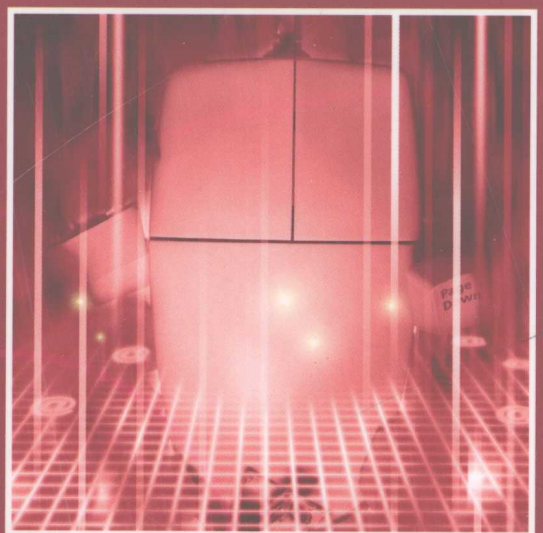
普通高等教育“十一五”计算机类规划教材

数据结构

戴敏 主编



免费电子课件

机械工业出版社
CHINA MACHINE PRESS

普通高等教育“十一五”计算机类规划教材

数据结构

主编　戴　敏
参编　董玉涛　刘凤连
肖迎元　孙志慧

机械工业出版社

本书共分9章，主要内容包括：算法设计与分析的基本知识，线性结构、树和图等各种基本数据结构的逻辑特点、存储结构、主要操作的实现与应用，递归、查找和排序等典型算法的实现及应用。本书每一章中都配有不同难度的例题和习题，帮助学生理解和掌握重点、难点问题。全书采用类C语言作为算法描述语言，各章的“简单应用举例”中含有很多实用的算法实例，既是本章算法的综合运用，也有助于培养学生根据求解的问题，合理选择数据结构，应用高级语言编写有效算法的能力。

本书可以作为全日制高等院校计算机科学与技术专业、信息与计算科学专业、电子信息科学与技术等信息类相关专业普通本科学生的专业基础课教材，也可作为广大从事计算机软件开发人员的参考书。为方便教师教学，本书配有教学课件，欢迎选用该书作为教材的老师索取，索取邮箱：llm7785@sina.com。

图书在版编目(CIP)数据

数据结构/戴敏主编. —北京：机械工业出版社，2008.2

普通高等教育“十一五”计算机类规划教材

ISBN 978-7-111-23201-8

Ⅰ.数… Ⅱ.戴… Ⅲ.数据结构—高等学校—教材 Ⅳ.TP311.12

中国版本图书馆CIP数据核字(2007)第206373号

机械工业出版社(北京市百万庄大街22号 邮政编码100037)

责任编辑：刘丽敏 责任校对：刘志文

封面设计：张 静 责任印制：洪汉军

北京铭成印刷有限公司印刷

2008年2月第1版第1次印刷

184mm×260mm·16印张·393千字

标准书号：ISBN 978-7-111-23201-8

定价：26.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

销售服务热线电话：(010)68326294

购书热线电话：(010)88379639 88379641 88379643

编辑热线电话：(010)88379726

封面无防伪标均为盗版

前　言

“数据结构”是计算机及信息等相关学科的一门重要专业基础课。用计算机来解决实际问题(特别是非数值计算类问题)时，要涉及数据的组织方法、存储结构以及数据的处理，而数据的表示及数据的处理正是数据结构课程的主要研究对象。通过本课程的学习，可以为后续课程，特别是软件方面课程的学习打下坚实的基础，同时也能培养和训练学生结合实际问题，合理选择数据结构，编写有效算法的能力。本课程所讨论的知识内容和提倡的方法为计算机程序设计提供了重要的理论基础，对于研发各种系统及应用软件具有不可替代的作用。本书正是针对这一背景，面向普通高等院校而编写的教材。

本书共分9章。第1章综述了数据结构中的一些基本概念，以及算法设计与分析的基本知识。第2~4章分别介绍了几种基本的线性结构，即线性表、栈和队列、多维数组与矩阵的压缩存储。第5章讨论了递归和递归算法的设计。第6章和第7章讨论了常用的非线性结构，即树和图。第8章和第9章介绍了信息处理中常用的运算查找和排序。

本书在内容表达上注重原理与实践的结合，配备大量详尽的例题和插图，以便读者能更好地对各种数据结构加深认识和理解。对于知识点和相关概念的叙述，本着从易到难的学习规律，概念叙述简洁，概念讨论和实际例子相结合。对各种算法均先讨论其设计思想和方法，然后对其逐步求精，再给出完整的算法描述，以使学生能抓住算法的本质和基本思想。各章的“简单应用举例”中含有很多实用的算法实例，既是本章算法的综合运用，也有助于培养学生运用各种数据结构解决实际问题的能力。另外，本书在每章的最后都总结了本章的学习要点，方便读者理解和掌握重点、难点问题。同时，每章都收集了不同难度的习题，供读者复习提高之用。

本书力求深入浅出，通俗易懂。既可作为全日制高等院校计算机科学与技术专业、信息与计算科学专业、电子信息科学与技术等信息类相关专业普通本科学生的专业基础课教材，也可作为广大从事计算机软件开发人员的参考书。本书讲授学时为50~80学时，同时应预留一定的上机时间。教师可根据学时及专业，酌情删去一些内容。本书文字通俗、便于自学，读者只需掌握程序设计基本技术和一门高级程序设计语言，便可学习本书。本书中采用类C语言作为数据结构和算法的描述语言，对各种抽象数据类型的定义和实现简明清晰，既不拘泥于C语言的细节，又容易转换成能上机执行的C或C++程序。

本书由戴敏主编并负责全书统稿。其中戴敏编写第1、2、6章；董玉涛编写第7章的7.1、7.2、7.3节和第8章；刘凤连编写第3章、4章的4.3节、9章；肖迎元编写第4章的4.1、4.2节；孙志慧编写第5章、第7章的7.4、7.5、7.6节。此外，在本书的编写过程中，还有部分同志参加了相关程序的调试和文稿整理及校对等工作。在此，本书作者对所有参与者表示感谢，是他们的辛勤工作使本书得以尽快与读者见面。

尽管编者在写作过程中非常认真和努力，但由于水平和时间有限，本书难免有不当和疏漏之处，恳请各位老师、专家与读者不吝赐教，批评指正。

编　者

目　录

第1章 绪 论

数据结构(Data Structure)是一门随着计算机科学的发展而逐渐形成的学科，目前已成为计算机类各专业的基础课之一。它介绍和研究用计算机解决一系列问题(特别是非数值计算类问题)时所用的各种数据的组织方法、存储结构及处理方法。随着计算机功能与速度的不断提高，计算机已深入到人类社会的各个领域。计算机的应用已不再局限于科学计算，而更多地用于控制、管理及数据处理等非数值计算的处理工作。与此相应，计算机加工处理的对象也由纯粹的数值发展到字符、表格和图像等各种具有一定结构的数据，这就给程序设计带来一些新的问题。在处理这些数据之前，不仅要研究处理数据的特性，还要研究处理数据间的相互关系及其对应的存储表示。因此，解决非数值性问题的关键是利用这些特性和关系设计出合适的数据结构。为了编出一个“好”的程序，必须分析待处理对象的特性以及各处理对象之间存在的关系，这就是“数据结构”这门学科形成与发展的背景。

1.1 数据结构研究内容

一般来说，当用计算机解决一个实际问题时，大致需要经过以下几个步骤：首先分析实际问题，从中抽象出一个适当的数学模型，然后设计一个解决此数学模型的算法，最后编程、调试、测试，直至得到最终的解答。寻求数据模型的实质是分析问题，从中提取出计算机处理的对象，并找出这些对象之间的关系，然后用数学的语言加以描述。

在计算机发展的初期，使用计算机主要是处理数值计算问题，人们面临的许多问题基本上可以用数学方程进行描述。例如，“鸡兔同笼”问题，可转化为对二元一次方程组进行求解；再如，人们在电视里天天看到的天气预报，它的数学模型是一个环流模式方程。由于当时所涉及的运算对象是简单的整型、实型或布尔类型数据，所以程序设计者的主要精力集中于程序设计的技巧上，而无须重视数据结构。但随着遇到的问题不断地增多并且更加复杂，许多问题无法用数学工具或数学方程来描述。例如，图书资料查询、电话号码自动管理、交通道路规划等问题。解决此类问题的关键已不再是分析数学和计算方法，而是要分析问题中所用到的数据是如何组织的，研究数据之间存在什么样的关系，也就是要建立有效的数据结构来描述要解决的问题，这样就引入了数据结构的知识概念。下面举例说明数据结构的概念。

【例 1-1】 学校人事信息检索系统：如果想获取学校教务处某个教职员工的有关信息；或者想查询教务处人员的职务分布情况的时候，只要建立相关的数据结构，按照某种算法编写了相关程序，可以通过计算机来实现自动搜索。很显然，对应于所选取的教职员工，其姓名、职务等应是一一对应的。各个教职员工的信息集合实际上是一种查找表的结构。因此，可以根据它们之间的这种关系建立数据结构，再按照某种算法编写出相关程序，就能够实现所需要的查询。由此，可以在学校人事信息检索系统中建立一张按教务处教职员工顺序号排列的教职员工信息表和分别按姓名、职务顺序排列的索引表，如图 1-1 所示。

序号	职工号	姓名	性别	出生日期	职务
1	0001	张东宇	男	1965.03.21	处长
2	0002	李晓敏	女	1966.04.17	科长
3	0003	宋　彦	女	1967.12.06	科员
4	0004	王　港	男	1969.07.25	科员
5	0005	刘蓓蓓	女	1970.09.11	科员
6	0006	肖　扬	男	1975.11.02	科员
7	0007	谢云飞	男	1965.06.24	科长
8	0008	樊凌志	男	1970.10.18	主任
9	0009	曹婷婷	女	1973.07.03	科员
10	0010	高　磊	女	1974.01.20	科员

a）教职员工信息表

曹婷婷	9
樊凌志	8
高　磊	10
李晓敏	2
刘蓓蓓	5
宋　彦	3
王　港	4
肖　扬	6
谢云飞	7
张东宇	1

b）姓名索引表

处长	1
科长	2,7
科员	3,4,5,6,9,10

c）职务索引表

图 1-1　人事信息查询系统中的数据结构

由这三张表构成的文件便是教务处教职员工信息检索的数学模型，表中每个职工的信息放在一行中，称为一个数据元素，教务处所有职工的信息按照职工编号顺序依次存放在表中。类似这样的表格在不同计算机软件系统中有很多，如学校学生信息管理系统中的学生学籍登记表、图书馆管理系统中的图书目录表等。这类表格有一个共同的特性，就是各数据元素之间的关系是线性的关系，即一个元素的前面只有一个元素，后面也只有一个元素，第一个数据元素前面和最后一个数据元素后面没有元素。这类数学模型可称为线性的数据结构，即对象之间通常存在着的是一种简单的线性关系。

【例 1-2】 学校行政机构管理问题。一个典型的高校行政机构如图 1-2 所示，这是一个层次结构。顶层节点“学校”代表整个系统，它的下一层节点代表这个系统的各个子系统，即部处与学院。再下一层节点代表更小的机构，如中心实验室、计算机科学技术系等，直到最底层一个小组或一个教研室等。

在图 1-2 中，每一个节点代表一个数据元素，这类元素之间所呈现的是一种层次关系，从上到下按层进行展开形成一棵倒立的“树”，最上层是“树根”，依层向下射出“节点”和“树叶”。这种“树”形结构的模型在生活中接触得也比较多，比如图家行政区域规划、书籍目录等。

【例 1-3】 比赛编排问题。有 6 支球队进行足球比赛，分别用 v1,v2,…,v6 表示这 6 支球队，它们之间的比赛情况也可以用一个称作图的数据结构来反映，有向图中的每个顶点表示一个球队，如果从顶点 v_i 到 v_j 之间存在有向边 $<v_i, v_j>$，则表示球队 i 战胜球队 j。这样，各球队之间的胜负情况可以用一张图表示。例如，图 1-3 表示 v1 队战胜 v2 队，v2 队战胜 v3 队，v3 队战胜 v6 队，如此等等。

由此可以看出，用点和点与点之间的线所构成的图可反映实际生产和生活中的某些特定对象之间的特定关系。此类问题还有铁路交通图、教学编排等。

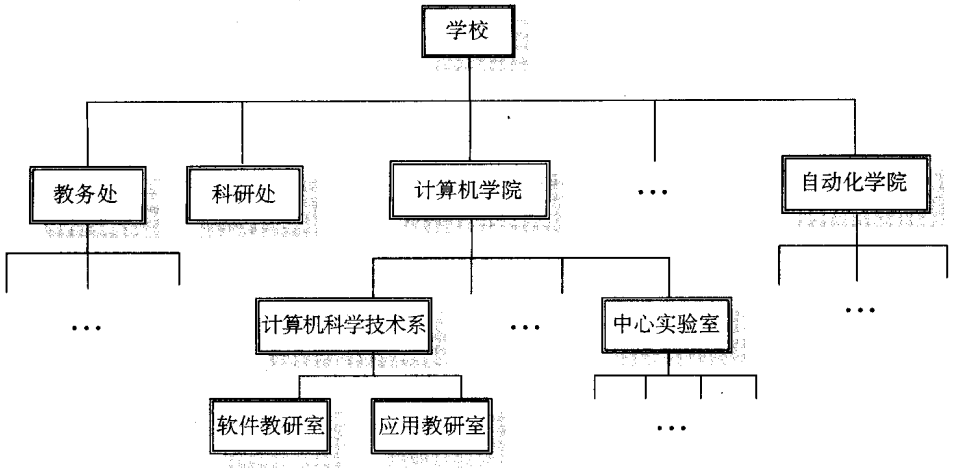


图 1-2　高校行政机构图

从前面的几个例子可以看出，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。随着计算机应用领域的扩大和软、硬件的发展，非数值计算问题越来越显得重要。据统计，当今处理非数值计算性问题占用了90%以上的机器时间[1]。大量非数值计算问题的数学模型正是这门课程所要讨论的对象。因此，概括地说，数据结构是一门讨论“描述现实世界实体的数学模型(非数值计算)及其上的操作在计算机中如何表示和实现”的学科。简单说来，数据结构是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作的学科[2]。

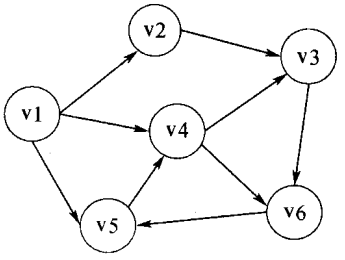


图 1-3　球队胜负关系示意图

国外把“数据结构”作为一门独立的课程而设立是在1968年。在此之前没有专门的“数据结构”课程，但在20世纪60年代初期，在“编译原理”和“操作系统”等课程中已经存在了有关“数据结构”的内容。之后，有些国家在高校中开始设立有关课程，但当时一般称为“表处理语言”而非“数据结构”。它的主要任务是分析当时的一些表处理系统，如简单表处理语言系统(SLIP)、信息处理语言系统(IPL-V)、串处理语言系统(SNOBOL)等[3]，它们的数据对象的结构是表或树，随后，“数据结构”的范围被扩大到了图、集合、代数结构等方面。由于数据必须在计算机中进行处理，因此不能局限于数据本身的数学问题的研究，还必须考虑数据的物理结构即数据在计算机中的存储结构，这进一步扩大了“数据结构”的内容。近年来，由于数据库、情报检索系统的发展，在“数据结构”课程中又增加了文件结构，特别是大型文件的组织等内容。

1968年，美国著名计算机科学家D·E克努特教授所著的《The Art of Computer Programming》Volume I(《计算机程序设计技巧》第一卷)出版，对计算机的发展做出了重大贡献，在书中作者论证了任何语言都可以采用表处理语言那样的技术。书中系统全面地论述了若干种数据的逻辑结构及物理结构。随后，数据的逻辑结构、存储结构及对应每种结构的操作被独立起来，形成了“数据结构”的主要内容。20世纪70年代以后，由于人们在软件开发领域对数据结构重要性的认识，数据结构开始越来越广泛地得到研究，各种版本的数据结构著作

相继出现。著名的瑞士科学家沃斯(N. Wirth)教授曾提出："算法 + 数据结构 = 程序"，得到计算机界广泛地认可[4]。

目前在我国，"数据结构"不仅是计算机及信息等相关专业的核心课程之一，也是很多非信息类专业的主要选修课程。"数据结构"是一门涉及广泛的专业基础课，它与数学、计算机硬件和软件有十分密切的关系。数据结构是介于数学、计算机硬件和计算机软件之间的一门计算机科学与技术专业的核心课程，它是计算机程序设计、编译原理、操作系统、数据库、人工智能等课程的基础。同时，数据结构技术也广泛应用于信息科学、系统工程、应用数学以及各种工程技术领域。

当然，"数据结构"的发展并未终结，它仍处于高速发展的时代。一方面，从抽象和具体这两种观点研究"数据结构"正在成为趋势；另一方面，计算机硬件系统及其他各学科的不断发展也必然对数据结构产生重大影响[2,3]。

1.2 基本概念和术语

为了便于以后各章节的学习，本节主要讲述数据结构中常用的基本概念和术语。

1. 数据

数据(Data)是指能输入到计算机中并能被计算机处理的一切对象。它是计算机程序加工的原料，应用程序处理各种各样的数据。在计算机科学中，所谓数据就是计算机加工处理的对象，它能够被计算机识别、存储和加工处理。这里必须强调指出，所谓数据不仅可以是数值数据，也可以是非数值数据。数值数据是一些整数、实数或复数，主要用于工程计算、科学计算和商务处理等；非数值数据包括字符、文字、图形、图像、语音等。例如，一个用某种程序语言编写的源程序、一篇文章、一张地图、一幅照片、一首歌曲等，都可以视为"数据"。今后随着计算机的发展，还将不断扩大数据的范围。

2. 数据元素

数据元素(Data Element)是数据的基本单位。在不同的条件下，数据元素又可称为元素、节点、顶点、记录等。由于数据的范围非常广泛，因此基本单位也是可大可小的。例如，人事信息检索系统中教职工信息表中的一个记录、文件系统结构树的一个节点、比赛编排问题中的一个顶点等，都被称为一个数据元素。

有时，一个数据元素可由若干个数据项(Data Item)组成。例如，人事信息管理系统中教职工信息表的每一个数据元素就是一个教职工记录。它包括教职工的职工号、姓名、性别、出生日期、职务、部门等数据项。

3. 数据对象

数据对象(Data Object)是具有相同性质的数据元素的集合，是数据的一个子集。在某个具体问题中，数据元素都具有相同的性质(元素值不一定相等)，属于同一数据对象。因为计算机不可能实时处理一切类型的数据，总是对特定的问题处理一种或几种对象。例如，用计算机求素数这种问题只涉及"整数"这种数据对象。整数数据对象是集合 $N = \{0, \pm 1, \pm 2, \cdots\}$，字母字符数据对象是集合 $C = \{'A','B',\cdots,'Z'\}$。

4. 数据结构

数据结构(Data Structure)是指互相之间存在着一种或多种关系的数据元素的集合。在任

何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间的关系称为结构。根据数据元素间关系的不同特性，通常有下列4类基本的结构：

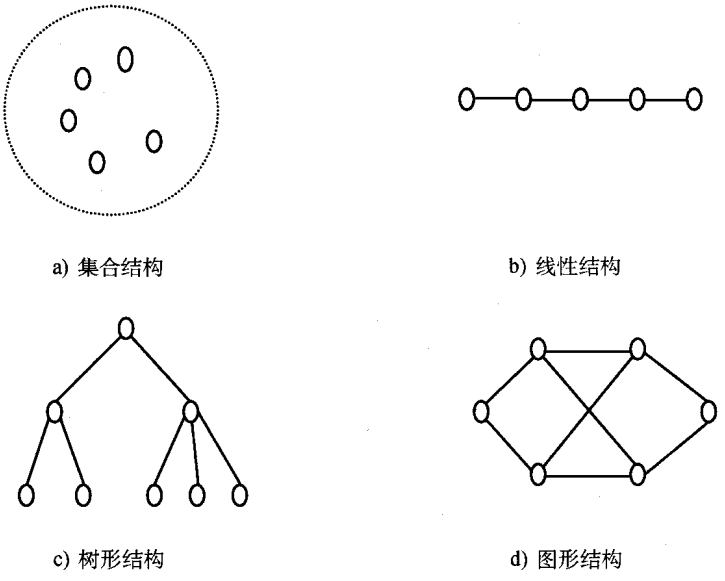
1）集合结构。在集合结构中，数据元素间的关系是“属于同一个集合”。集合是元素关系极为松散的一种结构。

2）线性结构。该结构的数据元素之间存在着一对一的关系。

3）树形结构。该结构的数据元素之间存在着一对多的关系。

4）图形结构。该结构的数据元素之间存在着多对多的关系，图形结构也称作网状结构。

图1-4为表示上述4类基本结构的示意图。



a) 集合结构　　b) 线性结构

c) 树形结构　　d) 图形结构

图1-4　4类基本结构的示意图

由于集合是数据元素之间关系极为松散的一种结构，因此也可用其他结构来表示它。

从上面所介绍的数据结构的概念中可以知道，一个数据结构有两个要素：一个是数据元素的集合，另一个是关系的集合。在形式上，数据结构通常可以采用一个二元组来表示。

数据结构的形式定义为：数据结构是一个二元组

$$\text{Data\_Structure} = (D, R)$$

其中，D是数据元素的有限集，R是D上关系的有限集。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，是数据之间的相互关系。它是从解决问题的需要出发，为实现必要的功能所建立的数据结构。它属于用户的视图，是面向问题的。例如，人事信息检索系统中建立的按职工号排列的有序表。数据的逻辑结构是独立于计算机的，它与数据在计算机中的存储位置无关。

研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的表示（又称映像）称为数据的物理结构，或称

存储结构。数据的物理结构是数据逻辑结构的物理存储方式，属于具体实现的视图，是面向计算机的。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。数据元素之间的关系在计算机中通常有两种不同的表示方法：顺序映像和非顺序映像，并由此得到两种不同的存储结构：顺序存储结构和链式存储结构。

顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构(见图 1-5a)。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

链式存储方法对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构(见图 1-5b)。链式存储结构通常借助于程序设计语言中的指针类型来实现。

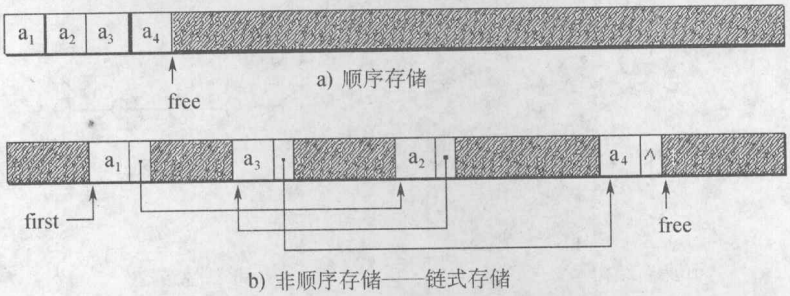


图 1-5　顺序存储与链式存储

除了通常采用的顺序存储方法和链式存储方法外，有时为了查找的方便还采用索引存储方法和散列存储方法。

5. 数据类型

数据类型(Data Type)是和数据结构密切相关的一个概念。它最早出现在高级程序设计语言中，用来描述程序中操作对象的特性。在用高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。类型显式地或隐含地规定了在程序执行期间，变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此，数据类型是一个值的集合和定义在这个值集上的一组操作的总称。

在高级程序设计语言中，数据类型可分为两类：一类是原子类型，另一类则是结构类型。原子类型的值是不可分解的，如 C 语言中整型、字符型、浮点型、双精度型等基本类型，分别用保留字 int、char、float、double 标识。而结构类型的值是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构的，也可以是结构的，例如，数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可被看成是由一种数据结构和定义在其上的一组操作所组成的。

1.3　算法和算法分析

在计算机领域，一个算法实质上是针对所处理问题，在数据的逻辑结构和存储结构的基础上施加的一种运算。算法与数据结构的关系紧密，在算法设计时先要确定相应的数据结

构，而在讨论某一种数据结构时也必然会涉及相应的算法。由于数据的逻辑结构和存储结构不是唯一的，算法的设计思想和技巧也不是唯一的，所以处理同一个问题的算法也不是唯一的。本节学习算法与算法分析的目的，就是要学会根据处理问题的需要，为待处理的数据选择合适的逻辑结构和存储结构，从而设计出比较满意的算法。

1.3.1　算法定义

1. 算法与算法的特性

算法(Algorithm)是对特定问题求解步骤的一种描述，是指令的有限序列。其中每一条指令表示一个或多个操作。虽然在计算机中研究算法只不过是最近几十年的事情，但是有很多著名的算法已经存在上千年了。例如，求两个正整数之间的最大公约数的欧几里德算法。该算法可以表述如下：如果 m、n 都是正整数，且 m > n。则 m = nq + r，0 < r < n，此处 r、q 都是正整数，于是(m,n) = (n,r)。符号(m,n)为 m、n 之间的最大公约数。根据欧几里德算法，可以得到求 m、n (m、n 都是正整数,且 m > n)之间最大公约数的公式如下：

$$g(m,n)=\begin{cases} n & \text{若 } \mathrm{mod}(m,n)=0 \\ g(n,\mathrm{mod}(m,n)) & \text{若 } \mathrm{mod}(m,n)!=0 \end{cases}$$

其中，mod(m,n)表示 m 除 n 之后的余数。这样，即可得到 m、n 之间的最大公约数(参见算法 1.1)。

算法 1.1　求 m、n 之间的最大公约数

```
int maximal _ common _ divisor ( ){
    //输入两个正整数 m、n,满足关系 m > n。本算法求 m 和 n 之间的最大公约数。
    int m,n,r;//r 用于保存 m% n 的结果
    printf(" Please enter two positive intergers m, n(m > n):");
    scanf("% d,% d",&m,&n);
    if(m <= n)return  -1; // 若 m > n,继续执行后面语句,否则程序中断。
  While(1){
        r = m% n; //得到 m 除 n 的余数。
        if(r == 0)   return n; // 若余数为 0,则 m 和 n 之间的最大公约数为 n。
        else{ m = n; n = r; // 更新被除数和除数。
    }
}//m > n 时,函数返回正整数 m 和 n 之间的最大公约数
```

分析算法 1.1 的执行过程，可得到算法具有的五个重要特征：

1）有穷性。一个算法必须在有穷步之后结束，即必须在有限时间内完成。像上述欧几里德算法，无论 m、n 为什么正整数，在执行若干次循环之后，终归可以得到它们之间的最大公约数。例如 m = 36，n = 24 时，最大公约数为 12。

2）确定性。算法的每一步必须有确切的定义，无二义性。算法的执行对应着的相同的输入仅有唯一的一条路经。例如算法 1.1 的每一步，都有确切定义。首先取得 m、n 的值，接着判断 m > n 是否成立，进入循环，取得 m/n 的余数，……。

3）可行性。算法中的每一步都可以通过已经实现的基本运算的有限次执行得以实现。

4）输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象集合，它就是在算法执行之前的初始值。例如，上述算法中 m、n 的值。

5）输出。一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。例如，上述算法最后得到的最大公约数。

算法的含义与程序十分相似，但又有区别。一个程序不一定满足有穷性。例如，操作系统，只要整个系统不遭破坏，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中，因此操作系统不是一个算法。另一方面，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。算法代表了对问题的解，而程序则是算法在计算机上的特定实现。一个算法若用程序设计语言来描述，则它就是一个程序。

2. 算法设计的要求

算法与数据结构是相辅相承的。一个具体的应用问题，通常可以使用不同的算法进行求解。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。

要设计一个好的算法通常要考虑以下的要求：

1）正确。算法的执行结果应当满足预先规定的功能和性能要求。“正确”一词的含义通常可分为以下四个层次：①程序不含语法错误；②程序对于几组输入数据能够得出满足规格说明要求的结果；③程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果；④程序对于一切合法的输入数据都能产生满足规格说明要求的结果。显然，达到第④层意义下的正确是极为困难的，所有不同输入数据的数量大得惊人，逐一验证的方法是不现实的。对于大型软件需要进行专业测试，一般情况下，通常以第③层意义的正确性作为衡量一个程序是否合格的标准。

2）可读。一个算法应当思路清晰、层次分明、简单明了、易读易懂。算法主要是为了人的阅读与交流，其次才是机器执行。可读性好有助于人对算法的理解；晦涩难懂的程序易于隐藏较多错误难以调试和修改。

3）健壮。当输入不合法数据时，算法应能作出反应或进行处理，而不至引起严重后果。例如，对于算法 1.1 中的欧几里德算法，要求输入的正整数 $m > n$，当输入的 m、n 值不满足这个条件时，算法不应继续计算，而应报告输入出错，并且，处理出错的方法应是返回一个表示错误或错误性质的值，并中止程序的执行，以便进行处理。

4）高效。有效使用存储空间和有较高的时间效率。通俗地说，效率指的是算法执行时间。对于同一个问题如果有多个算法可以解决，执行时间短的算法效率高。存储量需求指算法执行过程中所需要的最大存储空间。效率与低存储量需求这两者都与问题的规模有关。例如，对 100 个正整数进行排序和对 1000000 个正整数进行排序时，所花的执行时间或运行空间显然有一定的差别。

3. 算法的描述

算法可以使用各种不同的方法来描述。最简单的方法是使用自然语言。用自然语言来描述算法的优点是简单且便于人们对算法的阅读，缺点是不够严谨。可以使用程序流程图、N-S 图等算法描述工具描述算法，其特点是描述过程简洁、明了。

用以上两种方法描述的算法都不能够直接在计算机上执行，若要将它转换成可执行的程

序还有一个编程的问题。

也可以直接使用某种程序设计语言来描述算法，不过直接使用程序设计语言并不容易，而且不太直观，常常需要借助于注释才能使人看明白。

为了解决理解与执行这两者之间的矛盾，人们常常使用一种称为伪码语言的描述方法来进行算法描述。伪码语言介于高级程序设计语言和自然语言之间，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此它比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。它虽然不能直接执行但很容易被转换成高级语言。

1.3.2　算法分析预备知识

算法的好坏主要取决于算法在执行过程中所占用的空间和所耗费的时间，因此可以从一个算法的时间复杂度与空间复杂度来评价算法的优劣。

1. 影响算法运行时间的因素

将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于下列因素：

1）运行程序的计算机的机器指令的品质和速度。例如，运行程序使用486机还是使用586机，执行时间会有差别。

2）书写程序的语言。一般来说，实现语言的级别越高，其执行效率就越低。

3）编译程序所生成目标代码的质量。对于代码优化较好的编译程序其所生成的程序质量较高。

4）问题的规模。在进行问题求解时，算法所使用的空间、时间的多少，与求解问题的规模相关。例如，对100个正整数进行排序和对1000000个正整数进行排序时，所占用的空间和耗费的时间显然是不同的，这里待排序的正整数的个数就是问题的规模。在其他情况下，例如在矩阵中进行加、减操作时，可以将矩阵的行数和列数作为该问题的规模。算法运行耗费的时间和求解问题的规模n是紧密相连的，在通常情况下，问题的规模越大，运行时间将越长。

前3个因素表明算法的执行时间依赖于软件和硬件的环境。程序的运行时间不但与计算机的机器指令相关，而且与书写语言以及编译程序所生成的目标代码的质量相关。这就意味着，不能够把在某台具体的计算机上运行的时间，作为衡量相应算法的好坏标准。因为，一个好的算法，在一台老式的计算机上运行，其耗费的时间可能比一个差的算法在新式的计算机上运行所耗费的时间要长。也就是说，仅仅考虑算法在计算机中所耗费的绝对时间，判断算法的优劣是不全面的。换句话说，不能采用在某一计算机上运行的绝对时间单位，如秒、分、…，作为衡量某一算法时间复杂度的优劣标准。

显然，在各种因素都不能确定的情况下，很难比较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，可以将上述各种与计算机相关的软、硬件因素都确定下来，这样一个特定算法的运行工作量的大小就只依赖于问题的规模（通常用正整数n表示），或者说它是问题规模的函数。通常把算法所需的时间和问题规模n的函数T(n)称为时间复杂度，类似地，把算法所占用的空间和问题规模的函数S(n)称为空间复杂度。

2. 使用程序步分析算法

在讨论算法的时间复杂度T(n)时，通常认为是相应的算法在某一种计算模型或理想的

计算机上执行的指令的总条数。一个算法是由控制结构(顺序、循环、分支)和原操作(指固有数据类型的操作)构成，算法时间取决于两者的总和效果。撇开与计算机软、硬件有关的因素，假定每条语句的执行时间为单位时间，则

算法所耗费时间 = ∑(一条语句执行次数 × 执行该语句所需时间)

即一个算法所耗费时间是该算法中所有语句的频度之和。

为简化分析，可以使用下面定义的程序步(而不是直接计算语句次数)来分析算法。

一个程序步(Program Step)是指在语法上或语义上有意义的一段指令序列，该程序段的执行时间与问题实例的特征(如规模)无关[6]。例如，注释语句的程序步数为0；声明语句的程序步数为0；每个简单语句，如赋值语句、读语句、写语句的程序步数为1。

【例1-4】 求一个数组元素累加之和。下面以sum()函数为例来说明如何计算一个算法的程序步数。

```
float sum (float a[ ], const int n )
{   float s;
    s = 0.0;
    for( int i = 0;i < n;i ++ )
        s += a[i];
    return s;
}
```

表1-1统计了函数的程序步数。从表1-1可以看出，这里被计算的每一程序步的执行时间均与问题实例的规模n(数组元素的个数)无关。该程序的总程序步数为2n+3。

表1-1 计算累加和程序的程序步数

程序语句	一次执行所需程序步数	执行频度	程序步数
{ float s; int i;	0	0	0
s = 0.0;	1	1	1
for(i = 0; i < n; i ++)	1	n + 1	n + 1
s += a[i];	1	n	n
return s;	1	1	1
}	0	0	0
	总程序步数		2n + 3

使用程序步数分析算法的时间复杂度不依赖具体的物理计算机，从而可以更有效地比较算法的优劣。但是，各种不同的计算模型的指令功能不尽相同。例如，在一种模型下某算法的时间复杂度 $T(n) = 8n^2$，在另一种模型下可能会变为 $T(n) = 100n^2$。这也提示我们，某些系数并不是反映算法优劣的主要因素。为了进一步去除这些因素的影响，采用“级别”或者“阶”来评价算法的优劣更加合适。下面将首先引入相关的数学概念，然后讨论如何运用它们求解算法的时间复杂度的级别。

3. 大O表示法

在讨论算法的时间复杂度时，经常用到一种特殊的表示法，即大O表示。它是用于表

示时间复杂度函数的增长率的上界的。

定义 1.1 如果存在着正的常数 C 和自然数 n_0，当 $n \geqslant n_0$ 时，有 $f(n) \leqslant Cg(n)$ 成立，则称 f(n) 当 n 充分大时上有界，且 g(n) 是它的一个上界，记为 $f(n) = O(g(n))$，称为大 O 记号[6]。

大 O 记号用以表达一个算法运行时间的上界。说一个算法具有 O(g(n)) 的运行时间，是指当 n 足够大时，该算法在计算机上的实际运行时间不会超过 g(n) 的常数倍，g(n) 是它的一个上界。

【例 1-5】 设 $T(n) = (n+1)^2$。那么，取 $n_0 = 1$ 及 $C = 4$ 时，$T(n) \leqslant Cn^2$ 成立。所以，$T(n) = O(n^2)$。

【例 1-6】 设 $T(n) = 3n^3 + 2n^2$。选 $n_0 = 0$ 及 $C = 5$；$T(n) \leqslant Cn^3$。所以，$T(n) = O(n^3)$。同样的道理，若取 $n_0 = 0$ 及 $C = 5$；$T(n) \leqslant Cn^4$。所以，$T(n) = O(n^4)$ 同样是成立的。

从例 1-6 可以发现，时间复杂度函数 T(n) 的上界有多个。这种情况下，究竟取哪一个上界表示它的增长率的上界呢？

大 O 表示法以函数的渐进性为特征，给出了算法当问题的规模 n 增大时的上界。但是，大 O 表示法并没有给出这个函数和这个上界的接近程度。这就意味着，这个上界既可能非常接近（“紧贴”）的，也可能相距很远。从算法的时间复杂度的角度进行分析，$f(n) = O(g(n))$ 意味着找到了 f(n) 的一个上界 g(n)。但 g(n) 最好是一个“紧贴”的渐进界，或者说找到一个最小的上界。在例 1-5 中 $T(n) = O(n^3)$ 是没有意义的，因为有“紧贴”的渐进界 n^2 存在。所谓“紧贴”的上界，是和原函数 T(n) 最接近的一个上界。

定义 1.2 设存在一个函数 $f(n) = O(g(n))$，如果对于每一个函数 h(n) 都使得 $f(n) = O(h(n))$，同样也使得 $g(n) = O(h(n))$ 成立，就说 g(n) 是 f(n) 的紧贴渐进界[7]。

如果一个算法的时间复杂性的紧贴渐进界是 O(g(n)) 的，那么它的算法的时间增长率是 O(g(n)) 的，读作 g(n) 级（阶）的。O(1)、O(n)、$O(n^2)$ 通常称为常数级（阶）、线性级（阶）和平方级（阶）。

1.3.3 算法分析

前面引入了程序步的概念来分析算法的时间复杂度。不同的程序步在计算机上的实际执行时间通常是不同的，程序步数并不能确切反映程序运行的实际时间。而且，事实上，一个程序一次执行所需要的程序步的精确计算往往也很困难，那么，引入程序步的意义何在呢？本节中定义的渐近时间复杂度，使人们可以使用程序步在数量级上估计一个程序的执行时间。

1. 时间复杂度

一个程序的时间复杂度（Time Complexity）是指程序运行从开始到结束所需要的时间。

一个算法是由控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同的算法，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该原操作重复执行的次数作为算法的时间度量。一般情况下，算法中原操作重复执行的次数是规模 n 的某个函数 T(n)。许多时候要精确地计算 T(n) 是困难的，引入渐进时间复杂度在数量上估计一个算法的执行时间，也能够达到分析算法的目的。

当问题的规模逐渐增大，在极限的情况 $n \to +\infty$ 时的时间复杂度函数的极限称为渐进时