高等学校教材

数据结构

SHUJUJIEGOU

高宏宾　倪　铃　编著

1994年(修订本)

西北工业大学出版社

高等学校教材

数据结构

（1994 年修订本）

高宏宾　倪　铃　编著

西北工业大学出版社
1997 年 2 月　西安

（陕）新登字 009 号

【内容简介】 本书以通俗的语言系统地介绍了各种类型的数据结构的基本概念、逻辑特性及其存储结构，并对各种数据结构的有关运算给出了相应的实现算法，同时对有关算法进行了简要的分析。

全书共分 11 章。在第一章中，初步介绍了数据结构和算法的概念及其关系。在以后各章中，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查找、排序及文件组织等方面的应用。本书注意了理论与实践相结合，在算法设计上，强调其方法和技巧。为使读者熟悉掌握所学的内容，各章后面均附有一定数量的习题可供练习。

本书可作为大专院校计算机专业的教材，也可供从事计算机工程与应用工作的科技工作者参考。

高等学校教材

数据结构

（1994 年修订本）

高宏宝 倪 铃 编著

责任编辑 李 珂

责任校对 享 邑

*

（710072 西安市友谊西路 127 号 电话 8493844）

陕西省 新华书店发行

西安电子科技大学印刷厂印装

ISBN 7－5612－0940－1/TP・125（课）

*

开本：787×1092 毫米 1/16 印张：15.25 字数：365 千字

1988 年 4 月 1 版 1997 年 2 月第 2 版第 3 次印刷

印数：10501－14500 册 定价：16.00 元

修订版前言

数据结构课程的教学在我国已有十多年的历史。近几年来，我们在数据结构的教学实践和教学改革方面作了一定的努力。本书是对1988年版的修订，同时也是我们近几年来工作的一个小结。

我们认为，数据结构课程的教学目的是使读者通过学习各种典型数据结构及其实现方法，学会分析研究实际问题中数据对象的特性，并能根据问题的需要设计出合理组织这些数据的数据结构和存储结构，同时依照相应的存储结构设计出正确的、高质量的算法来实现所设计的数据结构。虽然这样的要求高了一些，但这是学习数据结构的最终要求。因此，我们在系统地介绍数据结构的基本内容的过程中，强调在一定的存储结构下，对算法设计的方法和技巧的介绍，并把这些方法技巧溶进大量的算法设计实例之中。各章后面所附的算法设计题目将有助于提高读者的算法设计能力。

在内容安排上，全书分为11章。第一章着重介绍数据结构与算法的有关概念及其关系，并对书中描述算法所用的语言以及算法的度量作了概要说明；第二章至第七章，分别介绍了串、线性表、链表、数组、树以及图等几种基本类型的数据结构的特征、基本运算、存储结构以及相应运算的实现算法；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 介绍了文件的组织方法。由于篇幅所限，在这次修订中我们删去了原版第十二章的内容。

只要读者掌握了一定的程序设计的基本技术以及离散数学和概率论的初步知识，便可以学习和掌握本书所介绍的基本内容。本书作为计算机系本科学生的教材，讲授学时为64学时。若作为专科或其他相关专业的数据结构教材时，内容可作不同程度的取舍。

本书由西北工业大学高宏宾编写第二、六、七、八、九、十章；无锡纺织工业管理干部学院倪铃编写第一、三、四、五章；第十一章由西北工业大学孙坤编写。另外，西北工业大学吴健提供了第九、十两章的初稿，最后高宏宾统稿。

西安电子科技大学罗昌隆教授、朱儒荣教授对本书进行了仔细的审阅，提出了许多宝贵的意见。在本书编写过程中，我们还得到了西北工业大学出版社的许多同志以及计算机系的张遵濂、蒋立源、徐秋元、韩兆轩、白中英、胡正国等教授的支持、关心和帮助。陈向东同志还帮助抄写了部分手稿，在此一并表示衷心的谢忱。

编　者

1994年1月于西安

原版前言

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，在国内一些院校才先后开出不同学时的“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储结构。第六章和第七章，分别介绍了非线性结构中的树和图，给出了定义、特征、在计算机中常用的几种存储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，对于其他专业（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。

目　录

第一章　数据结构与算法 …… 1

§1.1　数据结构的概念 …… 1
§1.1.1　什么是数据结构 …… 1
§1.1.2　数据结构的分类 …… 2
§1.2　算法 …… 4
§1.2.1　算法的概念 …… 4
§1.2.2　算法的分析 …… 5
§1.2.3　算法设计的基本步骤 …… 9
§1.2.4　数据结构与算法的关系 …… 9
§1.2.5　算法设计中所用的语言 …… 10
习题 …… 12

第二章　串 …… 14

§2.1　串的基本概念 …… 14
§2.2　串的存储结构 …… 14
§2.3　串的运算及其实现 …… 16
§2.3.1　串的联接 …… 16
§2.3.2　求串的长度 …… 17
§2.3.3　求子串 …… 17
§2.3.4　定位 …… 19
§2.3.5　置换 …… 19
§2.4　串的模式匹配 …… 20
习题 …… 24

第三章　线性表 …… 26

§3.1　线性表的定义及运算 …… 26
§3.2　线性表的顺序映象 …… 27
§3.3　栈 …… 29
§3.3.1　栈的定义及其运算 …… 29
§3.3.2　栈的应用 …… 31
§3.3.3　多个栈的情况 …… 35
§3.4　队列 …… 37

§3.4.1 队列的定义及运算 …… 37
§3.4.2 队列的顺序表示 …… 38
§3.4.3 循环队列 …… 39
习题 …… 42

第四章 链表 …… 44

§4.1 线性表的链接分配 …… 44
§4.2 链接的栈和队列 …… 49
§4.3 可利用空间表 …… 52
§4.4 循环链表 …… 53
§4.5 多项式加法 …… 54
§4.6 等价关系的处理 …… 60
§4.7 双重链表和动态存储管理 …… 64
§4.8 广义表 …… 73
习题 …… 75

第五章 数组 …… 79

§5.1 数组的顺序分配 …… 79
§5.2 稀疏数组 …… 83
§5.3 正交链表与稀疏数组 …… 86
习题 …… 89

第六章 树 …… 92

§6.1 树及其存储结构 …… 92
§6.1.1 树的定义 …… 92
§6.1.2 若干术语 …… 93
§6.1.3 树的存储结构 …… 94
§6.2 二叉树 …… 95
§6.2.1 二叉树的定义及几种特殊二叉树 …… 95
§6.2.2 二叉树的性质 …… 96
§6.2.3 二叉树的存储结构 …… 98
§6.3 遍历二叉树 …… 99
§6.3.1 遍历二叉树的递归算法 …… 99
§6.3.2 遍历二叉树的非递归算法 …… 101
§6.3.3 二叉树的唯一性问题 …… 102
§6.4 线索二叉树 …… 103
§6.4.1 直接后继和直接前驱的查找算法 …… 104
§6.4.2 二叉树的线索化 …… 107
§6.4.3 线索二叉树的插入算法 …… 108

§6.5 树和森林 …… 110
§6.5.1 树的二叉树表示 …… 110
§6.5.2 森林的二叉树表示 …… 111
§6.5.3 树和森林的遍历 …… 113
§6.5.4 树的其它表示形式 …… 114
§6.6 二叉树的路径长度及哈夫曼树 …… 116
§6.6.1 二叉树的路径长度 …… 116
§6.6.2 哈夫曼树及哈夫曼算法 …… 118
§6.7 树与等价问题 …… 121
习题 …… 125

第七章 图 …… 128

§7.1 图的基本概念 …… 128
§7.2 图的存储结构 …… 131
§7.2.1 图的矩阵表示 …… 132
§7.2.2 图的邻接表表示 …… 133
§7.2.3 图的其它表示形式 …… 135
§7.3 图的遍历和求图的连通分量 …… 138
§7.3.1 图的遍历 …… 138
§7.3.2 求图的连通分量 …… 141
§7.4 有向图的处理 …… 142
§7.4.1 单源最短路径 …… 142
§7.4.2 每对顶点之间的最短路径 …… 145
§7.4.3 拓扑排序 …… 147
§7.4.4 关键路径 …… 150
§7.5 无向图的处理 …… 153
习题 …… 155

第八章 数据查找 …… 159

§8.1 数据查找及其效率 …… 159
§8.2 顺序查找 …… 160
§8.3 二分查找 …… 161
§8.4 二叉排序树查找 …… 164
§8.5 哈希查找 …… 167
§8.5.1 哈希函数的构造技术 …… 168
§8.5.2 哈希冲突的处理方法 …… 170
§8.5.3 哈希法的分析 …… 175
§8.6 分块查找 …… 176
习题 …… 179

第九章　内部排序 …… 181

§9.1　插入排序 …… 181
§9.2　归并排序 …… 182
§9.3　快速排序 …… 188
§9.4　选择排序 …… 190
§9.5　堆排序 …… 194
§9.6　基数排序 …… 197
习题 …… 200

第十章　外部排序 …… 201

§10.1　外部设备简介 …… 201
§10.1.1　磁带 …… 201
§10.1.2　磁盘 …… 202
§10.2　2-路平衡归并排序 …… 203
§10.3　多路平衡归并排序 …… 206
§10.4　多阶段归并排序 …… 207
§10.5　初始归并段的产生 …… 210
§10.6　最佳归并排序 …… 211
习题 …… 215

第十一章　文件 …… 216

§11.1　文件的基本概念 …… 216
§11.1.1　术语 …… 216
§11.1.2　文件的存储与组织 …… 217
§11.2　顺序文件 …… 218
§11.3　随机组织文件 …… 220
§11.3.1　直接存取文件 …… 220
§11.3.2　索引文件 …… 222
§11.3.3　链表文件 …… 227
§11.4　B-树 …… 229
习题 …… 232

参考文献 …… 234

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的发展，计算机对信息的加工，已从简单的数值计算，发展到大量地解决非数值处理问题，其加工处理的信息，也由简单的数值发展到字符、图像、声音等具有复杂结构的数据。因此，对许多规模大、结构复杂的程序，若要考虑其效率及可靠性，则不仅要对程序设计的方法进行系统的研究，同时还要研究程序所加工的对象及其结构。也就是说，信息（数据）的表示和组织直接影响计算机程序处理信息的效率与质量。

计算机所加工处理的信息我们称它为“数据”。计算机科学可以看成是研究数据特性，以及数据在计算机中的表示和转换方法的一门科学。在大多数情况下，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”中要研究的问题。

当原始数据输入计算机后，要求设计一个算法，才能将输入的数据转换成要加工的信息，否则就不能达到转换的目的，而算法的设计与数据的结构有着密切的关系。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，常常涉及的几个术语的概念，这些术语包括数据、数据元素、数据类型和数据对象。

计算机把输入的原始数据加工处理成我们所需要的信息。这里，数据是计算机加工处理的“原料”，而信息则为加工后的“产品”。直观地说，**数据**（data）是描述客观事物的数、字符以及所有能输入到计算机中并能被计算机程序加工处理的符号集合。集合中的元素称为**数据元素**（data element），它是数据的基本单位。

数据类型（data type）是指在程序设计语言中各个变量所具有数据种类。在各种不同的程序设计语言中，它们都各有属于自己的一组基本数据类型，如FORTRAN语言中的一组基本数据类型为整型、实型、逻辑型和复型。又如PASCAL语言中的基本数据类型为整型、实型、逻辑型和字符型，用户还可以自己定义说明纯量类型以及构造类型等。程序设计语言中除有自己的数据类型外，还提供一组有意义的运算，以对这些变量进行处理。即数据类型是变量所能取的值与其可操作的运算的集合。有些数据类型是容易提供的，因为这些变量的运算，机器语言本身就实现了，如整数和实数的算术运算。实现其他数据类型，则需要很大的工作量。

数据对象（data object）是指某种数据类型的元素集合，如整数的数据对象是集合 I=

{0，±1，±2，…}；英文字符的数据对象是集合 C＝｛'A'，'B'，…，'Z'｝。在这两个集合中，前者是无限集合，后者是有限集合。由此看出，数据对象是数据一个子集，它可以是无限的，也可是有限的。

数据结构（data structure）是数据元素及其相互关系的集合。数据结构不仅要描述数据元素，而且要描述数据元素之间的相互关系，但不涉及元素的具体内容。数据元素之间的相互关系，表示了数据元素之间的某种联系，也称之为结构。更确切地说，数据结构是一个二元组 D＝（K，R），其中 K 是数据元素的有限集合，R 是 K 上的关系的有限集合。即 K 和 R 有机的结合，就是数据结构。如复数的数据结构 D_c＝（K，R），其中 K＝｛x|x 是实数｝，R＝｛<x，y>|x，y∈K，x 为实部，y 为虚部｝。由此，任何一对实数均可构成一个复数。同理，整数、实数也是数据结构，由于数据元素之间的关系十分简单，因此它们是一种简单的数据结构。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。应当指出，在上面的二元组中，R 是数据元素逻辑关系的集合。因此，这里的数据结构是指数据的逻辑结构，而把数据的物理结构看成是数据的逻辑结构在计算机中的存储方式。物理结构又称存储结构。在程序设计中，数据的逻辑结构和物理结构是不可分割的两个方面。这就决定了“数据结构”应该研究的问题。

回顾 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。这些语言的数据元素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。后来，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。在“数据结构”中对各运算给出相应的算法，且分析算法的效率。

近年来，由于数据库、情报检索系统、管理信息系统的不断发展，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。在研究过程中，我们还将给出若干例题，来说明各种数据结构在实际中的应用。本书所要介绍的数据结构类型有串、栈、队列、数组、链表、树、图、文件等。

pen
abacus
book
rubber
triargle
meter

图 1.1 文具表

§1.1.2 数据结构的分类

本书将介绍不同类型的数据结构。实际上，读者已经多次接触过它们。图 1.1 所给出的表，就是数据结构的一种类型。将表中所有顺序书写的单词依次存储在计算机内存单元中，是很容易做到的。为了访问这张表，我们要知道表在存储器中的起始地址和表的长度。如果我们要对表进行修改，如增加一些新单词，或将表中一些原有的单词删除，就会出现一些基本的问题。表若是无序的，只要把增加的内容添加在表的末端，或把要删去的内容从表中划去就可以了。因此，计算机中只要有足够的连续存储空间，在一个无序表里增加或删除单词都是很容易做到的。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。

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。

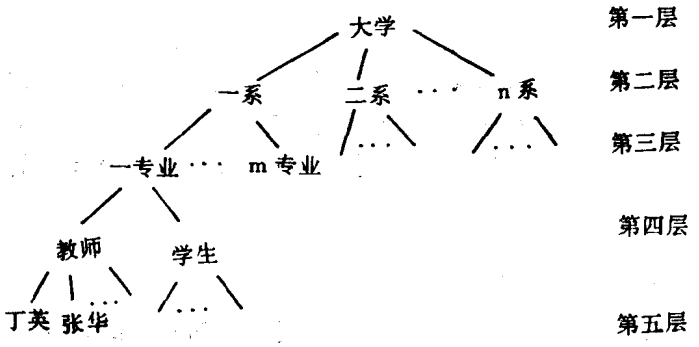

图 1.2　一所学校人事档案的逻辑结构

图是一种多路径结构，我们可以把图想象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。图在运输业中有着广泛的应用，例如当司机要行经某几个城市时，它有助于找到最短的路线。

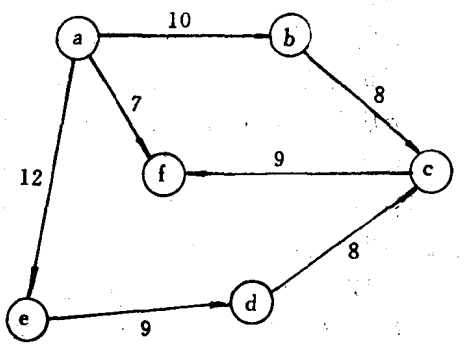

图 1.3　顶点代表城市、边为标以长度的公路图

上面例举了三种常见的数据结构，在这三种数据结构中，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储单元中，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（逻辑结构和物理结构）特性在该数据存在期间的变动情况，将它们划分为静态结构，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，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，又需要如图论、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

在数据结构中，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，都将以算法的形式给出。实际上，人们在进行程序设计时，基本问题在于算法，也就是说，程序设计就是在计算机上实现某种算法。

§1.2.1 算法的概念

算法就是解决问题的步骤，在本书中，算法就是解决数据结构问题的运算序列。人们有时将**算法**（algorithm）一词与欧几里德算法"Enclid's algorithm"联系在一起，该算法是求两个数的最大公因子的步骤，给出欧几里德算法，对帮助理解"算法"是有益的。

例 欧几里德算法。给定两个正整数 m 和 n，求它的最大公因子，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←m，N←n

运算过程为

① 求余数：R←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←N，N←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。一个算法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。

任意一个算法都应具有如下 5 个重要特性：

1. 有穷性　一个算法必须在执行有限步之后结束。

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≠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的执行次数是有限的，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≠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？这种情况不能在算法中出现。

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，就是说原则上都能够精确地进行，而且人们用笔和纸做有限次即可完成。

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。在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较好。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，求出其值为最大的元素 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←N，K←N，M←A [N]；

② 求前面一项的下标：K←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 是最大值吗？即 A [K] ≤M？是，转②；

⑤ 不是，保留新的最大值及其下标。I←K，M←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，所要求的存储量是一定的。下面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，则和所使用的机器有关。对于上面的算法，首先计算执行每一步的次数，如表 1.1 所示。在表 1.1 中，除了第⑤步中的数量 T 外，其余各步骤执行次数都是已知的。而数量 T 是改变当前最大值的次数，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←A [N] 处，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，…，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是相同的。假设 n=3，

那么有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（如表1.2）。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。

表 1.1

步骤	执行次数
1	1
2	n
3	n
4	n−1
5	T
6	1

表 1.2

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

通常T的平均值定义为

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

其中，P_i 为T取值i的概率，其值为

P_i=（n个元素的排列中满足T=i的个数）/n!

例如，在表1.1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。

下面我们来考虑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中的三个程序段。

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

$$\sum_{1\leqslant i\leqslant n} 1, \qquad \sum_{1\leqslant i\leqslant n} i, \qquad \sum_{1\leqslant i\leqslant n} i^2$$

上面三个式子的结果分别为

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

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

一般地有下面的公式：

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

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。斐波那契序列为

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \cdots$$

其中每个新项是前面两项之和。设序列的第一项为 F_0，且 $F_0=0$，$F_1=1$，一般项则为

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

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 [f_n1 : =0; f_n2 : =1;
9,                  for i : =2 to n do
10,                  [f_n : =f_n1+f_n2;
11,                   f_n1 : =f_n2;
12,                   f_n2 : =f_n];
13,                 Write (f_n)]
        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。表 1.3 列出了前面三种情况的频数。

表 1.3

步骤	n＜0	n＝0	n＝1	步骤	n＜0	n＝0	n＝1
1	1	1	1	5	0	1	0
2	1	1	1	6	0	0	1
3	1	0	0	7	0	0	1
4	0	1	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，这时才能真正进入 for 循环。步骤 1，2，4，6，8 只执行一次，而步骤 3，5，7 则不执行。当 n≥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 时的分析。对每个语句计数一次，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），称之为该算法的时间复杂性。其中 O 记号表示数量级与 n 成正比。

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

定义：如果存在两个常数 C>0 和 $n_0 \geqslant 0$，当 $n \geqslant n_0$ 时，$|f(n)| \leqslant C|g(n)|$ 成立，则称函数 f(n)是 O(g(n))，并记作 f(n)=O(g(n))。

f(n) 通常表示某个算法的计算时间。如果我们说一个算法的计算时间为 O(g(n))，或者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。其中 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等等。在斐波那契程序中，n 表示为一个输入量，该程序的时间复杂性可以写成 T(n)=O(n)，其中 g(n)=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。若一个算法执行的时间是 $O(\log_2 n)$，则当 n 充分大时，可以认为它比 O(n)的速度要快。同样，$O(n \log_2 n)$比 $O(n^2)$要快，但不如 O(n)快。下面这 7 个计算时间 O(1)、$O(\log_2 n)$、O(n)、$O(n\log_2 n)$、$O(n^2)$、$O(n^3)$和 $O(2^n)$是算法分析中最常见的。

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，第二个算法执行的时间要比第一个算法执行的时间大得多。当常数等于 1 时，图 1.6 表示出 6 种计算时间是如何随着 n 的增长而加大的。从图中可以看出，随着 n 的增大，时间 O(n) 和 $O(n \log_2 n)$ 的增长要比其他时间的增长慢得多。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，算法的时间复杂性 P(n) 只要考虑具有最大频数的语句就可以了。假设某一步骤执行 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储单元总数，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。