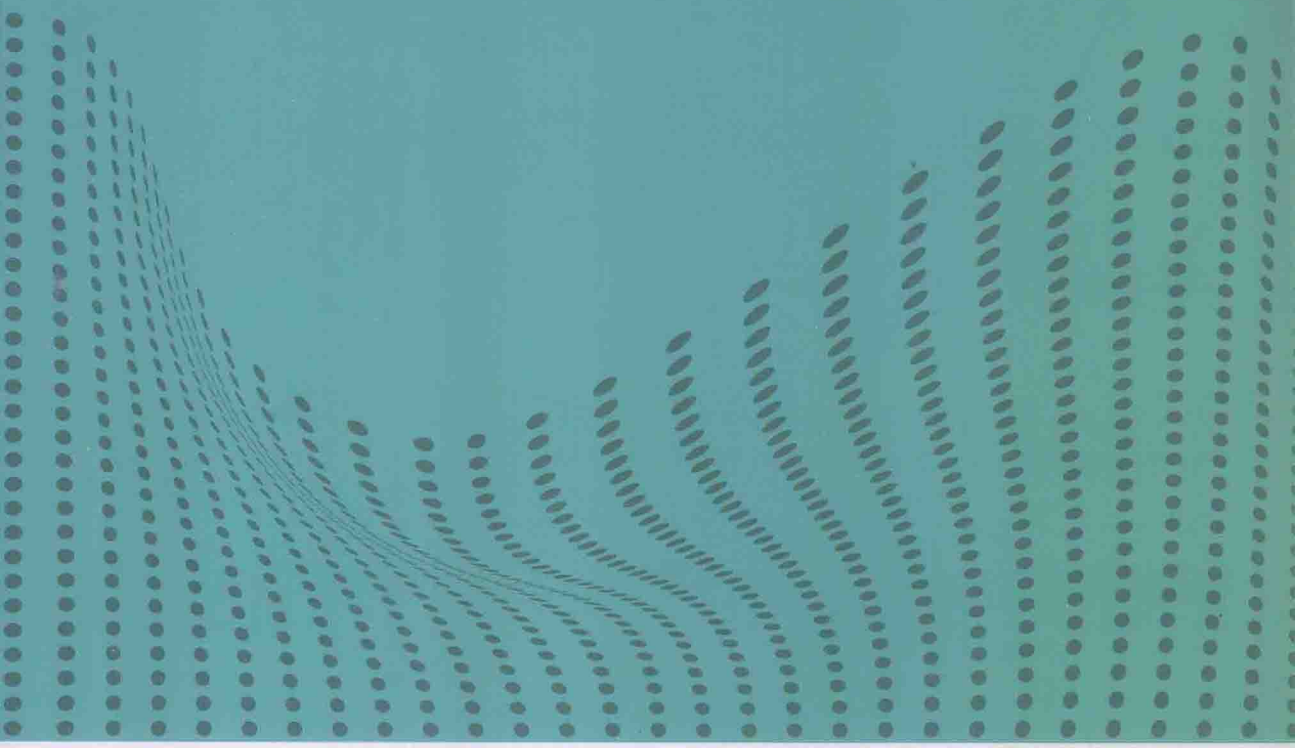
21世纪高等学校计算机系列规划教材



数据结构(C语言描述)

杨厚群　主编

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

21世纪高等学校计算机系列规划教材

数据结构（C语言描述）

主　编　杨厚群
副主编　王艳霞

上海交通大学出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内 容 提 要

本书以抽象数据类型为主轴，采用面向对象的思想，在基本概念、基本结构、基本技术等方面侧重深度，在算法的实现讲解上侧重广度，将计算机科学中的一些重要的问题求解技术贯穿其中，全面讲解了数据结构的基础知识和相应算法。本书内容包括线性表、栈和队列、串、数组和广义表、树和二叉树、图、查找与散列、排序以及文件。

本书可作为高等院校计算机专业学生的教材使用，也可作为相关人员学习参考资料使用。

图书在版编目(CIP)数据

数据结构：C 语言描述/杨厚群主编. —上海：上海交通大学出版社，2013

ISBN 978-7-313-09231-1

Ⅰ. ①数… Ⅱ. ①杨… Ⅲ. ①数据结构—高等学校—教材 ②C 语言—程序设计—高等学校—教材 Ⅳ. ①TP311.12 ②TP312

中国版本图书馆 CIP 数据核字(2012)第 281173 号

数据结构(C 语言描述)

杨厚群 主编

上海交通大学出版社出版发行

(上海市番禺路 951 号 邮政编码：200030)

电话：64071208 出版人：韩建民

北京振兴源印务有限公司印刷 全国新华书店经销

开本：787mm×1 092mm 1/16 印张：19 插页 1 字数：462 千字

2013 年 1 月第 1 版 2013 年 1 月第 1 次印刷

ISBN 978-7-313-09231-1/TP 定价：36.00 元

前 言 Preface

在计算机科学中，数据结构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的研究，还和计算机软件的研究有着密切的关系。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，还是设计和实现编译程序、操作系统、数据系统及其他系统程序和大型应用程序的重要基础。

作为计算机专业重要的主干课程，数据结构要求学习者学会分析和研究需解决的问题中的数据的特性，为其选择合适的数据结构进行描述，在此数据结构的基础上写出相应的算法，并初步掌握算法的时间复杂度和空间复杂度的分析技术。对初学者而言，难点在于如何掌握抽象的数据结构原理和方法，以及如何实现两者在实际问题中的应用。许多数据结构和算法的书籍多用伪代码和自然语言来描述，不提供完整算法，这给很多初学者带来不便。

本书以抽象数据类型为主轴，采用面向对象的思想，在基本概念、基本结构、基本技术等重要的基础知识上侧重深度，在算法的实现讲解上侧重广度，将计算机科学中的一些重要的问题求解技术贯穿其中，全面讲解了数据结构的基础内容和相应算法。所有算法都有算法功能说明、方法分析、实例描述，并且使用标准 C 语言编写实现，体现了数据抽象和方法抽象的结合。编者对本书内容的深度和广度方面都进行了仔细考虑。通过学习本书，读者可以快速理解和掌握数据结构原理和经典算法，真正领会数据结构和算法的本质。

全书共分 10 章，主要介绍了数据结构中的基本概念和术语，抽象数据类型，以及算法与算法分析；线性表的概念，线性表的顺序存储、链式存储和实现，双向链表，以及线性表的具体应用；栈的顺序存储结构和链式存储结构，队列的顺序存储结构和链式存储结构，循环队列以及相关的操作；串的表示、存储结构和实现，串的相关模式匹配算法；数组的存储结构和矩阵的压缩存储，广义表的基本概念、存储结构和操作；串和二叉树的基本概念、存储表示和实现，树和森林的存储表示和实现、相互转换，Huffman 算法的实现和 Huffman 编码；图的基本概念、存储结构，图的遍历算法，以及求最小生成树的 Prim 算法和 Kruskal 算法、求最短路径的 Dijkstra 算法和 Floyd 算法；查找的基本概念，包括各种查找方法及其算法实现和算法分析，以及散列表的基本概念、散列查找及性能分析；排序的概念及主要排序方法的原理、算法实现和算法分析；文件的基本概念和存储结构。

本书引入抽象数据类型，按照认知的规律安排章节内容。通过严谨的算法讨论和时空复杂度分析，并使用大量的实例讲解数据抽象与过程抽象，突出数据结构和算法的紧密结合。课程教学资源丰富，采用的例题都经过严格筛选、测试，全部在 Turbo C 2.0 环境下编

译通过。每章后面都附有大量的习题,通过完成习题可以加深读者对知识的理解。

本书可作为计算机科学与技术专业的本科教材,也可作为计算机应用专业的教材。教师可以根据本专业的特点、学生情况和教学学时选讲部分章节的内容。

本书由海南大学杨厚群教授任主编,王艳霞任副主编。其中,第1章至第3章由杨厚群、邢诒杏、陈静编写;第4章和第9章由马建强编写;第5章由王艳霞编写;第6章和第7章由陈海珠编写;第8章由闫庆华编写;第10章由符浅浅和杨厚群共同编写。全书由杨厚群统稿。陈绮教授提供了大量的例题和习题,何中市教授、徐光侠副教授对本书的编写给予了大力支持,对此编者表示由衷的感谢。

本书得到了海南大学科研启动项目(KYQD1118)的支持,在此一并表示感谢。

由于编者水平有限,疏漏之处在所难免,恳请专家和读者不吝指正。

编　者

目 录 Contents

第 1 章 绪 论

数据结构作为一门学科主要研究数据的各种逻辑结构和存储结构，以及对数据的各种操作。数据结构反映数据的内部构成，是数据存在的形式，即一个数据由哪些成分数据构成，以什么方式构成，呈现什么结构。在我们研究各种数据结构时，需要弄清楚：为什么要使用这个结构，如何使用这个结构，以及何时使用这个结构。数据结构有逻辑上的数据结构和物理上的数据结构之分。逻辑上的数据结构反映成分数据之间的逻辑关系，而物理上的数据结构则反映成分数据在计算机内部的存储安排。数据结构是信息的一种组织方式，其目的是提高算法的效率，它通常与一组算法的集合相对应，通过这组算法集合可以对数据结构中的数据进行某种操作。

在开始学习数据结构之前，需要弄清楚什么是数据结构，它研究什么，采用什么方法学习数据结构，相对应的算法集合如何评价，数据结构的描述规范等。

本章学习要点

- 数据结构的基本概念。
- 抽象数据类型。
- 算法描述。
- 算法分析。
- 数据结构的 C 语言表示。

1.1 数据结构的基本概念

本节我们将给出数据结构的一些基本概念，在这之前先看几个数据结构的实例。如表 1-1 所示的学生选课表中，学号、课程名、教师、成绩都是数据项(item)，是有独立含义的最小单位，其取值范围是数据域；一行称为一条记录(record)。我们可以用 C 语言或者其他语言描述这些记录，将其定义为结构，那么整个选课表就是一个线性表结构。

表 1-1　学生选课表

学　号	课 程 名	教　师	成　绩
101	操作系统	卢春燕	90
……	……	……	……

如图 1-1 所示的某报刊订阅管理系统结构图中，登录、用户、管理员、录入、查询等都是结点；登录和管理员、管理员和查询之间是层次关系或祖先后裔关系。这个订阅管理系统结构图就是一个非线性结构，我们称之为树。

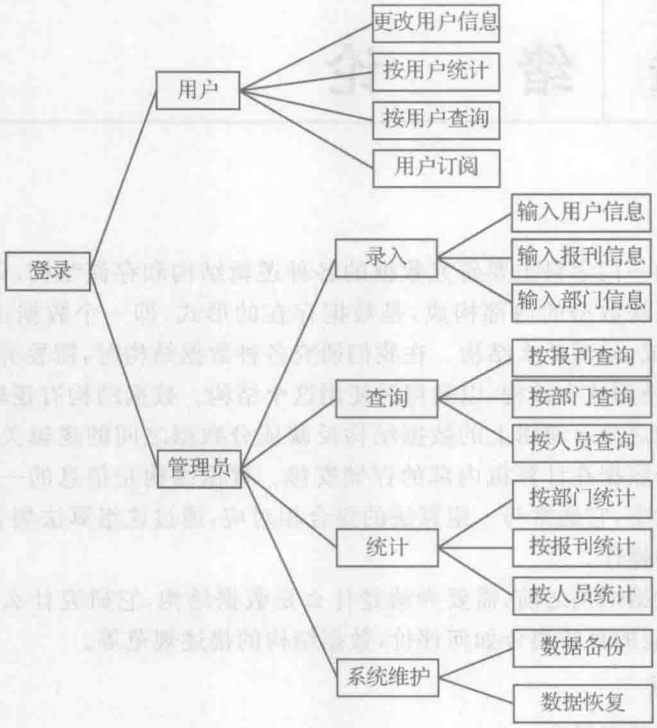


图 1-1　某报刊订阅管理系统结构图

1)数据结构中的一些基本概念

下面，我们给出数据结构中一些基本概念的定义。

(1)数据。数据(data)是对客观事物的描述，是能被输入到计算机中进行加工处理的各种符号集合，如数值、字符、声音、图像、视频等。对计算机科学而言，数据的含义非常宽泛。简而言之，数据就是计算机化的信息。

(2)数据元素。数据元素(data element)是构成数据的基本单位，在计算机程序中通常被作为一个整体进行考虑和处理。如表 1-1 中所示的选课记录，图 1-1 中所示的树都被称为数据元素。

(3)数据项。数据项(date item)是数据不可分割的最小单位，如学号、课程名、成绩等。一个数据元素可由一个或若干个数据项组成。例如，一个学生的选课信息作为一个数据元素，该数据元素包括学号、课程名、成绩等数据项。

(4)数据对象。数据对象(data object)是由性质相同的数据元素组成的集合，是数据的

一个子集。例如，由整数组成的数据集合 $D=\{0,\pm1,\pm2,\cdots\}$，由 26 个字母组成的集合 $C=\{A,B,\cdots,Z\}$，其中，D 是无限集，C 是有限集。表 1-1 所示的选课表也是一个数据对象。

(5)数据结构。数据结构(data structure)是指集合中数据元素相互之间存在一种或多种特定关系，由一组数据对象及其数据成员之间的联系组成，用以表述数据的组织形式。例如，表结构(表 1-1 所示的学生选课表)、树结构(图 1-1 所示的某报刊订阅管理系统结构图)。

数据结构的形式化为一个二元组，记为：

$$\text{Data\_Struture}=(D,R) \tag{1-1}$$

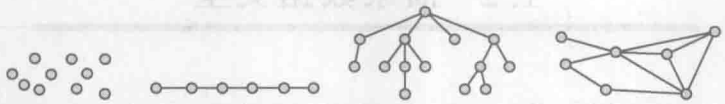
其中，D 为数据对象的有限集，R 是 D 上关系的有限集。

(6)数据类型。数据类型(data type)是一个值的集合以及定义在这个集合上的一组操作的总称。数据类型显式或隐式地定义了取值范围，以及在该值上允许进行的一组运算。例如，高级程序语言中的数据类型就是已经实现的数据结构的实例，分为非结构类型和结构类型两类，非结构类型如 C 语言中的整型、实型、字符型、枚举类型、指针类型等，结构类型如结构体和共用体。数据类型是程序语言中允许的变量种类，是已经实现的数据结构。例如，C 语言中的整型可能的取值范围是 $-32\ 768\sim+32\ 767$，可允许的操作集合为加、减、乘、除、乘方、取模。

2)数据结构的内容

数据结构包括逻辑结构、物理结构和操作集合。

(1)逻辑结构。数据对象内各元素之间的逻辑关系称为逻辑结构，其形式化描述见公式(1-1)所示。根据这些逻辑关系的性质划分，通常有 4 类基本结构，即集合结构、线性结构、树形结构和图形结构，如图 1-2 所示。



(a) 集合结构 (b) 线性结构 (c) 树形结构 (d) 图形结构

图 1-2 4 类基本结构

在这些数据对象内，元素之间的关系存在如下一些特点：

①集合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。

②线性结构中的数据元素之间存在一对一的关系。

③树形结构中的数据元素之间存在一对多的关系。

④图形结构中的数据元素之间存在多对多的关系。

因此，数据的逻辑结构可概括如下：

逻辑结构 { 线性结构——所有元素按次序排列在一个序列中
　　　　 { 非线性结构——每个元素可以与若干个其他元素关联

(2)物理结构。逻辑结构在计算机中的实现称为数据的物理结构，又称为存储结构。物理结构是逻辑结构和数据的物理映像，逻辑结构是数据结构的模型抽象，两者共同确立了数据对象中元素之间的结构关系。物理结构的形式化描述如下：

计算机中要存入 D，建立从 D 到存储空间 S 的映像 M，$D\rightarrow S$，即对每一个 $d(d\in D)$ 都有唯一的 $z\in S$，使得 $M(D)=Z$，且映像必须包含关系 R。

顺序存储结构和链式存储结构是最主要的两种存储方式。

①顺序存储结构。顺序存储结构是指用数据元素在存储器中的相对位置来表示元素之间的逻辑关系。

例如,令 y 的存储位置和 x 的存储位置之间差一个常量 C,C 是一个隐含值,整个存储结构中只含数据元素本身的信息,如图 1-3(a)所示。

②链式存储结构。链式存储结构是指用数据元素存储地址的指针来表示数据元素之间的逻辑关系。

例如,以指针表示后继关系,需要用一个和 x 在一起的附加信息指示 y 的存储位置,如图 1-3(b)所示。

(a) 顺序存储结构　　(b) 链式存储结构

图 1-3　存储结构示意图

(3)操作集合。建立数据结构的目的是在计算机中实现对数据的操作。例如,对数据的增加、删除、修改、查找、排序等操作。

综上所述,按某种逻辑模型组织起来的一批数据通过映像将它们存放在计算机的存储器中,并给这些数据定义操作的集合,称为数据结构。

1.2　抽象数据类型

抽象是人类认识世界的一种方式,是指从同类型的众多事物中舍弃个别的、非本质的属性和行为,而抽取共同的、本质的属性和行为的过程。例如,要表示空间上的一个四边形,就要对其进行抽象,四边形有正方形、长方形、菱形以及任意的不规则四边形,只要是由四条边组成的平面上的封闭图形就是四边形。它们的边长可能各不相同;各边的颜色也可能不一样;它们的形状也可能各异,如可以是凹的,也可以是凸的;甚至各图形在空间上的位置也各不相同。要表示这样的图形,可用四边形有序的 4 个顶点和边来描述,由这 4 个值及它们的先后顺序就可以描述四边形在某时刻的形状及在空间上的位置,借助类似的抽象方法我们还可以描述更为复杂的树形结构和网状结构。数据结构就是使用这种抽象的概念来描述各类程序中数据的组织形式。

通过对各种数据的抽象,C 语言提供了整型、实型、字符、指针等多种数据类型,利用这些数据类型定义更高级的数据抽象,可以构造出像栈、队列、树、图等复杂的抽象数据类型。

抽象数据类型(abstract data type,ADT)是一组数据对象以及其中各元素间的结构关系和其上的一组操作的集合。ADT 可以形式化地用三元组表示为:

```
ADT 抽象数据类型名{
    数据对象 D:{对象定义}
```

数据关系 R:{关系定义}

基本操作 P:{操作定义}

}ADT 抽象数据类型名

我们熟知的数据类型与抽象数据类型从本质上看是同一个概念。例如,字符类型是定义好的字符集合,每个字符用 ASCII 码表示,并且预定义了相关的操作。当在程序设计中使用字符时,并不需要了解字符数据在计算机内如何表示,也不需要了解对字符的操作是如何实现的,只需要掌握如何使用字符的操作即可。字符的具体存储方式和操作过程如何实现与字符的使用无关。而 ADT 定义的范围更为广泛,除了在不同的语言中已经定义并实现的数据类型外,还包括设计软件系统时用户自定义的复杂数据类型。

ADT 包括定义和实现两方面,其中定义是独立于实现的,定义仅需要给出一个 ADT 的逻辑特性,不需要考虑如何在计算机中实现。

下面通过一个例子来理解数据抽象和抽象数据类型的概念。

【例 1-1】 抽象数据类型整数的定义。

完整的定义包括两个部分:数据对象和操作。其中,对象定义为整数有序子序列,从 0 开始,到计算机上的最大整数结束,不涉及整数存储。符号 x、y 表示自然数集上的两个元素,True 和 False 是布尔集上的两个元素。定义在整数集上的操作包括加、减、相等以及小于等。定义的操作如表 1-2 所示。

表 1-2 ADT 整数

数据对象 / 操作集合	x,y,最大整数		
	实现功能	输入参数	返回值
Zero()	无	0	类似构造函数,没有参数
Is_Zero(x)	x	True/False	如果 x 值为 0,则返回 False,否则返回 True
Add(x,y)	x,y	整数	自然数求和
Equal(x,y)	x,y	True/False	自然数求相等
Subtract(x,y)	x,y	整数	自然数求差
Successor(x)	x	整数	累加器,返回 x 的下一个自然数。如果不存在,即 x 已经是机器的最大整数,那么返回机器最大整数

表 1-2 说明了 ADT 定义的一般形式。我们在具体数据结构的 ADT 定义中,一般不提供类似 C 函数的操作定义。从应用的角度出发,ADT 定义的实质在于避开具体的实现细节,即在一般情况下,用自然语言的形式来解释操作的意义,用户可以选择不同的语言和机器完成物理实现。因此,抽象数据类型的定义仅仅取决于客观存在的逻辑属性,与采用何种语言以及计算机如何表示和实现无关,之所以使用 ADT,就是为实现软件的构件化和可重用性提供理论保证,进而提高软件生产率。

ADT 通常是指由用户定义且用以表示应用问题的数据模型,一般由基本的数据类型组成,并包括相关服务操作集合。本课程中将要学习的表、堆栈、队列、串、树、图等结构就是不同的抽象数据类型。

一般来说,数据类型的抽象程度越高,包含该抽象数据类型的软件复用程度就越高。

ADT定义了抽象数据类型需要包含哪些信息,并根据功能确定服务接口,使用者可以使用接口对该抽象数据类型进行操作。从应用的角度看,只要了解该抽象数据类型的规格说明,就可以利用其接口的服务来使用这个类型,而不必关心其物理实现,从而集中精力考虑如何解决实际问题。

ADT物理实现作为私有部分封装在其实现模块内,具体做法就是将描述数据对象状态的属性和数据对象固有的行为分别用数据结构和方法来加以描述,并将它们捆绑在一起形成一个可供外界访问的独立的逻辑单元,外界只能通过客体所提供的方法来对其间的数据结构加以访问,而不能直接存取。很明显,封装是实现信息隐藏的有效手段,它尽可能地隐蔽对象的内部细节,只保留有限的对外接口,使之与外部发生联系。封装保证了数据的安全性,提高了应用系统的可维护性,也有利于软件的移植与重用。

1.3 算法描述

Pascal之父、结构化程序设计的先驱Nicklaus Wirth最著名的一本书是《算法+数据结构=程序》,该书提纲挈领地表明了数据结构和算法是程序的两大组成部分,二者相辅相成,缺一不可。

问题(problem)是需要研究讨论并完成的任务或需求;算法(algorithm)是对特定问题求解步骤的一种描述,它是指令的有限序列,其中,每一条指令表示一个或多个操作;程序(program)是用具体语言对算法的实现。

【例1-2】 计算1-1/2+1/3-1/4+1/5-…+1/99-1/100。

算法1:自左至右逐项相加或相减。

算法2:将多项式写成两个多项式之差:(1+1/3+1/5+…+1/99)-(1/2+1/4+1/6+…+1/100)。

算法3:先通分,再运算。

算法是抽象的,可利用自然语言、类语言进行描述。自然语言或类语言简单明了但容易产生歧义;而程序必须是准确和严谨的,只能用程序设计语言来表示。

近年来,在计算机科学研究、嵌入式系统开发、教学和实践中,C语言的使用范围越来越广,已成为计算机专业与非计算机专业必修的高级程序设计语言。C语言类型丰富,执行效率高,本教材采用标准C语言作为描述算法的工具(假设读者在学习数据结构课程之前,都已了解和掌握了C语言)。

算法具备以下5个重要特性:

(1)有穷性。对于任意一组合法输入值,在执行有限步骤之后必须能结束,即算法中的每个步骤都能在有限时间内完成。

(2)确定性。在任意情况下所需执行的操作,在算法中都有明确的规定,使得算法的执行机构都能明确其含义及如何执行。并且在任何条件下,算法都只有一条执行路径。

(3)可行性。算法中的所有操作都必须足够精确,都可以通过已经实现的基本运算执行有限次实现。

(4)输入。算法具有若干个或0个输入。有些算法需要在执行过程中输入,而有些算法表面上可以没有输入,实际上已被嵌入算法之中。

(5)输出。它是一组与"输入"有确定关系的量值,是算法进行信息加工后得到的结果,这种确定关系即为算法的功能。

通常设计一个"好"的算法应考虑达到以下目标:

(1)正确性。首先,算法应当满足以特定的"规格说明"方式给出的需求。其次,对算法是否"正确"的理解可以有以下4个层次:

- 程序中不含语法错误。
- 程序对于几组输入数据能够得出满足要求的结果。
- 程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据能够得出满足要求的结果。
- 程序对于一切合法的输入数据都能得出满足要求的结果。

(2)可读性。要有利于人们对算法的理解;有利于程序的调试和维护、交流和移植。

(3)健壮性。当输入的数据非法时,算法应当恰当地作出反应或进行相应处理,返回一个表示错误或错误性质的值,以便在更高的抽象层次上进行处理,而不是产生莫名其妙的输出结果或者中断程序运行。

(4)高效率与低存储量需求。通常,高效率指的是算法执行时间要尽可能少;低存储量指的是算法执行过程中所需的最大存储空间要尽可能小,两者都与问题的规模有关。

1.4 算法分析

数据结构的优劣可以用实现的算法来衡量,对数据结构的分析就转化成了对算法的分析,一是要验证算法是否能正确解决问题,二是要对算法的效率作性能评价。性能评价分为算法分析和性能测量。算法分析是复杂性理论的核心内容,目的是获取与具体机器无关的时间和空间的估计。在计算机程序设计中,程序是否有效地利用存储空间,程序的执行时间是否可以接受,是对算法效率进行评价的核心标准。本节我们关注对算法的分析,即对于一个实际问题的解决,如何从若干个可行的算法中找出最有效的算法,就是以算法执行所需的机器时间和所占用的存储空间作为标准。

对一个算法的效率作出分析,就是要度量算法的执行时间,通常使用以下两种方法。

(1)事后统计法。不同的算法可通过统计实际执行时间来分辨优劣。但该方法存在缺点:一是必须先运行依据算法编制的程序;二是所得时间的统计量依赖于计算机的硬件、软件等环境因素,有时容易掩盖算法本身的优劣。

(2)事前分析估算法。求出算法的时间规模函数。

与算法执行时间相关的因素包括:

- 算法选择的策略。
- 待解决问题的规模。
- 编写程序的语言。

• 编译程序产生的机器代码的质量。

• 计算机执行指令的速度。

显然,对于相同的算法,使用不同的编译器、不同的程序语言或者在不同的机器上运行,效率均不相同。因此用绝对的时间单位衡量算法的效率是不合适的,还应考虑与计算机硬件、软件有关的因素。可以认为:一个特定算法的"运行工作量"的大小只依赖于问题的规模(通常用整数 n 表示),或者说,它是问题规模的函数,而问题规模 n 对于不同的问题其含义是不同的,对矩阵是阶数,对多项式运算是多项式项数,对图是顶点个数,对集合运算是集合中的元素个数。

1.4.1 时间复杂度

一个算法的执行时间大致上等于其编译时间和所有语句执行时间的总和,由于编译时间不依赖于问题的特征,所以对算法效率的分析主要集中在算法的执行时间上。

算法=控制结构(顺序、分支和循环)+基本操作

基本操作是指操作时间与问题规模无关的操作。例如,给变量赋值、比较两个数的大小、四则运算等,而 n 个加法操作就不是基本操作,因为它与输入规模有关。算法的时间取决于控制结构和基本操作两者的综合效果。为了便于比较同一问题的不同算法,通常的做法是,从算法中选取一种相对于所研究的问题(或算法类型)的基本操作,以该基本操作重复执行的次数作为算法的时间度量。

设 n 为求解问题的规模,基本操作执行次数的总和称为语句频度,记为 $f(n)$,算法中语句总的执行次数记为 $T(n)$。通常情况下,算法中基本操作重复执行的次数是问题规模 n 的某个函数。这里用 O 来表示 $T(n)$ 随 n 的变化情况所达到的数量级,算法的时间量度记为:

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

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

由于算法的时间复杂度考虑的只是问题规模 n 的增长率,所以在难以精确计算基本操作执行次数(或语句频度)的情况下,只需求出它关于 n 的增长率或阶即可。一般情况下,随着 n 的增大,$T(n)$ 增长较慢的算法为最优算法。

定义:若 $O(f(n))=O(g(n))$,则当且仅当存在正的常数 c 和 n_0,使得对所有的 n,当 $n\geqslant n_0$ 时,有 $f(n)\leqslant cg(n)$。

【例 1-3】 时间复杂度分析。

```
①++x;                          O(1)     常量阶
②for(i=1;i<=n;i++)
  {++x;}                       O(n)     线性阶
③for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
  {++x;}                       O(n²)    平方阶
```

常用的时间复杂度的阶有 7 个,其复杂度从小到大依次为:$O(1)$ 常数阶,$O(\log_2 n)$ 对数阶,$O(n)$ 线性阶,$O(n\log_2 n)$ 二维阶,$O(n^2)$ 平方阶,$O(n^3)$ 立方阶,$O(2^n)$ 指数阶。

由图 1-4 可知，随着问题规模 n 的增大，不同阶的时间复杂度增长快慢不一，因此，应尽可能选用多项式阶算法，而避免使用指数阶的算法。

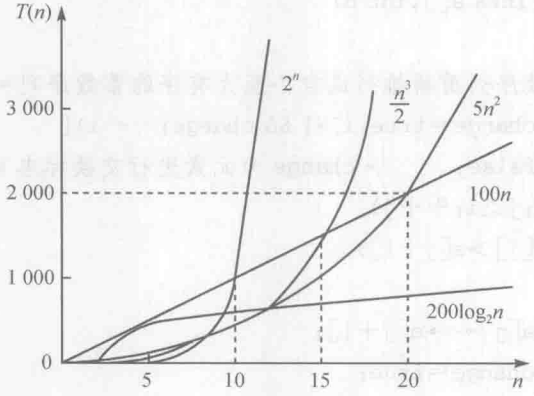


图 1-4　不同阶的时间复杂度示意图

【例 1-4】　两个矩阵相乘，判断其时间复杂度。

```
void mult(int a[],int b[],int & c[])
{
    /*以二维数组存储矩阵元素,c是a和b的乘积*/
    for(i=1;i<=n;++i)                          /*n次*/
        for(j=1;j<=n;++j){                     /*n²次*/
            c[i,j]=0;                          /*n²次*/
            for(k=1;k<=n;++k)                  /*n³次*/
                c[i,j]+=a[i,k]*b[k,j];         /*n³次*/
        }
}
```

算法的基本操作为乘法操作，语句频度为：$f(n)=n+n^2+n^2+n^3+n^3=2n^3+2n^2+n$。

时间复杂度为：$T(n)=O(f(n))=O(2n^3+2n^2+n)=O(n^3)$。

所以算法的时间代价是输入规模 n 的立方阶。

【例 1-5】　选择排序，判断其时间复杂度。

```
void select_sort(int& a[],int n)
{
    /*将a中整数序列重新排列成自小至大有序的整数序列*/
    for(i=0;i<n-1;++i){
        j=i;                                   /*选择第i个最小元素*/
        for(k=i+1;k<n;++k)
            if(a[k]<a[j])j=k;
        if(j!=i)a[j]←→a[i]
    }
}
```