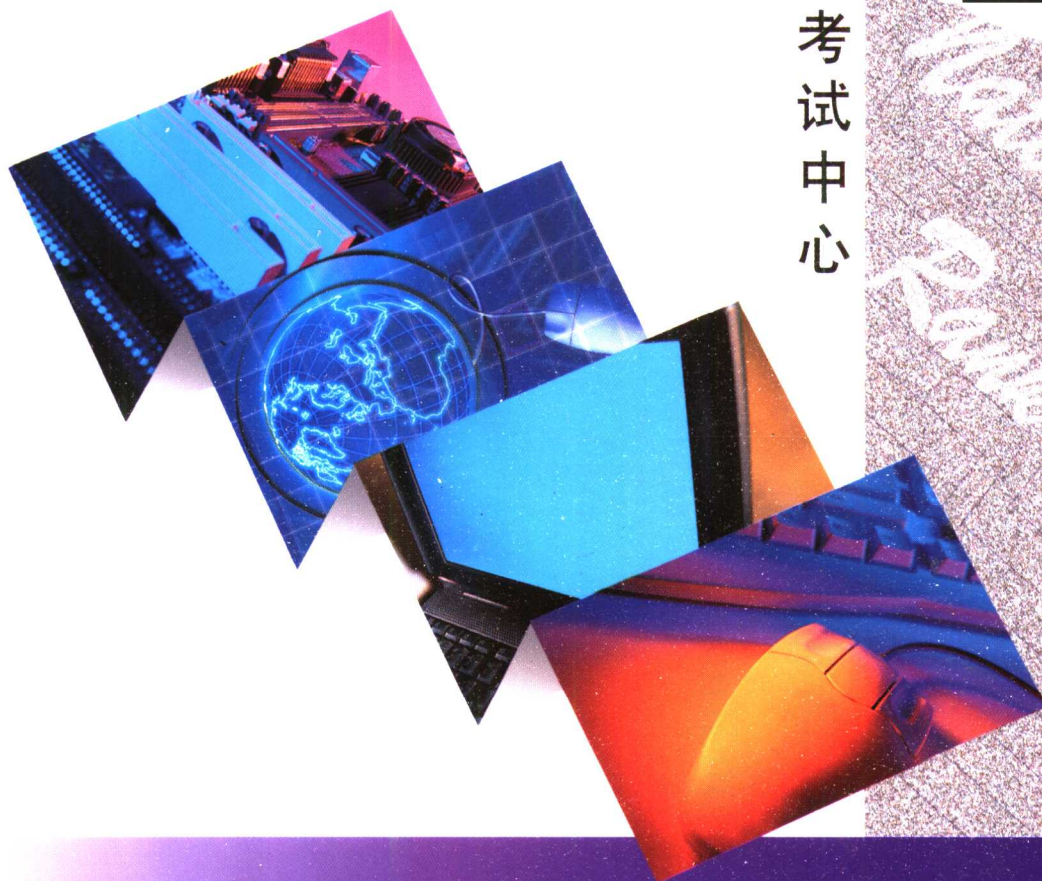


教育部
考试中心



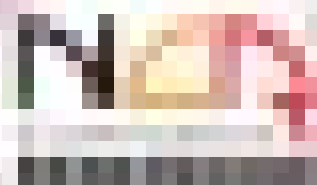
全国计算机等级考试

二级教程

—公共基础知识 (2004 年版)

高等教育出版社





2019年10月



全国计算机等级考试

二级教程

——公共基础知识

2019年10月

机械工业出版社

全国计算机等级考试

二 级 教 程

——公共基础知识（2004 年版）

教育部考试中心

高等教育出版社

内容简介

本书是根据教育部考试中心制订的《全国计算机等级考试考试大纲(2004年版)》中对二级公共基础部分的要求而编写的。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——公共基础知识:
2004年版/教育部考试中心. —北京:高等教育出版社, 2004.5

ISBN 7-04-015043-3

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

中国版本图书馆 CIP 数据核字(2004)第 031016 号

策划编辑 田晓兰 责任编辑 田晓兰 封面设计 于文燕
版式设计 马静如 责任校对 尤 静 责任印制 韩 刚

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 100011
总 机 010-82028899

购书热线 010-64054588
免费咨询 800-810-0598
网 址 <http://www.hep.edu.cn>
<http://www.hep.com.cn>

经 销 新华书店北京发行所
印 刷 北京市鑫霸印务有限公司

开 本 880×1230 1/16
印 张 7.75
字 数 230 000

版 次 2004 年 5 月第 1 版
印 次 2004 年 5 月第 1 次印刷
定 价 16.00 元

本书如有缺页、倒页、脱页等质量问题,请到所购图书销售部门联系调换。

版权所有 侵权必究

大力推行全国计算机等级考试 为发展知识经济、信息产业和培养计算机 专门人才作出贡献

(代 序)

中国科学院院士 北京大学信息与工程科学学部主任

全国计算机等级考试委员会主任委员

杨芙清

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。科教是经济发展的基础,知识是人类创新的源泉。基础研究的科学发现、应用研究的原理探索 and 开发研究的技术发明,三者之间的联系愈来愈紧密,转换周期日趋缩短。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,迅速改变着人们的观念、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,微电子是基础,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。软件产业是增长最快的朝阳产业,是具有高额附加值、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,是典型的知识型产业。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,大力推动信息产业的发展,需要在全民中普及计算机的基本知识,广开渠道,培养和造就一批又一批能熟练运用计算机和软件技术的各行各业的专门人才。

1994 年,原国家教委(现教育部)推出了全国计算机等级考试,它是一种重视应试人员对计算机和软件的实际掌握能力的考试。它不限制报考人员的学历背景,任何年龄段的人员都可以报考。这就为培养各行各业计算机的应用人才,开辟了一条广阔的道路。

1994 年是推出计算机等级考试的第一年,当年参加考试的有 1 万余人;到 2003 年,报考人数已达 251 万余人。截止至 2003 年底,全国计算机等级考试共开考 18 次,考生人数累计超过 1 050 万人,其中,有 350 多万考生获得了不同级别的证书。

事实说明,鼓励社会各阶层的人士通过各种途径掌握计算机应用技术,并运用等级考试对他们的才予以认真的、有权威性的认证,是一种较好的人才培养的有效途径,是比较符合我国具体情况的。等级考试也为用人单位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的评价。

计算机等级考试所取得的良好效果,也同全国各有关单位专家们在等级考试的大纲编写、试题设计、阅卷评分及效果分析等等多项工作中所付出的大量心血和辛勤的劳动密切相关,他们为这项工作的顺利开展作出了重要的贡献。

计算机与软件技术是一项日新月异的高新技术,计算机等级考试的考试内容和考核形式也将跟随新

技术的发展不断创新，需要及时推出新的考试科目，及时修订旧科目的考试大纲、教材，对考试命题以及上机考试系统进行改革和完善，从而使等级考试更能反映当前的应用实际，使培养计算机应用人才的基础工作更健康地向前发展。本书的出版正是为了满足新时期新技术发展的需要，满足社会主义市场经济人才培养的需要。

从面临知识经济的机遇与挑战这样一个社会大环境的背景出发，考察全国计算机等级考试，就会看到，这一举措是符合知识经济和发展信息产业的方向的，是值得大力推行的。

我们相信，在 21 世纪知识经济和加快发展信息产业的形式下，在教育部考试中心的精心组织领导下，在全国各有关专家们的大力配合下，全国计算机等级考试一定会以更新的面貌出现，从而为我国培养计算机应用专门人才的宏大事业作出更多的贡献。

2003 年 12 月

前 言

本书是根据教育部考试中心制订的《全国计算机等级考试考试大纲（2004 年版）》中关于二级公共基础知识部分的要求而编写的。

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

2004 年 3 月

目 录

第1章 数据结构与算法	1	2.2.2 结构化程序的基本结构与特点	42
1.1 算法	1	2.2.3 结构化程序设计原则和方法的应用	43
1.1.1 算法的基本概念	1	2.3 面向对象的程序设计	43
1.1.2 算法复杂度	4	2.3.1 关于面向对象方法	43
1.2 数据结构的基本概念	6	2.3.2 面向对象方法的基本概念	45
1.2.1 什么是数据结构	6	习题2	48
1.2.2 数据结构的图形表示	11	第3章 软件工程基础	50
1.2.3 线性结构与非线性结构	12	3.1 软件工程基本概念	50
1.3 线性表及其顺序存储结构	12	3.1.1 软件定义与软件特点	50
1.3.1 线性表的基本概念	12	3.1.2 软件危机与软件工程	51
1.3.2 线性表的顺序存储结构	13	3.1.3 软件工程过程与软件生命周期	52
1.3.3 顺序表的插入运算	14	3.1.4 软件工程的目标与原则	53
1.3.4 顺序表的删除运算	15	3.1.5 软件开发工具与软件开发环境	54
1.4 栈和队列	16	3.2 结构化分析方法	54
1.4.1 栈及其基本运算	16	3.2.1 需求分析与需求分析方法	55
1.4.2 队列及其基本运算	18	3.2.2 结构化分析方法	55
1.5 线性链表	20	3.2.3 软件需求规格说明书	59
1.5.1 线性链表的基本概念	20	3.3 结构化设计方法	60
1.5.2 线性链表的基本运算	23	3.3.1 软件设计的基本概念	60
1.5.3 循环链表及其基本运算	25	3.3.2 概要设计	62
1.6 树与二叉树	26	3.3.3 详细设计	67
1.6.1 树的基本概念	26	3.4 软件测试	70
1.6.2 二叉树及其基本性质	28	3.4.1 软件测试的目的	71
1.6.3 二叉树的存储结构	31	3.4.2 软件测试的准则	71
1.6.4 二叉树的遍历	32	3.4.3 软件测试技术与方法综述	71
1.7 查找技术	33	3.4.4 软件测试的实施	77
1.7.1 顺序查找	33	3.5 程序的调试	80
1.7.2 二分法查找	33	3.5.1 基本概念	80
1.8 排序技术	33	3.5.2 软件调试方法	81
1.8.1 交换类排序法	34	习题3	82
1.8.2 插入类排序法	35	第4章 数据库设计基础	84
1.8.3 选择类排序法	37	4.1 数据库系统的基本概念	84
习题1	38	4.1.1 数据、数据库、数据库管理系统	84
第2章 程序设计基础	40	4.1.2 数据库系统的发展	87
2.1 程序设计方法与风格	40	4.1.3 数据库系统的基本特点	88
2.2 结构化程序设计	41	4.1.4 数据库系统的内部结构体系	89
2.2.1 结构化程序设计的原则	41	4.2 数据模型	91

4.2.1 数据模型的基本概念	91	4.4.2 数据库设计的需求分析	104
4.2.2 E-R 模型	91	4.4.3 数据库概念设计	105
4.2.3 层次模型	95	4.4.4 数据库的逻辑设计	108
4.2.4 网状模型	95	4.4.5 数据库的物理设计	110
4.2.5 关系模型	96	4.4.6 数据库管理	110
4.3 关系代数	98	习题 4	111
4.4 数据库设计与管理	104		
4.4.1 数据库设计概述	104		

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。

1. 算法的基本特征

作为一个算法，一般应具有以下几个基本特征。

(1) 可行性 (effectiveness)

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

$$A=10^{12}, B=1, C=-10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，即

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

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

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。

2. 算法的基本要素

一个算法通常由两种基本要素组成:一是对数据对象的运算和操作,二是算法的控制结构。

(1) 算法中对数据的运算和操作

每个算法实际上是按解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。因此,计算机算法就是计算机能处理的操作所组成的指令序列。

通常,计算机可以执行的基本操作是以指令的形式描述的。一个计算机系统能执行的所有指令的集合,称为该计算机系统的指令系统。计算机程序就是按解题要求从计算机指令系统中选择合适的指令所组成的指令序列。在一般的计算机系统中,基本的运算和操作有以下四类:

- ① 算术运算:主要包括加、减、乘、除等运算。
- ② 逻辑运算:主要包括“与”、“或”、“非”等运算。
- ③ 关系运算:主要包括“大于”、“小于”、“等于”、“不等于”等运算。
- ④ 数据传输:主要包括赋值、输入、输出等操作。

前面提到,计算机程序也可以作为算法的一种描述,但由于在编制计算机程序时通常要考虑很多与方法和分析无关的细节问题(如语法规则),因此,在设计算法的一开始,通常并不直接用计算机程序来描述算法,而是用别的描述工具(如流程图,专门的算法描述语言,甚至用自然语言)来描述算法。但不管用哪种工具来描述算法,算法的设计一般都应从上述四种基本操作考虑,按解题要求从这些基本操作中选择合适的操作组成解题的操作序列。算法的主要特征着重于算法的动态执行,它区别于传统的着重于静态描述或按演绎方式求解问题的过程。传统的演绎数学是以公理系统为基础的,问题的求解过程是通过有限次推演来完成的,每次推演都将对问题作进一步的描述,如此不断推演,直到直接将解描述出来为止。而计算机算法则是使用一些最基本的操作,通过对已知条件一步一步的加工和变换,从而实现解题目标。这两种方法的解题思路是不同的。

(2) 算法的控制结构

一个算法的功能不仅取决于所选用的操作,而且还与各操作之间的执行顺序有关。算法中各操作之间的执行顺序称为算法的控制结构。

算法的控制结构给出了算法的基本框架,它不仅决定了算法中各操作的执行顺序,而且也直接反映了算法的设计是否符合结构化原则。描述算法的工具通常有传统流程图、N-S 结构化流程图、算法描述语言等。一个算法一般都可以用顺序、选择、循环三种基本控制结构组合而成。

3. 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。

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。递推本质上属于归纳法,工程上许多递推关系式实际上是通过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。

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 1.1 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。

1.1.2 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 1.2 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

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

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

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

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

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

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

$$W(n) = \max\{t_i | 1 \leq i \leq n+1\} = n$$

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率, 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。

本章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。

下面通过两个实例来说明对同一批数据用不同的表示方法后, 对处理效率的影响。

例 1.3 无序表的顺序查找与有序表的对分查找。

图 1.1 是两个子表。从图中可以看出, 在这两个子表中所存放的数据元素是相同的, 但它们在表中存放的顺序是不同的。在图 1.1 (a) 所示的表中, 数据元素的存放顺序是没有规则的; 而在图 1.1 (b) 所示的表中, 数据元素是按从小到大的顺序存放的。我们称前者为无序表, 后者为有序表。

下面讨论在这两种表中进行查找的问题。

首先讨论在图 1.1 (a) 所示的无序表中进行查找。由于在图 1.1 (a) 表中数据元素的存放顺序没有一定的规则, 因此, 要在这个表中查找某个数时, 只能从第一个元素开始, 逐个将表中的元素与被查数进行比较, 直到表中的某个元素与被查数相等 (即查找成功) 或者表中所有元素与被查数都进行了比较且都不相等 (即查找失败) 为止。这种查找方法称为顺序查找。显然, 在顺序查找中, 如果被查找数在表的前部, 则需要比较的次数就少; 但如果被查找数在表的后部, 则需要比较的次数就多。特别是当被查找数刚好是表中的第一个元素时 (如被查数为 35), 只需要比较一次就查找成功; 但当被查找数刚好是表中最后一个元素 (如被查数为 46) 或表中根本就没有被查数时 (如被查数为 67), 则需要与表中所有的元素进行比较, 在这种情况下, 当表很大时, 顺序查找是很费时间的。虽然顺序查找法的效率比较低, 但由于图 1.1 (a) 为无序表, 没有更好的查找方法, 只能用顺序查找。

35
16
78
85
43
29
33
21
54
46

(a) 无序表

16
21
29
33
35
43
46
54
78
85

(b) 有序表

图 1.1 数据元素存放顺序不同的两个表

现在再讨论在图 1.1 (b) 所示的有序表中进行查找。由于有序表中的元素是从小到大进行排列的, 在查找时可以利用这个特点, 以便使比较次数大大减少。在有序表中查找一个数可以如下进行:

将被查数与表中的中间元素进行比较: 若相等, 则表示查找成功, 查找过程结束; 若被查数大于表中的这个中间元素时, 则表示如果被查数在表中的话, 只能在表的后半部, 此时可以抛弃表的前半部而保留后半部; 若被查数小于表中的这个中间元素, 则表示如果被查数在表中的话, 只能在表的前半部, 此时可以抛弃表的后半部而保留前半部。然后对剩下的部分 (前半部或后半部) 再按照上述方法进行查找, 这个过程一直做到在某一次的比较中相等 (查找成功) 或剩下的部分已空 (查找失败) 为止。例如, 如果要在图 1.1 (b) 所示的有序表中查找 54, 则首先与中间元素 35 进行比较, 由于 54 大于 35, 再与后半部分的中间元素 54 进行比较, 此时相等, 共比较了 2 次就查找成功。如果采用顺序查找法, 在图 1.1 (a) 所示的无序表中查找 54 这个元素, 需要比较 9 次。这种查找方法称为有序表的对分查找。

显然, 在有序表的对分查找中, 不论查找的是什么数, 也不论要查找的数在表中有没有, 都不需要与表中所有的元素进行比较, 只需要与表中很少的元素进行比较。但需要指出的是, 对分查找只适用于有序表, 而对于无序表是无法进行对分查找的。

实际上, 在日常工作和学习中也经常遇到对分查找。例如, 当需要在词典中查找一个单词时, 一般不是从第一页开始一页一页的往后找, 而是考虑到词典中的各单词是以英文字母为顺序排列的, 因此可以根据所查单词的第一个字母, 直接翻到大概的位置, 然后进行比较, 根据比较结果再向前或向后翻, 直到找到该单词为止。这种在词典中查单词的方法类似于对分查找。

由这个例子可以看出, 数据元素在表中的排列顺序对查找效率是有很影响的。

例 1.4 设有一学生情况登记表如表 1.1 所示。在表 1.1 中, 每个学生的情况是以学号为顺序排列的。

显然,如果要在表 1.1 中查找给定学号的某学生的情况是很方便的,只要根据给定的学号就可以立即找到该学生的情况。但是,如果要在该表中查找成绩在 90 分以上的所有学生的情况,则需要从头到尾扫描全表,才能将成绩在 90 分以上的所有学生找到。在这种情况下,为了找到成绩在 90 分以上的学生情况,对于成绩在 90 分以下的所有学生情况也都要被扫描到。由此可以看出,要在表 1.1 中查找给定学号的学生情况虽然很方便,但要查找成绩在某个分数段中的学生情况时,实际上需要查看表中所有学生的成绩,其效率是很低的,尤其是当表很大时更为突出。

表 1.1 学生情况登记表

学 号	姓 名	性 别	年 龄	成 绩	学 号	姓 名	性 别	年 龄	成 绩
970156	张小明	男	20	86	970163	王 伟	男	20	65
970157	李小青	女	19	83	970164	胡 涛	男	19	95
970158	赵 凯	男	19	70	970165	周 敏	女	20	87
970159	李启明	男	21	91	970166	杨雪辉	男	22	89
970160	刘 华	女	18	78	970167	吕永华	男	18	61
970161	曾小波	女	19	90	970168	梅 玲	女	17	93
970162	张 军	男	18	80	970169	刘 健	男	20	75

为了便于查找成绩在某个分数段中的学生情况,可以将表 1.1 中所登记的学生情况进行重新组织。例如,将成绩在 90 分以上(包括 90 分,下同)、80~89 分、70~79 分、60~69 分之间的学生情况分别登记在四个独立的子表中,分别如表 1.2、表 1.3、表 1.4 与表 1.5 所示。现在如果要查找 90 分以上的所有学生的情况,就可以直接在表 1.2 中进行查找,从而避免了对成绩在 90 分以下的学生情况进行扫描,提高了查找效率。

表 1.2 成绩在 90 分以上的学生情况登记表

学 号	姓 名	性 别	年 龄	成 绩	学 号	姓 名	性 别	年 龄	成 绩
970159	李启明	男	21	91	970164	胡 涛	男	19	95
970161	曾小波	女	19	90	970168	梅 玲	女	17	93

表 1.3 成绩在 80~89 分之间的学生情况登记表

学 号	姓 名	性 别	年 龄	成 绩	学 号	姓 名	性 别	年 龄	成 绩
970156	张小明	男	20	86	970165	周 敏	女	20	87
970157	李小青	女	19	83	970166	杨雪辉	男	22	89
970162	张 军	男	18	80					

表 1.4 成绩在 70~79 分之间的学生情况登记表

学 号	姓 名	性 别	年 龄	成 绩	学 号	姓 名	性 别	年 龄	成 绩
970158	赵 凯	男	19	70	970169	刘 健	男	20	75
970160	刘 华	女	18	78					

表 1.5 成绩在 60~69 分之间的学生情况登记表

学 号	姓 名	性 别	年 龄	成 绩	学 号	姓 名	性 别	年 龄	成 绩
970163	王 伟	男	20	65	970167	吕永华	男	18	61

由例 1.4 可以看出,在对数据进行处理时,可以根据所做的运算不同,将数据组织成不同的形式,以便于做该种运算,从而提高数据处理的效率。

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