全国计算机等级考试二级教程

——公共基础知识

（2015年版）

教育部考试中心

高等教育出版社

全国计算机等级考试二级教程

——公共基础知识

（2015 年版）

Quanguo Jisuanji Dengji Kaoshi Erji Jiaocheng

——Gonggong Jichu Zhishi

教育部考试中心

主编　徐士良

编者　陈　英　刘晓鸿

高等教育出版社·北京

内容简介

本书是根据教育部考试中心颁布的《全国计算机等级考试二级公共基础知识考试大纲(2013年版)》的要求，在2013年版的基础上修订而成。主要内容包括：数据结构与算法，程序设计基础，软件工程基础，数据库设计基础。

本书不仅是应试者必备的自学和辅导材料，也可以作为一般院校相应课程的教材或自学参考书。

图书在版编目(CIP)数据

全国计算机等级考试二级教程:2015 年版. 公共基础知识/教育部考试中心编. --北京:高等教育出版社,2014. 11

ISBN 978-7-04-041371-7

Ⅰ. ①全… Ⅱ. ①教… Ⅲ. ①电子计算机-水平考试-教材 Ⅳ. ①TP3

中国版本图书馆 CIP 数据核字(2014)第 249507 号

策划编辑 何新权　　责任编辑 何新权　　封面设计 杨立新　　版式设计 范晓红

责任校对 王 雨　　责任印制 刘思涵

出版发行 高等教育出版社
社　　址 北京市西城区德外大街 4 号
邮政编码 100120
印　　刷 山东鸿杰印务集团有限公司
开　　本 787mm×1092mm 1/16
印　　张 9.5
字　　数 220 千字
购书热线 010-58581118

咨询电话 400-810-0598
网　　址 http://www. hep. edu. cn
http://www. hep. com. cn
网上订购 http://www. landraco. com
http://www. landraco. com. cn
版　　次 2014 年 11 月第 1 版
印　　次 2014 年 11 月第 1 次印刷
定　　价 20.00 元

物 料 号 41371-00

积极发展全国计算机等级考试 为培养计算机应用专门人才、促进信息 产业发展作出贡献

（序）

中国科协副主席　中国系统仿真学会理事长
第五届全国计算机等级考试委员会主任委员
赵沁平

当今，人类正在步入一个以智力资源的占有和配置，知识生产、分配和使用为最重要因素的知识经济时代，也就是小平同志提出的“科学技术是第一生产力”的时代。世界各国的竞争已成为以经济为基础、以科技（特别是高科技）为先导的综合国力的竞争。在高科技中，信息科学技术是知识高度密集、学科高度综合、具有科学与技术融合特征的学科。它直接渗透到经济、文化和社会的各个领域，迅速改变着人们的工作、生活和社会的结构，是当代发展知识经济的支柱之一。

在信息科学技术中，计算机硬件及通信设施是载体，计算机软件是核心。软件是人类知识的固化，是知识经济的基本表征，软件已成为信息时代的新型“物理设施”。人类抽象的经验、知识正逐步由软件予以精确地体现。在信息时代，软件是信息化的核心，国民经济和国防建设、社会发展、人民生活都离不开软件，软件无处不在。软件产业是增长快速的朝阳产业，是具有高附加值、高投入高产出、无污染、低能耗的绿色产业。软件产业的发展将推动知识经济的进程，促进从注重量的增长向注重质的提高方向发展。软件产业是关系到国家经济安全和文化安全，体现国家综合实力，决定21世纪国际竞争地位的战略性产业。

为了适应知识经济发展的需要，大力促进信息产业的发展，需要在全民中普及计算机的基本知识，培养一批又一批能熟练运用计算机和软件技术的各行各业的应用型人才。

1994年，国家教委（现教育部）推出了全国计算机等级考试，这是一种专门评价应试人员对计算机软硬件实际掌握能力的考试。它不限制报考人员的学历和年龄，从而为培养各行业计算机应用人才开辟了一条广阔的道路。

1994年是推出全国计算机等级考试的第一年，当年参加考试的有1万余人，2012年报考人数已达549万人。截至2013年年底，全国计算机等级考试共开考38次，考生人数累计达5 422万人，有2 067万人获得了各级计算机等级证书。

事实说明，鼓励社会各阶层人士通过各种途径掌握计算机应用技术，并通过等级考试对他们的能力予以科学、公正、权威性的认证，是一种比较好的、有效的计算机应用人才培养途径，符合我国的具体国情。等级考试同时也为用人部门录用和考核人员提供了一种测评手段。从有关公司对等级考试所作的社会抽样调查结果看，不论是管理人员还是应试人员，对该项考试的内容和

形式都给予了充分肯定。

计算机技术日新月异。全国计算机等级考试大纲顺应技术发展和社会需求的变化,从2010年开始对新版考试大纲进行调研和修订,在考试体系、考试内容、考试形式等方面都做了较大调整,希望等级考试更能反映当前计算机技术的应用实际,使培养计算机应用人才的工作更健康地向前发展。

全国计算机等级考试取得了良好的效果,这有赖于各有关单位专家在等级考试的大纲编写、试题设计、阅卷评分及效果分析等多项工作中付出的大量心血和辛勤劳动,他们为这项工作的开展作出了重要的贡献。我们在此向他们表示衷心的感谢!

我们相信,在21世纪知识经济和加快发展信息产业的形势下,在教育部考试中心的精心组织领导下,在全国各有关专家的大力配合下,全国计算机等级考试一定会以"激励引导成才,科学评价用才,服务社会选材"为目标,服务考生和社会,为我国培养计算机应用专门人才的事业作出更大的贡献。

前　言

本书是根据教育部考试中心颁布的《全国计算机等级考试二级公共基础知识考试大纲(2013年版)》的要求组织编写的。

全书共分四章。

第1章主要介绍算法的基本概念,数据结构的基本概念及其定义,线性表及其基本运算,栈和队列及其基本运算,线性链表及其基本运算,二叉树的基本概念、存储结构及其遍历,最后还介绍了几种常用的查找与排序算法。

第2章主要介绍程序设计方法与风格,结构化程序设计,面向对象的程序设计方法,对象,方法,属性及继承与多态性。

第3章主要介绍软件工程基本概念,结构化分析方法,结构化设计方法,软件测试的基本方法,程序的调试方法。

第4章主要介绍数据库,数据库管理系统,数据库系统的基本概念,数据模型,实体联系模型及E-R图等基本概念,关系代数理论中的基本运算,数据库设计的基本方法和步骤。

每章后面都附有一定数量的习题。

本书的第1章由徐士良老师编写,第2、3章由陈英老师编写,第4章由刘晓鸿老师编写,全书由徐士良、黄啸波统稿,徐士良审定,陈向群老师对全书进行了最后审阅。

由于时间紧迫,以及作者水平有限,书中难免有错误或不妥之处,恳请读者批评指正。

编者

目　录

第 1 章　数据结构与算法 …… 1
1.1　算法 …… 1
1.1.1　算法的基本概念 …… 1
1.1.2　算法设计基本方法 …… 2
1.1.3　算法复杂度 …… 6
1.2　数据结构的基本概念 …… 9
1.2.1　什么是数据结构 …… 9
1.2.2　数据结构的图形表示 …… 12
1.2.3　线性结构与非线性结构 …… 13
1.3　线性表及其顺序存储结构 …… 14
1.3.1　线性表的基本概念 …… 14
1.3.2　线性表的顺序存储结构 …… 15
1.3.3　顺序表的插入运算 …… 16
1.3.4　顺序表的删除运算 …… 18
1.4　栈和队列 …… 19
1.4.1　栈及其基本运算 …… 19
1.4.2　队列及其基本运算 …… 22
1.5　线性链表 …… 25
1.5.1　线性链表的基本概念 …… 25
1.5.2　线性链表的基本运算 …… 29
1.5.3　循环链表 …… 31
1.6　树与二叉树 …… 32
1.6.1　树的基本概念 …… 32
1.6.2　二叉树及其基本性质 …… 35
1.6.3　二叉树的存储结构 …… 38
1.6.4　二叉树的遍历 …… 39
1.7　查找技术 …… 41
1.7.1　顺序查找 …… 41
1.7.2　二分法查找 …… 42
1.8　排序技术 …… 42
1.8.1　交换类排序法 …… 42
1.8.2　插入类排序法 …… 44
1.8.3　选择类排序法 …… 46
习题 1 …… 48
第 2 章　程序设计基础 …… 50
2.1　程序设计方法与风格 …… 50
2.2　结构化程序设计 …… 52
2.2.1　结构化程序设计的原则 …… 52
2.2.2　结构化程序的基本结构与特点 …… 52
2.2.3　结构化程序设计原则和方法的应用 …… 53
2.3　面向对象的程序设计 …… 54
2.3.1　关于面向对象方法 …… 54
2.3.2　面向对象方法的基本概念 …… 57
习题 2 …… 60
第 3 章　软件工程基础 …… 61
3.1　软件工程基本概念 …… 61
3.1.1　软件定义与软件特点 …… 61
3.1.2　软件危机与软件工程 …… 62
3.1.3　软件过程与软件生命周期 …… 64
3.1.4　软件工程的目标与原则 …… 65
3.1.5　软件开发工具与软件开发环境 …… 66
3.2　结构化分析方法 …… 67
3.2.1　需求分析与需求分析方法 …… 67
3.2.2　结构化分析方法 …… 68
3.2.3　软件需求规格说明书 …… 72
3.3　结构化设计方法 …… 73
3.3.1　软件设计的基本概念 …… 74
3.3.2　概要设计 …… 75
3.3.3　详细设计 …… 80
3.4　软件测试 …… 85
3.4.1　软件测试的目的和定义 …… 85
3.4.2　软件测试的准则 …… 85
3.4.3　软件测试方法与技术综述 …… 86
3.4.4　软件测试的策略 …… 93
3.5　程序的调试 …… 96
3.5.1　基本概念 …… 96
3.5.2　软件调试方法 …… 98
习题 3 …… 99
第 4 章　数据库设计基础 …… 101

4.1 数据库系统的基本概念 …… 101
4.1.1 数据、数据库、数据库管理系统 …… 101
4.1.2 数据库系统的发展 …… 105
4.1.3 数据库系统的基本特点 …… 107
4.1.4 数据库系统的内部结构体系 …… 108
4.2 数据模型 …… 109
4.2.1 数据模型的基本概念 …… 109
4.2.2 E-R 模型 …… 110
4.2.3 层次模型 …… 114
4.2.4 网状模型 …… 115
4.2.5 关系模型 …… 116
4.3 关系代数 …… 118
4.4 数据库设计与管理 …… 125
4.4.1 数据库设计概述 …… 125
4.4.2 数据库设计的需求分析 …… 126
4.4.3 数据库概念设计 …… 127
4.4.4 数据库的逻辑设计 …… 130
4.4.5 数据库的物理设计 …… 133
4.4.6 数据库管理 …… 133
习题 4 …… 134
附录 1 全国计算机等级考试二级公共基础知识考试大纲(2013 年版) …… 136
附录 2 全国计算机等级考试二级公共基础知识样题及参考答案 …… 138
附录 3 习题参考答案 …… 140

第1章 数据结构与算法

1.1 算法

1.1.1 算法的基本概念

所谓算法是指解题方案的准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。当然，程序也可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序的编制不可能优于算法的设计。

作为一个算法，一般应具有以下几个基本特征。

(1) 可行性(Effectiveness)

算法的可行性包括以下两个方面：

① 算法中的每一个步骤必须能够实现。如在算法中不允许执行分母为0的操作，在实数范围内不可能求一个负数的平方根等。

② 算法执行的结果要能够达到预期的目的。

针对实际问题设计的算法，人们总是希望能够得到满意的结果。但一个算法又总是在某个特定的计算工具上执行的，因此，算法在执行过程中往往要受到计算工具的限制，使执行结果产生偏差。例如，在进行数值计算时，如果某计算工具具有7位有效数字(如程序设计语言中的单精度运算)，则在计算下列三个量

$$A^{①}=10^{12},B=1,C=-10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，即

$$A+B+C=10^{12}+1+(-10^{12})=0$$

$$A+C+B=10^{12}+(-10^{12})+1=1$$

① 为与计算机程序保持一致，本书中所有变量均用正体表示。

而在数学上,A+B+C 与 A+C+B 是完全等价的。因此,算法与计算公式是有差别的。在设计一个算法时,必须要考虑它的可行性,否则是不会得到满意结果的。

(2) 确定性(Definiteness)

算法的确定性,是指算法中的每一个步骤都必须是有明确定义的,不允许有模棱两可的解释,也不允许有多义性。这一性质也反映了算法与数学公式的明显差别。在解决实际问题时,可能会出现这样的情况:针对某种特殊问题,数学公式是正确的,但按此数学公式设计的计算过程可能会使计算机系统无所适从。这是因为根据数学公式设计的计算过程只考虑了正常使用的情况,而当出现异常情况时,此计算过程就不能适应了。

(3) 有穷性(Finiteness)

算法的有穷性,是指算法必须能在有限的时间内做完,即算法必须能在执行有限个步骤之后终止。数学中的无穷级数,在实际计算时只能取有限项,即计算无穷级数值的过程只能是有穷的。因此,一个数的无穷级数表示只是一个计算公式,而根据精度要求确定的计算过程才是有穷的算法。

算法的有穷性还应包括合理的执行时间的含义。因为,如果一个算法需要执行千万年,显然失去了实用价值。

(4) 拥有足够的情报

一个算法是否有效,还取决于为算法所提供的情报是否足够。通常,算法中的各种运算总是要施加到各个运算对象上,而这些运算对象又可能具有某种初始状态,这是算法执行的起点或是依据。因此,一个算法执行的结果总是与输入的初始数据有关,不同的输入将会有不同的结果输出。当输入不够或输入错误时,算法本身也就无法执行或导致执行有错。一般来说,当算法拥有足够的情报时,此算法才是有效的,而当提供的情报不够时,算法可能无效。

综上所述,所谓算法,是一组严谨地定义运算顺序的规则,并且每一个规则都是有效的,且是明确的,此顺序将在有限的次数下终止。

1.1.2 算法设计基本方法

计算机解题的过程实际上是在实施某种算法,这种算法称为计算机算法。计算机算法不同于人工处理的方法。

本小节介绍工程上常用的几种算法设计方法,在实际应用时,各种方法之间往往存在着一定的联系。

(1) 列举法

列举法的基本思想是,根据提出的问题,列举所有可能的情况,并用问题中给定的条件检验哪些是需要的,哪些是不需要的。因此,列举法常用于解决“是否存在”或“有多少种可能”等类型的问题,例如求解不定方程的问题。

列举法的特点是算法比较简单。但当列举的可能情况较多时,执行列举算法的工作量将会很大。因此,在用列举法设计算法时,使方案优化,尽量减少运算工作量,是应该重点注意的。通常,在设计列举算法时,只要对实际问题进行详细的分析,将与问题有关的知识条理化、完备化、系统化,从中找出规律;或对所有可能的情况进行分类,引出一些有用的信息,是可以大大减少列举量的。

下面举例说明利用列举算法解决问题时如何对算法进行优化。

例 1.1 今有鸡母一,值钱三;鸡翁一,值钱二;鸡雏一,值钱半。凡百钱买百鸡,问鸡母、鸡翁、鸡雏各几何?

假设买母鸡 I 只、公鸡 J 只、小鸡 K 只。根据题意,粗略的列举算法描述如下:

```
FOR I=0 TO 100 STEP 1   DO
FOR J=0 TO 100 STEP 1   DO
FOR K=0 TO 100 STEP 1   DO
{
    IF ((I+J+K==100)AND(3*I+2*J+0.5*K==100.0)) THEN
        PRINT   I, J, K
}
END
```

(注:本算法描述采用的是一种大家都能理解的算法描述语言书写的,并不是用某种具体的程序设计语言编写的程序。本章中以后均采用这种描述语言来描述算法。)

在这个算法中,共嵌套有三层循环,每层循环各需要循环 101 次,因此,总循环次数为101^3,大约为 100 万次。但只要对问题进行分析,发现可以对这个算法进行优化,可以减少大量不必要的循环次数。

首先,考虑到母鸡为 3 钱一只,因此,母鸡最多只能买 33 只,即算法中的外循环没有必要从 0 到 100,而只需要从 0 到 33 就可以了。

其次,考虑到公鸡为 2 钱一只,因此,公鸡最多只能买 50 只。又考虑到对公鸡的列举是在算法的第二层循环中,此时已经买了 I 只母鸡,且买一只母鸡的价钱相当于买 1.5 只公鸡。因此,由第一层循环已经确定买 I 只母鸡的前提下,公鸡最多只能买 50-1.5I,即第二层对 J 的循环只需从 0 到 50-1.5I 就可以了。

最后,考虑到买的总鸡数为 100,而由第一层循环已确定买 I 只母鸡,由第二层循环已确定买 J 只公鸡,因此,买小鸡的数量只能是 K=100-I-J,即第三层循环已经没有必要了。

经过以上分析,可以将上述算法修改如下:

```
FOR I=0 TO 33 STEP 1   DO
FOR J=0 TO 50-1.5*I STEP 1 DO
{
    K=100-I-J
    IF (3*I+2*J+0.5*K==100.0) THEN
        PRINT   I, J, K
}
END
```

不难分析,经修改后的算法的列举量(即循环次数)为

$$\sum_{I=0}^{33}(51-1.5I)\approx 894$$

这个算法的执行结果如下:

2	30	68
5	25	70
8	20	72
11	15	74
14	10	76
17	5	78
20	0	80

列举原理是计算机应用领域中十分重要的原理。许多实际问题,若采用人工列举是不可想象的,但由于计算机的运算速度快,擅长重复操作,可以很方便地进行大量列举。列举算法虽然是一种比较笨拙而原始的方法,其运算量比较大,但在有些实际问题中(如寻找路径、查找、搜索等问题),局部使用列举法却是很有效的,因此,列举算法是计算机算法中的一个基础算法。

(2) 归纳法

归纳法的基本思想是,通过列举少量的特殊情况,经过分析,最后找出一般的关系。显然,归纳法要比列举法更能反映问题的本质,并且可以解决列举量为无限的问题。但是,从一个实际问题中总结归纳出一般的关系,并不是一件容易的事情,尤其是要归纳出一个数学模型更为困难。从本质上讲,归纳就是通过观察一些简单而特殊的情况,最后总结出一般性的结论。

归纳是一种抽象,即从特殊现象中找出一般关系。但由于在归纳的过程中不可能对所有的情况进行列举,因此,最后由归纳得到的结论还只是一种猜测,还需要对这种猜测加以必要的证明。实际上,通过精心观察而得到的猜测得不到证实或最后证明猜测是错的,也是常有的事。

(3) 递推

所谓递推,是指从已知的初始条件出发,逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定,或是通过对问题的分析与化简而确定。递推本质上也属于归纳法,工程上许多递推关系式实际上是通过对实际问题的分析与归纳而得到的,因此,递推关系式往往是归纳的结果。

递推算法在数值计算中是极为常见的。但是,对于数值型的递推算法必须要注意数值计算的稳定性问题。

(4) 递归

人们在解决一些复杂问题时,为了降低问题的复杂程度(如问题的规模等),一般总是将问题逐层分解,最后归结为一些最简单的问题。这种将问题逐层分解的过程,实际上并没有对问题进行求解,而只是当解决了最后那些最简单的问题后,再沿着原来分解的逆过程逐步进行综合,这就是递归的基本思想。由此可以看出,递归的基础也是归纳。在工程实际中,有许多问题就是用递归来定义的,数学中的许多函数也是用递归来定义的。递归在可计算性理论和算法设计中占有很重要的地位。

下面用一个简单的例子来说明递归的基本思想。

例 1.2 编写一个过程,对于输入的参数 n,依次打印输出自然数 1 到 n。

这是一个很简单的问题,实际上不用递归就能解决。对应的算法描述如下:

```
wrt(int  n)
```

```
{
    FOR k=1 TO n STEP 1 DO   PRINT k
    RETURN
}
```

这个问题还可以用递归来解决。对应的算法描述如下：

```
wrt1(int n)
{
    IF (n≠0)   THEN
    {
        wrt1(n-1)
        PRINT n
    }
    RETURN
}
```

在递归过程 wrt1( )中，n 是形参。在开始执行过程 wrt1( )时，首先要判断形参变量值（开始时为 n）是否不等于 0，如果不等于 0，则将形参值减 1（即 n-1）后作为新的实参再调用过程 wrt1( )；在调用过程 wrt1( )时，又需判断形参值（此时已变为 n-1）是否不等于 0，如果不等于 0，则又将形参值减 1（即 n-2）后作为新的实参再次调用过程 wrt1( )，……。以此类推，直到过程 wrt1( )的形参值等于 0 为止。此时，由于在先前各层的过程调用中，过程 wrt1( )实际上没有执行完，即各层中的形参值还没有被打印输出，这就需要逐层返回，以便打印输出各层中的输入参数 1，2，…，n。为此，在递归算法的执行过程中，需要记忆各层调用中的参数，以便在逐层返回时恢复这些参数继续进行处理。具体来说，在函数 wrt1( )开始执行后，随着各次的递归调用，逐次记忆各层调用中的输入参数 n，n-1，n-2，…，2，1，在逐层返回时，又依次（按记忆的相反次序）将这些参数打印输出。

在程序设计中，递归是一个很有用的工具。对于一些比较复杂的问题，设计成递归算法其结构清晰，可读性也强。

由上例可以看出，自己调用自己的过程称为递归调用过程。

递归分为直接递归与间接递归两种。如果一个算法 P 显式地调用自己则称为直接递归。如果算法 P 调用另一个算法 Q，而算法 Q 又调用算法 P，则称为间接递归调用。

递归是很重要的算法设计方法之一。实际上，递归过程能将一个复杂的问题归结为若干个较简单的问题，然后将这些较简单的问题再归结为更简单的问题，这个过程可以一直做下去，直到最简单的问题为止。

有些实际问题，既可以归纳为递推算法，又可以归纳为递归算法。但递推与递归的实现方法是大不一样的。递推是从初始条件出发，逐次推出所需求的结果；而递归则是从算法本身到达递归边界的。通常，递归算法要比递推算法清晰易读，其结构比较简练。特别是在许多比较复杂的问题中，很难找到从初始条件推出所需结果的全过程，此时，设计递归算法要比递推算法容易得多。但递归算法的执行效率比较低。

（5）减半递推技术

实际问题的复杂程度往往与问题的规模有着密切的联系。因此,利用分治法解决这类实际问题是有效的。所谓分治法,就是对问题分而治之。工程上常用的分治法是减半递推技术。

所谓“减半”,是指将问题的规模减半,而问题的性质不变;所谓“递推”,是指重复“减半”的过程。

下面举例说明利用减半递推技术设计算法的基本思想。

例1.3 设方程f(x)=0在区间[a,b]上有实根,且f(a)与f(b)异号。利用二分法求该方程在区间[a,b]上的一个实根。

用二分法求方程实根的减半递推过程如下:

首先取给定区间的中点c=(a+b)/2。

然后判断f(c)是否为0。若f(c)=0,则说明c即为所求的根,求解过程结束;如果$f(c)\neq 0$,则根据以下原则将原区间减半:

若f(a)f(c)<0,则取原区间的前半部分;

若f(b)f(c)<0,则取原区间的后半部分。

最后判断减半后的区间长度是否已经很小:

若$|a-b|<\varepsilon$,则过程结束,取(a+b)/2为根的近似值;

若$|a-b|\geq\varepsilon$,则重复上述的减半过程。

(6) 回溯法

前面讨论的递推和递归算法本质上是对实际问题进行归纳的结果,而减半递推技术也是归纳法的一个分支。在工程上,有些实际问题很难归纳出一组简单的递推公式或直观的求解步骤,并且也不能进行无限的列举。对于这类问题,一种有效的方法是“试”。通过对问题的分析,找出一个解决问题的线索,然后沿着这个线索逐步试探,对于每一步的试探,若试探成功,就得到问题的解,若试探失败,就逐步回退,换别的路线再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

1.1.3 算法复杂度

算法的复杂度主要包括时间复杂度和空间复杂度。

1. 算法的时间复杂度

所谓算法的时间复杂度,是指执行算法所需要的计算工作量。

为了能够比较客观地反映出一个算法的效率,在度量一个算法的工作量时,不仅应该与所使用的计算机、程序设计语言以及程序编制者无关,而且还应该与算法实现过程中的许多细节无关。为此,可以用算法在执行过程中所需基本运算的执行次数来度量算法的工作量。基本运算反映了算法运算的主要特征,因此,用基本运算的次数来度量算法工作量是客观的也是实际可行的,有利于比较同一问题的几种算法的优劣。例如,在考虑两个矩阵相乘时,可以将两个实数之间的乘法运算作为基本运算,而对于所用的加法(或减法)运算忽略不计。又如,当需要在一个表中进行查找时,可以将两个元素之间的比较作为基本运算。

算法所执行的基本运算次数还与问题的规模有关。例如,两个20阶矩阵相乘与两个10阶矩阵相乘,所需要的基本运算(即两个实数的乘法)次数显然是不同的,前者需要更多的运算次数。因此,在分析算法的工作量时,还必须对问题的规模进行度量。

综上所述，算法的工作量用算法所执行的基本运算次数来度量，而算法所执行的基本运算次数是问题规模的函数，即

$$\text{算法的工作量} = f(n)$$

其中 n 是问题的规模。例如，两个 n 阶矩阵相乘所需要的基本运算（即两个实数的乘法）次数为 n^3，即计算工作量为 n^3，也就是时间复杂度为 n^3。

在具体分析一个算法的工作量时，还会存在这样的问题：对于一个固定的规模，算法所执行的基本运算次数还可能与特定的输入有关，而实际上又不可能将所有可能情况下算法所执行的基本运算次数都列举出来。例如，“在长度为 n 的一维数组中查找值为 x 的元素”，若采用顺序搜索法，即从数组的第一个元素开始，逐个与被查值 x 进行比较。显然，如果第一个元素恰为 x，则只需要比较 1 次。但如果 x 为数组的最后一个元素，或者 x 不在数组中，则需要比较 n 次才能得到结果。因此，在这个问题的算法中，其基本运算（即比较）的次数与具体的被查值 x 有关。

在同一个问题规模下，如果算法执行所需的基本运算次数取决于某一特定输入时，可以用以下两种方法来分析算法的工作量。

（1）平均性态（Average Behavior）

所谓平均性态分析，是指用各种特定输入下的基本运算次数的加权平均值来度量算法的工作量。

设 x 是所有可能输入中的某个特定输入，$p(x)$ 是 x 出现的概率（即输入为 x 的概率），$t(x)$ 是算法在输入为 x 时所执行的基本运算次数，则算法的平均性态定义为

$$A(n) = \sum_{x \in D_n} p(x)t(x)$$

其中 D_n 表示当规模为 n 时，算法执行时所有可能输入的集合。这个式子中的 $t(x)$ 可以通过分析算法来加以确定；而 $p(x)$ 必须由经验或用算法中有关的一些特定信息来确定，通常是不能解析地加以计算的。如果确定 $p(x)$ 比较困难，则会给平均性态的分析带来困难。

（2）最坏情况复杂性（Worst-Case Complexity）

所谓最坏情况分析，是指在规模为 n 时，算法所执行的基本运算的最大次数。它定义为

$$W(n) = \max_{x \in D_n} \{t(x)\}$$

显然，$W(n)$ 的计算要比 $A(n)$ 的计算方便得多。由于 $W(n)$ 实际上是给出了算法工作量的一个上界，因此，它比 $A(n)$ 更具有实用价值。

下面通过一个例子来说明算法复杂度的平均性态分析与最坏情况分析。

例 1.4 采用顺序搜索法，在长度为 n 的一维数组中查找值为 x 的元素。即从数组的第一个元素开始，逐个与被查值 x 进行比较。基本运算为 x 与数组元素的比较。

首先考虑平均性态分析。

设被查项 x 在数组中出现的概率为 q。当需要查找的 x 为数组中第 i 个元素时，则在查找过程中需要做 i 次比较，当需要查找的 x 不在数组中时（即数组中没有 x 这个元素），则需要与数组中所有的元素进行比较。即

$$t_i = \begin{cases} i, & 1 \leqslant i \leqslant n \\ n, & i = n+1 \end{cases}$$

其中 $i=n+1$ 表示 x 不在数组中的情况。

如果假设需要查找的 x 出现在数组中每个位置上的可能性是一样的，则 x 出现在数组中每一个位置上的概率为 q/n（因为前面已经假设 x 在数组中的概率为 q），而 x 不在数组中的概率为 1-q。即

$$p_i=\begin{cases} q/n, & 1\leqslant i\leqslant n \\ 1-q, & i=n+1 \end{cases}$$

其中 i=n+1 表示 x 不在数组中的情况。

因此，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素，在平均情况下需要做的比较次数为

$$A(n)=\sum_{i=1}^{n+1}p_it_i=\sum_{i=1}^{n}(q/n)i+(1-q)n=(n+1)q/2+(1-q)n$$

如果已知需要查找的 x 一定在数组中，此时 q=1，则 A(n)=(n+1)/2。这就是说，在这种情况下，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素，在平均情况下需要检查数组中一半的元素。

如果已知需要查找的 x 有一半的机会在数组中，此时 q=1/2，则

$$A(n)=[(n+1)/4]+n/2\approx 3n/4$$

这就是说，在这种情况下，用顺序搜索法在长度为 n 的一维数组中查找值为 x 的元素，在平均情况下需要检查数组中 3/4 的元素。

再考虑最坏情况分析。

在这个例子中，最坏情况发生在需要查找的 x 是数组中的最后一个元素或 x 不在数组中的时候，此时显然有

$$W(n)=\max\{t_i \mid 1\leqslant i\leqslant n+1\}=n$$

在上述例子中，算法执行的工作量是与具体的输入有关的，A(n)只是它的加权平均值，而实际上对于某个特定的输入，其计算工作量未必是 A(n)，且 A(n)也不一定等于 W(n)。但在另外一些情况下，算法的计算工作量与输入无关，即当规模为 n 时，在所有可能的输入下，算法所执行的基本运算次数是一定的，此时有 A(n)=W(n)。例如，两个 n 阶的矩阵相乘，都需要做 n^3 次实数乘法，而与输入矩阵的具体元素无关。

2. 算法的空间复杂度

一个算法的空间复杂度，一般是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间（例如，在链式结构中，除了要存储数据本身外，还需要存储链接信息）。如果额外空间量相对于问题规模来说是常数，则称该算法是原地（in place）工作的。在许多实际问题中，为了减少算法所占的存储空间，通常采用压缩存储技术，以便尽量减少不必要的额外空间。

1.2 数据结构的基本概念

利用计算机进行数据处理是计算机应用的一个重要领域。在进行数据处理时，实际需要处理的数据元素一般有很多，而这些大量的数据元素都需要存放在计算机中，因此，大量的数据元素在计算机中如何组织，以便提高数据处理的效率，并且节省计算机的存储空间，这是进行数据处理的关键问题。

显然，杂乱无章的数据是不便于处理的。而将大量的数据随意地存放在计算机中，实际上也是“自找苦吃”，对数据处理更是不利。

数据结构作为计算机的一门学科，主要研究和讨论以下三个方面的问题：

① 数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构；

② 在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构；

③ 对各种数据结构进行的运算。

讨论以上问题的主要目的是为了提高数据处理的效率。所谓提高数据处理的效率，主要包括两个方面：一是提高数据处理的速度，二是尽量节省在数据处理过程中所占用的计算机存储空间。

本节主要讨论工程上常用的一些基本数据结构，它们是软件设计的基础。

1.2.1 什么是数据结构

计算机已被广泛用于数据处理。实际问题中的各数据元素之间总是相互关联的。所谓数据处理，是指对数据集合中的各元素以各种方式进行运算，包括插入、删除、查找、更改等运算，也包括对数据元素进行分析。在数据处理领域中，建立数学模型有时并不十分重要，事实上，许多实际问题是无法表示成数学模型的。人们最感兴趣的是知道数据集合中各数据元素之间存在什么关系，应如何组织它们，即如何表示所需要处理的数据元素。

简单地说，数据结构是指相互有关联的数据元素的集合。例如，向量和矩阵就是数据结构，在这两个数据结构中，数据元素之间有着位置上的关系。又如，图书馆中的图书卡片目录，则是一个较为复杂的数据结构，对于列在各卡片上的各种书之间，可能在主题、作者等问题上相互关联，甚至一本书本身也有不同的相关成分。

数据元素具有广泛的含义。一般来说，现实世界中客观存在的一切个体都可以是数据元素。例如：

描述一年四季的季节名

春、夏、秋、冬

可以作为季节的数据元素；

表示数值的各个数

18、11、35、23、16、…

可以作为数值的数据元素；

表示家庭成员的各成员名