“十三五”普通高等教育规划教材

实用数据结构与算法

SHIYONG SHUJU JIEGOU YU SUANFA

主 编 胡 慧 闵娟娟 余玛俐

北京邮电大学出版社
www.buptpress.com

“十三五”普通高等教育规划教材

实用数据结构与算法

主　编　胡　慧　闵娟娟　余玛俐
副主编　邓　维　李　静
参　编　周小雄　吕　慧

北京邮电大学出版社
·北京·

内 容 简 介

本书主要介绍了常见的数据结构以及操作等内容，本书以实用性和可操作性为主，深入浅出地安排内容。全书的内容和体系设计充分考虑了应用型本科院校学生的实际情况，增设了数据结构实现基础一章，内容包括函数、数组、指针、结构体等在C语言中较难的知识点，给学生实现数据结构中的算法打下扎实的基础。

全书共分八章，内容包括：绪论、数据结构实现基础、线性表、串和特殊矩阵、树与二叉树、图、查找、排序等。每章均侧重基本概念、基础知识和基本操作的讲解，内容精炼，语言通俗易懂，在涉及理论、算法等一类知识内容时，注重知识内容的实用性和趣味性，从生活应用举例、解决实际问题出发达到抛砖引玉的目的。每章后均有小结和习题，同时还设有大量典型的解题指导与示例，为学生提供了解题的思路和答案，帮助学生对所学内容进行总结和消化。

本书既可作为高等学校计算机及相关专业“数据结构”课程的教材，也可作为各培训学校、培训机构等的培训教材，同时，也适合作为从事计算机工程与应用的广大读者的自学教材和参考书。

图书在版编目(CIP)数据

实用数据结构与算法/胡慧，闵娟娟，余玛俐主编. -- 北京：北京邮电大学出版社，2018.8
ISBN 978-7-5635-5497-3

Ⅰ.①实… Ⅱ.①胡… ②闵… ③余… Ⅲ.①数据结构②算法分析 Ⅳ.①TP311.12②TP312

中国版本图书馆CIP数据核字(2018)第157367号

书　　名	实用数据结构与算法
主　　编	胡　慧　闵娟娟　余玛俐
责任编辑	沙一飞
出版发行	北京邮电大学出版社
社　　址	北京市海淀区西土城路10号(100876)
电话传真	010-82333010　62282185(发行部)　010-82333009　62283578(传真)
网　　址	www.buptpress3.com
电子信箱	ctrd@buptpress.com
经　　销	各地新华书店
印　　刷	中煤(北京)印务有限公司
开　　本	787 mm×1 092 mm　1/16
印　　张	18.5
字　　数	460千字
版　　次	2018年8月第1版　2018年8月第1次印刷

ISBN 978-7-5635-5497-3　　定价：45.00元

preface 前 言

“数据结构”主要研究非数值计算问题中计算机的操作对象、它们之间的关系以及运算等内容，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。它不仅是一般程序设计（特别是非数值型程序设计）的基础，也是设计和实现操作系统、数据库系统、编译程序以及其他软件的重要基础。

本书根据教育部高等学校计算机科学与技术教学指导委员会制定的《高等学校计算机科学与技术专业发展战略研究报告暨专业规范》以及教育部关于应用型大学计算机课程基本要求，总结编者多年的教学和软件开发经验，以C语言为依托，采用Visual Studio 2010作为语言开发环境，介绍了用计算机解决一系列问题特别是非数值信息处理问题时所用的各种组织数据的方法、存储数据的方法以及基本操作的算法。本书的目的在于使读者掌握算法分析、算法设计的基本方法，掌握数据的逻辑结构、存储结构的基本设计方法和理论，培养读者分析问题、解决问题的能力，并为后续课程的学习打下良好的理论基础和实践基础。

本书以实用性和可操作性为主，不强调长篇大论的理论性讲解，结构合理、内容充实、概念清晰、通俗易懂，内容叙述深入浅出，突出重点，所有例题均调试通过。每章开始均设置了学习目标，使读者知道通过本章学习应达到的目标，每章结尾均有小结，提纲挈领，对知识点进行强化，注重理论联系实际，使读者能更好地理解关键知识。在每章还设置了解题指导与示例，为读者提供了解题的思路和答案，帮助读者对所学内容进行总结和消化。

本书共分为八章，第一章主要介绍了数据结构和算法的基本概念，第二章到第四章介绍了线性结构的概念及其实现，第五章和第六章介绍了树型结构和图型结构，最后两章分别介绍查找和排序操作。

本书由胡慧、闵娟娟、余玛俐、周小雄、吕慧编写，其中胡慧、闵娟娟、余玛俐为主编，邓维、李静为副主编。全书由邓安远教授担任主审，邓安远教授为本书的出版倾注了很多心血，提出了很多宝贵的修改意见。本书在编写过程中，参考了国内外相关教材、资料，得到了众多教学一线教师的支持和帮助，在此一并衷心地表示感谢。

本书既可作为高等学校计算机及相关专“数据结构”课程的教学用书，也可作为从事计算机工程与应用的广大读者的参考书。

由于时间仓促及编者水平有限，书中难免有不足及疏漏之处，恳请读者批评指正。

编 者

目　录

第一章 绪 论

【学习目标】

1. 掌握数据结构的相关概念。
2. 领会抽象数据类型的含义以及定义方式。
3. 理解算法的概念、表示、特性和目标。
4. 掌握算法的时间效率和存储空间的度量。

数据结构是一门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。本章讲述数据结构的基本概念及相关术语，介绍数据结构、数据类型和抽象数据类型之间的联系，介绍算法的特点及算法的时间与空间复杂度。

1.1 数据结构的基本概念

在讲解什么是数据结构之前，我们先来看一个项目开发的真实案例，众所周知，QQ 空间、微信朋友圈、微博是当前非常热门的信息分享和网络交流平台，大家或多或少在这几个平台上发表过信息或者回复过别人的评论。某校计算机专业的几个学生准备着手开发一个帮助高校师生取快递、带餐、购物等服务的校园跑腿平台，在做订单评价功能的需求分析时，参考了 QQ 空间的信息评论回复功能，在具体实现的时候，发现会被如下问题所困扰：

- 如何在计算机中存放所有订单的评论和回复？
- 所用方法能让某条评论的所有回复紧跟着呈现在该评论的下方吗？
- 所用方法能删除某条评论而不影响其他评论和回复的自然呈现吗？
- 若想将评论按照发表时间顺序排列，又应该如何去排呢？

实际上，大多数项目比上面讲的订单评价功能的实现要复杂得多，想象一下人机对弈软件的开发或者田径赛的时间安排问题；或者看一个航空售票软件，这个软件需要存储旅客和航班的各种信息。这些软件都包含了许多的数据且数据元素之间的联系复杂，但是需要处理的数据并不是杂乱无章的，它们一定有内在的联系，只有弄清楚它们之间本质的联系，才能使用计算机对大量的数据进行有效的处理。数据结构正是讨论这类程序设计问题所涉及的现实世界实体对象的描述、信息的组织方法及其相应操作的实现。

在借助于集成开发环境可以快速生成可视化程序的今天，程序设计已不再是计算机专业人员的特长。有些人认为，只要掌握几种开发工具就可以成为编程高手，其实不然，要想成为一个专业的软件开发人员，至少需要具备以下三个条件：(1)能够熟练地选择和设计各种数据结构和算法；(2)能够熟练地掌握至少一门程序设计语言；(3)熟知所涉及的相关应用领域的知

识。其中，后两个条件比较容易实现，而第一个条件则需要花费相当多的时间和精力才能够达到，它是区分一个软件开发人员水平高低的重要标志之一，数据结构贯穿程序设计的始终，缺乏数据结构和算法的深厚功底，很难设计出高水平且具有专业水准的应用程序。

1.1.1 基本概念和术语

计算机发展的初期主要应用于数值计算，数据量小且结构简单，那时的程序设计人员将主要精力放在程序设计的技巧上，面对如何在计算机中组织数据并不需要花费太多的时间和精力。然而，随着计算机软、硬件的发展，计算机的应用领域不断扩大，从客观事物中抽象出的数据日益显现出多样化的特征，简单的数据类型已经远远不能满足需要，各数据元素之间的复杂联系也已经不是普通的数学方程式所能表述的了。在这种背景下，一种专门研究数据之间结构关系的学科——数据结构应运而生。**数据结构是一门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。**

20 世纪 60 年代末到 70 年代初，出现了大型程序，软件也相对独立，结构程序设计成为程序设计方法学的主要内容，人们越来越重视数据结构。1968 年，美国高德纳(Donald E. Knuth)教授开创了数据结构的最初体系，在其所著的《计算机程序设计艺术》第一卷《基本算法》中较系统地阐述了数据的逻辑结构和存储结构及其操作。同年，数据结构作为一门独立的课程开始进入大学课堂。

注意："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综合性的专业基础核心课程。这门课程的内容不仅是一般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的重要基础。

下面介绍数据结构的相关术语。

数据(data)**是描述客观事物的数值、字符以及能输入且能被处理的各种符号集合。**如数学计算中用到的整数和实数、文本编辑中用到的字符、多媒体技术中涉及的视频和音频信号，经采集转换后都能形成计算机可操作的数据。数据是计算机加工的对象，是数据元素的集合。

数据元素(data element)**是组成数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。**数据元素可以是不可分割的"原子"，例如，整数"5"或字符"A"；也可以由若干个数据项组成，**数据项**(data item)**是有独立含义的、不可分割的最小单位。**例如，学生基本信息表中的学号、姓名、性别等都是数据项，而表中的一行就是一个数据元素，此时的数据元素通常称为记录。

数据对象(data object)**是性质相同的数据元素的集合，是数据的一个子集。**例如，正整数数据对象是集合 N={1,2,3,…}，大写字母的字符数据对象是集合 C={'A','B',…'Z'}，学生基本信息表也可以是一个数据对象。由此可见，不论数据元素集合是无限集(如正整数集)，还是有限集(如大写字母字符集)，还是多个数据项组成的复合数据元素集合(如学生基本信息表)，只要集合内元素的性质相同，都可称之为一个数据对象。

提示：由于数据对象是数据的子集，而在实际应用中，处理的数据元素通常具有相同的性质，所以在不产生混淆的情况下，将数据对象简称为数据。

数据结构(data structure)**是相互之间存在一种或多种特定关系的数据元素的集合。**计算

机所处理的数据并不是数据的简单汇集，而是具有内在联系的数据集合，数据结构主要研究数据元素之间的相互关系，即数据的组织形式，具体应包括三个方面的内容：

①数据元素之间的逻辑关系，也称为数据的逻辑结构；

②数据元素及其关系在计算机存储器内的表示，称为数据的存储结构；

③数据的运算集合，即对数据进行的操作以及实现这些操作的算法。

1.1.2 数据的逻辑结构

数据的逻辑结构是指数据元素之间的逻辑关系的描述，它与数据在计算机中的存储无关，是独立于计算机的。它可以看作是从具体问题抽象出来的数学模型，为了解决某个具体问题，在对问题理解的基础上，我们选择一个合适的“数学模型”来表示数据元素之间的逻辑关系。数据的逻辑结构有数据元素和数据元素间的逻辑关系两个要素，根据数据元素之间逻辑关系的不同特性，通常有下列四类基本结构：

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。例如，确定一名学生是否为班级成员，只需将班级看做一个集合结构。

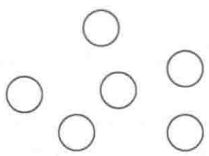

图 1-1 集合结构

②线性结构：结构中的数据元素之间存在一对一的线性关系。例如，字母表的逻辑结构就是一个线性结构，如图 1-2 所示。

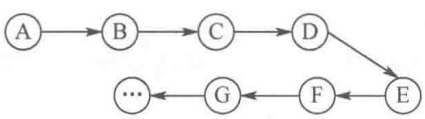

图 1-2 字母表的逻辑结构图

我们还可以用一个二元组 B=(D,R)来描述数据的逻辑结构，其中 D 是数据元素的有限集，R 是 D 上关系的有限集。如果只讨论字母表之间的顺序关系，用 r 表示，则字母表的逻辑结构可以用下面的二元组形式化的予以表达：

B=(D,R)

D={A,B,C,D,E,F,G…}

R={r}

r={<A,B>,<B,C>,<C,D>,<D,E>,<E,F>,<F,G>…}

③树状结构：结构中的数据元素之间存在着一对多的层次关系。例如，九江学院机构设置按类型分为行政单位、科研结构、教学单位、后勤管理处和附属机构，其中教学单位又包括多个二级学院，从而构成了树状结构，如图 1-3 所示。

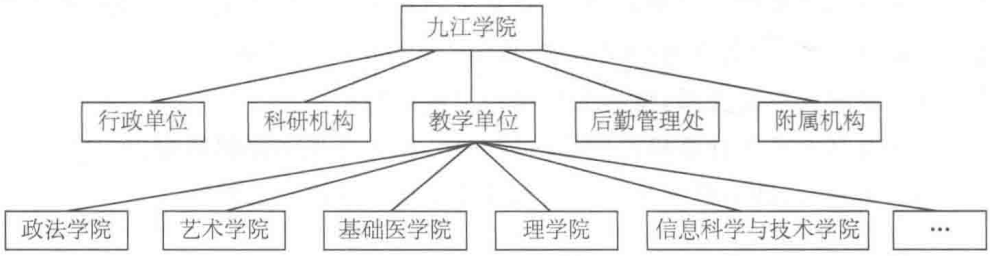

图 1-3 九江学院的机构设置图

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。例如，武汉有直飞丽江的航班，南昌到丽江没有直飞航班，坐飞机需要经过昆明中转，武汉、昆明、丽江、南昌四个城市之间的直飞航空路线图就是一个图状结构，如图 1-4 所示。

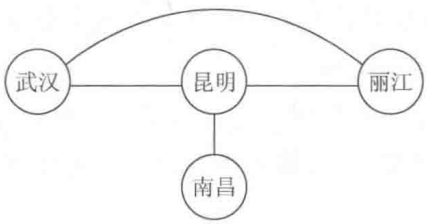

图 1-4 四个城市间的直飞航空路线图

思考：你能用二元组 B=(D,R)描述出图 1-3 和图 1-4 所对应的逻辑结构吗？

树状结构和图状结构都是非线性结构。在非线性结构中，一个数据元素可以有多个直接后继或者既有多个直接后继又有多个直接前驱。由于集合的关系非常松散，只有属于或不属于的简单关系，因此可以用其他结构代替它，故数据的逻辑结构可概括如下：

- 线性结构——线性表、栈、队列、字符串、数组、广义表；
- 非线性结构——树、图。

上面提到的这些逻辑结构在本书的后续章节我们将会进行详细地介绍。

1.1.3 数据的存储结构

要对数据进行处理，必须先将数据存储在计算机中，将数据无规律地存储在计算机中就像一本英语字典中随意编排单词一样，没有任何使用价值。**数据的存储结构（又称物理结构）是指数据的逻辑结构在计算机中的存储形式，包括数据元素的存储表示和逻辑关系的存储表示两个方面。**数据的逻辑结构是面向问题的，而数据的存储结构是面向计算机的，其基本的目标就是将数据及其逻辑关系存储在计算机中。

在计算机中，数据元素用一个结点来表示，最基本的存储结构主要有以下两种：

①顺序存储结构：要求所有的数据元素依次存放在一片连续的存储空间里，使得逻辑上相邻的结点在存储器中的物理位置也是相邻的，结点间的逻辑关系通过存储单元的邻接关系来体现，通常借助程序设计语言的数组来描述。

假定数据元素 1 存放的位置是 L_0，每个数据元素占用 m 个存储单位，则 n 个数据元素的顺序存储结构示意图如图 1-5 所示。

②链式存储结构：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。此时，数据元素的存储关系并不能反映出其逻辑关系，因此需要给每个结点附加指针字段，用于存放后继数据元素的存储地址，这样通过地址就可以找到相关联数据元素的位置，通常借助于程序设计语言中的指针类型来描述这种存储结构。

如图 1-6 所示，给出了一个链式存储结构示意图，为了更清楚地反映链式存储结构，可采用更直观的图示来表示，如图 1-7 所示。

存储地址	存储内容
L_0	元素1
L_0+m	元素2
	……
$L_0+(i-1)\times m$	元素i
	……
$L_0+(n-1)\times m$	元素n

图 1-5 顺序存储结构示意图

存储地址	数据域	指针域
1345	元素1	1400
1356	元素4	^
……	……	……
1400	元素2	1536
……	……	……
1536	元素3	1356

图 1-6 链式存储结构示意图 1

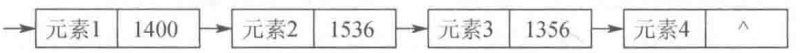

图 1-7 链式存储结构示意图 2

1.1.4 数据的运算集合

数据结构研究的内容除了数据的逻辑结构、数据的存储结构以外，还包括在这些数据上定义的一个运算集合，**即施加在这些数据上的操作的集合**。对于某一类型的数据结构，只有通过对定义的运算集合的研究，才能清楚地理解数据结构的定义和作用。

数据的运算是定义在数据的逻辑结构之上的，而运算的具体实现依赖于数据的存储结构。数据的运算集合视情况而定，例如，定义在整数上的有加、减、乘、除、求余等操作。一般而言，数据的运算包括插入、删除、查找、输出和排序等操作。

插入是指在一个结构中增加一个新的结点。

删除是指在一个结构中删除一个结点。

查找是指在一个结构中检索满足条件的结点。

输出是指将一个结构中所有结点的值打印、输出。

排序是指将一个结构中所有结点按某种顺序重新排列。

例如，对于学生基本信息表，表中的一行就是一个数据元素，数据元素与数据元素之间的逻辑关系是一种简单的线性结构。在计算机中存放时，可采用顺序存储结构也可以采用链式存储结构。当有新生转入时要增加数据元素，学生退学时要删除相应的数据元素，可以根据条件查看某个学生的信息，发现学生信息有误时要修改数据元素。这里的增、删、查、改就是数据的运算集合。

1.2 抽象数据类型

数据类型(data type)**是一组性质相同的值集合以及定义在这个值集合上的一组操作的总称**。例如,高级程序设计语言中的整型变量,其值集为某个范围上的整数(范围大小依赖于不同的机器和软件系统),定义在其上的操作为加、减、乘、除和取模等算术运算。而实型变量也有自己的取值范围和相应运算,如取模运算是不能用于实型变量的。C 语言除了提供整型、实型、字符型等基本类型外,还有数组、结构体、共用体等结构类型,数据类型反映了程序设计语言的数据描述和处理能力。

抽象数据类型(abstract data type,ADT)**是一个数据模型及定义在该模型上的一组运算**。抽象数据类型的定义取决于客观存在的一组逻辑特性,而与其在计算机内如何表示和实现无关。它抽取反映问题的本质点,对用户隐藏数据存储和操作实现的细节,从而大大提高软件的开发效率。其实整数类型也是一个抽象数据类型,人们在程序设计中大量使用整数类型及其相关的算术运算,而没有人去追究这些算术运算是如何实现的。

抽象数据类型是数据类型的进一步抽象,是大家熟知的基本数据类型的延伸和发展。除了包括那些已经定义并实现的数据类型外,还包括用户在设计软件时自己定义的数据类型。例如,在开发地图类软件系统时,经常会用到坐标,x 和 y 变量总是成对出现,我们就可以定义一个叫 point 的抽象数据类型,它有 x、y 两个实型变量,根据实际应用可以定义求两点之间的距离、偏移量、坐标的转换等操作。本课程将要学习的线性表、栈、队列、树、图等结构也是一个个不同的抽象数据类型。

抽象数据类型具体包括三部分:数据对象、数据对象上关系的集合以及对数据对象的基本操作的集合。定义一个抽象数据类型时,必须给出它的名字及各基本操作的函数名,并且规定这些函数的参数性质,一旦定义了一个抽象数据类型及具体实现,在程序设计中就可以像使用一般数据类型那样,十分方便地使用抽象数据类型。下面给出描述抽象数据类型的标准格式:

```
ADT 抽象数据类型名{
数据对象:<数据对象的定义>
数据关系:<数据关系的定义>
基本操作:
操作名 1(参数表)
初始条件:<初始条件描述>
操作结果:<操作结果描述>
操作名 2(参数表)
初始条件:<初始条件描述>
操作结果:<操作结果描述>
……
操作名 n(参数表)
初始条件:<初始条件描述>
操作结果:<操作结果描述>
}ADT 抽象数据类型名
```

ADT 定义格式中的“初始条件”描述了操作执行之前数据结构和参数应满足的条件，若初始条件为空，则省略。“操作结果”说明了操作正常完成之后，数据结构的变化状况和应返回的结果。

下面以复数为例，给出一个完整的抽象数据类型的定义。

ADT Complex{
数据对象：D={e_1，e_2 | e_1，e_2 ∈ R，R 是实数集}
数据关系：S={<e_1，e_2> | e_1 是复数的实部，e_2 是复数的虚部}
基本操作：
Creat(&C，x，y)
操作结果：构造复数 C，其实部和虚部分别被赋以参数 x 和 y 的值。
GetReal(C)
初始条件：复数 C 已存在。
操作结果：返回复数 C 的实部值。
GetImag(C)
初始条件：复数 C 已存在。
操作结果：返回复数 C 的虚部值。
Add(C1，C2)
初始条件：C1，C2 是复数
操作结果：返回两个复数 C1 和 C2 的和。
Sub(C1，C2)
初始条件：C1，C2 是复数
操作结果：返回两个复数 C1 和 C2 的差。
}ADT Complex

1.3 算法和算法性能分析

我们讨论数据对象的各种不同的组织方式，是为了得到处理这些数据对象的高效算法。当我们在研究图书馆的图书摆放问题时，发现用不同方法摆放图书，会直接影响图书查找、插入等工作的效率。在计算机的世界里，“图书”就是待处理的“数据对象”，图书的摆放方法就是数据对象在计算机中的组织方式，“查找”“插入”图书等工作就是对数据进行的“操作”，而完成这些操作所用的方法就是“算法”。在讨论“数据结构”这个概念时，关心的不仅仅是数据对象本身以及它们在计算机中的组织方式，还要关心与它们相关联的一个操作集及实现这些操作的高效算法。

在本节，我们来学学有关算法的基本知识。

1.3.1 算法的基本概念

算法是对特定问题求解步骤的一种描述，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。简单来说为解决一个问题而采取的方法和步骤，就称算法。

对于给定的问题，可以有多种不同的算法来解决。例如，求 1+2+3+…+10 000 结果，

根据不同的解题思路，我们给出了如下两段代码。

代码 1：

```
int i,sum=0,n=10000;
for(i=1;i<=n;i++)
   sum=sum+i;
printf("sum=%d",sum);
```

代码 2：

```
int i,sum=0,n=10000;
sum=(1+n)*n/2;
printf("sum=%d",sum);
```

很明显代码 2 采用了等差数列解题的算法，计算机只要做一次算术运算，而代码 1 采用的是累加的操作步骤，计算机则要循环一万次加法运算。所以，为了有效地解决问题，我们不仅要保证算法正确，还要考虑算法的效率，从多个可以解决问题的算法中选择合适的算法。

算法可用自然语言、伪代码、流程图或高级程序设计语言进行描述，在不同层面上讨论算法有不同的描述方法。本书在高级程序设计语言的基础上讨论算法，选取 C 语言作为描述算法的工具。

注意：算法和程序是有区别的，算法是解决问题的步骤，程序是算法在计算机中的实现，与所用的计算机及所用的编程语言有关。算法需要程序来完成功能，程序需要算法作为灵魂。由于本书直接选取 C 语言作为描述算法的工具，所以给出的大多数代码不能简单地认为就是程序，而应更多关注代码所实现的算法的思想。

1.3.2 算法的特性

算法具有 5 个基本特性：有穷性、确定性、可行性、输入、输出。

有穷性：一个算法必须总是（对任何合法的输入值）在执行有限的操作步骤之后结束，且每一步骤都可在有限时间内完成。现实中写出的带有死循环的代码，就不满足有穷性。有穷也指在实际应用中的执行时间合理，如果一个算法需要几十年才能执行完，它的意义也就不大了。

确定性：算法中每一个步骤应当是明确的，不能存在二义性。算法的每个步骤有明确的定义，在一定条件下，算法都只有一条执行路径，相同的输入只能有唯一的输出结果。

可行性：算法中的每一个步骤都必须是可行的，都可以通过基本操作的有限次运行完成。可行性意味着算法可以转换成程序上机运行。

输入：根据问题的不同，算法可以有 0 到多个输入，例如，求任意数的绝对值，就必须输入这个数，而如果只是打印出“hello,world”这样的代码，则不需要输入参数。

输出：至少有一个或多个输出。算法的执行应当是有结果的，没有结果我们就不知道问题的解决情况，无输出的算法没有任何意义。

1.3.3 算法设计的要求

当用算法来解决某问题时，算法设计的目标是正确、可读、健壮、高效、低耗。所以设计算

法时，通常应遵循以下设计原则：

正确性：算法应该满足具体问题的求解需求，能得到问题的正确答案。对于合法的输入数据能够产生满足要求的输出结果。

可读性：一个好的算法首先应便于人们阅读、理解和交流，其次才是计算机执行性。可读性高有助于人们理解算法，而晦涩难懂的算法易于隐藏错误，难于调试和修改。

健壮性：当输入不合理的数据时，算法也应当能作出合适的处理，而不是产生异常或莫名其妙的输出结果。

高效率与低存储量：算法的效率指的是算法执行时间，执行时间越短效率越高；存储量需求指的是算法执行过程中所需的最大存储空间。设计算法应该尽量满足时间效率高和存储量低的需求。

思考：*求一个实数的平方根，当输入一个负数时，算法应该如何处理？*

1.3.4 算法的时间性能分析

通常对于一个实际问题的解决，可能有多种解决方案，从而对应多个可行的算法，那么如何从这些算法中找出最有效的算法？对于一个解决实际问题的算法，又如何来评价它的优劣？

衡量算法效率的方法主要有两类：事后统计方法和事前分析估算方法。事后统计方法必须把算法转换成可执行的程序，在计算机上运行，然后测算其时间和空间开销。这种方式测算结果受计算机的软、硬件等环境因素（如编译产生的代码质量、机器执行指令的速度、输入的数据量等）的影响，难以精确估计，所以我们通常采用事前分析估算方法。目前，测定算法运行时间最可靠的事前分析估算方法就是计算对运行时间有消耗的基本操作的执行次数，算法的运行时间与这个计数成正比。

我们把一个算法中某语句重复执行的次数称为该语句的频度。算法中所有基本语句频度之和是问题规模 n 的某个函数 $f(n)$，**问题规模是算法求解问题输入量的多少，是问题大小的本质表现**。例如，排序运算中的问题规模 n 为参加排序的记录数。下面我们给出代码段 1 对应算法的语句频度。

例 1.1

```
                                    语句频度
(1)int i,sum=0,n=10000;             /* 执行 1 次 */
(2)for(i=1;i<=n;i++)                /* 执行 n+1 次 */
(3)    sum=sum+i;                   /* 执行 n 次 */
(4)printf("sum=%d",sum);            /* 执行 1 次 */
```

该算法的所有语句频度之和为 $f(n)=2n+3$。

在大多数情况下，算法的执行时间是随问题规模增长而增长的，因此对算法的评价通常只需考虑其随问题规模增长的趋势。在这种情况下，我们只需考虑当问题规模充分大时，算法中基本语句的执行次数在渐进意义下的阶。例 1.1，当 n 趋向无穷大时，显然有

$$\lim_{n\to\infty} f(n)/n=\lim_{n\to\infty}(2n+3)/n=2$$

当 n 充分大时，$f(n)$ 和 n 之比是一个不等于 0 的常数，即 $f(n)$ 和 n 是同阶的或者说 $f(n)$ 和 n 的数量级相同，记作 $O(f(n))=O(n)$，这里用“O”来表示数量级。

算法的时间复杂度 $T(n)$ 是该算法的时间度量，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**。

时间复杂度并不是所有语句的精确执行次数之和，而是估算的数量级。它重在体现随着问题规模的增大，算法执行时间增长的变化趋势。**分析算法时间复杂度的基本方法为：找出所有语句中执行次数最多的那条语句作为基本语句，计算基本语句的频度得到问题规模 n 的某个函数 $f(n)$，取其数量级用符号"O"表示即可**。

下面再举例说明如何求算法的时间复杂度。

例 1.2 常数阶示例　　语句频度

```
(1)int i,sum=0,n=10000;  /* 执行 1 次 */
(2)sum=(1+n) * n/2;      /* 执行 1 次 */
(3)printf("sum=%d",sum); /* 执行 1 次 */
```

由于以上三条语句的频度都为 1，可以选择其中任意一条作为基本语句。由于 $f(n)=1$，所以该代码段的执行时间是一个与问题规模 n 无关的常数。算法的时间复杂度为**常数阶**，记作 $T(n)=O(1)$。

提示：如果算法的执行时间不随问题规模 n 的增加而增加，那么算法中基本语句的频度就是某个常数，即使这个常数再大，算法的时间复杂度都是 $O(1)$。

一般情况下，对循环语句只需考虑循环体中语句的执行次数，当有若干个循环语句时，算法的时间复杂度是由最深层循环内的基本语句的频度 $f(n)$ 决定的。因此对于例 1.1 给出的代码段，频度最大的是语句(3)，$f(n)=n$，所以该代码段的时间复杂度为 $T(n)=O(n)$，$O(n)$ 又称**线性阶**。

例 1.3 平方阶示例

```
(1)x=1;
(2)for(i=1;i<=n;++i)
(3)   for(j=1;j<=i;++j)
(4)         ++x;
```

该代码段中频度最大的是语句(4)，$f(n)=1+2+3+\cdots+n=n(n+1)/2=n^2/2+n/2$，所以该代码段的时间复杂度为 $T(n)=O(n^2/2+n/2)=O(n^2)$，即时间复杂度为平方阶。

数据结构中常见的时间复杂度数量级递增排列依次为：常量阶 $O(1)$、对数阶 $O(\log_2 n)$、线性阶 $O(n)$、线性对数阶 $O(n\log_2 n)$、平方阶 $O(n^2)$、立方阶 $O(n^3)$、指数阶 $O(2^n)$ 等。

注意：一般情况下，随着 n 的增大，$T(n)$ 增长较慢的算法为较优算法。当 n 值较大时，指数阶 $O(2^n)$ 的算法效率极低，失去了应用价值。因此在选择算法的时候，应该避免使用指数阶的算法。

有些情况下，算法的时间复杂度不仅仅依赖于问题的规模，还与问题输入数据集的初始状态有关。例如，在一个有 n 个数据元素的线性结构中检索指定数据元素 x，如果 x 正好就是线性结构中的第一个结点，则检索时间为 $O(1)$。如果位于最后一个结点或线性结构中没有该数据元素，则检索时间为 $O(n)$。这样就有了算法在最好、最坏以及在平均状态下的时间复杂度的概念。平均时间复杂度是指所有可能的输入实例均以等概率出现的情况下，算法的期望运行时间。由于最坏情况下的时间复杂度是算法在任何输入实例上运行时间的上界，所以如果没有特别说明，讨论的时间复杂度均是最坏情况下的时间复杂度。

1.3.5 算法的空间性能分析

一个算法在计算机上所占用的存储空间，包括存储算法本身所占用的存储空间、算法的输入数据所占用的存储空间和算法在运行过程中临时占用的辅助存储空间这三个方面。由于输入数据所占空间只取决于问题本身，是通过参数表由调用函数传递而来的，它不随算法的不同而改变。存储算法本身所占用的存储空间与算法书写的长短成正比，要压缩这方面的存储空间，就必须编写出较短的算法，而算法在运行过程中临时占用的辅助存储空间随算法的不同而不同。

算法的空间复杂度 $S(n)$定义为该算法所耗费的辅助存储单元的数量级，记作 $S(n)=O(f(n))$，其中 n 为问题的规模，$f(n)$为语句关于问题规模 n 所占用的辅助存储空间的函数。算法的空间复杂度表示随着问题规模 n 的增大，算法运行所需存储量的增长率与 $f(n)$的增长率相同。

例 1.4 将一维数组 a 中的 n 个数组元素依次循环前移一位放在原数组中，实现该问题的两个算法如下：

算法 1：

```
for(i=0;i<n;i++)
   b[i]=a[(i+1)%(n-1)];
for(i=0;i<n;i++)
   a[i]=b[i];
```

算法 2：

```
t=a[0]
for(i=0;i<n-1;i++)
   a[i]=a[i+1];
a[n-1]=t;
```

算法 1 借助了一个大小为 n 的辅助数组 b，所以其空间复杂度为 $O(n)$；算法 2 仅用了一个临时变量 t，所以其空间复杂度为 $O(1)$。

算法的时间复杂度和空间复杂度是衡量一个算法优劣的两个主要方面，我们往往希望一个算法运行时间短、占用存储空间少，而且其他性能也不错。而实际这两者往往相互影响，要节约算法的执行时间往往要以牺牲更多的空间为代价，而为了节省空间可能要耗费更多的计算时间。因此，当设计一个算法（尤其是大型算法）时，要综合考虑算法的各项性能，如算法的使用频率、算法处理的数据量的大小、算法描述语言的特性、算法运行的机器系统环境等各方面因素，才能够设计出比较好的算法。

1.4 小 结

本章主要讲述了数据结构和算法的基本概念。

数据结构包括数据的逻辑结构、存储结构和运算集合这三个部分。数据的逻辑结构定义了数据元素之间的逻辑关系，分为线性结构和非线性结构。数据的存储结构是逻辑结构在计