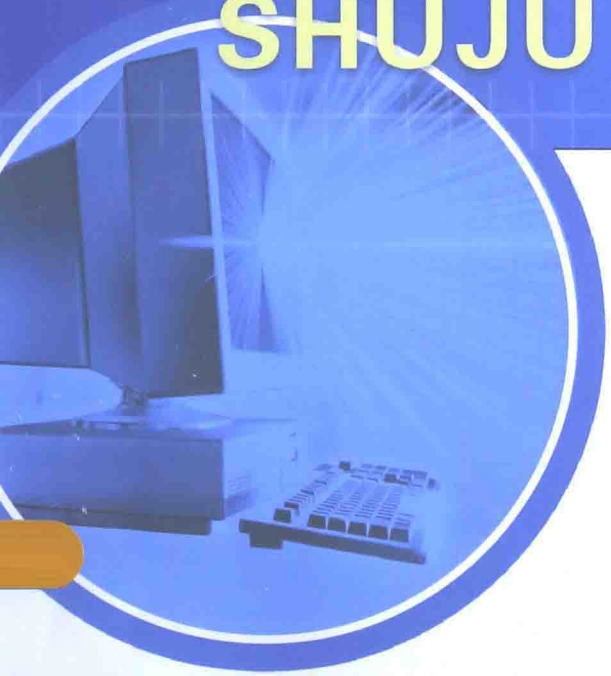
面向高等院校人才培养规划教材

数据结构

SHUJU JIEGOU



主　编　韩利凯　李　军

副主编　高寅生　徐东升
　　王　帆　赵世磊

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

面向高等院校人才培养规划教材

数据结构

主　编　韩利凯　李　军
副主编　高寅生　徐东升
　　王　帆　赵世磊

ZHEJIANG UNIVERSITY PRESS
浙江大学出版社

内 容 简 介

本书是为“数据结构”课程编写的教材，书中系统地介绍各种常用的数据结构与算法。全书共分为11章。第1章为绪论，引入数据结构与算法的一些基本概念，是全书的综述；第2～7章分别介绍线性表、栈与队列、串、数组与广义表、树与二叉树和图等几种基本的数据结构；第8章、第9章和第10章分别介绍各种查找和内、外排序的基本运算，它们都是数据处理中广泛使用的技术；第11章介绍数据结构实验系统开发的全过程并通过一周或两周的时间由学生独立完成一个课程设计。

本书注重应用、选材精练、图文并茂，对基本理论的叙述深入浅出、通俗易懂；精选的应用实例涉及领域相当广泛，给读者提供了思路与方法，有助于提高读者分析问题和解决问题的能力。书中自始至终使用C语言来描述算法和数据结构，全部程序都在TURBO C 2.0或Visual C＋＋ 6.0中调试通过，每章后进行小结并配有适量习题，便于读者掌握各章的重点和难点并进行必要的训练。

本书可作为高等院校计算机及相关专业的教科书，也可作为从事计算机应用的工程技术人员的自学参考书。

图书在版编目(CIP)数据

数据结构/韩利凯，李军主编. —杭州：浙江大学出版社，2013.8

ISBN 978-7-308-12082-1

Ⅰ. ①数… Ⅱ. ①韩… ②李… Ⅲ. ①数据结构—高等学校—教材Ⅳ. ①TP311.12

中国版本图书馆CIP数据核字(2013)第189281号

数据结构

主编 韩利凯 李 军

责任编辑 邹小宁
文字编辑 刘 郡
封面设计 朱 琳
出 版 浙江大学出版社
（杭州市天目山路148号 邮政编码 310007）
（网址：http://www.zjupress.com）
排 版 杭州教联文化发展有限公司
印 刷 浙江华人数码印刷有限公司
开 本 787mm×1092mm 1/16
印 张 14.5
字 数 353千
版 印 次 2013年8月第1版 2013年8月第1次印刷
书 号 ISBN 978-7-308-12082-1
定 价 32.00元

编　委　会

主　编　韩利凯　李　军

副主编　高寅生　徐东升

　　王　帆　赵世磊

参　编　马宗保　谢巧玲　何可可

　　冯永亮　赵宁社

前　言

教育部高校计算机科学与技术教学指导委员会指出："数据结构是计算机学科本科教学计划中的骨干基础课程，对学生基本的计算机问题求解能力的培养具有重要意义。作为一门必修课程，该课程既是对以往课程的深入和扩展，也是为将来更加深入学习其他专业课程打下基础。课程中所学习的排序问题的算法以及基本的树、图等数据结构，是计算机学科的基本功。B＋树、散列等高级数据结构，也是数据库、操作系统、编译原理、计算机网络等后续课程的基础。"上述论述充分说明了"数据结构"课程在整个计算机学科领域中的地位和意义。

目前，数据结构不仅是计算机专业的一门重要的专业基础课程，而且也是大多数高等院校的非计算机专业的主干课程。数据结构不仅是计算机专业考研的必考科目之一，还是全国计算机等级考试、软件资格（水平）考试的主要考试内容。

本书的特点是注重应用、层次清晰、结构合理，对基本理论的叙述深入浅出、通俗易懂；选材精练、图文并茂，精选的应用实例涉及领域相当广泛，给读者提供了思路与方法，有助于提高读者分析问题和解决问题的能力；书中自始至终使用C语言来描述算法和数据结构，全部程序都在Turbo C 2.0或Visual C＋＋ 6.0中调试通过，每章后进行小结并配有适量习题，便于读者掌握各章的重点难点并进行必要的训练。

本书的内容主要分为以下几个部分。

第1章：如果读者刚接触数据结构，这一章将告诉您数据结构是什么，以及本书的学习目标、学习方法和学习内容；另外，还介绍了本书对算法的描述方法。

第2章：主要介绍了线性表。首先讲解线性表的逻辑结构，然后介绍线性表的各种常用存储结构，在每一节均给出了算法的具体应用。通过学习这一章，读者可以掌握顺序表、动态链表和静态链表的操作。

第3章：主要介绍操作受限的线性表——栈和队列，内容包括栈的定义，栈的基本操作及栈与递归的转化，队列的概念，顺序队列和链式队列的运算。

第4章：主要介绍一种特殊的线性表——串。首先介绍串的概念，然后介绍串的各种存储表示，以及串的模式匹配算法。

第5章：主要介绍数组与广义表。首先介绍数组的概念，数组（矩阵）的存储结构及运算，几种特殊矩阵；然后介绍广义表的概念，广义表的两种存储方式，广义表的操作实现。

第6章：主要介绍非线性数据结构——树和二叉树。首先介绍树和二叉树的概念，

然后介绍树和二叉树的存储表示，二叉树的性质，二叉树的遍历和线索化，树、森林与二叉树的转换及哈夫曼树。

第7章：主要介绍非线性数据结构——图。首先介绍图的概念和存储结构，然后介绍图的遍历、最小生成树、拓扑排序、关键路径及最短路径。

第8章：主要介绍数据结构的常用技术——查找。首先介绍查找的概念，然后结合具体实例介绍各种查找算法，并给出完整程序。

第9章：主要介绍数据结构的常用技术——内排序。首先介绍排序的相关概念，然后介绍各种内排序技术，并给出具体的实现算法。

第10章：主要介绍外部排序。首先介绍了有关外存信息存取的特点，然后介绍外部排序的基本方法。

第11章：通过本章学习，可以复习数据结构的基本算法，学习数据结构实验系统开发的全过程，了解系统设计的一般要求；掌握文件包含处理的基本方法；提高程序编写和程序调试的能力。

本书由西安文理学院韩利凯教授和李军副教授担任主编，高寅生、王帆、徐东升、赵世磊担任副主编。其中，第1章由马宗保编写；第2章、第3章由谢巧玲编写；第4章、第5章由何可可编写；第6章由徐东升编写；第7章、第8章由冯永亮编写；第9章由赵宁社编写；第10章、第11章由韩利凯和李军编写。

在本书编写中，编者参考了国内外众多数据结构与算法方面的优秀教材，其中大多列举在书后的参考文献中。在此，我们对这些教材的编著者表示衷心感谢。

由于编著者水平有限、时间仓促，书中难免存在一些不足之处，殷切希望广大读者批评指正。

编　者

2013年4月

目　录

第1章 绪 论

随着科学技术的进步，计算机扮演的角色越来越重要，已从单纯的数值处理进入到社会的方方面面，带来了关于非数值数据处理的新问题。为了更好地处理这些数据，我们必须根据这些数据特有的结构关系，针对性地进行程序设计，通过合理的数据结构设计来提高处理数据的效率，这使得数据结构的研究变得重要起来，数据结构（Data Structure）这门学科便应运而生。

数据结构是一门主要研究非数值计算的程序设计问题中的操作对象，以及它们之间的关系和操作等相关问题的学科。程序设计就是数据结构和算法的研究。

“数据结构”作为一门独立的课程在国外是从1968年才开始设立的。1968年美国唐·欧·克努特教授开创了“数据结构”的最初体系，他所著的《计算机程序设计技巧》第一卷《基本算法》是第一本较系统地阐述数据的逻辑结构和存储结构及其操作的著作。

目前在我国，“数据结构”也已经不仅仅是计算机专业的教学计划中的核心课程之一，而且是其他非计算机专业的主要选修课程之一。

“数据结构”在计算机科学中是一门综合性的专业基础课。“数据结构”的研究不仅涉及计算机硬件（特别是编码理论、存储装置和存取方法等）的研究范围，而且和计算机软件的研究有着更密切的关系。无论是编译程序还是操作系统，都涉及数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以便查找和存取数据元素更为方便。因此，可以认为“数据结构”是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。在计算机科学中，“数据结构”不仅是一般程序设计（特别是非数值计算的程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

到目前“数据结构”的发展并未终结，从各专门领域中特殊问题的数据结构得到研究和发展，如多维图形数据结构等，到从抽象数据类型的观点来讨论数据结构，使得这门学科不断得到延伸和扩充。

1.1 数据结构的基本概念

为了系统的学习本门课程，我们先熟悉一些概念和术语，从而为以后的章节学习奠定基础。

1.1.1 数据(Data)

数据是客观事物的符号表示,是对能输入计算机中并能被计算机中程序处理的符号的总称。

1.1.2 数据元素(Data Element)

数据元素是数据的基本单位,是数据集合的个体,在计算机中通常作为一个整体进行考虑和处理。例如在学籍表中,可以把学生作为一个数据元素或记录(Record),则学籍表中的每一项(如姓名、学号等)就为一个数据项。数据项(Data Element)是数据不可分割的最小单位。

1.1.3 数据对象(Data Object)

数据对象是性质相同的数据元素的集合,是数据的一个子集。例如整数数据对象是集合N=$\{0,\pm1,\pm2,\cdots\}$,字母字符数据对象是集合C=$\{'A','B',\cdots,'Z'\}$,学籍表也可看作一个数据对象。由此可看出,不论数据元素集合是无限集(如整数集)、有限集(如字符集),还是由多个数据项组成的复合数据元素(如学籍表),只要性质相同,都是同一个数据对象。

1.1.4 数据类型(Data Type)

数据类型是一组性质相同的值的集合以及定义在这个值集合上的一组操作的总称。数据类型中定义了两个集合,即该类型的取值范围,以及该类型中可允许使用的一组运算。例如在高级语言中,整型类型可能的取值范围是$-32768\sim+32767$,可用的运算符集合为加、减、乘、除等。

1.1.5 数据结构(Data Structure)

在现实世界中,不同数据元素之间不是独立的,而是存在特定的关系,我们将这些关系称为结构。数据结构是相互之间存在一种或多种特定关系的数据元素集合。通过分析待处理对象的特性及各处理对象之间存在的关系,从而编写出一个高效的程序,这就是数据结构研究的主要内容。

1.2 数据结构的内容

通过对于数据结构的了解,我们知道数据结构就是研究数据元素相互之间存在的一种或多种特定关系。而数据元素间的相互关系具体来说包括三个方面:数据的逻辑结构、数据的物理结构和数据的运算集合。

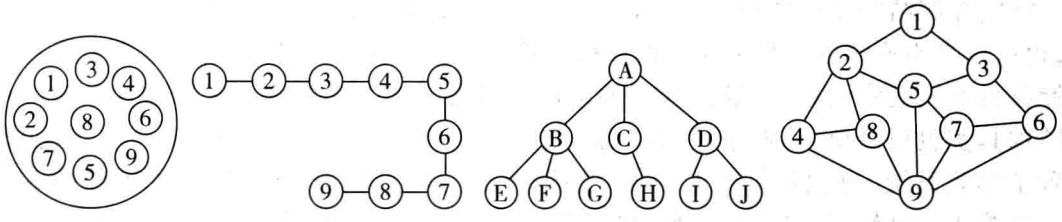
数据的逻辑结构是指数据元素之间的逻辑关系描述。数据结构的形式定义为：Data_Structure＝[D，S，P]。其中：D是数据元素的有限集，S是上下关系的有限集，P是对数据对象的基本操作。根据数据元素之间关系的不同特性，通常有下列四类基本的结构，如图1-1所示。

(1)集合结构：结构中的数据元素之间除了同属于一个集合的关系外，无任何其他关系。各个数据元素是“平等”的，它们的共同属性是“同属于一个集合”。数据结构中的集合关系类似于数学中的集合。

(2)线性结构：结构中的数据元素之间存在着一对一的线性关系。

(3)树形结构：结构中的数据元素之间存在着一对多的层次关系。

(4)图状结构或网状结构：结构中的数据元素之间存在着多对多的任意关系。



(a)集合结构　(b)线性结构　(c)树形结构　(d)图状结构(网状结构)

图1-1　数据元素四类基本的结构

我们在用示意图表示数据的逻辑结构时，要注意以下两点。

(1)将每一个数据元素看做一个结点，用圆圈表示，用数字或字母表示不同结点。

(2)元素之间的逻辑关系用结点之间的连线表示，如果这个关系是有方向的，那么用带箭头的连线表示。

由于集合的关系非常松散，可以用其他的结构代替，故数据的逻辑结构可概括为如图1-2所示。

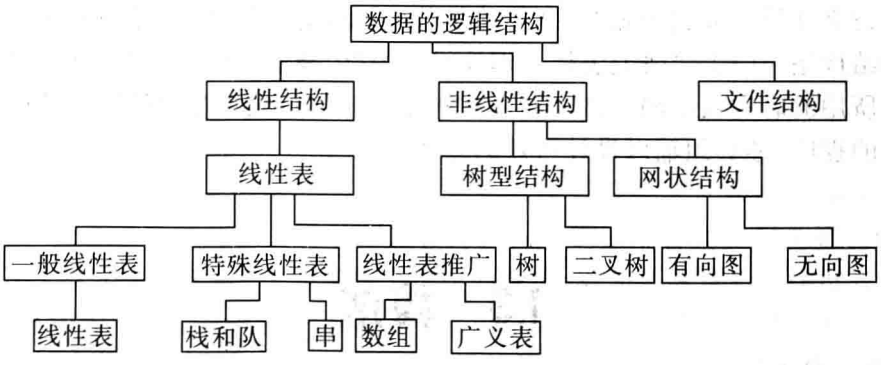


图1-2　数据的逻辑结构

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，又称存储结构。它包括数据元素的表示和关系的表示。数据是数据元素的集合，那么根据物理结构的定义，实际上就是如何把数据元素存储到计算机的存储器中，这里的存储器主要是指内存。像硬盘、软盘、光盘等外部存储器的数据组织通常用文件结构来描述。

数据的存储结构应正确反映数据元素之间的逻辑关系，这才是最关键的，如何存储数据元素之间的逻辑关系，是实现物理结构的重点和难点。

数据元素的存储结构形式有两种:顺序存储和链式存储。

顺序存储结构是把数据元素存放在地址连续的存储单元里，其数据间的逻辑关系和物理关系是一致的。这种存储结构其实很简单，类似于我们排队占位。大家都按顺序排好，每个人占一小段空间，大家谁也别插谁的队。我们之前学计算机语言时，数组就是这样的顺序存储结构。当你告诉计算机，你要建立一个有10个整型数据的数组时，计算机就在内存中找了片未占用空间，按照一个整型所占位置的大小乘以10，开辟一段连续的空间，于是第一个数组数据就放在第一个位置，第二个数据放在第二个，这样依次摆放。这就是顺序存储结构。

链式存储结构是把数据元素存放在任意的存储单元里，这组存储单元可以是连续的，也可以是不连续的。数据元素的存储关系并不能反映其逻辑关系，因此需要用各个指针存放数据元素的地址，这样通过地址就可以找到相关联数据元素的位置。这种存储结构其实也很简单，类似于我们利用排号系统进行排队。现在如银行、医院等地方，设置了排队系统，也就是每个人去了，先领一个号，等着叫号，叫到时去办理业务或看病。在等待的时候，你爱在哪在哪，可以坐着、站着或者走动，甚至出去逛一圈，只要及时回来就行了。你关注的是前一个号有没有被叫到。叫到了，下一个就轮到你了。显然，链式存储就灵活多了，数据存在哪里不重要，只要有一个指针存放了相应的地址就能找到它了。

逻辑结构是面向问题的，而物理结构就是面向计算机的，其基本的目标就是将数据及其逻辑关系存储到计算机的内存中。

综上所述，数据结构的内容可归纳为三个部分:逻辑结构、存储结构和运算集合。按某种逻辑关系组织起来的一批数据，按一定的映象方式把它存放在计算机的存储器中，并在这些数据上定义了一个运算的集合，即Data_Structure＝[D，S，P]。其中，D是数据元素的有限集，S是上下关系的有限集，P是对数据对象的基本操作，这就叫做数据结构。

数据结构是一门主要研究怎样合理地组织数据，建立合适的数据结构，提高计算机执行程序所用的时空效率的学科。然而在程序设计中只建立合适的数据结构并不能设计出高效的程序，还必须辅以设计巧妙的算法。

1.3 算法

算法(Algorithm)是解决特定问题时求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。

关于算法，德国数学家高斯小时候的故事就很具有代表性，对于一个求1＋2＋3＋…＋100的结果的程序，你应该怎么写呢?

```
(1)int i,sum=0,n=100;
   for(i=0; i<=n; i++)
   {
      sum=sum+i;
   }
```

其实,sum= 1 +2 +3 +…+99 +100

sum=100+99 +98 +…+2 + 1

‖ ‖ ‖ ‖ ‖ ‖

2*sum=101+101+101+…+101+101

则,sum=(1+n)*n/2就可以这样编写:

```
(2)int i,sum=0,n=100;
   sum=(1+n)*n/2;
```

这两种求和程序中的第二种等差数列的算法,不仅仅可以用于1加到100,就是加到一千、一万、一亿(需要更改整型变量类型为长整型,否则会溢出),也就是瞬间之事。但如果用第一种编写的程序,显然计算机要循环一千、一万、一亿次的加法运算。人脑比电脑算得快,似乎成为了现实。这就是算法的魅力。

我们来了解下算法还具有的下列5个重要特性。

(1)有穷性:一个算法必须总是(对任何合法的输入值)在执行有穷步之后结束,且每一步都能在有穷时间内完成。但是写一个算法,计算机需要算上个二三十年,一定会结束,它在数学意义上是有穷了,可是就其效率性而言,算法的意义就不大了。

(2)确定性:算法中每一条指令必须有确切的含义,读者理解时不会产生歧义。并且,在任何条件下,算法只有唯一一条执行路径,即对于相同的输入只能得出相同的输出。

(3)可行性:一个算法是能运行的,即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

(4)输入:一个算法有零个或多个的输入,这些输入取自某个特定的对象的集合。

(5)输出:一个算法有一个或多个的输出,这些输出是同输入有着某些特定关系的量。

通常,设计一个“好”的算法除了应具备算法设计的特性外,还应达到以下几个算法设计的要求。

(1)正确性(Correctness):算法的正确性是指算法至少应该具有输入、输出、加工处理无歧义性、能正确反映问题的需求和能够得到问题的正确答案。但是算法的“正确”通常在用法上有很大的差别,大体有四个层次的考虑:首先要看算法程序没有语法的错误;然后要算法程序对于合法的输入数据能够产生满足要求的输出结果;其次算法程序对于非法的输入数据能够得出满足规格说明的结果;最后算法程序对于精心选择的,甚至刁难的测试数据都要有满足要求的输出结果。

(2)可读性(Readability):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。晦涩难懂的程序容易隐藏较多错误,难以调试和修改。

(3)健壮性(Robustness):当输入的数据非法时,算法也能适当地做出反应或进行处

理。不是产生莫明其妙的输出结果，而应报告输入出错。处理出错的方法应是返回一个表示错误或错误性质的值，而不是打印错误信息或异常。同时中止程序的执行，以便在更高的抽象层次上进行处理。

(4)效率与低存储量需求：通俗地说，效率指的是算法执行的时间。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就高。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，比如求100个人的平均分与求1 000个人的平均分所花的执行时间或运行空间显然有一定的差别。

1.4 算法描述

描述算法的方法很多，根据描述方法的不同，可以将算法描述分为以下几种。

(1)用自然语言描述算法：自然语言就是人们日常生活中使用的语言。如汉语、英语、日语等。此种语言的特点有通俗易懂，缺乏直观性，不够简洁，且易产生歧义。使用此种语言描述要注意描述要求尽可能精确、详尽。

(2)框图算法描述：这是一种图示法，描述算法形象、直观，容易理解。但用来描述比较复杂的算法就显得不够清晰和方便。

(3)伪代码描述算法：这种描述方法很像程序，但不能直接在计算机上编译、运行。它描述的算法易编、易懂，修改容易，格式统一，容易转化为程序语言代码。

(4)用程序或函数实现算法：用计算机能理解和执行的程序设计语言把算法表示出来，然后把程序输入到计算机并执行，计算机就能按照预定的算法去解决问题，获得结果。用程序表示算法时，必须按照程序设计语言适用某类计算机的具体规定来进行。

本书主要介绍算法的思路和实现过程，并尽可能地将算法对应于C语言函数或程序(或类C语言算法描述)，使具备C语言基础的读者能阅读或上机运行，以便更好地理解算法。

1.5 算法性能评价

一种数据结构的优劣由实现其各种运算的算法具体体现，对数据结构的分析实质上就是对实现运算的算法分析，除了要验证算法是否满足算法设计的基本要求之外，还需要对算法的效率作性能评价。因此算法分析是每个程序设计人员应该掌握的技术。评价算法的标准很多，评价一个算法主要看这个算法所占用机器资源的多少，而这些资源中时间代价与空间代价是两个主要的方面。通常是以算法执行所需的机器时间和所

占用的存储空间来判断一个算法的优劣。

当一个算法转换成程序并在计算机上执行时，其运行所需的时间一般取决于以下几个因素。

(1)硬件的速度：即机器执行指令的速度。

(2)实现算法的程序设计语言：对于同一个算法，实现语言的级别越高，执行效率就越低。

(3)编译程序所产生的机器代码的质量。

(4)依据的算法选用何种策略。

(5)问题的规模：例如求100以内还是1 000以内的素数。

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行，效率均不相同。这表明使用绝对的时间单位去衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为一个特定算法“运行工作量”的大小，只依赖于问题的规模(通常用整数量 n 表示)。或者说，它是问题规模的函数。这种函数被称为算法的时间复杂度和空间复杂度。

1.5.1 时间性能分析

算法执行时间需通过该算法编制的程序在计算机上运行时所消耗的时间来度量。而对于算法分析，我们关心的是算法中语句总的执行次数 $T(n)$ 关于问题规模 n 的函数，进而分析 $T(n)$ 随 n 的变化情况并确定 $T(n)$ 的数量级(Order of Magnitude)。在这里，我们用“O”来表示数量级，这样我们可以给出算法的时间复杂度概念。所谓算法的时间复杂度，即是算法的时间量度，记作

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

它表示随问题规模 n 的增大，算法的执行时间的增长率和 $f(n)$ 的增长率相同，称作算法的渐进时间复杂度，简称时间复杂度。

我们来看看下面三种求和的算法。

```
(1)int i,sum=0,n=100;                    /*执行1次*/
   for(i=0;i<=n;i++)                     /*执行n+1次*/
   {
      sum=sum+i;                         /*执行n次*/
   }
(2)int i,sum=0,n=100;                    /*执行1次*/
   sum=(1+n)*n/2;                        /*执行1次*/
```

显然，第一种算法，执行了 $1+(n+1)+n$ 次 $=2n+2$ 次；而第二种算法，是 $1+1=2$ 次。我们把循环看作一个整体，忽略头尾循环判断的开销，那么这两个算法其实就是 n 次与1次的差距，算法好坏显而易见。

```
(3)int i,j,x=0,sum=0,n=100;              /*执行1次*/
   for(i=1;i<=n;i++)
   {
```

```
    for(j=1;j<=n;j++)
    {
        x++;
        sum=sum+x;                          /*执行n*n次*/
    }
}
```

第三个例子中，i从1到100，每次都要让j循环100次。而当中的x++和sum=sum+x，从1+2+3+，…，+10000，执行了100^2次。所以这个算法当中，循环部分的代码整体需要执行n^2（忽略循环体头尾的开销）次。显然这个算法的执行次数对于同样的输入规模$n=100$，要多于前面两种算法，这个算法的执行时间随着n的增加也将远远多于前面两个。

可以从以上问题描述中得到启示，同样问题的输入规模是n，求和算法的第一种，求1+2+，…，+n需要一段代码运行n次，那么这个问题的输入规模使得操作数量是$f(n)=n$，显然运行100次的同一段代码规模是运算10次的10倍。而第二种，无论n为多少，运行次数都为1，即$f(n)=1$。第三种，运算100次是运算10次的100倍。因为它是$f(n)=n^2$。这样用大写O()来体现算法时间复杂度的记法，我们称之为大O记法。一般情况下，随着n的增大，T(n)增长最慢的算法为最优算法。

显然，由此算法时间复杂度的定义可知，我们的三个求和算法的时间复杂度分别为$O(n)$，$O(1)$，$O(n^2)$。

通常，$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)<O(2^n)$。

1.5.2 空间性能分析

类似于算法的时间复杂度，算法中以空间复杂度(Space Complexity)作为算法所需存储空间的量度，记作

$$S(n)=O(f(n))$$

其中n为问题的规模(或大小)。一个上机执行的程序除了需要存储空间来寄存本身所用指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间，否则应同时考虑输入本身所需空间(和输入数据的表示形式有关)。若额外空间相对于输入数据量来说是常数，则称此算法为原地工作，空间复杂度为O(1)。又如果所占空间量依赖于特定的输入，则除特别指明外，均按最坏情况来分析。

通常，我们都使用“时间复杂度”来指运行时间的需求，使用“空间复杂度”指空间需求。当不用限定词地使用“复杂度”时，通常都是指时间复杂度。本书重点要讲的还是算法的时间复杂度的问题。