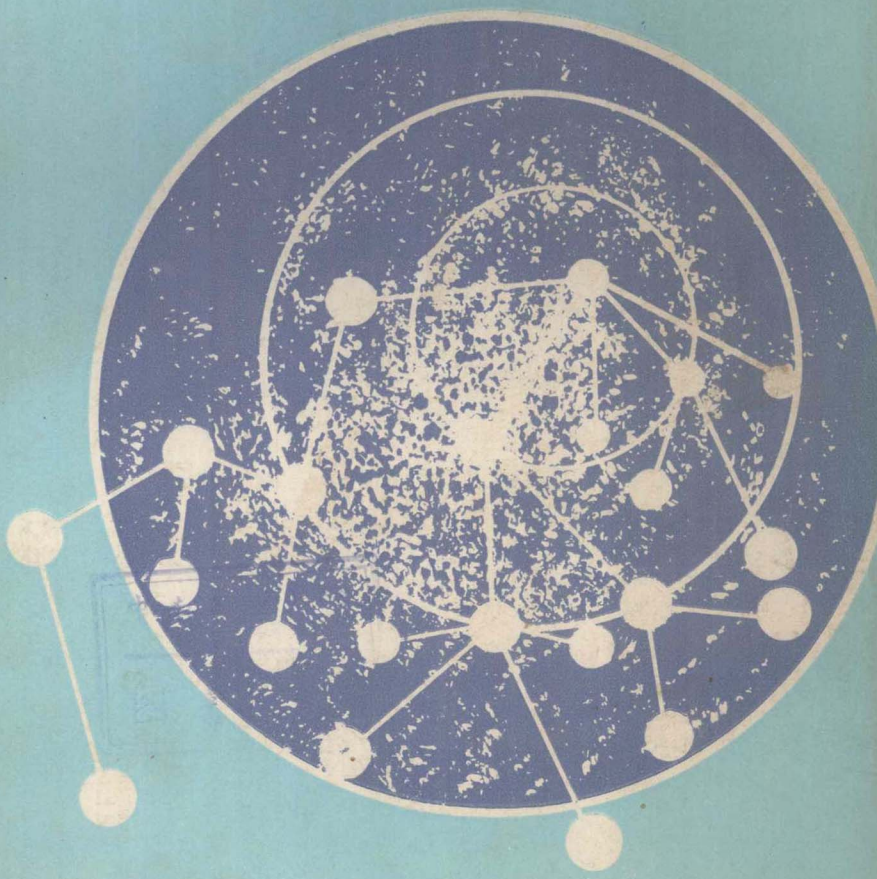
高等学校教材

数据结构

倪 铃 主编



西北工业大学出版社

高等学校教材

数 据 结 构

倪 铃 主编

西北工业大学出版社

内 容 提 要

本书以通俗的语言阐述了各种数据结构的基本概念、逻辑特性及其存贮结构。对各种数据结构给出了有关算法，並使用文字或类PASCAL语言对算法进行描述，同时对各种算法进行了分析。

全书共分十二章。在第一章中，对数据结构与算法的关系进行了初步的研究，以后各章，除依次介绍串、线性表、链表、数组、树和图等主要数据结构及有关算法外，还较详细地介绍了数据结构在排序、查找及文件处理等方面的应用。最后给出了数据结构的一些示例。本书注意了理论与实践相结合，结合各章节的内容，给出了适当的例题。各章后面还附有习题可供练习。

本书可作大专院校计算机专业的教材，也可供从事计算机科学与软件工程的科技工作者参考。

高等学校教材

数 据 结 构

主 编 倪 铃

责任编辑 李 珂

*

西北工业大学出版社出版

（西安市友谊西路127号）

陕西省新华书店发行

西北工业大学出版社印刷厂印装

*

开本 787×1092 毫米 1/16 16.125印张 388千字

1988年4月第1版 1988年4月第1次印刷

印数 1—8000册

ISBN 7-5612-0047-1/TP·12(课) 定价：2.70元

前　言

从本质上分析，人类大部分工作都是非数学的。只有一小部分活动，才采用数学公式加以描述，并利用计算机作数值计算。即使像“机械工程”那样的“硬”科学，大部分“思考”也是靠“符号推理”，而不是靠计算来完成的。生物学、医学、法律、管理学等都是如此。正是在这样的背景下，随着计算机的飞速发展，处理大量的非数值问题（数据处理或信息处理），已成为计算机愈来愈重要的任务。“知识信息处理系统”，即所谓第五代计算机研制任务的提出，则标志着计算机功能从单纯的数据处理向知识智能处理的转变。E.A.费吉鲍姆预言：“90年代是软件技术空前大发展的时期，而最最重要的是那些新软件思想将完全改变‘计算’的概念”。数据结构就是为研究和解决计算机处理这种非数值问题而产生的理论、技术和方法。因此，对于学习或从事计算机软件工程的人们来说，它已成为必不可少的核心课程。

《数据结构》成为计算机科学的各种教学计划中的核心课程时间不长。在国外，1968年开始正式以《数据结构》名称而作为课程设置。由于众所周知的原因，直到1978年秋季，在国内院校先后开出不同学时型的《数据结构》课。从而产生了对于相应教材的迫切需求。根据航空高等院校电子计算机专业的要求，我们编写了这本教材。

在编写本书的过程中，我们认为把对数据结构的阐述与用某种现有的程序设计语言实现数据结构加以区别是必要的。

本书在第一章中，着重介绍了数据结构与算法的关系，并为以后对各章节中的各种算法分析打下一定的基础。第二章至第五章，分别介绍了线性结构中的串、线性表、链表和数组的结构特征，基本运算及在内存中的存贮结构。第六章和第七章，分别介绍了非线性结构中的树和图，先后给出了定义、特征、在计算机中常用的几种存贮方式以及常用的一些算法。第八章至第十章，在上述基本数据结构的基础上，介绍了几种常用的查找和排序的方法。第十一章，介绍了文件系统，为学习数据库打下一定的基础。第十二章，结合各章所介绍的内容，用PASCAL语言书写的完整程序，给出一些典型数据结构示例，以帮助初学者提高解题能力。

本书对给出的各种算法，都不同程度地作了时间复杂性和空间复杂性的分析，同时给出了适当的例题，以加深读者对各种数据结构的理解。此外，由于《数据结构》是一门实践性环节较强的专业基础课，所以在每一章（除第十二章）的最后，都备有一定数量的习题，以供读者练习。

本书作为计算机软件专业教材，讲授学时为70学时左右，对于其它专业（例如，信息处理，计算机应用，应用数学等），内容可作不同程度的取舍。通过本书的学习，对数据结构会有较全面的认识，并为独立进行有关领域的科研及软件工程设计打下良好的基础。

学习本课程前，读者应具有程序设计的基本技术。同时应具有PASCAL语言、离散数学和概率论的知识。

本书由倪铃主编，各章执笔人为：第一、二、三、四、五、六章倪铃，第七、十章高宏

宾，第八、九章吴健，第十一章孙坤，第十二章石志强。

北京航空学院王人骅副教授对本书进行了认真地审阅，并提出许多宝贵意见。在编写过程中，还得到西北工业大学计算机软件教研室有关同志的大力支持，在此，我们一并表示衷心地感谢。

我们虽从事计算机软件的教学和科研多年，但由于水平所限，又缺乏经验，加上编写时间仓促，因此书中的错误和不妥之处在所难免，恳切希望读者不吝指正。

编　者

1987 年 4 月于西安

目　　录

第一章　数据结构与算法

近几年来电子计算机的应用得到了迅速地发展，计算机的信息加工，已从简单的数值计算，发展到大量地解决非数值问题，其加工处理的信息，也由简单的数值发展到字符。对许多规模大、结构复杂的程序，要考虑其效率及可靠性，不仅必须对程序设计的方法进行系统地研究，同时还要研究程序所处理的信息。因为信息的表示和组织，直接影响计算机程序处理信息的效率。

计算机所加工处理的信息我们称它为“数据”，计算机科学就可以看成是研究数据，以及在计算机中的表示和转换数据方法的一门科学。大多数情况，这些数据并不是杂乱无章的，数据之间往往存在着重要的结构关系，这就是数据结构的重要内容。

当原始数据输入计算机后，要求设计一个算法，才能将输入数据转换成要加工的数据，否则就不能达到转换的目的。因此，我们可以看出数据结构和算法两者之间有着密切的联系。事实上，应该把数据结构和算法看成一个整体，缺少一方另一方即失去意义，这一点从下面几节中也可以看出。

§1.1　数据结构的概念

数据结构与数学、计算机硬件和计算机软件有着十分密切的关系，它是操作系统、编译原理、数据库、情报检索、人工智能等课程的重要基础。

§1.1.1　什么是数据结构

什么是数据结构？在回答该问题之前，先讨论在数据结构中，常常涉及到的几个术语的概念，这些术语包括数据、数据类型和数据对象。

我们知道，计算机是处理数据的工具，它把输入的原始数据经过加工处理成我们所要求的数据。因此，数据可以说成是一个能输入计算机并能被计算机程序加工处理的符号集合。集合中的元素，是一些能够描述客观事物的数字、字符和符号等，并称之为数据元素。

数据类型是指在程序设计语言中各个变量所具有的数据种类。在 FORTRAN 语言中，数据类型可为整型、实型、逻辑型和复型，在 PASCAL 语言中除上述数据类型外，还有由用户自己定义说明的纯量类型以及用户定义的构造类型等。每种程序设计语言都配备有一组内部数据类型。也就是说，程序设计语言允许变量指定数据类型，并提供一组有意义的运算以对这些变量进行处理。有些数据类型是容易提供的，因为这些类型的运算，机器语言本身就实现了。整数和实数的算术运算就是这样的例子。实现其它数据类型，则需要很大的工作量。

数据对象是指某种数据类型的元素集合，如整数的数据对象是集合$D=\{0, \pm1, \pm2, \cdots\}$；按字母顺序排序的英文字符的数据对象是集合$D=\{'A', 'B', \cdots, 'Z'\}$。在这两个集合中，前者是无限的，后者是有限的。因此，数据对象是数据的一个子集，它可以是无限

的，也可以是有限的。

数据结构与数据对象不同，数据结构不仅要描述数据对象，而且要描述数据对象各元素之间的相互关系，但它并不涉及元素的具体内容。所谓关系，即描述数据对象中各元素之间的运算及运算的规则。如数据对象整数I，再加上如何进行+、-、*、/等运算的描述，其中包括运算集合P和P中各运算的运算规则的集合B，就构成了一个整数数据结构D的定义。也就是说，D是由I，P和B组成的，从而得出下面的结论：在数据集合中，数据元素和其相互关系结合在一起就称作数据结构。又可给出如下的形式定义，数据结构D是一个二元组D=(K，R)，其中K是数据元素的有限集合，而R是K上的关系的有限集合。数据结构有时很复杂，一个数据结构中的数据元素可能是另一个数据结构。

数据结构不仅要研究数据的逻辑结构和物理结构，而且对每种数据结构均定义相应的运算，同时给出相应的算法，分析算法的效率。其中数据的逻辑结构是指数据元素之间的结构关系，物理结构是指数据元素在计算机内如何表示。在研究过程中，我们还将给出若干重要的例子，来说明各种数据结构在实际中的应用。本书所要研究的数据结构类型有串、栈、队列、数组、链表、树、图、文件等。

§1.1.2 为什么要学习数据结构

首先让我们回顾一下数据结构这门课程的发展过程，对于理解数据结构内容和学习数据结构的重要性是有意义的。

早期的电子计算机仅用于科学计算，在60年代中期，有些国家为了研究当时已经出现的几个表处理系统，分别开设了与数据结构有关的“表处理语言”，如LISP系统，SNOBOL系统等。这些语言的数据对象的结构形式多为表结构或树结构。1968年在美国某些大学计算机科学系的教学计划中，把“数据结构”作为一门独立的课程，但该课程的内容范围并没有具体的规定。最初，数据结构几乎和图论，特别和表、树的理论是同义语。随后，这个概念又扩充到包括网络、代数、集合论、格、关系等方面，统称为“数据结构”的内容。由于数据必须在计算机中进行处理，因此不能只考虑数据本身的数学性质，还必须考虑数据的物理结构。这样一来，就进一步扩大了数据结构的内容。后来，人们逐渐将数据的有关数学性质独立出来，形成了现在的“离散数学结构”，而把数据的逻辑结构、物理结构以及对每种结构所定义的运算作为“数据结构”的主要内容。

近年来，由于数据库，情报检索系统的不断发展，在数据结构的课程中，又增加了文件结构，特别是大型文件的组织等方面的内容。

《数据结构》是计算机科学专业的一门核心课程。因此学好数据结构这门课是十分重要的。

pen
abacus
book
rubber
triangle
meter

图1.1 文具表

§1.1.3 数据结构的分类

在本书中，将介绍不同类型的数据结构。实际上，读者也许已经接触过它们，但没有认识到这就是各种不同类型的数据结构。

图1.1所给出的表，就是数据结构的一种类型。表中所有顺序书写的单词可以顺序存贮在计算机内存单元中，这一点很容易做到。为了访问这张表，我们必须知道表的起始地址和

表的长度。当我们要往这张表上增加一些单词，或要删除一些单词时，就会出现一些基本的问题。当表是无序的，只要把增加的内容添加到表的末端上去，把要删去的内容从表中划去就可以了。因此，只要有足够的连续存贮空间，在一个无序表里增加或删除名字都是很容易的。但如果表是有序的，插入和删除就不那么简单了。为了保持表的有序性，对修改过的表要不断地进行整理。

对于表格（或矩阵可作为二维数组）的存贮方法，同样可以采用按行或按列顺序存放，要访问其中某一个元素，只要知道该元素在此表格中的行号和列号就可以了。在进行表格处理时，必须要考虑对元素的插入、删除和修改等，这些在后面几章中要进行专门的讨论。

形式上更为复杂的一种数据结构是树，树通常用来表示元素的分层组织。建立人事档案就是树型结构的一个例子，其结构如图 1.2 所示。它说明了查找一位教师或一位学生的档案材料时必须遵循的次序。树的重要特征是建立了层次的概念。层次可以说明什么更为重要，什么必须先完成，或者什么更为概括。这是一种单一路径结构的概念，即从最上层的项寻找至最底层的每一项，有且仅有一条路径可走。

图是一种多路径结构，我们可以把图想像成是描绘若干个城市及它们之间的公路网，且标上公路的长度，如图 1.3 所示。图在运输中有着广泛地应用，当司机要行径某几个城市时，它有助于找到最短的路线。

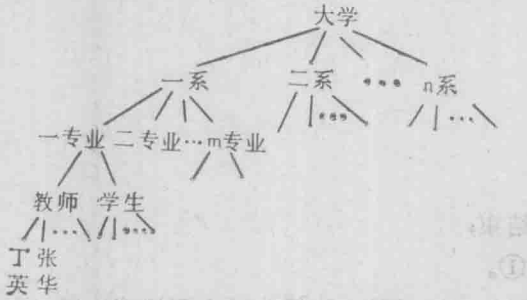


图 1.2 一所学校人事档案的逻辑结构

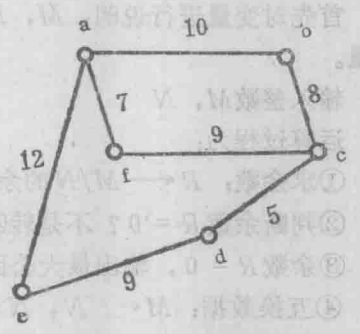


图 1.3 顶点代表城市，边为标以长度的公路图

上面例举了四种常见的数据结构，在这四种数据结构中，若考虑它们中间各个数据元素之间的相对关系，则可将其结构分为两大类：即线性结构和非线性结构。表和数组等为线性结构，树和图均为非线性结构。（对于非线性结构，读者可以提前思考这样一个问题：如何把这种非线性结构存放到顺序表示的存贮单元中，并且能准确地表示出它们的结构？）

我们还可以按照数据的结构（辑逻结构和物理结构）特性在该数据结构存在期间的变动情况，将它们划为静态结构，半静态结构和动态结构。其静态结构就是在数据结构存在期间总是不变动的，例如向量和数组。而半静态结构，则对其结构允许有较小的变动，如允许向量的上限和下限可以改变，这就是所谓的栈和队列。当一种结构在某种模式范围内，可以随机地重新组织时，称为动态结构，如链表结构。从不同的角度对数据结构进行分类，是为了更好地认识它们，更深入地了解各种结构的特性。如栈和队是线性结构，又是半静态结构；树是非线性结构，又是动态结构。

数据结构在本书中的重点侧重于实践技术，而不是理论方面，它是以程序设计，离散数学

为基础。但对数据结构及其算法进行分析时，就需要如图论、集合论、组合分析、概率论等方面的理论基础。它们在为分析算法提供性能公式时是一种很重要的手段，我们将依据这些公式来选择合适的算法。

§1.2 算　法

数据结构中算法是一个十分重要的内容。实现算法、表达算法、验证算法、分析算法、研究和改进算法，构成了计算机科学的各个分支，有人这样说过："计算机科学就是研究算法的科学"。本书在讨论各种数据结构的基本运算时，也都将给出相应的算法，那么什么是算法呢？

§1.2.1 算法的概念

算法（Algorithm）一词经常同欧几里德算法"Enclid's algorithm"联系在一起，该算法是求两个数的最大公因子的过程，给出欧几里德算法，对帮助理解"算法"是有好处的。

例 欧几里德算法。给定两个正整数m和n，求它们的最大公因子，即能够同时整除m和n的最大的正整数。

首先对变量进行说明：M，N分别为存放正整数的变量，R为存放N除M所得之余数的变量。

输入整数M，N

运算过程为：

①求余数：$R \longleftarrow M/N$的余数；

②判断余数$R=0$？不是转④；

③余数$R=0$，输出最大公因子N，运算结束；

④互换数据：$M \longleftarrow N$，$N \longleftarrow R$，返回①。

上述的运算过程就是一个算法，如果$m=544$，$n=119$，连续执行上面的算法后，在步骤③处输出两个数的最大公因子$N=17$，且算法到此结束。我们可以看出算法的意义十分类似于过程、方法、规程、程序。一个算法就是一个有穷规则的集合，其中的规则规定了一个解决某特定问题的运算序列。

一个算法应具有如下五个重要的特性：

1. 有穷性：一个算法必须保证在执行有穷步之后结束。

欧几里德算法是满足这个条件的，因为余数R的值总是小于N。若$R \neq 0$，经过④中互换数据后，N的值减小。正整数的递减序列最后必然要终止。所以，对于任意给定的正整数n，步骤①的执行次数是有穷的，尽管有时执行的次数可能很大。

2. 确定性：算法的每一个步骤必须是确切定义的。

如欧几里德算法中的第①步，当M和N为正整数时，余数R的值是确切知道的，并且R是一个非负的整数。当$R \neq 0$时，步骤④的执行并不影响第①步的确切定义。但当我们写出一个烹调方法时，经常是缺乏确定性的，如"加糖少许"。糖加到哪里？少许到什么程度？这种情况不能出现在算法中。

3. 输入：一个算法有零个或多个输入，也就是在算法开始之前，对算法给出的初始量。

欧几里德算法中，有两个取自正整数集合中的输入，即M和N。

4. 输出：一个算法有一个或多个输出，它是与输入有某个特定关系的量。

欧几里德算法中，有一个输出，即两个输入的最大公因子。

5. 能行性：一般说来，期望一个算法是能行的，就是说原则上都是能够精确地进行，而且人们用笔和纸做有穷次即可完成。

欧几里德算法中，仅仅使用一个正整数除以另一个正整数的除法，测试一个整数是否为零，将一个变量的值置于另一个变量中这样一些运算，这些运算显然都是能行的。

§1.2.2 算法的分析

在实际问题中，我们不仅需要算法，而且要求有一个好的算法。当解决一个具体的问题时，可能会有若干个算法供选用，只有对这些算法进行分析，才能知道哪一个算法是最好的。因此，对算法进行分析是十分必要的。下面我们对一个求最大值的算法进行分析。

对给定的元素，a_1，a_2，…，a_{n-1}，a_n，求出最大的元素a_i，并使其i尽可能地大。

先将N个元素存放在一维数组$A(N)$中，M为存放最大值的变量，I、K为下标变量，其中$A(I)$为最大值。

因为要求I尽可能地大，因此从最后一个元素开始选择。

具体算法如下：

①变量置初值：$I \longleftarrow N$，$K \longleftarrow N$，$M \longleftarrow A(N)$；

②求前面一项的下标：$K \longleftarrow K-1$；

③前面还有元素吗？即$K=0$？没有元素，转⑥；

④前面还有元素，M内还是最大值吗？即$A(K) \leqslant M$？是，转②；

⑤不是，保留新的最大值及其下标。$I \longleftarrow K$，$M \longleftarrow A(K)$并转②；

⑥输出最大元素的下标I和最大值M，运算结束。

在上述算法中，所要求的存贮量是一定的。我们仅仅分析执行所有步骤时所需要的时间。对每一个执行步骤，我们需要知道两个量。第一个是一次执行所需要的时间；第二个是执行的次数。这两个数的乘积就是该步骤所占用的时间总量。第二个数字称为频数(frequency count)，估计频数是件比较困难的事。而执行一条命令所需要的时间，则和所使用的机器等有关。对于上面的算法，首先计算执行每一步的次数，可列表如下：

步　骤	执行次数
1	1
2	n
3	n
4	$n-1$
5	T
6	1

在右面的表中，除了第⑤步中的数量T外，其余各步骤执行的次数都是已知的。而数量T是改变当前的最大值的次数，为了完成分析，必须知道T的变化情况。

当a_n为最大元素时，除第①步中$M \longleftarrow A(N)$外，M中的值以后一直没有被改变，这时$T=0$为最小值。

当a_1为最大元素，且元素有下面的关系：$a_1 > a_2 > \cdots > a_n$，则最大值M将被改变$n-1$次，这时$T=n-1$为最大值。

因此，T的平均值一定在0和$n-1$之间。假定序列中的元素$a_k(k=1,2,\cdots,n)$的值是互不相同的，则它们有$n!$种排列，每一种排列的可能性我们都认为它是相同的。假设$n=3$，

那么有以下 3! 种排列情况，且每种都是同等可能的。对每种情况，依据上述算法可以得到一个 T 值（如右表）。

排列情况	T 值
$a_3>a_2>a_1$	0
$a_3>a_1>a_2$	0
$a_2>a_3>a_1$	1
$a_2>a_1>a_3$	1
$a_1>a_3>a_2$	1
$a_1>a_2>a_3$	2

T 的平均值是：$(0+0+1+1+1+2)/6=5/6$，可见，T 的平均值并不依赖于具体元素 a_k 的大小，而仅仅与它们之间的相对次序有关。通常 T 的平均值定义为：

$$T_n=\sum_{i=1}^{n-1} iP_i$$

T 取值 i 的概率

$P_i=$（n 个元素的排列中满足 $T=i$ 的个数）$/n!$

上表中 $n=3$，$P_0=2/6=1/3$，$P_1=3/6=1/2$，$P_2=1/6$，数学上可以证明 $T_n=H_n-1$，其中

$$H_n=1+\frac{1}{2}+\cdots+\frac{1}{n}=\sum_{i=1}^{n}\frac{1}{i}$$

到此，我们基本上完成了上述算法的分析，求出了执行的次数，在一台确定的计算机上运行，所花费的时间也就很容易知道了。

下面再看一个根据算法中语句执行的最大频数来分析一个算法执行时间的数量级。请看图1.4中的三个例子：

```
⋮                  for i:=1 to n do        for i:=1 to n do
x:=x+1                 x:=x+1                  for j:=1 to n do
⋮                  end                             x:=x+1
                                                   end
                                               end
```

(a)　　(b)　　(c)

图 1.4　计算频数的三个简单程序

在程序(a)中，我们假设语句 x:=x+1 不包含在任何循环中，因此，它的频数是 1。在程序(b)中，同一个语句将执行 n 次。而在程序(c)中则执行 n^2 次（假设 $n>1$）。1，n 和 n^2 具有各不相同的递增的数量级。在具体进行分析时，我们所关心的主要是把一个算法的数量级确定下来，也就是说去确定那些具有最大频数的语句。在确定数量级时，会经常遇到如下的一类公式：

$$\sum_{1\le i\le n} 1\quad,\quad \sum_{1\le i\le n} i\quad,\quad \sum_{1\le i\le n} i^2$$

上面三个式子的结果分别为：

$$n\quad,\quad n(n+1)/2\quad,\quad n(n+1)(2n+1)/6$$

在图 1.4(c)的程序段中，语句 x:=x+1 被执行 $\sum_{1\le i\le n}\sum_{1\le j\le n} 1=\sum_{1\le i\le n} n=n^2$ 次。

一般有下面的公式：

$$\sum_{1\leqslant i\leqslant n} i^k=\frac{n^{k+1}}{k+1}+\text{低次项},\quad k\geqslant 0$$

为了弄清这些概念，我们来看一个计算第 n 个斐波那契（Fibonacci）数的简单程序。斐波那契序列开始为：

0, 1, 1, 2, 3, 5, 8, 13, 21, 24, 55,…

其中每个新项是前面二项之和。设序列的第一项为 F_0，那么，$F_0=0$，$F_1=1$，而一般项为：

$$F_n=F_{n-1}+F_{n-2};\quad n\geqslant 2$$

下面的程序用来求取第 n 项斐波那契数 F_n

```
        procedure  fibonacci
        {n 为任一非负整数}
        begin
1       read(n)
2,3,    if  n<0  then  [write('error'); stop]
4,5,    if  n=0  then  [write('0'), ; stop]
6,7,    if  n=1  then  [write('1'); stop]
8,                 else  [fn1:=0; fn2:=1
9,                       for  i:=2  to  n  do
10,                      fn:=fn1+fn2;
11,                      fn1:=fn2
12,                      fn2:=fn]
13,                write(fn)
        end;    {fibonacci}
```

首先分析 n 可能取什么值，n 的完整集合应包括四种情况，即 $n<0$，$n=0$，$n=1$ 和 $n>1$。下表列出了前面三种情况的频数。

步骤	$n<0$	$n=0$	$n=1$
1	1	1	1
2	1	1	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	1
8～13	0	0	0

这三种情况都没有意义，它们都不能使程序的作用发挥出来。重要的是要对 $n>1$ 的情况进行分析，这时才真正进入 for 循环。步骤 1，2，4，6，8 只执行一次，而步骤 3，5，7 根本不执行。步骤 9 中的两条命令各执行一次。当 $n\geqslant 2$ 时，第 9 步执行 n 次。因为虽然从 2 到 n 只有 $n-1$ 次，但最后有一次还是返回步骤 9，这时 i 已增加到 $n+1$，测试 $i>n$ 成立，于是转移到步骤 13。因此步骤 10,11,12 都将执行 $n-1$ 次，我们用图 1.5 中的表来概括 $n>1$

的分析。

步骤	1	2	3	4	5	6	7	8	9	10	11	12	13
频数	1	1	0	1	0	1	0	2	n	$n-1$	$n-1$	$n-1$	1

图 1.5 计算 F_n 时各步骤执行次数

每个语句计数一次，因为第 8 步有两个语句各执行一次，于是总的计数是 $4n+4$。以后，我们常常略去两个常数 4，把这种计数写成 $O(n)$。这个 O 记号表示数量级与 n 成正比。

定义：当且仅当有两个常数 C 和 n_0，对所有 $n \geqslant n_0$，使得 $|f(n)| \leqslant C|g(n)|$ 成立，则 $f(n)=O(g(n))$。

$f(n)$通常表示某个算法的时间。如果我们说一个算法的计算时间为 $O(g(n))$，通常称为算法的时间复杂性，则表明它所需要的执行时间只是 $g(n)$的某个常数倍。n 是说明输入或输出的一个参数，它可以是输入的个数或输出的个数或者是它们的和。在斐波那契程序中，n 表示输入的量值，该程序的时间复杂性可以写成 $T(n)=O(n)$。

以后，当我们写 $O(1)$ 就意味着计算时间为一常数，$O(n)$ 称为线性的，$O(n^2)$ 称为平方的，$O(n^3)$称为立方的，$O(2^n)$ 称为指数的。若一个算法所用的时间是 $O(\log n)$，则当 n 充分大时，它比 $O(n)$速度快。同样，$O(n\log n)$比 $O(n^2)$ 要好，但不如 $O(n)$。下面这七个计算时间 $O(1)$、$O(\log n)$、$O(n)$、$O(n\log n)$、$O(n^2)$、$O(n^3)$ 和 $O(2^n)$ 都是全书中最常见的。

若有两个执行同一任务的算法，前者的计算时间为 $O(n)$，后者为 $O(n^2)$，通常取前者较好。因为，随着 n 的增大，第二个算法的时间要比第一个算法大得多。图 1.6 表示：当常数等于 1 时，下面六种计算时间是如何随着 n 增长的。

$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	6	24	64	512	256
4	16	64	256	4 096	65 536
5	32	160	1 024	32 768	2 147 483 648

图 1.6 计算函数的值

从表中可以看出，时间$O(n)$和 $O(n\log n)$的增长要比其它时间的增长慢得多。综上所述，给出一个算法，分析每个句子的频数并求出总和，可得到一个多项式

$$P(n)=C_k n^k + C_{k-1} n^{k-1} + \cdots + C_1 n + C_0$$

其中 C_i 为常数，$C_k \neq 0$，n 为一个参数。如利用大O记号，则 $P(n)=O(n^k)$。由此可以看出，$P(n)$的时间复杂性只要考虑具有最大频数的语句就可以了。假设任一步骤执行 2^n 次或 2^n 次以上，则表达式应为

$$C2^n + P(n) = O(2^n)$$

空间复杂性是测量算法性能的另一种方法。所谓一种算法的空间复杂性，是按该算法编写的程序在计算机中运行时所占用的存贮单元总数，其中包括程序本身的长度以及所有工作单元之长度。人们常常可以利用空间来换取时间，以得到一个较快的算法，在后面将会看到这样的实例。