普通高等教育“十三五”规划教材
21世纪全国高校应用人才培养信息技术类规划教材

数据结构与算法（C语言版）

李忠月　虞铭财　主编

非外借

北京大学出版社
PEKING UNIVERSITY PRESS

普通高等教育“十三五”规划教材
21 世纪全国高校应用人才培养信息技术类规划教材

数据结构与算法（C 语言版）

李忠月　虞铭财　主　编

北京大学出版社
PEKING UNIVERSITY PRESS

图书在版编目（CIP）数据

数据结构与算法：C语言版/李忠月，虞铭财主编. —北京：北京大学出版社，2019.3
21世纪全国高校应用人才培养信息技术类规划教材

ISBN 978-7-301-30284-2

Ⅰ. ①数…　Ⅱ. ①李…　②虞…　Ⅲ. ①数据结构–高等学校–教材　②算法分析–高等学校–教材　③C语言–程序设计–高等学校–教材　Ⅳ. ①TP311.12　②TP312.8

中国版本图书馆CIP数据核字（2019）第033947号

书　　名	数据结构与算法（C语言版） SHUJU JIEGOU YU SUANFA（C YUYAN BAN）
著作责任者	李忠月　虞铭财　主编
策划编辑	温丹丹
责任编辑	温丹丹
标准书号	ISBN 978-7-301-30284-2
出版发行	北京大学出版社
地　　址	北京市海淀区成府路205号　100871
网　　址	http://www.pup.cn　　新浪微博：@北京大学出版社
电子信箱	zyjy@pup.cn
电　　话	邮购部010-62752015　发行部010-62750672　编辑部010-62756923
印刷者	北京富生印刷厂
经销者	新华书店
	787毫米×1092毫米　16开本　15.75印张　383千字 2019年3月第1版　2019年3月第1次印刷
定　　价	45.00元

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究
举报电话：010-62752024　电子信箱：fd@pup.pku.edu.cn
图书如有印装质量问题，请与出版部联系，电话：010-62756370

此为试读，需要完整PDF请访问：www.ertongbook.com

前言

数据结构是计算机专业及相关专业的重要基础课。它所讨论的知识内容和提倡的技术方法，无论是对进一步学习计算机相关领域的其他课程，还是对从事大型信息工程的开发，都有着关键的作用。

本书采用C语言作为数据结构与算法的描述语言，在对数据的存储结构与算法进行描述时，尽量考虑C语言的特色，同时兼顾数据结构和算法的可读性，便于读者将书中的数据结构与算法转换成C语言程序。

本书有如下特色。

（1）算法步骤和算法描述，以非常适合读者学习的方式呈现。本书的作者一直处在教学的第一线，本书是作者所在教学团队多年教学经验的积累、教学改革的成果。

（2）算法讲解细致。本书对算法思想进行了详细的阐述，将用文字描述的算法步骤与用C语言表述的算法描述一一对应，能够提高读者将算法转化为程序的能力以及算法设计与算法实现的能力。

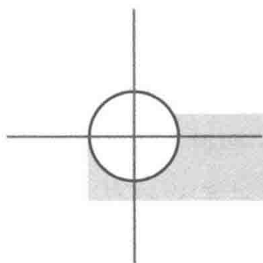
（3）本书图文并茂。一图值千言，用上千个字描述不明白的事情，很可能一张图就能解释清楚。本书基本上做到难理解部分的讲解都有相关的图示，有的内容通过多图逐步分解剖析。

（4）为满足读者对在线开放学习的需求，本书对一些重要的知识点、重要的算法和难懂的算法，都有配套的微课，读者可以通过扫描二维码观看。通过这种方式，读者能重复学习，做到攻克重点、难点，不留学习的死角。

由于作者水平有限，对于书中难免存在的疏漏和不妥之处，敬请读者批评指正。

作者

2019年2月



目 录

第 1 章 概论 …… 1

1.1 引言 …… 1

1.2 基本概念 …… 3

1.3 逻辑结构与存储结构 …… 4

1.4 抽象数据类型 …… 5

1.5 算法 …… 6

本章小结 …… 14

习题 …… 15

第 2 章 线性表 …… 19

2.1 线性表的定义 …… 19

2.2 线性表的顺序存储 …… 20

2.3 线性表的链式存储 …… 25

2.4 单链表 …… 25

2.5 循环单链表 …… 34

2.6 双链表 …… 34

2.7 顺序表与链表的比较 …… 35

2.8 应用实例:一元多项式 …… 35

本章小结 …… 39

习题 …… 39

第 3 章 栈和队列 …… 44

3.1 栈 …… 44

3.2 一般顺序栈 …… 45

3.3 双端栈 …… 47

3.4 一般链栈 …… 48

3.5 多链栈 …… 49

3.6 应用实例:栈的应用 …… 50

3.7 队列 …… 55

3.8 循环队列 …… 56

3.9 链队列 …… 59

本章小结 …… 61

习题 …… 61

第 4 章 串 …… 65

4.1 串的定义 …… 65

4.2 串的存储结构 …… 66

4.3 串的模式匹配 …… 66
本章小结 …… 71
习题 …… 71
第5章 数组和广义表 …… 73
5.1 数组 …… 73
5.2 特殊矩阵的压缩存储 …… 74
5.3 稀疏矩阵的压缩存储 …… 75
5.4 广义表 …… 76
本章小结 …… 76
习题 …… 77
第6章 树和二叉树 …… 79
6.1 树 …… 79
6.2 二叉树 …… 83
6.3 二叉树的遍历 …… 88
6.4 二叉树遍历的非递归算法 …… 91
6.5 二叉树遍历算法的应用 …… 93
6.6 创建二叉树 …… 95
6.7 树、森林与二叉树 …… 100
6.8 哈夫曼树 …… 103
6.9 哈夫曼编码 …… 108
本章小结 …… 111
习题 …… 111
第7章 图 …… 118
7.1 图的基本概念 …… 118
7.2 图的存储结构 …… 122
7.3 图的遍历 …… 130
7.4 图的最小生成树 …… 133
7.5 最短路径 …… 142
7.6 有向无环图及其应用 …… 149
本章小结 …… 157
习题 …… 157
第8章 查找 …… 166
8.1 查找的基本概念 …… 166
8.2 查找的基本方法 …… 167
8.3 顺序查找 …… 167
8.4 折半查找 …… 168
8.5 分块查找 …… 170
8.6 二叉排序树 …… 172
8.7 平衡二叉树 …… 179
8.8 散列查找 …… 183
本章小结 …… 193

习题 …… 193
第9章 排序 …… 199
9.1 排序的基本概念与分类 …… 199
9.2 冒泡排序 …… 201
9.3 快速排序 …… 204
9.4 简单选择排序 …… 209
9.5 堆排序 …… 211
9.6 直接插入排序 …… 217
9.7 希尔排序 …… 219
9.8 归并排序 …… 221
9.9 基数排序 …… 224
9.10 排序算法的比较 …… 227
本章小结 …… 228
习题 …… 228
附录A 测试函数的运行时间 …… 233
附录B 并查集 …… 234
附录C C++语言中stack的用法 …… 235
附录D C++语言中queue的用法 …… 236
参考文献 …… 237

第1章 概　论

1.1 引　言

1968年，美国的Donald E. Knuth（高德纳）教授在其所写的《计算机程序设计艺术卷1：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，开创了数据结构的课程体系。同年，数据结构作为一门独立的课程，在计算机科学的学位课程中开始出现。

之后，20世纪70年代初，出现了大型程序，软件也开始相对独立，结构化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”，认为程序设计的实质是“对确定的问题选择一种好的结构”+“设计一种好的算法”。可见，数据结构在程序设计中占据了重要的地位。

什么是数据结构？事实上，这个问题在计算机科学界至今没有公认的、标准的定义。

这里先尝试解决下面几个简单的问题，在解决问题的过程中，或许读者可以得到对于数据结构的理解。

【问题1】书店往往是书的海洋，店主应该如何摆放书店里的书？

方法1：随便摆放。这种方法存放书非常方便，但是顾客找书却非常麻烦。最坏的情况是书店里根本没有顾客要找的书，顾客却需要找遍书店中的每一本书，最后才发现没有要找的书。

方法2：按照书名的拼音字母顺序摆放。这种方法使得查找方便了一些，但是可能会使新书的插入比较麻烦。如果新书是以A开头的，为了给新书腾出空间，要把很多书都向后挪动。

方法3：把书架划分成几块区域，每块区域指定摆放某种类别的图书，在每种类别内，按照书名的拼音字母顺序摆放。这种方法与方法2相比，无论是查找还是插入图书，工作量都减少了很多。

所以，解决问题方法的效率与数据的组织方式有关。

【问题2】按照顺序输出从1到N的全部正整数。

方法1：用循环算法实现。

```
void print(int N)
{
    int i;
    for(i =1;i < =N;i ++){
        printf("%d\n",i);
    }
}
```

方法2:用递归算法实现。

```
void print(int N)
{
    if(N!=0){
        printN(N-1);
        printf("%d\n",N);
    }
}
```

上面两种方法看上去似乎都能完成任务。然而,上机测试后发现,当 N 很大时,用递归算法实现的程序会拒绝运行,而用循环算法实现的程序仍然正常运行。

所以,解决问题方法的效率与空间的利用效率有关。

【问题3】多项式求和。多项式的标准表达式可以写为 $f(x)=a_0+a_1x+\cdots+a_{x-1}x^{n-1}$。现给定一个多项式的阶数 n,并将全部系数存放在数组中。请编写程序计算这个多项式在给定点 x 处的值。

方法1:直接算法。

```
double fun(double a[],double x,int n)
{
    int i;
    double p=a[0];
    for(i=1;i<=n;i++){
      p=p+a[i]*pow(x,i);
    }
    return p;
}
```

方法2:秦九韶算法。

早在800多年前,中国南宋时期的数学家秦九韶通过不断提取公因式 x 来减少乘法的运算次数,把多项式改写为 $f(x)=a_0+x(a_1+x(\cdots(a_{n-1}+x(a_n))))$。

```
double fun(double a[],double x,int n)
{
    int i;
    double p=a[n];
    for(i=n;i>0;i--){
        p=a[i-1]+x*p;
    }
    return p;
}
```

直接算法执行 n 次语句"p=p+a[i]*pow(x,i);",每次涉及 i 次乘法和1次加法运算,于是全部计算涉及 $(1+2+\cdots+n)=(n^2+n)/2$ 次乘法和 n 次加法。

秦九韶算法执行 n 次语句"p=a[i-1]+x*p;",每次涉及1次乘法和1次加法运算,于是全部计算涉及 n 次乘法和 n 次加法。

那么,秦九韶算法究竟比简单的直接算法快了多少呢?通过上机测试可以发现,秦九韶算法的计算速度明显比直接算法快了一个数量级。

所以,解决问题方法的效率与算法的巧妙程度有关。

本章将要介绍的是有关数据组织、算法设计、时间和空间效率的概念,以及通用分析方法,这些都是后续所有数据结构及其相关算法的基础。

1.2 基本概念

1.2.1 数据

数据(Data)是描述客观事物的符号,是计算机中可以操作的对象,而且能被计算机识别并输入给计算机处理的符号集合。数据不仅仅包括整型、实型等数值类型,还包括字符及声音、图像、视频等非数值类型。

也就是说,这里所说的数据,其实就是符号,而且这些符号必须具备两个前提:可以输入到计算机中和能被计算机程序处理。

1.2.2 数据元素

数据元素(Data Element)是组成数据的、有一定意义的基本单位,在计算机中通常作为整体处理。

数据元素在线性表中称为元素,在树中称为结点,在图中称为顶点,在查找和排序中称为记录。

1.2.3 数据项

一个数据元素可由若干个**数据项**(Data Item)组成。例如,人有姓名、年龄、性别、出生地址、联系电话等数据项。

数据项是数据不可分割的最小单位。但真正讨论问题时,数据元素才是数据结构中建立数据模型的着眼点。

1.2.4 数据对象

数据对象(Data Object)是性质相同的数据元素的集合,是数据的子集。

既然数据对象是数据的子集,在实际应用中,处理的数据元素通常具有相同性质,所以在不产生混淆的情况下,人们都将数据对象简称为数据。

例如,在表1.1所示的人员信息管理中,整张表格就是数据;每一个人员的信息是数据元素,这里有3个人员的信息;学号、姓名、性别、籍贯、出生年月、家庭住址,这些都是数据项。

表1.1 人员信息管理

学号	姓名	性别	籍贯	出生年月	家庭住址
101	赵红玲	女	河北	1983.11	北京
102	李勇	男	安徽	1980.3	杭州
103	朱一帆	男	江西	1977.7	上海

1.2.5 数据结构

结构,简单的理解就是关系。严格来说,结构是指各个组成部分相互搭配和排列的方式。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,将这些关系称为**结构**。那数据结构是什么?

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。

关于数据对象在计算机中的组织方式,还包含两个概念:一是数据对象的逻辑结构;二是数据对象在计算机中的存储结构(也称为物理结构)。

此为试读,需要完整PDF请访问: www.ertongbook.com

1.3 逻辑结构与存储结构

1.3.1 逻辑结构

数据的逻辑结构是从逻辑关系上描述数据的,它与数据的存储无关,是独立于计算机的。因此,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。

数据的逻辑结构有两个要素:一是数据元素,二是关系。数据元素的含义如前所述,关系是指数据元素之间的逻辑关系。根据数据元素之间关系的不同特性,通常可以分为以下四类基本结构。

1. 集合结构

集合结构中的数据元素除了同属于一个集合外,它们之间没有其他关系。各个数据元素之间是“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,如图 1.1 所示。

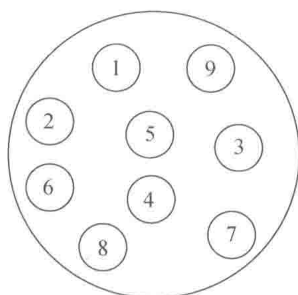


图 1.1 数据结构中的集合关系

2. 线性结构

线性结构中的数据元素之间是一对一的关系,如图 1.2 所示。

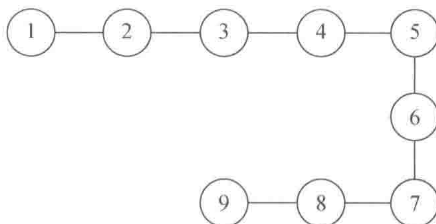


图 1.2 线性结构中数据元素的对应关系

3. 树结构

树结构中的数据元素之间是一对多的层次关系,如图 1.3 所示。

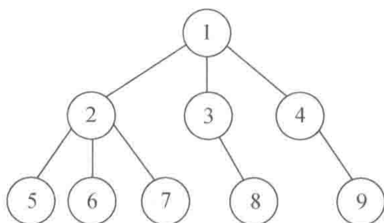


图 1.3 树结构中数据元素的一对多关系

4. 图结构

图结构中的数据元素之间是多对多的关系，如图1.4所示。

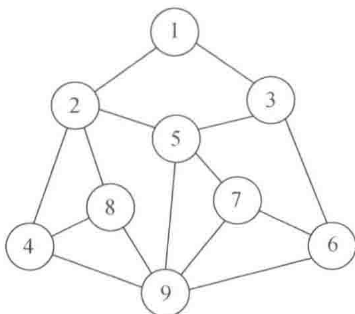


图1.4 图结构中数据元素的多对多关系

1.3.2 存储结构

存储结构是指数据的逻辑结构在计算机中的存储形式。

如何存储数据元素之间的逻辑关系，是实现存储结构的重点和难点。数据元素的存储结构形式有两种：顺序存储结构和链式存储结构。

1. 顺序存储结构

顺序存储结构是把数据元素存放在地址连续的存储单元中，其数据之间的逻辑关系和存储关系是一致的。计算机语言中的数组就是这样的顺序存储结构。

2. 链式存储结构

链式存储结构是把数据元素存放在任意的存储单元中，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。

1.4 抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是一种对数据类型的描述，这种描述是抽象的。

首先，数据类型描述两个方面的内容：一是数据对象集，二是与数据集相关联的操作集。抽象的意思是指描述数据类型的方法不依赖于具体实现，即数据对象集和操作集的描述与存储数据的计算机无关、与数据存储的物理结构无关、与实现操作的算法和编程语言均无关。

也就是说，抽象数据类型只描述数据对象集和相关操作集“是什么”，并不涉及“如何做”的问题。

为了便于在之后的讲解中对抽象数据类型进行规范的描述，下面给出了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:
数据关系:
    数据元素之间的逻辑关系的定义
基本操作:
        操作1
        操作2
```

```
    ……
    操作 n
}ADT 抽象数据类型名
```

在后面的章节中，每介绍一种数据结构时，我们首先会用抽象数据类型来描述这个结构，以方便大家理解。

例 1.1 在日常数据处理中经常会碰到的一个问题是，需要对一组数据进行基本的统计分析。例如，分析一个班级中学生在某门学科中的平均成绩、最高成绩、最低成绩、中位数、标准差等，或者统计家庭每年或每个月的开支情况，或者统计生产线上各位员工计件任务的完成情况等。

如果为每个具体应用都编写一个程序，显然不是一个很好的方法，因为这些程序具有很大的相似性。数据结构的处理方法是从这些具体应用中抽象出其共性的数据组织，进而采用程序设计语言实现相应的数据存储与操作。因此，对于上述例子可以抽象出一种针对基本统计要求的数据类型。

```
ADT 数据集合的基本统计{
数据对象:
    S = {x1, x2, …, xn}
数据关系:
    数据元素之间是集合关系
基本操作:
    (1)求 S 中元素的平均值。
    (2)求 S 中元素的最大值。
    (3)求 S 中元素的最小值。
    (4)求 S 中元素的中位数。这里的中位数是指将 S 中的元素按从大到小的顺序依次排列，
       处在中间位置([N/2]，大于等于 N/2 的最小整数)的那个元素。
}ADT 数据集合的基本统计
```

可以看到，针对上述数据抽象方式的具体程序可以用来求解不同领域的基本统计问题，这样既能保证程序设计的逻辑清晰，又在很大程度上实现了代码的重用。如何利用程序设计语言实现上述抽象数据类型？应用程序设计语言实现抽象数据类型，首先，必须考虑如何存储数据，即集合 *S* 中的数据在程序设计语言中怎样存储；其次，必须考虑如何实现操作，即在确定数据存储方式的基础上，如何实现相应的操作（如求平均值、最大值等函数）。

1.5 算　法

算法（Algorithm）是描述解决问题的方法。如今普遍认可的对算法的定义是：算法是解决特定问题求解步骤的描述，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。

著名的计算机科学家图灵奖获得者 N. Wirth（沃思）教授给出了一个著名的公式：

算法 + 数据结构 = 程序

这说明数据结构和算法是程序的两大要素，二者相辅相成，缺一不可。

数据结构与算法之间存在着本质联系，在某一类型数据结构上，总要涉及其上施加的运

算,而只有通过对运算的研究,才能清楚地理解数据结构的定义和作用;在涉及运算时,总要与该算法所处理的对象和结果数据等联系起来进行考虑。

"数据结构"课程中会遇到大量的算法问题,因为算法联系着数据在计算机过程中的组织方式。为了描述要实现的某种操作,常常需要设计算法,因而算法是研究数据结构的重要途径。

1.5.1 算法的特性

算法接收输入(有些情况下不需要输入),产生输出,并在有限的步骤之后终止。算法的每一条指令必须有充分明确的目标,不可以有歧义;必须在计算机能处理的范围之内;且其描述应不依赖于任何一种计算机语言及具体的实现手段。当然,用某一种计算机语言进行描述往往会使算法更容易理解,故本书采用C语言作为描述算法的工具。

概括来说,算法具有如下5个基本特性。

(1) 输入:一个算法有零个或多个输入。

(2) 输出:一个算法至少有一个或多个输出。

(3) 有穷性:一个算法必须在有限的步骤之内正常结束,不能形成无穷循环。

(4) 确定性:算法中的每个步骤必须有确定含义,无二义性。

(5) 可行性:原则上,一个算法能精确进行,操作可通过已实现的基本运算执行有限次来完成。

注意 算法不是程序。一个明显的区别是,程序可以无限运行(如操作系统),但算法必须在有限的步骤之后终止。算法与程序的主要不同之处,还在于算法比程序"抽象",强调表现"做什么",而忽略细节性的"怎样做"。

1.5.2 算法设计的要求

算法的设计是一门艺术。同一个问题,可以有多种解决问题的算法,但不同的算法往往有天壤之别。好的算法,应该具有正确性、可读性、健壮性、高效性4个特性。

1. 正确性

算法的正确性是指算法至少应该具有输入、输出和加工处理无歧义性,能正确反映问题的需要,能够得到问题的正确答案。

2. 可读性

算法设计的另一目的是为了便于阅读、理解和交流。可读性高有助于人们理解算法,可读性是算法好坏的重要标志。

3. 健壮性

健壮性是指一个算法对不合理数据输入的反应能力和处理能力,也称为容错性。例如,当输入的时间或距离是负数时,算法能做出相关处理,而不是产生异常或莫名其妙的结果。

4. 高效性

高效性包括时间和空间两个方面。时间高效是指算法设计合理,执行效率高,可以用时间复杂度来度量;空间高效是指算法占用存储容量合理,可以用空间复杂度来度量。时间复杂度和空间复杂度是衡量算法的两个主要指标。

1.5.3 算法效率的度量方法

分析算法效率的目的是看算法实际是否可行,并在同一问题存在多个算法时,可进行时

间和空间性能上的比较,以便从中挑选出较优的算法。衡量算法效率的方法主要有两种:事后统计方法和事前分析估算方法。

1. 事后统计方法

事后统计方法主要是通过设计好的测试程序和数据,利用计算机计时器对不同算法编制的程序的运行时间进行比较,从而确定算法效率的高低。但这种方法有很大缺陷的。

(1) 必须依据算法事先编制好的程序,这通常需要花费大量的时间和精力。

(2) 时间的比较依赖计算机硬件和软件等环境因素,有时会掩盖算法本身的优劣。

(3) 算法的测试数据设计困难,并且程序的运行时间往往还与测试数据的规模有很大关系,效率高的算法在小的测试数据面前往往得不到体现。

基于事后统计方法的缺陷,算法效率的度量不予采纳该方法。

2. 事前分析估算方法

事前分析估算方法是指,在计算机程序编制前,依据统计方法对算法进行估算。

一个用高级语言编写的程序在计算机上运行时所消耗的时间取决于下列因素。

(1) 算法采用的策略、方法。

(2) 编译产生的代码质量。

(3) 问题的输入规模。

(4) 机器执行指令的速度。

第(1)条当然是算法好坏的根本,第(2)条要由软件来支持,第(4)条要看硬件性能。也就是说,抛开这些与计算机硬件、软件有关的因素,一个程序的运行时间,依赖于算法的好坏和问题的输入规模。所谓问题的输入规模,是指输入量的多少。

1.5.4 函数的渐近增长

一个算法的执行时间大致上等于其所有语句执行时间的总和。语句的执行时间是指,该条语句的执行次数(又称为语句频度)与执行一次所需时间的乘积。

事实上,精确地比较程序执行的次数是没有意义的,因为每个步骤的执行时间可能不同。例如,递归调用的“1 次”,实际上涉及对系统堆栈的很多处理步骤,比循环中的“1 次”计算慢得多。所以在比较算法优劣时,人们只考虑宏观渐近性质,即当输入规模 n“充分大”时,观察不同算法复杂度的“增长趋势”,以判断哪种算法的效率更高。为此,引入下面 4 个数学定义。

定义 1.1 如果存在正常数 c 和 n_0,使得当 $n \geqslant n_0$ 时 $T(n) \leqslant c \cdot f(n)$,则记为 $T(n) = O(f(n))$。

定义 1.2 如果存在正常数 c 和 n_0,使得当 $n \geqslant n_0$ 时 $T(n) \geqslant c \cdot g(n)$,则记为 $T(n) = \Omega(g(n))$。

定义 1.3 $T(n) = \Theta(h(n))$,当且仅当 $T(n) = O(h(n))$ 和 $T(n) = \Omega(h(n))$。

定义 1.4 如果 $T(n) = O(p(n))$ 且 $T(n) \neq \Theta(p(n))$,则记为 $T(n) = o(p(n))$。

通常,可以使用传统的不等式来计算增长率,第一个定义是说,$T(n)$的增长率小于等于

$f(n)$的增长率;第二个定义 $T(n)=\Omega(g(n))$ 是说,$T(n)$的增长率大于等于$g(n)$的增长率;第三个定义 $T(n)=\Theta(h(n))$ 是说,$T(n)$的增长率等于 $h(n)$ 的增长率;第四个定义 $T(n)=o(p(n))$ 说的则是,$T(n)$的增长率小于 $p(n)$ 的增长率,它不同于 O,因为 O 包含增长率相同这种可能性。

函数的渐近增长:给定两个函数 $f(n)$ 和 $g(n)$,如果存在一个整数 N,使得对于所有的 $n>N$,$f(n)$总是比 $g(n)$大,那么可以说$f(n)$的增长渐近快于 $g(n)$。

下面来看第一个例子,假设两个算法的输入规模都是 n,算法 A 要做 $2n+3$ 次操作,算法 B 要做 $3n+1$ 次操作。那么算法 A 和算法 B 哪个更好?

准确来说,答案是无法确定的,算法 A 和算法 B 的计算结果如表 1.2 所示。

表 1.2 算法 A 和算法 B 的计算结果

次　数	算法 A($2n+3$)	算法 B($3n+1$)	算法 A′($2n$)	算法 B′($3n$)
$n=1$	5	4	2	3
$n=2$	7	7	4	6
$n=3$	9	10	6	9
$n=10$	23	31	20	30
$n=100$	203	301	200	300

当 $n=1$ 时,算法 A 的效率不如算法 B 的(因为算法 A 的执行次数比算法 B 的要多一次);当 $n=2$ 时,两者的效率相同;当 $n>2$ 时,算法 A 的效率就开始优于算法 B 的,而且随着 n 的增加,算法 A 的效率越来越优于算法 B 的(即算法 A 执行的次数比算法 B 要少)。于是可以得出结论,算法 A 总体上要好过算法 B。

从表 1.2 中可以发现,随着 n 的增大,不管算法后面是"+3"还是"+1",其实是不会影响算法最终的变化的(如表 1.2 所示的算法 A′与算法 B′的计算结果)。所以,可以忽略这些加法常数。

下面来看第二个例子,算法 C 是 $4n+8$,算法 D 是 $2n^2+1$,它们的计算结果如表 1.3 所示。

表 1.3 算法 C 和算法 D 的计算结果

次　数	算法 C($4n+8$)	算法 D($2n^2+1$)	算法 C′(n)	算法 D′(n^2)
$n=1$	12	3	1	1
$n=2$	16	9	2	4
$n=3$	20	19	3	9
$n=10$	48	201	10	100
$n=100$	408	20 001	100	10 000
$n=1\,000$	4 008	2 000 001	1 000	1 000 000

从表 1.3 中可以看出,当 $n\leq3$ 时,算法 C 的效率要差于算法 D 的(因为算法 C 的次数比较多);但当 $n>3$ 后,算法 C 的效率就越来越优于算法 D 的。甚至去掉相加的常数和与 n 相乘的常数,也会发现算法 C′的效率优于算法 D′的,算法 C′的次数随着 n 的增长,运行次数还是远小于算法 D′的。也就是说,算法中与最高次项相乘的常数并不重要。

下面再来看第三个例子。算法 E 是 $2n^2+3n+1$，算法 F 是 $2n^3+3n+1$，它们的计算结果如表 1.4 所示。

表 1.4　算法 E 和算法 F 的计算结果

次　数	算法 E($2n^2+3n+1$)	算法 F($2n^3+3n+1$)	算法 E′(n^2)	算法 F′(n^3)
$n=1$	6	6	1	1
$n=2$	15	23	4	8
$n=3$	28	64	9	27
$n=10$	231	2 031	100	1 000
$n=100$	20 301	2 000 301	10 000	1 000 000

当 $n=1$ 时，算法 E 与算法 F 结果相同；但当 $n>1$ 时，算法 E 优于算法 F，随着 n 的增大，差异非常明显。通过观察发现，最高次项的指数大的，随着 n 的增长，函数结果也增长很快。

下面来看最后一个例子。算法 G 是 $2n^2$，算法 H 是 $3n+1$，算法 I 是 $2n^2+3n+1$，它们的计算结果如表 1.5 所示。

表 1.5　算法 G、算法 H 和算法 I 的计算结果

次　数	算法 G($2n^2$)	算法 H($3n+1$)	算法 I($2n^2+3n+1$)
$n=1$	2	4	6
$n=2$	8	7	15
$n=5$	50	16	66
$n=10$	200	31	231
$n=100$	20 000	301	20 301
$n=1\ 000$	2 000 000	3 001	2 003 001
$n=10\ 000$	200 000 000	30 001	200 030 001
$n=100\ 000$	20 000 000 000	300 001	20 000 300 001
$n=1\ 000\ 000$	2 000 000 000 000	3 000 001	2 000 003 000 001

从上述这组数据可以看出，当 n 的值越来越大时，$3n+1$ 已经无法与 $2n^2$ 的结果相比较，最终几乎可以忽略不计。也就是说，随着 n 的值变得非常大以后，算法 G 其实已经很趋近于算法 I。于是可以得到这样一个结论，判断一个算法的效率时，函数中的常数和其他次要项常常可以忽略，而更应该关注主项（最高阶项）的阶数。

根据上面几个例子，对比这几个算法的关键执行次数函数的渐近增长性，基本就可以分析出：对于某个算法，随着 n 的增大，它会越来越优于另一个算法，或者越来越差于另一个算法。这其实就是事前分析估算方法的理论依据，通过算法的时间复杂度来估算算法的时间效率。

1.5.5　算法的时间复杂度

通常情况下，如果存在几种算法思想，则人们总愿意尽早放弃那些不好的算法思想，因此，通常需要对算法进行分析。不仅如此，进行分析的能力还有助于洞察到如何设计有效的

此为试读，需要完整PDF请访问：www.ertongbook.com