计算机等级考试教程

（四级）

数据结构与算法

全国高等学校计算机教育研究会
教材与课程建设委员会 组编

李大友 主编

陈文博 朱青 编著

机械工业出版社

计 算 机 等 级 考 试 教 程

（四　级）

数据结构与算法

全国高等学校计算机教育研究会
教 材 与 课 程 建 设 委 员 会　组编
李大友　主编
陈文博　朱　青　编著

机 械 工 业 出 版 社

本书详细介绍了常用数据结构的特性和类型表示方法。算法用 Pascal 和 C 语言两种语言加以对照描述。书中有大量调试过的数据结构实用算法例题。全书主要内容包括：数据结构概论、线性表、栈和队列、串、数组、树、图、广义表、查找、排序和文件等。

读者对象：全国计算机等级考试四级应试人员、大专院校师生、相关科技人员及电脑爱好者等。

图书在版编目（CIP）数据

计算机等级考试教程（四级）：数据结构与算法/李大友主编；陈文博　朱青编著．—北京：机械工业出版社，1996.4
ISBN 7-111-04995-0

Ⅰ.计…　Ⅱ①李…②陈…③朱…　Ⅲ.①电子计算机-技术等级标准-考核-教材②数据结构-技术等级标准-考核-教材③算法-技术等级标准-考核-教　Ⅳ.TP3

中国版本图书馆 CIP 数据核字（96）第 00274 号

出 版 人：马九荣（北京市百万庄南街 1 号 邮政编码 100037）
责任编辑：何文军　版式设计：张世琴　责任校对：孙志筠
封面设计：郭景云　责任印制：卢子祥
三河市宏达印刷厂印刷　·新华书店北京发行所发行
1996 年 4 月第 1 版第 1 次印刷
787mm×1092mm 1/16 · 21.75 印张 · 529 千字
0 001—5 000 册
定价：32.00 元

前　言

计算机等级考试是国家教委考试中心向全国推出的面向社会的、测试应试者计算机基础理论知识和应用能力的水平考试。它越来越被社会各界所认同，成为判断计算机专业人员水平和素质的公平尺度和权威性依据。四级考试是计算机等级考试的最高一级，它要求考生较全面地掌握计算机基础理论知识和软件、硬件方面的应用技能，以达到高级程序员水平。数据结构作为计算机等级考试三级B类和四级的必考课程之一，是计算机科学技术的专业基础课，同时也是我国计算机各专业教学计划中的一门主干课程。

本书介绍各种最常用的数据结构，阐明各种数据结构内在的逻辑关系，讨论它们在计算机中的存储表示，以及在这些数据结构上的运算和程序设计的算法，并对算法的效率进行了简要的分析。程序是算法的具体体现，书中的算法大都用 Pascal 和 C 两种常用工具语言在 Borland C++3.1 和 Turbo Pascal5.0 上调试通过，以便读者加深对所学内容的理解。在每一章之后，都附有大量的习题。这些题目强调数据结构算法的使用，不少题目都有应用背景。建议读者酌情选其中的一些题目上机练习。

数据结构讨论的对象是非数值性的数据以及它们之间的相互关系。根据这些关系的不同数学特性，可分为线性结构与非线性结构。本书详细地介绍了线性结构中的线性表、栈和队列以及字符串；非线性结构中的二叉树、树和图。查找和排序是数据处理中经常遇到的两种运算，它们也是数据结构讨论的主要内容之一。在当前数据库流行的同时，文件系统仍是一种有效的常用数据管理手段，而且它是学习数据库的基础，因此我们也为文件单列一章进行讨论。尽管数组是许多语言工具都已具备的数据类型，但它的使用价值值得我们在此对其做进一步的讨论。广义表的结构较为复杂，但应用前景看好，虽然考试大纲对此内容没有要求从数据结构的完整性角度来说，是一个不可缺少的内容。有些相对陈旧的内容，实际使用率很低，例如二叉树线索化问题，则进行了删繁就简。

本书既可作为计算机及相关专业学生的教材，亦可供从事计算机应用的科研和工程技术人员参考，使用本书作为四级考试的教材时，除广义表和栈与递归等章节外，都是必修内容。本书也可做为三级考试的教材，只是根据三级考试大纲酌情精减。

学习本书的读者应预先修完 Pascal 语言或 C 语言课程，并具有一般程序设计能力。

本书由陈文博和朱青共同编写。作者在学习和工作中得到严蔚敏老师的帮助，在此表示感谢。

由于作者水平所限，加之时间仓促，书中难免存在缺点和疏漏，欢迎广大读者予以批评指正。

作　者

1996 年 4 月

目　录

第1章 绪 论

1.1 学习数据结构的意义

数据结构是计算机应用专业的一门重要基础课程，目前又成为众多理工科专业学生所喜爱的选修课之一。数据结构是计算机水平考试三级 B 和四级的必考科目。由此可见，它在计算机知识结构中的地位与作用。事实说明，要想有效地使用计算机，仅掌握计算机语言而缺乏数据结构和算法的有关知识，难以应付众多复杂的应用课题。为此，让我们简单回顾一下数据结构学科形成和发展的历史。

在电子计算机发展的初期阶段，人们使用计算机主要是处理数值性问题，解决人们用手工或机械计算机难于胜任的数值计算。当时所涉及的操作对象还都比较简单，不外乎是整型、实型和布尔型数据。以此为对象的程序设计可称为数值型程序设计，对应的软件称为“数学软件”。随着电子计算机使用领域的扩大和深入解决“非数值性问题”越来越引起人们的关注，例如金融和工商企业领域的管理信息系统，支持多媒体的文献资料查询、神经元和模式识别，网络与通信，图形化用户界面技术等等。解决此类问题使用的数学工具已不是分析数学及其计算方法，而更多地用到离散数学和计算机的有关知识，所涉及的数据也更复杂，其突出的特点是，数据元素之间所具有的特定联系已不能用分析数学的方程式来简单描述。

为了帮助读者建立对数据结构的感性认识，下面先举几个简单例子。

例 1-1 车务管理中的查询问题。

在交通管理部门车务管理系统中，与车辆有关的数据都以文件形式存放在车辆档案库中，如图 1-1-1a 所示。若要求查找一辆“红色奥迪，车牌末尾数字是 8 的新车”的车主，仅从这个文件查找起来是很不方便的。针对这一类型的询问查找问题，可事先建立几个“倒排文件”，突出反映出具有相同属性的车辆和车牌号的对应关系，如图 1-1-1b 所示。

新车（估且取 1993 年以后领照的车）、奥迪、红色分别对应着车辆属性的三个集合，计算这三个集合之积，可得到一组车牌号，再从中排除尾号是 0，…，7，9 的车牌号，就能把查找对象迅速缩小到一个很小的特定范围之内。这个例子反映出运用文件这种数据结构的组织形式，可以提高查找速度和精度，为信息系统提供有效的数据管理手段。数据结构中涉及的文件组织方法也是数据库设计所必须具备的基本知识。

例 1-2 酒店管理系统中的客房分配问题。

在酒店的客房房态管理中，要求每间客房出租率的机会均等，以保证维持一个平均的磨损率。我们可以把所有的空客房排成一个队，每次前台接待客人入住时，从队头分配客房；客人结帐离店后，把退掉的客房排回队尾，这样就保证每间客房都有均等的机会被出租，也有均等的机会进行维护。图 1-1-2 表示了客房排队的模型。这种队列是一种元素之间先后操作次序很强的数据结构。在许多有“先来后到”要求的问题中都会用到这种数据结构。

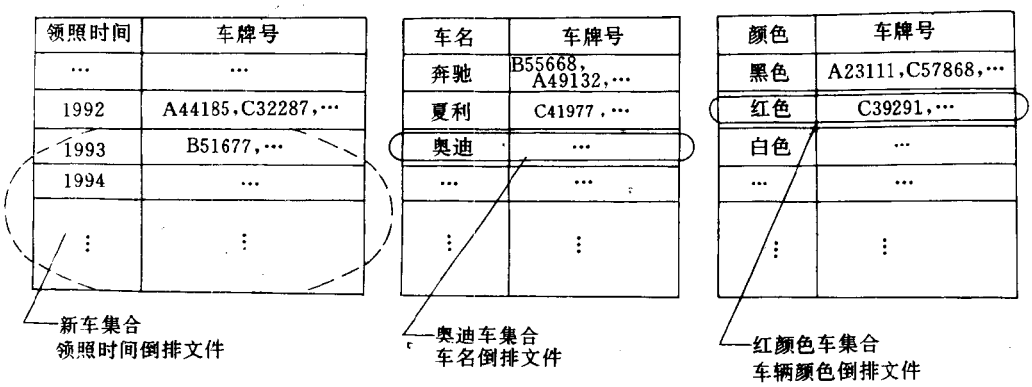
例 1-3 敷设煤气管路的最小费问题。

在几个居民点之间敷设煤气管路，每两个居民点之间的敷设费用可以估算出来，设计方

案要求每个居民点都可以通达煤气，且总的敷设费用最小。图 1-1-3a 是用图（Graph）的数据模型表达了这类问题的一个示例。其中小圆圈是图的顶点，代表居民点。每两个顶点之间的

车牌号	车主	车名	车型	颜色	购车地点	领照时间	过户变更
A83150	四达公司	奥迪	卧车	黑色	长春	93.4	
B42971	工业学院	大众	卧车	白色	上海	90.7	
A96685	通用机厂	北京	轻货	灰色	北京	88.5	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	

a)



b)

图 1-1-1　车务管理系统中文件数据结构模型

a）车辆档案文件　b）车辆属性倒排文件

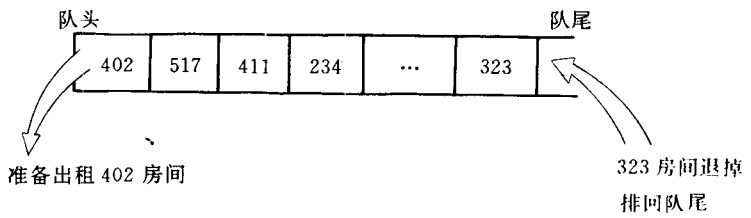


图 1-1-2　客户分配中使用的队结构

边及附带的数据表示这对居民点之间敷设管路的估算费用。这种图结构也是数据结构中研究的问题之一。我们可以用相应的数据结构来描述图、存储图，并且通过求图的最小生成树算法找到符合要求的解，如图 1-1-3b 所示按这种方案施工费用最少。有关网络规划的问题都会遇到图结构的应用。

例 1-4　识别手写体汉字问题

计算机脱机识别一个手写体的汉字时，一般采用分层次的办法，自顶向下进行。先用汉字的某些局部特征与识别主体相匹配，尽量容忍畸变和干扰，以逐步缩小识别范围直至最后

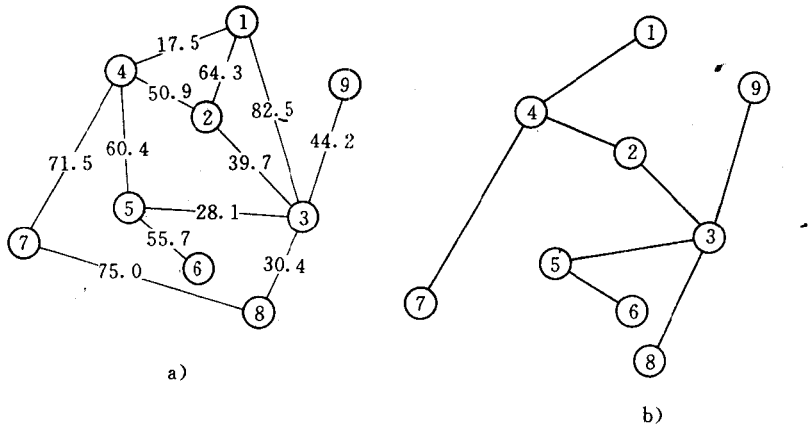


图 1-1-3　敷设煤气管路网

a）费用核算　b）敷设方案

确认为止。如识别“体”字时，先判断部首，“体”属于“单人旁”，因部首为“单人”的汉字还很多，故尚需按四角号码进行匹配，相同的只有“体”和“休”两字，最后再进行局部的细节匹配，被确认为“体”字。上述识别过程可用图 1-1-4 所示的“树”来描述。

像这样按分支和层次组织的数据，称为“树形结构”，识别“体”字的过程是对图 1-1-4所示的树进行某种操作或运算，在查找某类问题中树结构是有效的数据组织方法，应用范围很广。

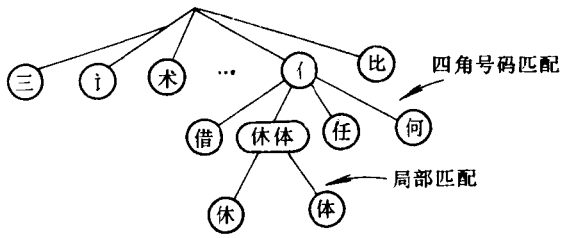


图 1-1-4　手写体汉字的识别过程

综合以上几个例子已不难看出，大量的实际应用问题中涉及的数据元素之间存在着非数值性的关系，我们称这种关系为“结构”。目前，多数高级语言尚不能直接操作这些带结构的数据，本书将在高级语言的层次上讨论如何表示这些数据和如何对它们进行操作。

1.2　数据结构的基本概念和术语

在系统学习数据结构知识之初，有必要介绍有关数据结构的基本概念和术语。数据结构的学科历史很短，各种术语还在不同的使用群体和应用场合之间互相碰撞、磨合、尚未形成严格统一的规范定义。另一方面，目前面向对象的技术和方法在计算机软件领域的发展势头十分迅猛，面向对象的有关术语和名词与数据结构原来的称谓和概念又有许多相似易混之处。我们兼顾习惯性和新情况，对数据结构的概念和术语进行解释性定义。

数据（Data）是信息的载体，它可在计算机内加以表示并加工处理。随着技术的进步，数据所描述的内容越来越丰富。像多媒体技术中涉及到的视频和音频信号，经采集转换后都能形成被计算机所接受的数据。

数据元素（Data Element）是数据的基本单位。一般来讲，数据元素是数据结构中不必再分的最小数据单位。例如，酒店客房分配问题中，排队的房间号就是数据元素。“最小数据单位”也是相对的。在文件系统中，我们把一个数据记录作为一个数据元素看待，数据记录可以由若干数据项组成。例如，车务管理中的车辆档案文件，每条记录就由车牌号、车主等很多数据项组成。通常，数据元素在除文件以外的所有数据结构中是操作的最小对象。

数据元素类（Data Element Class）是具有相同性质的数据元素的集合。在某个具体问题的讨论中，数据元素都具有相同的性质（元素值不一定相等），属于同一数据元素类，数据元素是数据元素类的一个实例。例如，在管路敷设的图中，所有顶点是一个数据元素类，顶点 1 和顶点 2 各自代表一个居民点，是顶点这种数据元素类中的实例。数据元素的值分别为 1 和 2。

结构（Structure）是指同一数据元素类中各数据元素之间存在的关系。如客房分配问题中，房间排队的次序关系；手写汉字识别过程中的分支匹配关系；管路敷设问题中，居民点之间的施工费用关系等等。通常这种关系是以问题为背景的一种特定联系。

数据结构（Data structure）简单地说，是带结构的数据元素类，它有两层含义：第一，它包括一个具有共同特性的数据元素的集合，即数据元素类。第二，它还包括一个定义在这个集合上的一组关系，即数据元素之间的“结构”。

从 1.1 节所举的例子可以看出，对每个具体问题而言，其数据元素和关系均具有特定的含义，但对数据结构学科而言，所研究的不是这些具体的含义，而是它们所具有的数学特性。根据不同的结构特性，本书讨论的数据结构有：表或文件、栈和队列、线性表、数组、树和图，广义表。表中的元素具有松散的集合关系，或者说，元素之间除了同属某个集合外，不存在任何关系；线性表中的元素前后之间存在着一对一的线性关系；数组的元素之间形成有规则的行和列的排列关系；树的元素之间存在着一对多的分支或层次关系，图的元素之间是多对多的网络关系，广义表除具有线性表的性质外，其数据元素本身有一种递归关系，即元素还可以是一个广义表。数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构研究的是数据元素及其关系的数学特性，而数据的物理结构研究的是它们在计算机内的实现方法。对机器语言来说，这种实现是具体的，本书仅在高级语言的层次上来讨论这种实现，用高级语言中的数据类型来描述这种实现细节。为简明起见，以后我们将简称数据的逻辑结构为数据结构，简称数据的物理结构为存储结构。

数据类型（Data Type）是指程序设计语言中允许的变量种类。不同的语言提供的基本数据类型不同。在用程序设计语言书写的程序中出现的每个变量，仅与一个数据类型相联系，它不仅规定了该变量可以设定的值的集合，还规定了在这个集合上的一组操作运算。每一种程序设计语言中都有一组它所允许的基本数据类型，如 FORTRAN 语言提供了五种基本数据类型：整数、实数、双精度数、复数和布尔量，Pascal 语言提供了整型、实型、布尔型和字符型四种基本类型。除了基本类型之外，程序设计语言中还提供结构的数据类型（简称为结构类型），如数组等。每种类型的数据都有属于自己范畴的合法运算。C 语言所提供的数据类型比 Pascal 语言提供的还要丰富。一般来讲，基本类型变量的值是不可分的，而结构类型变量的值是可分的，有一定的组织层次，或者说它是带结构的数据。因此，结构类型也可看成是工具语言中已经实现的数据结构。

抽象数据类型（Abstract Data Type—ADT）是指一个数据结构的规范化描述及该数据结构所涉及到的操作集，ADT 不强调在哪一级工具语言中具体实现，首先关注点是这些操作的

外部特性，即如何通过这些操作去完成某个算法任务。

例 1-5 利用字符串抽象数据类型实现一个字符串的逆置算法。

字符串抽象数据类型数据结构：字符串（String）是由字符组成的有限序列，记作 String $='a_1a_2\cdots a_n'$ $(n\geqslant 0)$，其中 $a_i\in$ 字符集。

字符串操作运算：

1）串赋值 CREATE (s, s_1) 把串 s_1 的值赋给串 s；

2）求串长 LEN (s) 函数。返回串 s 的字符个数；

3）求子串 SUB (s, i, j) 函数，返回从串 s 中第 i 到第 j 的字符序列；

4）串联接 CONCAT (s, t) 函数，返回由串 s 和 t 联接成的字符序列；

⋮

问题：设计算法把串 $s='b_1b_2\cdots b_m'$ 逆置为 $t='b_mb_{m-1}\cdots b_1'$。算法思想：构造一个新串 $t='b_m'$，然后依次把 $'b_{m-1}'$，b_{m-2}'，…，$'b_1'$ 联接到 t 中。

算法：

```
FUNC invert(s:string):string;
VAR t,s:string;
BEGIN
    m:=LEN(s);      {求 s 的串长度}
    IF m>0
    THEN [t:=SUB(s,m,m);  {构造新串 t='bm'}
          FOR i:=m-1 DOWNTO 1 DO
          [ u:=SUB(s,i,i);  {取'bi'}
            t:=CONCAT(t,u);  {联接到 t 尾部}
          ]
    RETURN (t)
END;
```

至此，尚未涉及字符串和它的运算在工具语言层的具体实现，仅通过串抽象数据类型的 LEN、SUB 和 CONCAT 操作运算完成了串的逆置算法。可见凭借抽象数据类型的支持，可以有效地研究问题、设计算法、分析算法。按计算机硬件处理问题的办法，可把抽象数据类型看成一个软芯片(Software Integrated Circuit SIC)。SIC 封装了数据结构，把操作作为对外接口留给了用户，串的抽象数据类型用软芯片来表示，如图 1-2-1 所示。

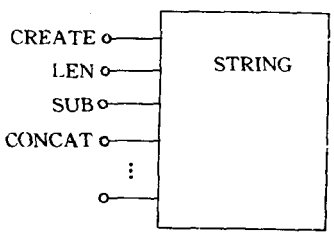


图 1-2-1 串的软芯片示意图

一个复杂的问题可能会用到许多 ADT，有些是本书将要讨论的，也有些要由用户创造。问题的解可以方便地用这些 SIC 构造出来。在具体程序设计时，需要通过工具语言，用数据类型把 ADT 具体化。这一过程相当于依据概念图把芯片制造出来。ADT 的思想使我们暂时回避细微末节，集中精力面对问题。

算法（Algorithm）就是解决某一特定类型问题的有限运算序列。一个算法必须是在执行有限步之后结束；算法中的每一步必须有确切定义；并且每一步有待实现的运算都是相当基

本的，换句话说，它们原则上都是能够精确地进行运算的；一个算法有 0 个或多个的输入，也必然有一个或多个的输出。算法不等同于程序。一个程序，譬如一个操作系统，只要不关机，它就不会结束。算法的设计可以避开具体的计算机和程序设计语言，通过 ADT 在抽象的层次进行；算法也可以借助程序设计语言中提供的数据类型及运算在具体的层次上实现，讨论算法是数据结构课程中的重要内容之一，Pascal 语言的发明人著名瑞士计算机科学家沃斯（N. Wirth）提出一个精辟论点：算法＋数据结构＝程序。可见算法和数据结构是利用计算机解决实际问题时不可缺少的两个方面。因此，本书在讨论数据结构的同时，也将介绍若干和数据结构有关的算法。

1.3 算法的描述工具

在本书中，所讲述的各种基本数据结构都用软芯片的方式给出抽象数据类型。在描述数据结构的操作算法时使用拟 Pascal 语言，因为 Pascal 语言本身拥有丰富的数据类型和控制结构，书写的程序结构性强；另一方面拟 Pascal 语言不拘泥于某些细节规定，对 Pascal 语言进行了技术上的简化处理，以便更方便地编制算法和增加算法的可读性。读者对拟 Pascal 语言书写的算法稍加技术处理即可用 Pascal 语言直接上机运行。

书中有大量关于数据结构的算法实例。对于这些实例，除用拟 Pascal 语言描述外，还给出 Pascal 语言或 C 语言的程序，有时两者兼有，目的在于通过各种工具语言环境熟练掌握数据结构的实用技术。一般说，Pascal 语言描述算法简捷，可读性好；用 C 语言描述算法灵活、实用性强。

拟 Pascal 语言简化处理表现在两个方面。第一，省略了形式化的类型说明和变量说明，代之以用较为自由的自然语言作非形式化的描述，且省略说明一些容易理解的整型变量；第二，增加了一些语句的功能，如扩充了 CASE 语句的范围，增添了出错处理语句 ERROR，将语句组的一对括弧 BEGIN—END 改为一对黑体方括号 []。如读者需对某些算法进行上机调试，只须稍加处理便可将用拟 Pascal 语言书写的算法转换成其它语言编写的程序。书中已给出部分拟 Pascal 语言书写的算法以及它的 Pascal 语言和 C 语言程序，有助于读者模仿改编其它程序。

本书中采用的拟 Pascal 语言的非形式化说明如下：

1）语言中允许整型、实型、字符型、布尔型等基本类型，还允许数组、指针等结构类型。

2）表达式可以是算术表达式、字符表达式和布尔表达式。在布尔表达式中含关系运算符和逻辑运算符。

3）赋值语句、IF 语句的形式与原 Pascal 语言相同。

4）情况语句有两种格式：

```
CASE 〈表达式〉 OF
〈常量 1〉: S1;
〈常量 2〉: S2;
   ⋮
〈常量 n〉: Sn
END
```

```
CASE
C1: S1;
C2: S2;
   ⋮
Cn: Sn;
ELSE Sn+1
END
```

5）循环语句有下列四种格式：

“当”语句

WHILE C DO S

“重复”语句

REPEAT S_1；S_2；…S_n UNTIL C

“步进循环”语句

FOR 变量：＝初值 TO 终值 DO S

或

FOR 变量：＝初值 DOWNTO 终值 DO S

它们与 Pascal 语言中的循环相同。

6）过程调用语句 CALL 过程名（参量表）；或过程名（参量表）；

7）所有的算法都是以过程或函数的形式出现。过程头 PROCEDURE 也可以简写成 PROC；函数头 FUNCTION 也可以简写成 FUNC。形式参数表中的参数不加类型说明，但在每个变参之前冠以 VAR。凡是需将过程加工中的结果传到过程外的参数必须是变参。其它方面与 PASCAL 语言相似。

1.4 算法分析技术初步

一个可执行的正确的算法也有优劣之分，通常以算法执行时在时间和空间资源方面消耗的多寡作为评价算法优劣的标准。

在时间消耗方面，可以采用事后统计的办法得出执行算法所用的具体时间。因为很多计算机环境都有这种计时的功能，甚至可以得到精确到毫秒级的统计数据。但这种办法有两个缺陷：第一、必须先运行才能分辨出算法的好坏；第二、所得时间的统计量过分依赖于计算机的硬件、软件等环境因素，这些因素容易掩盖算法本身的优劣。因为一个笨拙的算法在先进的大型机上运行可能比一个同功能的优化算法在微机上运行更省时间。

因此，人们常常对算法进行事前的时间估算。在此，可利用语句的“频度”和算法的渐近时间复杂度这两个和软、硬件无关的量度来讨论算法执行的时间消耗。先看一个具体的例子。

下列是一个 n×n 的矩阵 A 乘以 A，得到 $B=A^2$ 的算法。

```
PROC MTXMCT (A, n, VAR B);
{二维数组 A (1：n, 1：n) 中存放已知矩阵 A，A 自乘的结果送入 B (1：n, 1：n) 中。}
BEGIN
1    FOR i:=1 TO n DO
2        FOR j:=1 TO n DO
3        〔 B〔i,j〕:=0
4          FOR k:=1 TO n DO
5          B〔i,j〕:=B〔i,j〕+A〔i,k〕*A〔k,j〕
         〕
END;  {MTXMLT}
```

上述算法中，语句 3 在二重循环之内，重复执行的次数为 n^2。语句 5 在三重循环之内，重复执行的次数为 n^3。假设语句 3 执行一次的时间是 t_1，语句 5 执行一次的时间是 t_2，若只考虑

算法中这两个主要赋值语句的执行时间，而忽略步进语句中其它成分，如步长加1、终值判别，控制转移等所需时间，则可认为此算法耗用的时间近似为

$$T(h)=t_1n^2+t_2n^3$$

式中，矩阵的阶 n 表示问题的规模，当 n 很大时，显然有

$$\lim_{n\to\infty}\frac{T(n)}{n^3}=\lim_{n\to\infty}\frac{t_1n^2+t_2n^3}{n^3}=t_2$$

这表明，在 n 充分大时，T (n) 和 n^3 的比值为一常数，即 T (n) 和 n^3 是同阶的，可记作 T (n) =O (n^3)。其中 O 是数学符号，其严格的数学定义为：若 f (n) 是正整数 n 的一个函数，则 T (n) =O (f (n)) 表示存在一个正的常量 M，使得当 $n\geqslant n_0$ 时都满足 $|T(n)|\leqslant M|f(n)|$。

我们称 T (n) =O (n^3) 为算法 MTXMLT 的渐近时间复杂度；n^2 和 n^3 分别为语句 3 和语句 5 的频度。

一般情况下，表征问题规模的数量参数称为问题的尺寸 (Size)，如矩阵的阶，字符串的长度，图的顶点数和边数，文件中的记录数等等，通常用字母 n 表示。算法中语句的频度，即语句重复的次数，往往是问题尺寸 n 的函数，记作 F (n)。其值由算法的控制结构决定，是执行算法时耗用的时间的主要依赖因素。算法的渐近时间复杂度，简称为时间复杂度，以算法中频度最大的语句频度 f (n) 作为算法执行时间的一个量度，记作 T (n) =O (f (n))。

下面列举几个程序段中标有@语句的频度和该程序段的时间复杂度。

1) @y: =SIN (X)

单个语句的频度为 1，F (n) =1，则该程序段的时间复杂度为 T (n) =O (1)。

```
2) FOR i: =1 TO n-1 DO
   〔@1  y: =y+1;
         FOR j: =0 To 2 * n DO
         @2  x: =x+1
   〕
```

语句@1 的频度为 n-1，语句@2 的频度为 (n-1) (2n+1) $=2n^2-n-1$，则该程序段的时间复杂度 T (n) =O (n^2+n) =O (n^2)。可见，算法的时间复杂度是由嵌套最深层语句的频度决定的。

```
3) x: =1; y: =1;
FOR k: =1  TO n DO @1  x: =x+1;
FOR i: =1 TO n DO
    FOR j: =1 TO n DO @2  y: =y+1;
```

语句@1 的频度为 n，语句@2 的频度为 n^2。显然，此程序段的时间复杂度为 T (n) =O (n^2)。可见，当有多个串行的循环语句时，算法的时间复杂度由嵌套层次最多的循环语句中最里层语句的频度决定的。

```
4) i: =1;
WHILE (i<n) AND (X<>A 〔i〕) DO
@  i: =i+1;
IF A 〔i〕 =x THEN RETURN (i)
```

此程序段中的语句@的频度不仅是n的函数，而且与x及数组A中各分量的具体值有关，在这种情况下，通常按最坏的情况考虑。由于WHILE循环执行的最大次数为n－1，即语句@的频度的最大值为F（n）＝n－1，则认为此程序段的时间复杂度为T（n）＝O（n）。

求递归算法的频度不容易直接估算，须由递推公式计算求得。以有名的Hanoi塔问题为例，其解是一个递归形式的算法

```
PROC Hanoi (n, i, j);
BEGIN
  IF n=1
  THEN move (i, j);
  ELSE [Hanoi (n-1, i, 6-i-j);
        move (i, j);
        Hanoi (n-1, 6-i-j, j)
       ]
END;
```

我们不打算分析这个算法的来龙去脉，只关心算法的时间特性。显见问题的尺寸是n，move（i，j）语句的频度为1。若整个算法的频度为F（n），则递归调用Hanoi（n－1，i，6－i－j）和Hanoi（n－1，6－i－j，j）语句的频度应为F（n－1），于是有

$$F(n)=F(n-1)+1+F(n-1)$$

即

$$F(n)=2F(n-1)+1$$

进行递推

$$\begin{aligned}F(n)&=2(2F(n-2)+1)+1\\&=4F(n-2)+3\\&=4(2F(n-3)+1)+3\\&=8F(n-3)+7\\&\vdots\\&=2^kF(n-k)+2^k-1\end{aligned}$$

当n－k＝1时，

$$F(n)=2^{n-1}F(1)+2^{n-1}-1$$

F（1）是n＝1时的算法的频度，它只执行move（i，j）语句，其值为1。最后得到F（n）＝2^n－1。时间复杂度T（n）＝O（2^n）。

总之，频度和时间复杂度虽不能精确地确定一个算法或程序的执行时间，但可以让人们知道，随着问题尺寸的增大，算法耗用时间的增长趋势。由于讨论算法的好坏不是针对某个特定大小的问题（这样做的本身没有意义），因此，时间复杂度对算法来说是一个较恰当的量度。

较常见的时间复杂度有O（1）（常量型），O（n）、O（n^2）、…、O（n^k）（多项式型）和O（$\log_2 n$）、O（$n\log_2 n$）（对数型）。显然，时间复杂度为O（2^n）或O（e^n）的指数型算法的效率极低，在n值较大时无法实用。图1-4-1表明了各种时间复杂度的增长率。

实践中我们可以把事前估算和事后统计两种办法结合起来使用。还以矩阵自乘为例，若

上机运行一个 10×10 的矩阵模型，执行时间为 12ms。则由算法时间复杂度 T (n) =O (n^3) 可估算一个 31×31 的矩阵自乘的时间大致为 (31/10)^{3}12ms≈358ms。这种办法很有实用价值，它用小模型可推得大尺寸问题的耗用机时。

对空间占用的讨论可类似于时间复杂度，估算渐近空间复杂度简称为空间复杂度。由于问题中原始数据所占用的空间与算法无关，故一般考虑空间复杂度时，只估算算法中所需增添的辅助空间。如本节中讨论频度的第 (4) 个例子中，除了原始数据外只用了两个变量 (x 和 i) 的辅助空间，因为和问题的尺寸 n 无关，故空间复杂度 S (n) =O (1)。

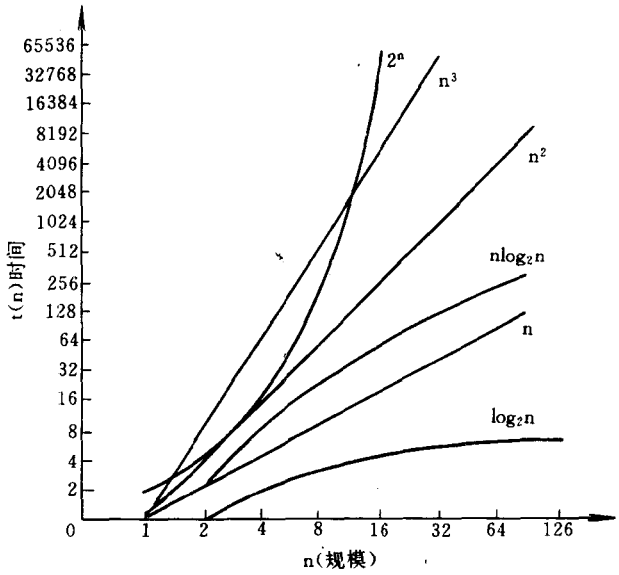


图 1-4-1　各种时间复杂度的增长率

一般情况下，时间和空间的开销是一对矛盾，要想空间比较节约，往往时间消耗就大；反之，时间开销小，常常得用空间资源的消耗来换取。具体在一个问题中到底注重哪一方面，这要看实际的需要和可能而定。也有一种观点，把时间和空间都考虑进来，以时空复杂度的乘积 T (n) * S (n) 为衡量算法的标准。但从实际的计算机发展趋势看，存储器的容量在不断扩大，成本又不断锐减。早期机器内存仅有几 KB，达到 640KB 已很满足了，现在动辄 16MB 已不是新鲜事；而机器速度的提高则不那么轻而易举。设计的算法也就应把时间开销作为首要的考虑因素，从更惜重时间的角度看问题，在本书中我们着重考虑时间效率，而假设内存足够大。

习　题

1. 仿照 1.1 节中给出的示例，举出一个实际问题中应用数据结构的例子，具体说明数据元素、数据元素类和数据结构的使用。

2. 估计下面三个程序段中标有@的语句频度。

(1) FOR i：=1 To n DO
　　FOR j：=1 To i DO
　　FOR k：=1 To j DO
　　@　X：=x+1；

(2) i：=1；
　　WHILE i<=n DO
　　〔@1　x：=x+1；
　　@2　i：=i+1
　　〕

(3) x：=n；y：=0；
WHILE x>=（y+1）*（y+1）DO
@y：=y+1；

3. 如果两个算法的频度分别为 n^2 和 $2^n/4$，当 n 大于何值的正整数时 $n^2 \leqslant 2^n/4$？

4. 若一个算法的频度为 $\sum_{i=1}^{n} i^k$，证明算法的时间复杂度 $T(n)=O(n^{k+1})$。

5. 二阶 Fibonacci 序列定义为 $F_0=1$，$F_1=1$，对于任何正整数 n 有 $F_n=F_{n-1}+F_{n-2}$（$n\geqslant 2$），用 PASCAL 和 C 两种语言编写，求二阶 Fibonacci 序列的算法，分析该算法的时间复杂度。

6. 用 PASCAL 和 C 两种语言编写求多项式

$A(x)=a_n x^n+a_{n-1}x^{n-1}+\cdots+a_1 x+a_0$

的值 $A(x_0)$ 的算法。通过估算频度求算法的时间和空间复杂度。

（提示：$A(x_0)=(\cdots((a_n x_0+a_{n-1})x_0+\cdots+a_1)x_0+a_0)$