印刷厂
发 行 者：新华书店总店科技发行所
开 本：787×1092 1/16 印张：16 字数：377 千字
版 次：1995 年 4 月第 1 版 1995 年 4 月第 1 次印刷
书 号：ISBN 7-302-01663-1/TP・712
印 数：0001-8000
定 价：13.80 元

前　言

随着计算机应用领域的不断拓展，使得由计算机处理的数据愈加丰富和多样化。如何根据实际应用的要求，对数据进行有效地组织和存储，进而编制出相应运算的算法，是数据结构这门课程所要研究并加以解决的问题。数据结构也为操作系统和数据库等后续课程奠定基础。

从逻辑角度看，数据可归结为三种基本结构：线性结构、树结构和图结构；从存储角度看，数据可归结为四种基本结构：顺序结构、链接结构、索引结构和散列结构。每一种逻辑结构可根据不同需要采用不同的存储结构，或者不同存储结构的组合。数据的逻辑结构和存储结构确定后，再结合指定运算的算法，就容易利用一种程序设计语言编写出程序。通过数据结构课程的学习，能够大大提高自己的软件开发能力和设计水平。

《数据结构简明教程》是为大专类学生和广大自学读者学习数据结构课程而精心编著的一本教材。本书内容虽然取材广泛，但注意删繁就简，着重实用；在叙述上力求概念明确，由浅入深、前后衔接和呼应；在算法分析上较细致，读者便于自学。全书共分为八章。第一章为绪论，它为学习以后各章做好准备。第二至第五章分别为线性表、链接表、树和图，它以线性结构、树结构和图结构为主线，分别讨论了每一种逻辑结构所对应的存储结构以及相应运算的算法。第六章和第七章分别为查找和排序，它包含了在数据处理领域中主要使用的几种查找方法和内排序方法。第八章为文件，简述了外部磁盘和磁带存储器的存取特性以及文件的各种存储结构。本书讲授时数为 60～70 学时。

对于大专类学生，本书目录中标记 * 号的内容可不作为教学要求。若把这些内容也考虑进去，则本书可作为本科学生学习数据结构的教材或参考书。

本书中的算法都是以“类 pascal 语言”编写的，建议读者在使用本书前最好具有 pascal 语言的基础。

本书由北京大学许卓群教授和北京航空航天大学唐发根副教授共同审定，认为该书系统性强、结构紧凑、内容充实、注重实用，叙述条理清楚、层次分明、循序渐进、便于自学，具有一定特色。在审定过程中，两位专家认真审阅了全部书稿，提出了宝贵意见，在此向他们表示衷心感谢。

由于本人学识浅薄，加之时间仓促，错误和不足之处在所难免，敬请专家和广大读者批评指正。

徐　孝　凯

1994 年 10 月

目　录

第一章 绪 论

自1946年美国第一台电子计算机问世以来，计算机科学和软硬件技术得到了飞速的发展，与此同时，计算机的应用领域也从最初的科学计算逐步发展到人类活动的各个领域。现在，计算机处理的对象不仅是简单的数值或字符，而且是带有不同结构的各种数据。因此，要设计出一个较好的软件，除了要掌握所用的计算机语言外，还要研究各种数据的特性和数据之间存在的关系。这就是“数据结构”这门学科形成和发展的背景。

1-1 基本术语

这一节，我们将对全书中最常用的名词和术语赋以确定的含义。

数据(data)是人们利用文字符号、数字符号以及其它规定的符号对现实世界的事物及其活动所做的描述。因此，大到一本书、一篇文章、一张图表等是数据，小到一条句子、一个单词、一个算式、以至一个数值、一个字符等都是数据。在计算机领域，人们把能够被计算机加工的对象，或者说能够被计算机输入，存储、处理和输出的一切信息都叫做数据。

数据元素(data element)是一个数据整体中相对独立的单位。如对于一个文件来说，每个记录就是它的数据元素；对于一个字符串来说，每个字符就是它的数据元素；对于一个数组来说，每一个成分就是它的数据元素。数据和数据元素是相对而言的，如对于一个记录来说，它相对于所在的文件被认为是数据元素，而它相对于所含的数据项又被认为是数据。因此，在本教材中，对数据和数据元素这两个术语的使用并不加以严格区别。

数据记录(data record)简称**记录**，它是数据处理领域组织数据的基本单位。它又由更小的单位——**数据项**(item)所组成，一个记录一般包括一个或若干个固定的数据项(当然每一个数据项还可以是记录的形式)。就拿对图书目录管理来说，每个记录表示一本图书的目录信息，如表1-1所示。

表1-1 图书目录表

登录号	书 号	书 名	作者	出版社	定价
00001	ISBN 7-04-003907-9/TP·103	计算方法	唐 珍	高等教育	4.80
00002	ISBN 7-111-03247-0/TP·158	计算机辅助制造	李德庆	机械工业	3.30
00003	ISBN 7-118-00994-6/TP·126	FORTRAN90学习指南	王文才	国防工业	11.00
00004	ISBN 7-302-00984-8/TP·363	数据结构	严蔚敏	清华大学	6.05
00005	ISBN 7-5609-0657-5/TP·65	微型计算机及其应用	周细等	华中理工大学	5.15
00006	ISBN 7-302-00860-4/TP·312	C程序设计	谭浩强	清华大学	7.30
00007	ISBN 7-03-001460-X/TP·101	计算机图学	孟粹娟	科 学	11.30
⋮	⋮	⋮	⋮	⋮	⋮

在表 1-1 中，第一行为表目行或目录行，它给出了该表中每条记录的结构。从表目行向下的每一行为一条记录，每一列为一个数据项，每条记录都由 6 个数据项组成，其名称分别为登录号、书号、书名、作者、出版社和定价。当记录不同时，所对应的同一数据项的值可能相同，也可能不同。例如对于第 4 条和第 6 条记录来说，出版社数据项的值就相同，同为“清华大学”；对于第 1 条和第 3 条记录来说，书名数据项的值就不同，一个为《计算方法》，另一个为《FORTRAN 90 学习指南》。

在一个表或文件中，若所有记录的某个数据项的值都不同，也就是说，每个值能够唯一地标识一个记录时，则可把这个数据项作为记录的**关键数据项**，简称**关键项**(key item)，关键项中的每一个值称作为所在记录的**关键字**(key word 或 key)。在表 1-1 中，登录号数据项的值都不同，所以可把登录号作为记录的关键项，其中的每一个值都是所在记录的关键字，如 00002 为第 2 条记录的关键字，00005 为第 5 条记录的关键字等。

在一个表或文件中，能作为关键项的数据项可能没有，可能只有一个，也可能多于一个。当没有时，可把多个有关的数据项联合起来，构成一个组合关键项，用组合关键项中的每一个值来唯一地标识一个记录。

引入了记录的关键项和关键字后，为简便起见，在以后的讨论中，经常利用关键项来代替所有记录，利用关键字来代替所在的记录。

数据处理(data processing)是指对数据进行查找、插入、删除、合并、排序、统计、简单计算、输入、输出等的操作过程。在早期，计算机主要用于科学和工程计算，进入 80 年代以后，计算机主要用于数据处理。据有关统计资料表明，现在计算机用于数据处理的时间比例平均高达百分之八十以上，随着时间的推移和计算机应用的进一步普及，计算机用于数据处理的时间比例必将进一步增大。像计算机情报检索系统、经济信息管理系统、图书管理系统、物资调配系统、银行核算系统、财务管理系统等都是计算机在数据处理领域的具体应用。数据结构是数据处理的软件基础，因此，数据结构课程是计算机所有专业的主干课程之一。

数据结构(data structure)，简单地说是指数据以及数据之间的联系。上面提到，数据的描述对象是现实世界的事物及其活动，而任何事物及其活动都不是孤立存在的，都是在一定意义上相互联系、相互影响的，所以数据之间必然存在着联系。由于这种联系是内在的，或根据需要人为定义的，所以被看作为“逻辑”上的联系，因此，又把数据结构称作为数据的**逻辑结构**。数据结构在计算机存储器上的存储表示称作为数据的**物理结构**或**存储结构**。由于存储表示的方法有顺序、链接、索引、散列等多种，所以，一种数据结构可表示成一种或多种物理结构。

为了更确切地描述数据结构，通常采用二元组表示：

$$B=(K,R)$$

B 是一种数据结构，它由数据元素的集合 K 和 K 上二元关系的集合 R 所组成。其中

$$K=\{k_i \mid 1\leqslant i\leqslant n, n\geqslant 0\}$$

$$R=\{r_j \mid 1\leqslant j\leqslant m, m\geqslant 1\}$$

k_i 表示第 i 个数据元素，n 为 B 中数据元素的个数，特别地，若 $n=0$，则 K 是一个空集，因而 B 也就无结构而言，或者说它具有任何结构；r_j 表示第 j 个二元关系(以后均简称

关系),m 为 K 上关系的个数。

在本书所讨论的数据结构中,一般只讨论 m=1 的情况,即 R 中只包含一个关系(R={r})的情况。对于包含有多个关系的数据结构,可分别对每一个关系进行讨论。

K 上的一个关系 r 是序偶的集合。对于 r 中的任一序偶<x,y>(x,y∈K),我们把 x 叫做序偶的第一元素,把 y 叫做序偶的第二元素,又称序偶的第一元素为第二元素的**直接前驱**,简称**前驱**,称第二元素为第一元素的**直接后继**,简称**后继**。如在<x,y>的序偶中,x 为 y 的前驱,而 y 为 x 的后继。

数据结构还能够利用图形形象地表示出来,图形中的每个结点(或叫顶点)对应着一个数据元素,两结点之间带箭头的连线(称作有向边或弧)对应着关系中的一个序偶,其中序偶的第一元素为有向边的起始结点,第二元素为有向边的终止结点。

作为例子,下面根据表 1-2 构造一些典型的数据结构。

表 1-2 教务处人事简表

职 工 号	姓 名	性 别	出生年月	职 务	单 位
01	万明华	男	1952 年 8 月	处 长	
02	赵 宁	男	1958 年 6 月	科 长	教材科
03	张 利	女	1954 年 12 月	科 长	考务科
04	赵书芳	女	1962 年 8 月	主 任	办公室
05	刘永年	男	1949 年 8 月	科 员	教材科
06	王明理	女	1965 年 4 月	科 员	教材科
07	王 敏	女	1962 年 6 月	科 员	考务科
08	张 才	男	1957 年 3 月	科 员	考务科
09	马立仁	男	1965 年 10 月	科 员	考务科
10	邢怀常	男	1966 年 7 月	科 员	办公室

表中共有 10 条记录,每条记录都由六个数据项所组成,由于每条记录的职工号各不相同,所以可把每条记录的职工号作为该记录的关键字,并在下面的例子中,我们将用记录的关键字来代表整个记录。

例 1 一种数据结构 linearity=(K,R),其中

K={01,02,03,04,05,06,07,08,09,10}

R={r}

r={⟨05,01⟩,⟨01,03⟩,⟨03,08⟩,⟨08,02⟩,⟨02,07⟩,⟨07,04⟩,
⟨04,06⟩,⟨06,09⟩,⟨09,10⟩}

对应的图形如图 1-1 所示。

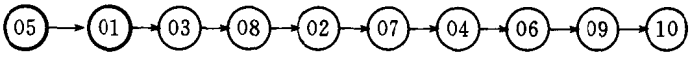


图 1-1 线性结构示意图

结合表 1-2,细心的读者不难看出;r 是按职工年龄从大到小排列的关系。

在 linearity 中,每个数据元素有且只有一个直接前驱元素(除结构中第一个元素 05

外),有且只有一个直接后继元素(除结构中最后一个元素 10 外)。这种数据结构的特点是数据元素之间的 1∶1 联系,即**线性关系**,我们把具有这种特点的数据结构叫做**线性结构**。

例 2 一种数据结构 tree=(K,R),其中

K={01,02,03,04,05,06,07,08,09,10}

R={r}

r={⟨01,02⟩,⟨01,03⟩,⟨01,04⟩,⟨02,05⟩,⟨02,06⟩,⟨03,07⟩,⟨03,08⟩,⟨03,09⟩,⟨04,10⟩}

对应的图形如图 1-2 所示。

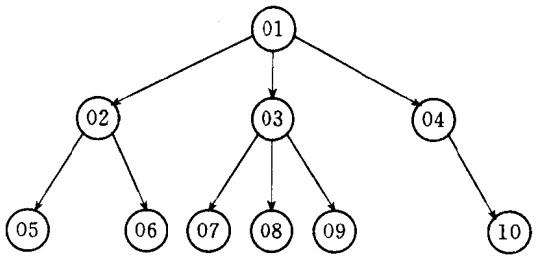


图 1-2 树结构示意图

结合表 1-2,细心的读者不难看出:r 是人员之间领导与被领导的关系。

图 1-2 象倒着画的一棵树,在这棵树中,最上面的一个没有前驱只有后继的结点叫作树根结点,最下面一层的只有前驱没有后继的结点叫作树叶结点,除树根和树叶之外的结点叫作树枝结点。在一棵树中,每个结点有且只有一个前驱结点(除树根结点外),但可以有任意多个后继结点(树叶结点可看作为具有 0 个后继结点)。这种数据结构的特点是数据元素之间的 1∶N 联系(N≥0),我们把具有这种特点的数据结构叫作**树型结构**或**树结构**。

例 3 一种数据结构 graph=(K,R),其中

K={01,02,03,04,05,06,07}

R={r}

r={⟨01,02⟩,⟨02,01⟩,⟨01,04⟩,⟨04,01⟩,⟨02,03⟩,⟨03,02⟩,⟨02,06⟩,⟨06,02⟩,⟨02,07⟩,⟨07,02⟩,⟨03,07⟩,⟨07,03⟩,⟨04,06⟩,⟨06,04⟩,⟨05,07⟩,⟨07,05⟩}

对应的图形如图 1-3 所示。

从图 1-3 可以看出,r 是 K 上的对称关系,为了简化起见,我们把⟨x,y⟩和⟨y,x⟩这两个对称序偶用一个无序对(x,y)或(y,x)来代替;在图形中,我们把 x 结点和 y 结点之间两条相反的有向边用一条无向边来代替。这样 r 关系可改写为:

r = {(01,02),(01,04),(02,03),(02,06),(02,07),(03,07),(04,06),(05,07)}

对应的图形如图 1-4 所示。

如果说 r 中每个序偶里的两个元素所代表的人员是好友的话,那么 r 关系就是人员之间的好友关系。

从图 1-3 或 1-4 可以看出,结点之间的联系是 M∶N 联系(M≥0,N≥0),也就是说,

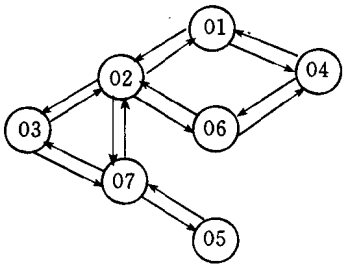


图 1-3　图型结构示意图

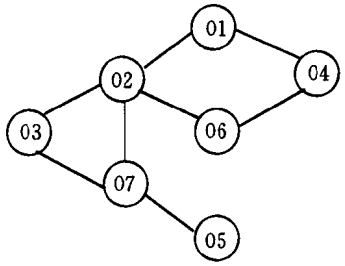


图 1-4　图型结构示意图

每个结点可以有任意多个前驱结点和任意多个后继结点。我们把具有这种特点的数据结构叫做**图型结构**。

从图型结构、树型结构和线性结构的定义可知，树型结构是图型结构的特殊情况（即 M＝1 的情况），线性结构是树型结构的特殊情况（即 N＝1 的情况）。为了区别于线性结构，我们把树型结构和图型结构统称为**非线性结构**。

例 4　一种数据结构 B＝(K,R)，其中

$K=\{k_1,k_2,k_3,k_4,k_5,k_6\}$

$R=\{r_1,r_2\}$

$r_1=\{\langle k_3,k_2\rangle,\langle k_3,k_5\rangle,\langle k_2,k_1\rangle,\langle k_5,k_4\rangle,\langle k_5,k_6\rangle\}$

$r_2=\{\langle k_1,k_2\rangle,\langle k_2,k_3\rangle,\langle k_3,k_4\rangle,\langle k_4,k_5\rangle,\langle k_5,k_6\rangle\}$

若用实线表示关系 r_1，虚线表示关系 r_2，则对应的图形如图 1-5 所示。

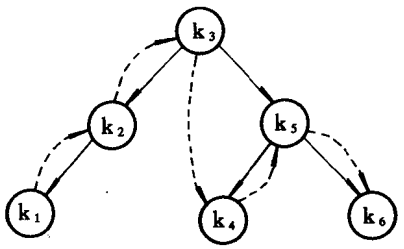


图 1-5 带有两个关系的数据结构示意图

从图 1-5 可以看出数据结构 B 是一种非线性的图型结构。但是，若只考虑关系 r_1 则为树型结构，若只考虑关系 r_2 则为线性结构。

算法(algorithm)，简单地说就是解决特定问题的方法（关于算法的严格定义，在此不作讨论）。特定的问题可分为数值的和非数值的两类。解决数值问题的算法叫做数值算法，科学和工程计算方面的算法都属于数值算法，如求解数值积分，求解线性方程组，求解代数方程，求解微分方程等。解决非数值问题的算法叫做非数值算法，数据处理方面的算法都属于非数值算法，如各种排序算法、查找算法、插入算法、删除算法、遍历算法等。数值算法和非数值算法并没有严格的区别，一般说来，在数值算法中主要进行算术运算，而在非数值算法中，则主要进行比较和逻辑运算。另一方面，特定的问题可能是递归的，也可能是非递归的，因而解决它们的算法就有递归算法和非递归算法之分。当然，从理论上讲，任何递归算法都可以通过循环、堆栈等技术转化为非递归算法。

在计算机领域，一个算法实质上是针对所处理问题的需要，在数据的逻辑结构和存储结构的基础上施加的一种运算。由于数据的逻辑结构和存储结构不是唯一的，在很大程度上可以由用户自行选择和设计，所以处理同一个问题的算法也不是唯一的。另外，即使对

于具有相同的逻辑结构和存储结构而言，其算法的设计思想和技巧不同，编写出的算法也大不相同。我们学习数据结构这门课的目的，就是要会根据数据处理问题的需要，为待处理的数据选择合适的逻辑结构和存储结构，进而设计出比较满意的算法。

1-2 算法描述

本书在进行算法描述时，采用“类 pascal 语言”作为工具。类 pascal 语言是我们在标准 pascal 语言的基础上所做的修改，它忽略了标准 pascal 语言中语法规则的一些细节，同时增加了一些功能较强的语句，这样使得描述出的算法清晰、直观、便于阅读和分析。读者根据这些算法不难编写出符合任一种 pascal 语言（如标准 pascal、IBM-PC pascal 或 Turbo pascal 等）或其它任一种语言语法规则的算法来。

类 pascal 语言所包含的语句如下：

1. 赋值语句

变量名：＝表达式；

2. 转向语句

goto 语句标号；

3. 调用过程语句

过程名(参数表)；

4. 退出循环语句

exit；

用于各种循环语句中，该语句被执行时，将立即退出当前循环，相当于 goto 到该循环语句后的第一条语句上，它是循环语句的一个非正常出口。

5. 返回语句

return；

用于过程或函数体中，该语句被执行时，将立即退出当前过程或函数，返回到调用前所保存的位置继续向下执行，它是执行过程或函数的一个非正常出口。

在函数体中使用该语句还可以书写成 return(表达式)的形式，表示把表达式的值赋给函数名后再退出当前被执行的函数，它相当于如下两条语句：

函数名：＝表达式;return；

6. 出错处理语句

error(字符串)；

在算法中为了避免非法操作，需要进行出错处理时统一使用该语句，它表示此算法的执行到此中止。在实际算法中，它可能是转去执行一个错误处理程序，也可能是简单地打印出错误信息并停止运行等。该语句括号内的字符串用以说明出错的原因，如使用字符串'overflow'表示溢出，'out of range'表示超出范围或越界等。

7. 复合语句

begin 语句 1;语句 2;…;语句 n end;

8. 条件语句

if 条件 then 语句 1[else 语句 2];

其中中括号部分可以省略。

9. 情况语句

格式 1：

```
case 变量名 of
  常量 1:语句 1;
  常量 2:语句 2;
    ⋮   ⋮
  常量 n:语句 n
end;
```

格式 2：

```
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;
```

10. for 循环语句

格式 1：

for 变量名：＝初值 to 终值 do 语句；

格式 2：

for 变量名：＝初值 downto 终值 do 语句；

若格式 1 中的初值大于终值，或格式 2 中的初值小于终值则都不会执行其循环体。

11. while 循环语句

while 条件 do 语句；

12. repeat 循环语句

repeat 一组语句 until 条件；

在这 12 种语句中，第 4、5、6 种语句和第 9 种的格式 2 语句是标准 pascal 语句所没有的，但在其它的 pascal 语言中具有类似的语句。

采用类 pascal 语言描述算法的书写规则如下：

1. 所有算法均以过程或函数说明的形式书写。算法的输入数据一般来自参数表，输出数据一般也由变参或函数名带回，这样就使得每个算法成为功能相对独立的一个模块。

2. 参数表中的参数一般不进行类型说明，若参数是变参，则规定为大写字母或大写字母开头的标识符表示，若参数是值参，则规定以小写字母或小写字母开头的标识符表示，当然作指定说明的除外。

3. 过程和函数体中的定义和说明部分一般被省略，语句部分通常按内容层次分别对语句进行编号，以便分析。第一层编号用(1)、(2)、(3)、……表示，第二层编号用(Ⅰ)、

(Ⅰ)、(Ⅱ)、……表示，第三层编号用(a)、(b)、(c)、……表示，并规定除第一层编号的外面不省略语句括号 begin 和 end 外，其余层次均省略。

例如，对于一维整型数组 A(1：n)，其中 1 和 n 分别表示该数组下标的下界和上界，若要求从下标 1 至 n(n≥1)的元素中查找出最大值和最小值，并把它们分别赋给变参 Max 和 Min，则算法描述为：

```
procedure find(A,n,Max,Min);
  begin
    (1) Max:=A[1]; Min:=A[1]; {赋初值}
    (2) for i:=2 to n do {查找过程}
      (Ⅰ) if A[i]>Max then Max:=A[i];
      (Ⅱ) if A[i]<Min then Min:=A[i]
  end;
```

此算法的描述完全符合上面所述的三条规则。

4. 在条件或循环语句中，出现多条简单语句时，为简单起见，也可不采用编号，而采用方括号代替 begin 和 end 语句括号。如：

```
if R[i]=T[j] then [i:=i+1; j:=j+1]
             else [i:=i-j+2; j:=1];
```

5. 当需要从若干个表达式中取其值最大者或最小者时，可简记为：

max(表达式1,表达式2,…);

min(表达式1,表达式2,…);

实际上这两个函数均可通过条件语句的嵌套或其它方法来实现。

6. 当两个变量 x 和 y 需要相互交换值时，可简记为：

x↔y;

实际上它对应下面三条赋值语句：

```
t:=x;x:=y;y:=t; {t 为临时工作变量}
```

此外，在利用类 pascal 语言描述算法时，还需要使用标准 pascal 语言中的所有数据类型、标准过程和函数等。

1-3 算法评价

对于解决同一个问题，往往能够编写出许多不同的算法。例如，对于排序问题，在第七章中将介绍多种算法。进行算法评价的目的，既在于从解决同一问题的不同算法中选择出较为合适的一种，也在于知道如何对现有算法进行改进，从而有可能设计出更好的算法。

一般从以下四个方面对算法进行评价

一、正确性

正确性是设计和评价一个算法的首要条件，如果一个算法不正确，其它方面就无从谈起。一个正确的算法是指在合理的数据输入下，能在有限的运行时间内得出正确的结果。

通过对数据输入的所有可能情况的分析和上机调试可以证明算法是否正确。当然，要从理论上证明一个算法的正确性，并不是一件容易的事，也不属于本课程所研究的范围，故不作讨论。

二、运行时间

运行时间是指一个算法在计算机上运算所花费的时间。它大致等于计算机执行一种简单操作(如赋值操作、转向操作、比较操作等)所需的时间与算法中进行简单操作次数的乘积。因为执行一种简单操作所需的时间随机器而异，它是由机器本身硬软件环境决定的，与算法无关，所以我们只讨论影响运行时间的另一个因素——算法中进行简单操作的次数。

不管一个算法是简单还是复杂，最终都是被分解成简单操作来具体执行的，因此，每个算法都对应着一定的简单操作的次数。显然，在一个算法中，进行简单操作的次数越少，其运行时间也就相对地越少；次数越多，其运行时间也就相对地越多。所以，通常把算法中包含简单操作次数的多少叫做算法的**时间复杂性**，它是一个算法运行时间的相对量度。

若解决一个问题的规模为 n，例如在排序问题中，n 表示待排元素的个数；在矩阵求逆中，n 表示矩阵的阶数；在图的遍历中，n 表示图中的顶点数。那么，算法的时间复杂性就是 n 的一个函数，通常记为 T(n)。下面通过例子来分析算法的时间复杂性。

算法1：累加求和

```
function sum(A,n):real;
                {A 表示一维实型数组,n 表示数组的大小}
  begin
    (1) s:=0; {给累加变量 s 赋初值}
    (2) for i:=1 to n do {进行累加求和}
          s:=s+A[i];
    (3) sun:=s {把 s 值(即累加和)赋给函数名}
  end;
```

计算机执行这个算法时，第(1)步和第(3)步都只需一次赋值操作，为了分析第(2)步包含多少简单操作的次数，可改写为：

```
(2)       i:=1;                          1次
     1:   if i>n then goto 2;            n+1次
          s:=s+A[i];                     n 次
          i:=i+1;                        n 次
          goto 1;                        n 次
(3)  2:   sum:=s
```

把第(2)步分解后的每一条语句的执行次数加起来，就得到了它包含的简单操作的次数，即为4n+2。因此，算法1的时间复杂性为：

$$T(n) = 4n + 4$$

算法2：矩阵相加