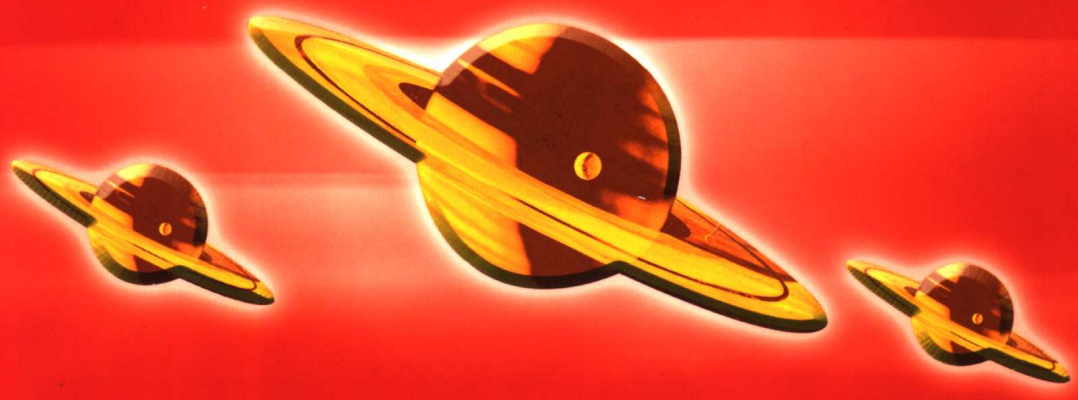
全国计算机等级考试 名师名导

谭浩强 主编

二级公共基础教程

徐士良 编著



- 名师编著，紧扣最新大纲，精辟讲解
- 专家指导，令您事半功倍，轻松掌握
- 内容全面，教学自学培训，考生必备

清华大学出版社

二级公共基础教程

徐士良　编著

全国计算机等级考试名师名导

谭浩强 主编

清华大学出版社
北 京

内 容 简 介

本书是根据教育部考试中心2004年最新制定的《全国计算机等级考试大纲》中对二级公共基础部分的要求编写的。书中内容紧扣考试大纲，并对考试大纲所要求的数据结构与算法、程序设计基础、软件工程基础以及数据库设计基础等内容从概念到应用进行了详细介绍，为考生的学习和备考可起到引导和帮助作用。

本书不仅是应试者必备的学习教材，也可以作为高等院校相应课程的教材或参考书。

图书在版编目(CIP)数据

二级公共基础教程/徐士良编著. —北京：清华大学出版社，2004

(全国计算机等级考试名师名导/谭浩强主编)

ISBN 7-302-08660-5

Ⅰ. 二… Ⅱ. 徐… Ⅲ. 电子计算机－水平考试－自学参考资料 Ⅳ. TP3

中国版本图书馆CIP数据核字(2004)第046956号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦

http://www.tup.com.cn　　**邮　　编**：100084

社 总 机：010-62770175　　**客户服务**：010-62776969

责任编辑：索　梅

封面设计：艺铭设计

印 刷 者：北京四季青印刷厂

装 订 者：三河市金元装订厂

发 行 者：新华书店总店北京发行所

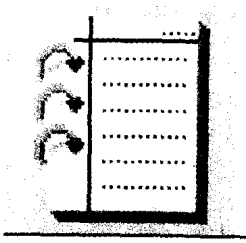
开　　本：185×260　**印张**：8.5　**字数**：195千字

版　　次：2004年6月第1版　2004年6月第1次印刷

书　　号：ISBN 7-302-08660-5/TP·6211

印　　数：1～5000

定　　价：12.00元



序

跨入21世纪，我国已掀起了第三次计算机普及高潮。在这次高潮中，将向一切有文化的人普及计算机知识和应用。随着社会主义市场经济的发展，近年来面向社会和面向学校的各种计算机考试如雨后春笋般涌现。许多人认为，学历是从整体上反映了一个人的知识水平，而证书则反映了一个人在某一方面的能力。证书制度是学历制度必要的补充，符合人才市场的需要，因而受到各方面的欢迎。

在众多的计算机考试中，由国家教育部考试中心主办的“全国计算机等级考试”是最权威、影响最广、最受欢迎的一种社会考试。自1994年推出了“全国计算机等级考试”以来，至2003年底，累计已有1000多万人报名参加考试，其中300多万人获得了等级证书。不少单位已经把通过全国计算机等级考试作为任职或晋升的条件。

全国许多地区和部门也组织了本地区或本系统的计算机统一测试。考试内容和方法大多与全国计算机等级考试类似。

随着计算机应用技术的发展，教育部考试中心对全国计算机等级考试的考试科目、考核内容和考试形式进行了一定程度的调整，推出了新的2004年版《考试大纲》。调整后的全国计算机等级考试分为四个等级：

一级：要求具有计算机的初步知识和使用办公软件及因特网(Internet)的初步能力。包括以下内容：

一级 MS Office

一级 WPS Office

一级 B

(可从中任选一种应试)

二级：要求具有计算机基础知识和语言程序设计或数据库程序设计及上机调试的能力。包括以下内容：

二级 C 语言程序设计

二级 C ++ 语言程序设计

二级 Java 语言程序设计

二级 Visual Basic 语言程序设计

二级 Visual FoxPro 数据库程序设计

二级 Access 数据库程序设计

(可从中任选一种应试)

三级：要求具有计算机应用基础知识和计算机硬件系统或软件系统开发的初步能力。包括以下内容：

三级 PC 技术

三级 信息管理技术

三级 网络技术

三级 数据库技术

四级：要求具备深入而系统的计算机知识和较高的计算机应用能力。

为了帮助广大应考者准备考试，我们于 1998 年和 2002 年分别根据当时的大纲编写出版了《全国计算机等级考试丛书》，由清华大学出版社出版，很受读者欢迎。根据考试内容的变化以及多年来我们所积累的经验，我们对本丛书进行了必要的调整和补充。该丛书由以下五个系列构成。

(1) 教程系列：全面而系统地介绍考试大纲所规定的内容。

(2) 应试辅导系列：概括而简洁地介绍知识点及考试难点，提供经典例题解析、练习题、模拟试卷及相应的参考答案。

(3) 样题汇编系列：按照全国计算机等级考试的内容和试题形式，提供了大量样题及其参考答案，供应试者选用。

(4) 上机考试指导系列：提供全真上机考试环境光盘，指导上机考试应试技巧，详细解析大量上机试题及相关程序设计方法。

(5) 全真模拟试卷系列：提供身临其境的考试样卷(包括笔试模拟试卷、上机模拟试题)以及参考答案。

本丛书不仅适用于全国计算机等级考试，也适用于内容类似的其他计算机统一考试，对大中学生和其他计算机学习者也有一定的参考价值。

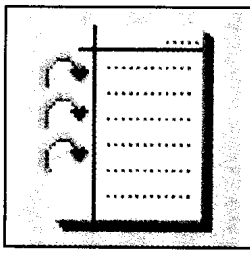
丛书中各书的作者都是高等学校或计算机应用部门中具有丰富教学经验并对计算机等级考试有较深入研究的教授、专家。相信该丛书的出版一定会受到广大准备参加计算机等级考试读者的欢迎。

欢迎读者对本丛书提出宝贵意见，以便不断完善。

《全国计算机等级考试名师名导》主编

谭浩强

2004 年 5 月



前　言

本书是《全国计算机等级考试名师名导》之一。编写本书的根据是教育部考试中心2004年最新颁布的《全国计算机等级考试大纲》中关于二级公共基础部分的要求。

根据新的大纲要求，对于二级考试中的所有科目增加基础知识部分，并且对基础知识统一要求，使用统一的基础知识大纲和教程。二级基础知识的内容在原有的基础上做了重大调整，调整后的二级基础知识主要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4个部分。二级基础知识在各科笔试中占30%的题量。

本书共分以下4章，每一章的内容都是按照大纲要求进行编写的。

第1章是基本数据结构与算法。在这一章中，主要介绍算法的基本概念、数据结构的基本概念及其定义，并具体介绍了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、二叉树的基本概念和存储结构及其遍历。最后还介绍了几种常用的查找与排序算法。

第2章是程序设计基础。在这一章中，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章是软件工程基础。在这一章中，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章是数据库设计基础。在这一章中，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。

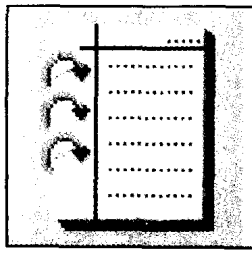
每章后面都附有一定数量的习题，习题的形式与实际考试时的笔试题相同。

本书的特点是内容精炼，语言通俗易懂。它不仅可以满足全国计算机等级考试(二级公共基础知识)的需要，而且也可以作为一般院校相应课程的教材或自学参考书。

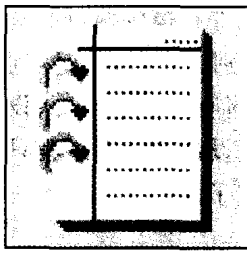
由于时间紧迫以及作者水平有限，书中的错误或不妥之处，恳请读者批评指正。

作　者

2004年5月



目　录



第 1 章
基本数据结构与算法

1.1 算法

计算机科学的发展，为科学计算与数据处理提供了高速和高精度的计算工具。但计算机在本质上只能机械地执行人的命令，它本身不会主动地进行思维，也不可能发挥任何创造性。因此，在用计算机解决问题时，首先要进行程序设计。通常，程序设计主要包括两个方面，一是行为特性的设计，二是结构特性的设计。所谓行为特性的设计，一般是指将解决问题过程中的每一个细节准确地加以定义，并将全部的解题过程用某种工具完整地描述出来。这一过程称为算法的设计。所谓结构特性的设计，是指为问题的解决确定合适的数据结构。数据结构与算法之间有着密切的关系。特别是对于数据处理问题，算法的效率通常与数据结构在计算机中的表示有着直接的关系。本节主要介绍算法的基本概念以及与算法效率有关的问题。

1.1.1 算法的基本概念

所谓算法是指对解题方案的准确而完整的描述。

通常，算法分为数值型算法与非数值型算法两种。非数值型算法又称为符号处理。

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

算法实际上是一种抽象的解题方法，它具有动态性。作为一个算法，一般应具有以下几个基本特征。

(1) 能行性(effectiveness)

算法的能行性主要包括两个方面。一是算法中的每一个步骤必须是能实现的，例如，

在算法中，不允许出现分母为零的情况，在实数范围内不能求一个负数的平方根等；二是算法执行的结果要能达到预期的目的。通常，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。因此，在设计一个算法时，必须要考虑它的能行性，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有明确定义，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。例如，对于指令“如果小明是大学生，则输出字母 Y，否则输出字母 N”，如果在该指令的执行过程中提供了小明一定不是大学生的某种信息时(如提供了所有不是大学生的名单，小明被列在其中)，执行的结果将输出字母 N；如果只提供了部分大学生的名单，且小明恰在其中，则执行的结果将输出字母 Y。但如果在提供的部分大学生的名单中找不到小明的名字，则在执行该指令时就无法确定小明是否是大学生。

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，算法并不有效。

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，而且是明确的，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 4 类：

- 算术运算　主要包括加、减、乘、除等运算。
- 逻辑运算　主要包括“与”、“或”、“非”等运算。
- 关系运算　主要包括“大于”、“小于”、“等于”、“不等于”等运算。
- 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 4 种基本功能操作考虑，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地加工和变换，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 结构化流程图及算法描述语言等。一个算法一般都可以用顺序、选择及循环 3 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。

这里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，应该重点注意使方案优化及尽量减少运算工作量。通常，在设计列举算法时，只要对实际问题进行详细分析，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，即可以大大减少列举量。

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），局部地使用列举法是很有效的。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，最后总结出有用的结论或解决问题的有效途径。

例 1.1　求前 n 个自然数平方的和

$$S_n=1^2+2^2+3^2+\cdots+n^2$$

为了导出 S_n 与 n 之间的一般关系，波利亚（G. Polya）曾提出如表 1.1 所示的归纳模式。

表 1.1　波利亚归纳模式

N	1	2	3	4	5	6	…
$1+2+3+\cdots+n$	1	3	6	10	15	21	…
$1^2+2^2+3^2+\cdots+n^2$	1	5	14	30	55	91	…
$\dfrac{1^2+2^2+3^2+\cdots+n^2}{1+2+3+\cdots+n}$	3/3	5/3	7/3	9/3	11/3	13/3	…

由表 1.1 很容易看出其中的规律性，即可以得到如下猜测：

$$\frac{1^2+2^2+3^2+\cdots+n^2}{1+2+3+\cdots+n}=(2n+1)/3$$

再由

$$1+2+3+\cdots+n=n(n+1)/2$$

可以得出最终的猜测如下：

$$1^2+2^2+3^2+\cdots+n^2=n(n+1)(2n+1)/6$$

实践证明，自然数平方之和的这个猜测是正确的。

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得到确定。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。

例 1.2　斐波那契(Fibonacci)数列。

斐波那契是意大利数学家，他在 1202 年出版的《算盘的书》中提出这样一个问题：假定一对刚出生的小兔子，一个月后就能长成大兔子，再过一个月就开始生小兔子，并且也正好生一对兔子(一雄一雌)，问在没有兔子死亡的情况下，一对初生的兔子一年后可繁殖成多少对兔子？

用 $F(n)$表示第 n 个月的兔子对数。

第 1 个月有 1 对兔子，即 $F(1)=1$。过一个月后(即第 2 个月)它们长大，但还没有生小兔子，因此，第 2 个月也只有 1 对兔子，即 $F(2)=1$。第 3 个月它们开始生下一对小兔子，加上原来的一对老兔子，共有 2 对兔子，即 $F(3)=2$。第 4 个月时，原来的 1 对老兔子又生下 1 对小兔子，而第 3 个月生下的 1 对兔子长大但还不能生小兔子，因此，共有 3 对兔子，即 $F(4)=3$。第 5 个月时，原来的 1 对老兔子又生下 1 对小兔子，第 3 个月生下的兔子也生下 1 对小兔子，而第 4 个月生下的 1 对小兔子长大但还不能生小兔子，因此，共有 5 对兔子，即 $F(5)=5$……一直递推到第 13 个月，便可以得到以下数列：

1,1,2,3,5,8,13,21,34,55,89,144,233

这个数列称为斐波那契数列。

显然，当 n 较大时，按照上面的方法逐步往后推算就显得太麻烦了。实际上，只要稍微进行分析，就可以知道，第 n 个月的兔子对数 $F(n)$由两部分组成：一是第 $n-1$ 个月的兔子对数 $F(n-1)$；二是第 n 个月具有生殖能力的兔子所生下的小兔子对数，而第 n 个月具有生殖能力的兔子对数应是第 $n-2$ 个月已经有的兔子对数，即 $F(n-2)$。由此可知，第 n 个月的兔子对数应为 $F(n)=F(n-1)+F(n-2)$。

综上所述，可以得到如下的递推关系：

$$F(1)=1$$
$$F(2)=1$$
$$F(n)=F(n-1)+F(n-2) \qquad (n=3,4,\cdots)$$

其中，$F(1)=1$，$F(2)=1$ 称为递推的初始条件，$F(n)=F(n-1)+F(n-2)$称为递推公式。

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。

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每一个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

解决实际问题的复杂程度往往与问题的规模有着密切的关系。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。

二分法求方程实根的减半递推过程如下。

① 取给定区间的中点 $c=(a+b)/2$。

② 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。

③ 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 $|a-b|\geqslant\varepsilon$，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，其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。

例 1.4　求解皇后问题。

由 n^2 个方块排成 n 行 n 列的正方形称为“n 元棋盘”。如果两个皇后位于棋盘上的同一行或同一列或同一对角线上，则称它们为互相攻击。现要求找出使 n 元棋盘上的 n 个皇后互不攻击的所有布局。

n 个皇后在 n 元棋盘上的布局共有 n^n 种，为了从中找出互不攻击的布局，需要对此 n^n 种方案逐个进行检查，将有攻击的布局剔除掉。这是一种列举法，但这种方法对于较大的 n，其运算量会急剧地增加。实际上，逐一列举是没有必要的。例如，如果第 1 行上的皇后在第 1 列，则第 2 行上的皇后就不可能在第 1 列。

下面用回溯法来求解皇后问题。

假设棋盘上的每一行放置一个皇后，分别用自然数进行编号为 $1,2,\cdots,n$。

首先定义一个长度为 n 的一维数组 q，其中每一个元素 $q[i](i=1,2,\cdots,n)$ 随时记录第 i 行上的皇后所在的列号。

初始时，先将各皇后放在各行的第 1 列，即数组 q 的初值为

$$q[i]=1 \quad (i=1,2,\cdots,n)$$

第 i 行与第 j 行上的皇后在某一对角线上的条件为

$$|q[i]-q[j]|=|i-j|$$

而它们在同一列上的条件为

$$q[i]=q[j]$$

回溯法的步骤如下：

从第 1 行(即 $i=1$)开始进行以下过程。

设前 $i-1$ 行上的皇后已经布局好，即它们均互不攻击。现在考虑安排第 i 行上的皇后的位置，使之与前 $i-1$ 行上的皇后也都互不攻击。为了实现这一目的，可以从第 i 行皇后的当前位置 $q[i]$ 开始按如下规则向右进行搜索。

① 若 $q[i]=n+1$，则说明第 i 个皇后暂时无法安排，将第 i 个皇后放在第 1 列(即置 $q[i]=1$)，且回退一行，考虑重新安排第 $i-1$ 行上的皇后与前 $i-2$ 行上的皇后均互不攻击的下一个位置。此时如果已退到第 0 行(实际没有这一行)，则过程结束。

② 若 $q[i]\leqslant n$，则需检查第 i 行上的皇后与前 $i-1$ 行的皇后是否互不攻击。若有攻击，则将第 i 行上的皇后右移一个位置(即置 $q[i]=q[i]+1$)，重新进行这个过程；若无攻击，则考虑安排下一行上的皇后位置，即置 $i=i+1$。

③ 若当前安排好的皇后是在最后一行(即第 n 行)，则说明已经找到了 n 个皇后互不攻击的一个布局，将这个布局输出(即输出 $q[i](i=1,2,\cdots,n)$)，然后将第 n 行上的皇后右移一个位置(即置 $q[n]=q[n]+1$)，重新进行这个过程，以便寻找下一个布局。

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两种。

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）运算可以忽略不计。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，比较两个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1个元素开始，逐个与被查值 x 进行比较。显然，如果第1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带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加以确定，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。