数据结构

刘美轮 编著

SHUJU JIEGOU

中央广播电视大学出版社

数据结构

数据结构

刘美轮　编著

中央广播电视大学出版社

前　言

《数据结构》在本世纪六十年代末期才开始单独作为一门课程出现，随着电子计算机科学的飞速发展和《数据结构》本身内容的不断充实，在不到二十年的时间内，这门课程已成为计算机科学的核心课程，它不仅是《操作系统》、《编译原理》、《算法设计与分析》、《计算机图形学》、《数据库》、《人工智能》等课程的基础，也是各行各业的计算机应用软件工作者必不可少的基本知识。

本书是为中央广播电视大学计算机应用专业编写的教材，选材基本上覆盖了有关数据结构的全部主要内容，计划讲授 60 学时，并配有一定的习题和上机实践练习。书中第一章介绍数据结构与算法的基本概念、本书所采用的算法描述语言及评价算法的一些原则；第二、三、四章分别讲述数组与线性表、链接表和串等基本的数据结构以及有关的算法和应用举例；第五章介绍以后几章要用到的递归概念及为递归问题设计非递归算法的方法；第六、七章介绍在实际应用中很重要的两种非线性数据结构——树和图；第八、九章讲述各种常用的排序和查找方法；最后一章介绍文件。

考虑到电大的要求和电视授课的特点，本书对各种数据结构和有关的算法多以实例来讲解，叙述较为详尽，更着重于实际应用，而略去一些理论推导和证明。为了便于读者自学，书中需要的预备知识在用到处大多先加以复习，因此本书可大体自成体系。

学习本书需要掌握一种程序设计语言，在各种高级语言中，以PASCAL 语言最为合适。但对于不懂 PASCAL 语言的读者，只要看懂了第一章关于算法描述语言的说明，也能够用其它高级语言将书中介绍的大部分算法编成相应的程序。

本书亦可供从事计算机应用的科学技术工作者和高等学校非计算机专业的师生参考。内容可视需要取舍，要求较低时可舍去如串、图、文件等章的全部或部分内容。书末的参考书目，尽可能列出了国内能见到的有关数据结构的译、著，需进一步深入学习的读者可以查阅。

在本书编写过程中，自始至终得到天津广播电视大学电气工程系负责人孙亦昌老师的关心和帮助，提出了不少原则性意见和具体意见。天津大学计算机系系主任柏家球老师在百忙之中认真审阅了全部书稿。根据他的宝贵建议和指正，作者作了再一次修改。此外，天津大学的张大钧老师提供了一些素材和教学经验，还有的老师参加了底图绘制和文字修改工作。谨在此一并致谢。

由于作者学识水平和教学经验的限制，编写过程又较仓促，错误和不足之处在所难免，希望读者和同行专家们批评指正。

作　者

一九八七年三月

目　录

第一章 绪 论

§1-1 数据结构与算法

什么是数据结构(Data structure)，目前尚没有一个统一的定义，我们这里采用比较一般的、通行的说法。所谓数据(Data)，就是一切能够由计算机接受和处理的对象，亦即信息。随着计算机硬件、软件和外部设备的不断发展，数据这一概念的含义越来越广泛。不仅整数、实数、复数等是数据，字符、人名、文字、表格、图形等也都能够由计算机接受和处理，也都是数据。一般来说，数据不是单个孤立的数或字符，而是由一批基本单位以一定的关系组织起来的。组成某种数据的基本单位叫做数据元素。所谓"数据结构"，不仅要描述某种数据，而且要描述此种数据中各数据元素之间的相互关系。这种关系往往是用一组有关数据元素的运算给出来。例如，数组是由有限数目的相同类型基本单位组成的，其数据元素可以是整数、实数、字符等，每个基本单位在数组中对应着一定的下标。作为一种数据结构，除了要定义每个数组的元素个数及类型以外，更重要的是要定义有关数组的一些运算。如读出某数组元素值、给某数组元素赋值或修改某数组元素的值、两数组元素互换位置以及更复杂一些的运算，如向数组中插入新元素或删除某数组元素等。又例如，集合也是一种数据结构，我们定义了两个集合的并、交、差运算以及判断一个元素是否属于某集合的运算等。此外，在 PASCAL 语言中的记录、文件、字符串等都是不同的数据结构。

数据结构可以是多层次的，即某数据结构的数据元素本身又是某种数据结构。例如，一数组的每个元素是一个记录，该记录的某个域是字符串，而字符串的元素又是字符，等等。

研究数据结构，包括研究数据的逻辑结构和数据的物理结构。所谓数据的逻辑结构，是指数据元素之间的逻辑关系，例如，同一数组中各元素的先后次序关系等；而数据的物理结构，则是考虑数据元素在计算机存储器中如何表示和安排，如数组元素或字符串各个字符在内存中是如何安排的等。本书以研究数据的逻辑结构为主，只在第二、四、十章中讨论关于数组、串和文件等的物理结构。因此本书中如不特别加以说明，所谓"数据结构"一般是指数据的逻辑结构，而将数据的物理结构称为存储结构。

除了程序设计语言本身所具有的数据结构以外，为了便于解决各种实际问题，在程序编制中常常还需定义其它一些数据结构，如后面将讲到的链接表和树等。即使是编程语言已有的数据结构，有时也需再增加一些该语言中原来没有的运算功能，例如对于数组，必要时还可增加数组元素的插入、删除运算等功能。不论新定义数据结构还是给原有的数据结构增添新的运算功能，都需在程序中编制相应的运算过程说明。

对一个特定的题目，如何组织数据(即采用哪些种数据结构)，以便于对数据进行存储和处理、

节省内存、提高运算速度，在程序编制中是很重要的问题。除此之外，另一个需要考虑的重要问题是解决此题目应采用什么算法(Algorithm)。数据作为计算机所能接受和处理的对象，相当于机械加工中的原材料、半成品和成品，而算法则相当于为加工出某一种产品所采用的加工工序和加工方法。

给"算法"这一概念下一个严格、准确的定义不很容易，在此我们仅对其做一个较简单的说明。算法是一个有穷的规则序列，这些规则决定了解决某一特定题目的一系列运算。由此题目的一定输入，依照这些规则，令计算机按部就班地进行计算，经过有限的计算步骤后能得到一定的输出。

进行计算机的程序设计，除了需要某种程序设计语言以外，最主要的是要掌握算法的分析、设计和数据结构的有关知识，因此有人称算法和数据结构是计算机程序设计的"两大支柱"。PASCAL 语言的设计者，瑞士的 N. 沃斯教授更提出了如下的公式:

算法＋数据结构＝程序

并以此公式作为他所著的一本书的书名。由此可看出算法和数据结构对于程序设计的重要性，也说明了算法和数据结构二者之间的密切关系。本书以讲述数据结构为主，也结合介绍与各种数据结构有关的一些算法，但对算法的设计与分析一般不进行详细讨论。

在计算机出现的早期，它主要是用来进行数值计算的，随着计算机应用的发展，越来越多的非数值计算也用计算机进行。数值计算包括求解线性代数方程组、求解非线性代数方程、求解常微分方程、求解偏微分方程、函数逼近等等，以及其它各种科学计算。研究数值计算的学科叫做《数值分析》(过去也叫做"计算方法")。所谓非数值计算，严格来讲也并非完全不处理数值，有时它也包含一些简单的加、减、乘、除等运算，但非数值计算只处理有限数目的不连续数值，较简单的例子如数据的排序、查找，复杂的例子如经济管理、图书情报的检索以及有关人工智能的许多问题等等。据统计，目前世界上的计算机 90% 以上的机时是用在非数值计算方面的。因此，本书中介绍的数据结构和算法也主要是针对非数值计算问题的。

§1-2 算法的描述

本书在描述算法时，将采用一种与 PASCAL 语言相仿、但较其简单、并适当加入一些中文语句的语言。读者很容易由本书介绍的算法用 PASCAL 语言编出计算机程序来。此外，因为 PASCAL 语言的保留字(字符号)很接近英语的相应的单词，在粗略描述算法时用中文语句又较简单明了，所以这种算法描述语言对于对 PASCAL 语言不太熟悉的读者，也是比较合适的。如果需要，也不难将本书讲述的算法用其它高级语言编出程序来。

现在对本书采用的算法描述语言中常用的一些语句说明如下。

1. 一般语句

赋值语句用冒号加等号作为赋值号。例如

A:＝1;

表示给变量 A 赋以数值 1 。又如

B:=C;

表示将变量 C 的值赋给变量 B。一般各语句均用分号来分隔。

加、减、乘、除算术运算，按习惯分别采用＋、－、*、/ 四种符号。对整型数的除法则有 div 和 mod 两种运算符，即整除和取余运算符，前者表示两整型数相除后舍去小数部分的整型结果，例如(13 div 5) 等于 2，(12 div 4)等于 3；后者则表示取余运算，例如 (13 mod 5)等于 3，(12 mod 4)等于 0 。此外，我们用向上的箭头↑表示幂运算，这是PASCAL 语言所没有的，例如(2↑3)等于 8，(3↑4)等于 81。

关系运算符有等于、不等于、小于、大于、不大于、不小于六种，分别表示成

= <> < > <= >=

关系运算经常在条件语句或循环语句中出现，其结果只有“成立”(即真)或“不成立”(即假)两种，例如(3＋2=5)或(5<=6)均为成立，(4－3>2)或(3*2<>6)均为不成立。

对于数组，数组名用字符串(一般均由英文字母组成)表示，数组元素的下标则写在其后的方括号中，如 NAME[3]，A[2, 3]等。下标也可以是已赋值的整型变量或简单的整型量算术运算表达式。

输入语句写成 read(…)的形式；输出语句写成 write(…)或 witeln(…)的形式，但往往也简单地用中文写，例如“打印某某量”。对变量和数组元素的初始置零语句，一般均省略，认为初始自动置零。

本课程所涉及的数据类型大多数为整数型，故为简单起见，所有类型定义和变量说明一般均略去，只在讲述动态链接结构时，为了阅读方便才加上这些说明。此外，用单引号括起来的表示是字符串，例如 write('A')表示打印字符 A 而不是打印变量 A 的数值。

凡用大括号括起来的文字{…}为注解，是为了帮助理解而对某些语句所做的说明，不是算法本身的语句。

2. 过程和函数

本书中介绍的算法多数不是完整的程序，而是某个局部的过程。我们规定，过程说明的开头写成如下形式

procedure 过程名称(……);

其中过程名称可以用英文字符串或中文来写，以尽量使人易于看出此过程的功能为原则；括号中包括调用此过程时最重要的一些形式参数，但它们的类型定义和一些非关键的参数在此均略去。过程内部局部变量的变量说明及 PASCAL 语言中过程执行语句前后的 begin、end 也省去不写。为使条理清楚，过程中的各语句用(1)、(2)、(3)、…等编号标明，有时一个编号中也可包括几条语句，中间用分号分隔。

主程序调用某过程或某过程调用另一个过程时，在需调用处只写被调用过程名称或用中文写“调用某某过程”即可，且在过程名称后的括号中写上调用时相应的实际参数。被调用的过程如果是中途返回，需在该处用中文写出“返回”语句，但被调用过程执行完了的返回则不必特别加

以说明。

函数说明在本书中出现较少，故未对其加以简化，就按 PASCAL 语言的格式写出。

3. 条件语句

条件语句即“如果”语句，形式为

if 〈某条件〉 then 〈语句 1 〉 else 〈语句 2 〉;

其对应的框图如图 1-1 (a)所示，如果该条件成立则执行“语句 1 ”，否则执行 “语句 2 ”。有时条件语句没有“语句 2 ”，即

if 〈某条件〉 then 〈语句 1 〉;

如果该条件成立则执行“语句 1 ”，否则不再执行什么，转而进行下面的语句，如图 1-1 (b)所示。

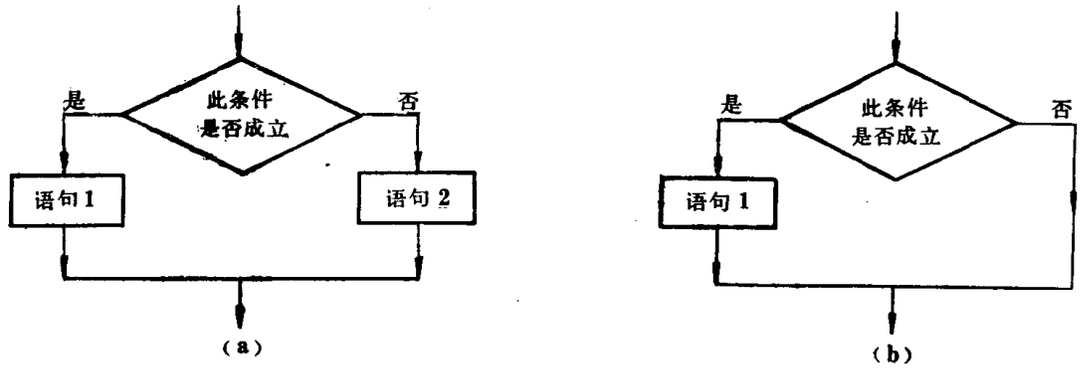

图 1-1 条件语句的框图

条件语句中的 “条件” 通常是由关系运算构成的布尔表达式，其中还可能包含布尔运算符 not、and 和 or，例如

(A>1) and not ((B=C) or (B<>D))

就是一个较复杂的布尔表达式。

语句 1 和语句 2 也可能是复合语句，即包括多条语句。对这种情况，各条语句按次序用(1)、(2)、(3)…等标明，而语句组前后的 begin 和 end 一般则省略不写了。

条件语句有时是多重的，即在语句 1 或语句 2 中又再包含条件语句。此时我们分别用(1)、(2)…，(i)、(ii)…及(a)、(b)…等不同的符号表示不同层次的语句。例如

```
if 〈条件 1 〉 then
  (1)……;
  (2)……;
  (3)if 〈条件 2 〉 then
        (i)……;
        (ii)……;
     else
        (i)……;
        (ii)……;
```

else ……;

4. 循环语句

以后主要要用到两种循环语句，即 while 语句和 for 语句，循环语句也即重复语句。

while 语句是一种有条件循环语句，形式为

while 〈某条件〉 do 〈语句〉;

它表示只要该条件成立，就执行 do 后面的语句，一直执行到条件不成立时，循环才结束，如图 1-2 的框图所示。其中的“条件”与 if 语句中的条件类似，是一个布尔表达式。do 后面的语句如是语句组，其前后的 begin 和 end 也同样省略，且给各条语句按顺序编号。

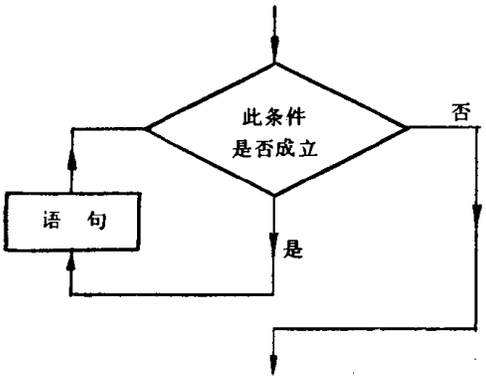

图 1-2 while 语句的框图

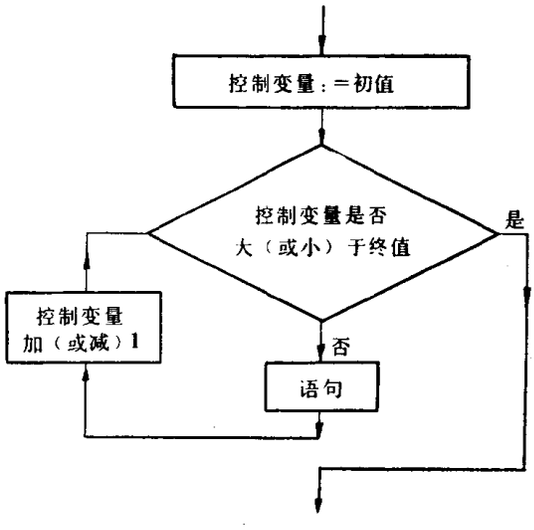

图 1-3 for 语句的框图

for 语句是一种无条件循环语句，一般常用于事先已确定了循环次数的情况，其形式为

for 控制变量:=初值 to 终值 do 〈语句〉;

它表示开始令控制变量(某整数型变量)等于给定的初值，然后重复执行 do 后面的语句。每执行一个循环，控制变量自动加 1，一直到控制变量大于给定的终值时才停止。如果原给定的初值已大于终值，此语句一次也不执行。

for 语句还有另外一种形式，即

for 控制变量:=初值 downto 终值 do 〈语句〉;

它与前一种 for 语句的区别在于，每执行一个循环，控制变量自动减 1，一直到控制变量小于给定的终值为止。一般后一种形式的 for 语句，其初值大于终值，否则此语句一次也不执行。这两种形式 for 语句的框图如图 1-3 所示。

for 语句中 do 后面的语句也可能是语句组，与以前几种情况中的语句组相同，省略其前后的 begin 和 end，并给各语句顺序编号。

while 语句和 for 语句这两种循环语句都可能有多重循环，也是用不同形式的编号表示不同层次的语句。下面是一个三重 for 语句的例子

```
for    i:=1    to    m    do
  (1)……;
  (2) for    j:=2    to    n    do
      (i)for    k:=1    to    (n-1)    do
            (a)……;
            (b)……;
            (c)……;
      (ii)……;
  (3)……;
```

5. 其它语句

有时为了书写简单明了，使用中文和一些数学符号来描述某一部分算法。读者很容易将它们改写成相应的语句。例如中文写的

将 K[1]与 K[m]对调;

对应的语句可写成

(1) w:=K[1]; {w 为某临时工作变量}

(2) K[1]:=K[m];

(3) K[m]:=w;

又如用数学符号写的

A:=min(B,C);

表示以 B 和 C 两个变量中数值较小的值给变量 A 赋值，其对应的语句可写成

```
if    B<C    then    A:=B    else    A:=C;
```

含最大值符号 max 的语句与此类似，不再赘述。

最后应说明，本书采用的算法描述语言是以简明易懂为原则的，并未完全包括实际高级语言(例如 PASCAL 语言)的全部功能。读者根据本书介绍的算法，如能充分采用高级语言的各种功能，可以更巧妙、灵活地编出程序。

§1-3 如何评价一种算法

对同一种问题，往往可以有多种不同的算法，如本书第八章就介绍了许多不同的排序方法。评价算法的目的，既在于从解决同一问题的不同算法中选出较为适用的一种，也在于有助于考虑如何对现有的算法进行改进或设计出新的算法。

1. 评价算法的一般原则

一般来说，一个好的算法主要应具有以下几个特点:

(1) 正确性: 也即要求该算法在合理的输入数据下，能在有限的时间内得出正确的结果。分析算法的正确性，一般需要用到有关的数学定理(例如线性代数、图论、组合学等方面的定理)，数学归纳法往往也是很有用的。对于较复杂的问题，可以将其算法分成一些局部的过程来分析，只

有每个过程都是正确的，才能保证整个算法的正确性。本书所介绍的算法，有些其正确性是不言而喻的，有些对其正确性做了证明，也有些将其证明从略了。

（2）运算工作量较少：这里所谓的运算工作量并非指的是计算机真正的运算时间，因为它随机型的不同而不同，也不是指需执行的指令或语句数目，因为它与所用的程序设计语言和程序员的编程习惯有关。此处是分析算法本身的特点，我们希望有一个足够准确而又一般化的衡量标准。通常是指计算所需的一些基本的操作（运算）次数，例如所需的比较和移动次数，所需的加法和乘法次数等，而且通常是对不同的算法做相对地比较。在评价一个算法时，运算量是非常重要的一个因素，本节后面还要对此着重进行讨论。

（3）所占空间量较省：这也不是具体指真正占多少计算机的内存或外存，因为这同样与所采用的机型、所用的程序设计语言和程序员惯用的格式等有关。此处也是对不同的算法进行相对地比较。

有些问题其输入数据有一定的形式，例如排序问题一般输入数据存于一个一维数组中，则我们要分析进行某算法还需多少额外的单元。当所需要占用的额外存储单元数不随输入数据的规模大小变化时，我们称这种算法是"就地"进行运算的，是较节省空间的算法。例如当排序运算仅在原数组中进行，只需少数的额外存储单元协助做数据元素的交换或移动之用时，就属于这种算法。但有时需要的额外存储单元与输入数据规模的大小成正比，对于大型问题这就是较浪费存储空间的算法。例如有的排序算法还需要另一个与原数组同样大小的数组以辅助运算，从占用空间的角度说来就是不好的算法。还应说明，这里所说的"存储单元"，并无严格的定义，是因问题而异的。

有的问题输入数据可以表示成不同的形式，例如将在第七章讲到的"图"就有不同的表示方法。对这类问题还需比较输入数据的不同形式所占空间的多少。

（4）简单性：最简单最直接的算法往往不是最有效的，其运算工作量可能较大。但算法的简单性使得证明其正确性比较容易，编写程序、改错和必要时对程序进行修改都比较方便，可以节省人的时间，还是应当强调的。即使如此，对于需反复多次使用的程序来说，算法的效率还是比其简单性更为重要些。

2. 算法的复杂性

复杂性是指实现和运行某一算法所需"资源"的多少，包括时间复杂性（所需运算时间）、空间复杂性（所占存储空间）和人工复杂性（编程及改错等所需的人工）。随着计算机硬件的发展，扩展内存或外存容量并不算困难；由于各种高级语言的出现，编程、改错等也较以前容易得多了，所以我们现在研究算法的复杂性，一般重点是指时间复杂性。

显然，一算法所需的运算时间与所解决问题的规模大小有关。我们用 n 作为表示问题规模的量，例如排序问题中 n 为需排序的元素个数；图的问题中 n 是图的顶点数或边数；矩阵的求逆中 n 为矩阵的阶数等。时间复杂性则表示成 n 的函数 T(n)。下表表示各种 T(n) 函数的算法在不同 n 值时的运算时间。表中凡未注明者时间的单位均为微妙。因实际上运算时间 T(n) 随不同的计算机而不同，此处并未指出所用的机型，主要是用它来进行相对的比较。

T(n)微秒	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	n^5	2^n	3^n	n!
n=20	4.3	20	86.4	400	8 毫秒	3.2秒	1.05秒	3487秒	771世纪
n=40	5.3	40	213	1600	64毫秒	1.7分	12.7天	3855世纪	2.59×10^{32}世纪
n=60	5.9	60	354	3600	216毫秒	13分	366世纪	1.3×10^{13}世纪	2.64×10^{66}世纪

由此表可以看出，不同 T(n)的算法，当 n 增长时运算时间增长的快慢很不相同。T(n)为指数函数的算法，当 n 较大时实际上是无法应用的，而 T(n)为阶乘函数的算法，其运算时间随 n 的增长比指数函数的算法还要快得多。凡是 T(n)为 n 的对数函数、线性函数或多项式的（幂函数也是多项式的特例），我们称这种算法为有效的算法，或好的算法；反之，T(n)为指数函数或阶乘函数的算法，则称之为坏的算法，这类算法一般没有实用价值。

讨论时间复杂性，我们最感兴趣的是所谓“渐近时间复杂性”，即当 n 逐渐增大时 T(n) 的极限情况。因此，为了分析方便，时间复杂性通常用数量级的形式来表示，即表示成

$$T(n)=O(f(n))$$

式中大写字母 O 为英文“数量级”一词(Order)的字头。这样的写法表示：存在某足够大的正整数 n_0 和一个正的常数 C，使得对所有 $n\geqslant n_0$ 都有 $T(n)\leqslant Cf(n)$。

用数量级的形式表示 T(n)，当 T(n)为多项式时，可只取其最高次幂项，且它的系数也可略去不写。例如若有

$$T(n)=3n^3+5n+12$$

则 $T(n)=O(n^3)$。因为若令 $n_0=1$，对所有 $n\geqslant n_0$，都有

$$3n^3+5n+12\leqslant 3n^3+5n^3+12n^3=20n^3$$

当取 $C=20$ 时，即有 $T(n)\leqslant Cn^3$。

用数量级表示 T(n)时，只需取最高次幂项，也可以这样来理解：无论该项的系数比起低次幂项的系数小多少，当 n 大到一定程度时总是最高次幂项占 T(n)的主要部分。例如若有

$$T(n)=n^2+1000n$$

当 n 较小时，第二项较n^2 项数值要大，但当$n>1000$ 以后，n^2 项就占T(n)的主要部分了，故$T(n)=O(n^2)$。由此可以看出，用数量级的形式表示，对于 n 较大的情况是很合适的。

由于用数量级表示时函数 f(n)前面的系数可略去不写，所以当 f(n)为对数函数时，对数的底也可以不必写出了，因为不同底的对数只是前面的系数不同，例如

$$\log_2 n=\log_2 10\times\log_{10} n=3.32\ \log_{10} n$$

与数学上的习惯不同，以后凡是对数函数没有标明其底的，我们一般认为是以 2 为底的对数。

对于足够大的 n，存在以下顺序

$$O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<\cdots\cdots<O(2^n)<O(3^n)<\cdots\cdots<O(n!)$$

还应说明，某算法的运算时间除与问题的规模 n 有关外，往往还和具体输入的数据有关。例如在进行排序时，原来的数据是杂乱的排列还是已大体有序，对运算的快慢是会有影响的；又如在一个一维数组中欲查找某一数据，若采用从头至尾顺序查找的算法，当碰巧待查找的数据在数组的第一个单元时，只需比较一次即可查找到，而最坏的情况，即当待查找的数据在数组的最后

一个单元时，则需比较 n 次才可查找到。因此，在研究某算法的运算时间时可有两种考虑办法：一是考虑平均情况，即研究同样 n 值时各种可能的输入，取它们运算时间的平均值；另一是考虑最坏情况，即研究各种输入中运算最慢的一种情况的运算时间。初看来似乎取平均的办法较好，但考虑所有可能的输入情况，在数学上分析起来比较困难，有时甚至是不可能做到的，而且各种可能性出现的概率也不一定相同。所以更多的情况下我们是研究最坏的输入数据下的运算时间。对于"实时"应用的运算问题，如巡回检测、计算机控制等，考虑最坏情况尤为重要。如若不然，可能在某些输入数据的情况下所留的运算时间不够用，而出现错误。

内 容 提 要

1. 数据是一切能够由计算机接受和处理的对象。一般来说，数据并非是单个孤立的数或字符，而是由一批基本单位以一定的关系组织起来的，组成数据的基本单位叫做数据元素。"数据结构"不仅要描述某种数据，而且要描述此种数据中各元素间的相互关系，往往是给出有关这些数据元素的一组运算。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构指数据元素之间的逻辑关系，数据的物理结构则指数据元素在计算机存储器中的表示和安排。一般前者简称为数据结构，而后者称为存储结构。

一般在程序设计语言中，包含有一些基本的数据结构，在程序编制中需要时，还可再定义其它的数据结构或给已有的数据结构再增加一些运算功能，为此，应在程序中给出相应的运算过程说明。

算法是一个有穷的规则序列，这些规则决定了解决某一特定题目的一系列运算。由此题目的一定输入，依照这些规则令计算机按部就班地进行计算，经过有限的计算步骤后能得到一定的输出。

算法和数据结构是计算机程序设计的"两大支柱"，或者说

$$\text{算法} + \text{数据结构} = \text{程序}$$

由此可看出二者对于程序设计的重要性和二者之间的密切关系。

计算机进行的计算包括数值计算和非数值计算，本书中介绍的数据结构和算法主要是针对非数值计算问题的。

2. 本书中描述算法，是用一种仿照 PASCAL 语言但较其简化并适当加入一些中文语句的语言。必须熟悉这种语言并能熟练地将本书中讲述的算法用 PASCAL 语言编出程序来。

在本书所用的算法描述语言中，赋值语句、过程调用语句、函数说明语句、读语句、写语句以及四则运算、关系运算等基本上与 PASCAL 语言相同，数组元素和字符串的表示也与PASCAL 语言没什么区别。但在过程说明中，类型定义、变量说明、执行语句前后的 begin、end 以及非关键的形式参数均省略了，过程中各语句一般用编号标明。对 if 语句、while 语句和 for 语句，都是仿照 PASCAL 语言的形式写出的，只是在有多条执行语句组成语句组时，省去这些语句前后的 begin 和 end 而给各条语句顺序编号。对于多重的 if 语句或多重循环语句则用不同形式的

编号表示不同层次的语句。

还有一些用中文或数学符号描述的语句，其含义比较明显，不难理解。

3. 评价一个算法，主要考虑其正确性、运算工作量的多少、所占存储空间是否节省以及其简单性。除了算法必须正确以外，一般运算工作量的多少是最重要的考虑因素。

所谓算法的复杂性，包括时间复杂性、空间复杂性和人工复杂性，但我们重点是研究算法的时间复杂性。

设以 n 作为表示问题规模的量，时间复杂性则表示成 n 的函数 T(n)。凡 T(n)为 n 的对数函数、线性函数或多项式(包括幂函数)的，我们称这种算法为有效的或好的算法。而 T(n)为指数函数或阶乘函数的算法，因 T(n)随 n 的增长而增长得太快，当 n 较大时是不实用的，故称之为坏的算法。

为了分析方便，复杂性通常用数量级的形式来表示，当 T(n)为多项式时，用这种表示法只需取其最高次幂项，且此项的系数也可略去不写。

一种算法的运算时间除与 n 有关外，还与具体的输入数据有关。在研究某算法的运算时间时，有时是考虑平均的情况，但多数是考虑输入数据最不利时的情况。

习　题

1-1　按本书所采用的算法描述语言写出的一段程序为

```
(1) n:=10;
(2) for  i:=1  to  n  do
        if  (i  mod  2)=0  then
          write(i)
        else
          (i) j:=i*2;
          (ii)if  j<n then  write(j);
```

试写出此段程序运行时的输出数据。

1-2　按本书所采用的算法描述语言写出的一段程序为

```
(1) n:=6;
(2) i:=1; j:=4;
(3) while  i<=n  do
      (i) j:=j+1;
      (ii) write(j);
      (iii) i:=i+1;
```

试说明此段程序运行时第(3)步的循环语句一共循环几次，并写出程序的输出结果。

1-3　拟向一个原有 n 个元素的数组中第 i 个元素前插入一新的元素，$1 \leqslant i \leqslant n < 10$，已知数组的类型定义用 PASCAL 语言写出为

type atype=array[1..10] of real;

以简化了的形式写出的插入元素过程说明为

procedure 插入(List, n, i, x);

(1) for j:=n downto i do

List[j+1]:=List[j];{一些数组元素后移一个单元}

(2) List[i]:=x;{插入数值为 x 的新元素}

(3) n:=n+1; {元素数目增加一个}

试将此过程说明用 PASCAL 语言写成完整的形式。

1-4 有$T_1(n)$和$T_2(n)$两个函数，当 n 等于或大于某一整数值 n_0 时，恒有 $T_1(n) \geqslant T_2(n)$，试求此 n_0 值。若

(i) $T_1(n)=n^2, T_2(n)=10n$;

(ii) $T_1(n)=2^n, T_2(n)=2n^3$;

(iii) $T_1(n)=n!, T_2(n)=3^n$。

1-5 对于§1-3 中各种不同函数 T(n) 值的表加以增补，要求再增加 n=10 和 30 的两行，和增加 $T(n)=\log_2(\log_2 n)$和 n^n 的两列。

1-6 已输入 x、y 和 z 三个不相等的整数，试设计一个算法将其中数值居中者打印出来。说明最好情况、最坏情况所需进行的比较次数及平均所需的比较次数。

第二章　数组和线性表

§2-1　数组及其存储结构

数组是最常见的一种数据结构，在一些高级程序设计语言中，例如早期的FORTRAN、ALGOL语言等，数组是组织数据的唯一结构。后来出现的一些语言(例如PASCAL语言)，虽然包括了其它一些数据结构，但数组仍是应用最广泛的一种结构。因此，数组一般最为大家所熟知，由它开始讲述"数据结构"是很合适的。

读者在学习程序设计语言时，可能已经了解了数组的一些应用，例如可以用一维数组表示一个向量或一组有序的数据；用二维数组表示一个矩阵等。本书后面还会经常用到数组这种基本的数据结构。

数组是由一些单元组成的，每个单元对应着一组下标值和一个数组元素。虽然PASCAL语言可用不同类型的量作为下标值，但在本书中下标值一般是取整数型的。一维数组的每个单元对应一个下标值；二维数组则对应两个；依此类推，n维数组的每个单元对应n个下标值。数组元素可以是整数型的、实数型的、字符型的、布尔型的等，也可以是有多个数据项的一种结构。但数组是一种"均匀"结构，同一数组中各个元素必须是同一类型的。

数组是一种随机存取结构，只要给定一组下标，就可访问与其对应的单元，例如存取或修改此单元的元素值或此单元中某个数据项的值等。这种访问对同一数组的各个单元来说是"平等"的。与此相反，有些别的数据结构，例如后面将讲到的链接表、树或图等就不同，访问对每个单元不是"平等"的，要有一定的先后次序。

在计算机中，表示数组的最普通的方式是，采用一组连续的存储单元顺序地存储各数组元素。只要建立起数组元素的下标值和存储地址之间的对应关系，就可由下标值随机地访问该数组的任一个元素。

先看一维数组，设数组名为A，其下标的下界为LB，上界为UB，用PASCAL语言写出的说明语句为

var A:array[LB··UB] of 成分类型;

先假设每个数组元素只占用一个存储单元，即每元素占一个字节，我们用Loc(i)表示下标为i的数组元素A[i](设其值表示成a_i)的地址，并假设已知数组中第一个元素(其下标等于LB)的地址为Loc(LB)，则有

$$Loc(i)=Loc(LB)+(i-LB)$$

这个关系(图2-1为此关系的示意图)，该式称为寻址公式。

对于二维数组，每个单元对应两个下标值，设它们的下界和上界分别为LB1、LB2和UB1、