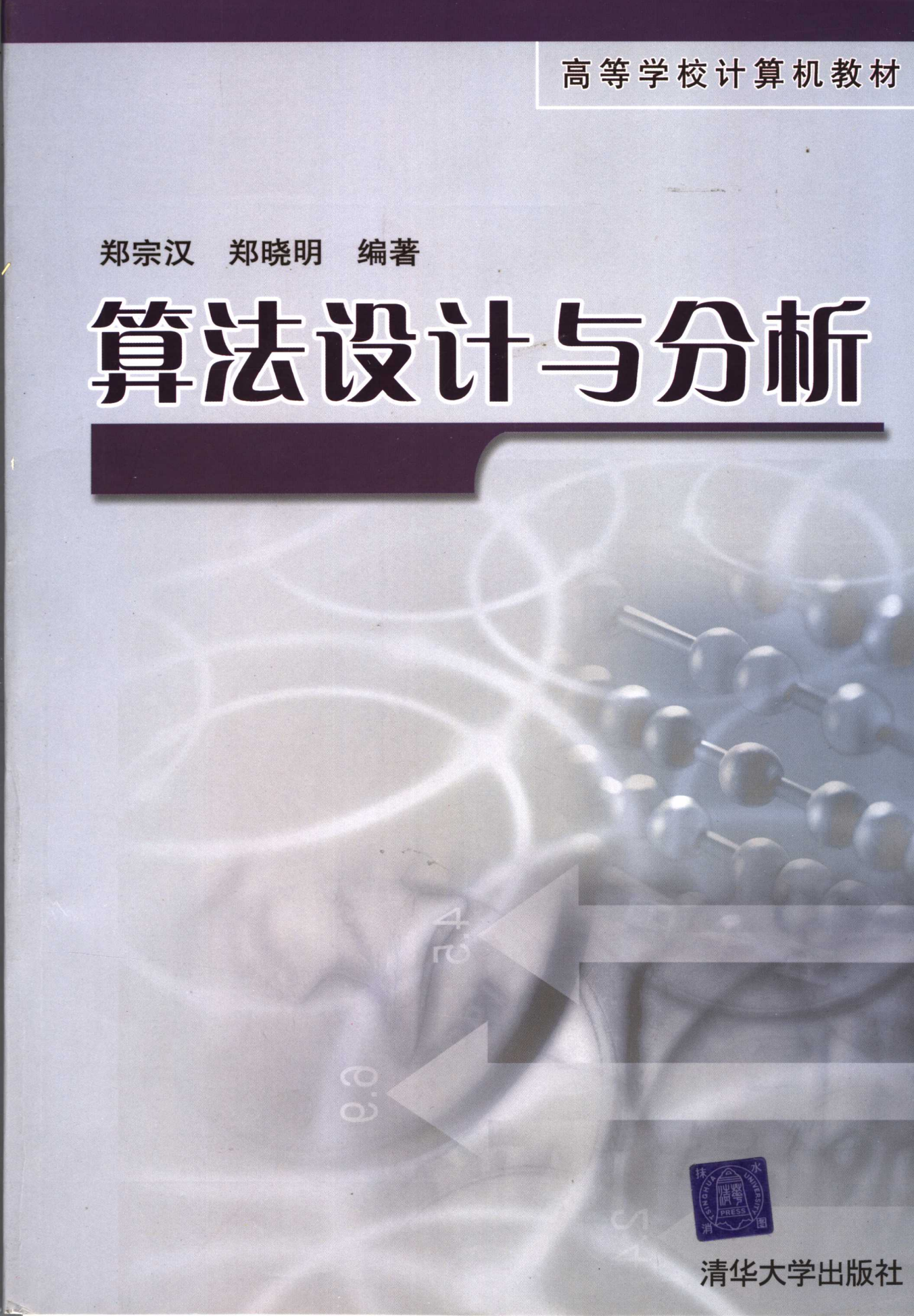
高等学校计算机教材

郑宗汉　郑晓明　编著

算法设计与分析



清华大学出版社

高等学校计算机教材

算法设计与分析

郑宗汉　郑晓明　编著

清华大学出版社
北　京

内 容 简 介

本书系统地介绍算法设计与分析的概念和方法，共四部分内容，第一部分包括前两章，介绍算法设计与分析的基本概念及必要的数学工具，对算法的时间复杂性的概念及算法的分析方法作了较为详细的叙述。第二部分包括第3~9章，以算法设计技术为纲，从排序问题和离散集合的操作开始，进而介绍递归技术、分治法、贪婪法、动态规划、回溯法、分支与限界法以及随机算法等算法设计技术及其复杂性。第三部分包括第10章和第11章，介绍计算机应用领域里的一些算法，如图和网络中的一些问题，以及计算几何中的一些问题。第四部分包括第12~15章，介绍算法设计与分析中的一些理论问题，如 *NP* 完全问题、计算复杂性问题、下界理论问题，最后介绍了近似算法及其性能分析。

本书内容选材适当，编排合理，由浅入深，循序渐进，互相衔接，逐步展开。可作为高等院校计算机专业本科生和研究生的教材，也可作为计算机科学与应用的科学技术人员的参考资料。

图书在版编目（CIP）数据

算法设计与分析/郑宗汉，郑晓明编著. —北京：清华大学出版社，2005.6
ISBN 7-302-10894-3

Ⅰ. 算… Ⅱ. ①郑… ②郑… Ⅲ. ①电子计算机-算法设计-高等学校-教材 ②电子计算机-算法分析-高等学校-教材 Ⅳ. TP301.6

中国版本图书馆 CIP 数据核字（2005）第 037973 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
组稿编辑：许存权
文稿编辑：马晓娟
封面设计：姜凌娜
版式设计：冯彩茹
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：23.5　**字数**：518 千字
版　　次：2005 年 6 月第 1 版　2005 年 6 月第1 次印刷
书　　号：ISBN 7-302-10894-3/TP · 7247
印　　数：1 ~ 4000
定　　价：32.00 元

前　言

计算机系统中的任何软件，都是按一个个特定的算法来予以实现的。算法性能的好坏，直接决定了所实现软件性能的优劣。如何判定一个算法的性能、用什么方法来设计算法、所设计的算法需要多少运行时间、多少存储空间，在实现一个软件时，这些都是必须予以解决的。计算机的操作系统、语言编译系统、数据库管理系统以及各种各样的计算机应用系统软件，都离不开用具体的算法来实现。因此，算法设计与分析是计算机科学与技术的一个核心问题，也是大学计算机专业本科生及研究生的一门重要的专业基础课程。通过算法设计与分析这门课程的学习，使读者能够掌握算法设计与分析的方法，利用这些方法去解决在计算机科学与技术中所遇到的各种问题，去设计计算机系统的各种软件中所可能遇到的算法，并对所设计的算法作出科学的评价。因此，算法设计与分析，不仅对计算机专业的科学技术人员，而且对使用计算机的其他专业技术人员，都是非常重要的。

本书的内容选材适当，循序渐进，互相衔接，逐步展开。先以最简单的穷举法为例，说明算法设计技术及算法分析的重要性；接着以排序问题中的某些算法为例，说明算法分析的一般方法；然后以算法设计技术为纲，按照所叙述算法的思想方法、实现步骤、所涉及到的数据结构、算法的具体描述以及复杂性分析等几个方面，逐个介绍各种算法设计技术及其分析方法。

全书分为四部分，第一部分包括第 1 章和第 2 章，介绍算法设计与分析的基本概念。第 1 章介绍算法的定义及算法设计与分析的基本概念，特别是对时间复杂性的概念及算法的分析方法作了较为详细的叙述。第 2 章简单叙述与算法分析有关的最基本的数学工具。第二部分包括第 3~9 章，介绍算法设计的基本技术。第 3 章介绍排序问题和离散集合的操作，进一步对算法分析进行了阐述，并为下面各章中所涉及到的问题作了技术上的准备。第 4 章介绍递归技术及分治方法，从理论上分析了分治算法的效率。第 5 章介绍贪婪法的设计方法及其正确性的证明。第 6 章介绍动态规划法的设计技术。第 7 章介绍回溯法的设计技术。第 8 章在回溯法的基础上，介绍分支与限界方法的应用及其分析。第 9 章介绍 3 种类型的随机算法及其性能分析。第三部分包括第 10 章和第 11 章，介绍计算机应用领域里的一些算法。第 10 章介绍图和网络中的一些问题。第 11 章介绍计算几何中的一些问题。第四部分包括第 12~15 章，介绍算法设计与分析中的一些理论问题。第 12 章介绍 *NP* 完全问题。第 13 章介绍计算复杂性问题。第 14 章介绍下界理论问题。第 15 章介绍近似算法及其性能分析。

本书是在许存权编辑的大力支持和努力下才得以出版的；王淑娟同志为本书做了大量的工作，在此一并表示诚挚的谢意。

由于水平有限，书中不当之处，敬请读者指正。

编者

2004 年 11 月

目　录

第 1 章　算法的基本概念

计算机系统中的任何软件，都是由大大小小的各种软件组成部分构成，各自按照特定的算法来实现，算法的好坏直接决定所实现软件性能的优劣。用什么方法来设计算法，所设计算法需要什么样的资源，需要多少运行时间、多少存储空间，如何判定一个算法的好坏，在实现一个软件时，都是必须予以解决的。计算机系统中的操作系统、语言编译系统、数据库管理系统以及各种各样的计算机应用系统中的软件，都必须用一个个具体的算法来实现。因此，算法设计与分析是计算机科学与技术的一个核心问题。

1.1　引　　言

在 20 世纪 50 年代，西方某些著名的词典中，还未曾收录过算法（Algorithm）一词。根据西方数学史家的考证，古代阿拉伯的一位学者写了一部名著，名字为“Kitāb al-jabr Wa'lmuqābɪla”（《复原和化简的规则》），作者的署名是 Abū 'Abd Allāh Muhammad ibn Mūsa al-Khwārizmī，从字面看，意思是“穆罕默德（Muhammad）的父亲，摩西（Moses）的儿子，Khwārizm 地方的人”。后来，这部著作流传到了西方，结果，从作者署名中的 al-Khwārizmī 派生出 Algorithm（算法）一词，而从作品名字中的 al-jabr 派生出 Algebra（代数）一词。随着时间的推移，Algorithm（算法）这个词的含义，已经完全与它原来的含义不同了。

1.1.1　算法的定义和特征

欧几里德曾在他的著作中描述过求两个数的最大公因子的过程。20 世纪 50 年代，欧几里德所描述的这个过程，被称为欧几里德算法，算法这个术语在学术上便具有了现在的含义。下面是这个算法的例子及它的一种描述。

算法 1.1　欧几里德算法
输入：正整数 m,n
输出：m,n 的最大公因子

```
1. int euclid(int m,int n)
2. {
3.     int r;
4.     do {
5.         r = m % n;
6.         m = n;
7.         n = r;
8.     } while(r)
```

```
 9.     return m;
10. }
```

在此用一种类C语言来叙述最大公因子的求解过程。今后，在描述其他算法时，还可能结合一些自然语言的描述，以代替某些繁琐的具体细节，而更好地说明算法的整体框架。同时，为了简明地访问二维数组元素，假定在函数调用时，二维数组可以作为参数传递，在函数中可以动态地分配数组。读者可以很容易地用其他方法来消除这些假定。

在算法的第5行，把m除以n的余数赋予r，第6行把n的值赋予m，第7行把r的值赋予n，第8行判断r是否为0，若非0，继续转到第5行进行处理；若为0，就转到第9行处理，第9行返回m，算法结束。按照上面这组规则，给定任意两个正整数，总能返回它们的最大公因子。可以证明这个算法的正确性。

根据上面这个例子，可以给算法如下的定义：

定义1.1 算法是解某一特定问题的一组有穷规则的集合。

算法设计的先驱者唐纳德.E.克努特（Donald E.Knuth）对算法的特征作了如下的描述：

（1）有限性。算法在执行有限步之后必须终止。算法1.1中，对输入的任意正整数m、n，在m除以n的余数赋予r之后，再通过r赋予n，从而使n值变小。如此往复进行，最终或者使r为0，或者使n递减为1。这两种情况，都最终使$r=0$，而使算法终止。

（2）确定性。算法的每一个步骤，都有精确的定义。要执行的每一个动作都是清晰的、无歧义的。例如，在算法1.1的第5行中，如果m、n是无理数，那么，m除以n的余数是什么，就没有一个明确的界定。确定性的准则意味着必须确保在执行第5行时，m和n的值都是正整数。算法1.1中规定了m、n都是正整数，从而保证了后续各个步骤中都能确定地执行。

（3）输入。一个算法有0个或多个输入，它是由外部提供的，作为算法开始执行前的初始值，或初始状态。算法的输入是从特定的对象集合中抽取的。算法1.1中有两个输入m、n，就是从正整数集合中抽取的。

（4）输出。一个算法有一个或多个输出，这些输出与输入有特定的关系，实际上是输入的某种函数。不同取值的输入，产生不同结果的输出。算法1.1中的输出是输入m、n的最大公约数。

（5）能行性。算法的能行性指的是算法中有待实现的运算，都是基本的运算。原则上可以由人们用纸和笔，在有限的时间里精确地完成。算法1.1用一个正整数来除另一个正整数，判断一个整数是否为0以及整数赋值等，这些运算都是能行的。因为整数可以用有限的方式表示，而且，至少存在一种方法来完成一个整数除以另一个整数的运算。如果所涉及的数值必须由展开成无穷小数的实数来精确地完成，则这些运算就不是能行的了。

必须注意到，在实际应用中，有限性的限制是不够的。一个实用的算法，不仅要求步骤有限，同时要求运行这些步骤所花费的时间是人们可以接受的。如果一个算法需要执行数十百亿亿计的运算步骤，从理论上说，它是有限的，最终可以结束，但是，以当代计算机每秒数亿次的运算速度，也必须运行数百年以上时间，这是人们所无法接受的，因而是不实用的算法。

算法设计的整个过程，可以包含对问题需求的说明、数学模型的拟制、算法的详细设计、

算法的正确性验证、算法的实现、算法分析、程序测试和文档资料的编制。本书所关心的是串行算法的设计与分析，其他相关的内容以及并行算法，可参考专门的书籍。这里，只在涉及有关内容时，才对相应的内容进行论述。

1.1.2　算法设计的例子，穷举法

例 1.1　百鸡问题。

公元 5 世纪末，我国古代数学家张丘建在他所撰写的《算经》中，提出了这样的一个问题："鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一。百钱买百鸡，问鸡翁、母、雏各几何？"意思是公鸡每只 5 元、母鸡每只 3 元、小鸡 3 只 1 元，用 100 元钱买 100 只鸡，求公鸡、母鸡、小鸡的只数。

令 a 为公鸡只数，b 为母鸡只数，c 为小鸡只数。根据题意，可列出下面的约束方程：

$$a+b+c=100 \tag{1.1.1}$$

$$5a+3b+c/3=100 \tag{1.1.2}$$

$$c\%3=0 \tag{1.1.3}$$

其中，运算符"/"为整除运算，"%"为求模运算，式（1.1.3）表示 c 被 3 除余数为 0。

这类问题用解析法求解有困难，但可用穷举法来求解。穷举法就是从有限集合中，逐一列举集合的所有元素，对每一个元素逐一判断和处理，从而找出问题的解。

上述百鸡问题中，a、b、c 的可能取值范围为 0~100，对在此范围内的 a、b、c 的所有组合进行测试，凡是满足上述 3 个约束方程的组合，都是问题的解。如果把问题转化为用 n 元钱买 n 只鸡，n 为任意正整数，则式（1.1.1）、式（1.1.2）变为：

$$a+b+c=n \tag{1.1.4}$$

$$5a+3b+c/3=n \tag{1.1.5}$$

于是，可用下面的算法来实现：

算法 1.2　百鸡问题
输入：所购买的 3 种鸡的总数目 n
输出：满足问题的解的数目 k，公鸡，母鸡，小鸡的只数 g[],m[],s[]

```
 1. void chicken_question(int n,int &k,int g[],int m[],int s[])
 2. {
 3.     int  a,b,c;
 4.     k = 0;
 5.     for (a=0;a<=n;a++){
 6.         for (b=0;b<=n;b++){
 7.             for (c=0;c<=n;c++) {
 8.                 if ((a+b+c==n)&&(5*a+3*b+c/3==n)&&(c%3==0)) {
 9.                     g[k] = a;
10.                     m[k] = b;
11.                     s[k] = c;
12.                     k++;
13.                 }
14.             }
```

```
15.         }
16.     }
17. }
```

这个算法有三重循环，主要执行时间取决于第 7 行开始的内循环的循环体的执行次数。外循环的循环体执行 $n+1$ 次，中间循环的循环体执行 $(n+1)(n+1)$ 次，内循环的循环体执行 $(n+1)(n+1)(n+1)$ 次。当 $n=100$ 时，内循环的循环体执行次数大于 100 万次。

考虑到 n 元钱只能买到 $n/5$ 只公鸡，或 $n/3$ 只母鸡。因此，有些组合可以不必考虑。而小鸡的只数又与公鸡及母鸡的只数相关，上述的内循环可以省去。这样，算法 1.2 可以改为：

算法 1.3 改进的百鸡问题
输入： 所购买的 3 种鸡的总数目 n
输出： 满足问题的解的数目 k，公鸡，母鸡，小鸡的只数 g[],m[],s[]

```
1. void chicken_problem(int n,int &k,int g[],int m[],int s[])
2. {
3.     int  i,j,a,b,c;
4.     k = 0;
5.     i = n/5;
6.     j = n/3;
7.     for (a=0;a<=i;a++){
8.        for (b=0;b<=j;b++) {
9.            c = n-a-b;
10.           if ((5*a+3*b+c/3==n)&&(c%3==0)) {
11.               g[k] = a;
12.               m[k] = b;
13.               s[k] = c;
14.               k++;
15.           }
16.       }
17.    }
18. }
```

算法 1.3 有两重循环，主要执行时间取决于第 8 行开始的内循环，其循环体的执行次数是 $(n/5+1)(n/3+1)$。当 $n=100$ 时，内循环的循环体的执行次数为 $21\times34=714$ 次，这和算法 1.2 的 100 万次比较起来，仅是原来的万分之七，有重大的改进。

对某类特定问题，在规模较小的情况下，穷举法往往是一个简单有效的方法。

例 1.2 货郎担问题。

货郎担问题是一个经典问题。某售货员要到若干个城市销售货物，已知各城市之间的距离，要求售货员选择出发的城市及旅行路线，使每一个城市仅经过一次，最后回到原出发城市，而总路程最短。

如果对任意数目的 n 个城市，分别用 1~n 的数字编号，则这个问题归结为在有向赋权图 $G=<V,E>$ 中，寻找一条路径最短的哈密尔顿回路问题。其中，$V=\{1,2,\cdots n\}$，表示城市顶点，边 $(i,j)\in E$ 表示城市 i 到城市 j 的距离，$i,j=1,2,\cdots n$。这样，可以用图的邻接矩阵 C 来

表示各个城市之间的距离，把这个矩阵称为费用矩阵。

售货员的每一条路线，对应于城市编号1,2…n的一个排列。用一个数组来存放这个排列中的数据，数组中的元素依次存放旅行路线中的城市编号。n个城市共有$n!$个排列，于是，售货员共有$n!$条路线可供选择。采用穷举法逐一计算每一条路线的费用，从中找出费用最小的路线，便可求出问题的解。下面是用穷举法解这个问题的算法。

算法 1.4　穷举法版本的货郎担问题
输入：城市个数 n，费用矩阵 c[][]
输出：旅行路线 t[]，最小费用 min

```
1. void salesman_problem(int n,float &min,int t[],float c[][])
2. {
3.     int p[n],i = 1;
4.     float    cost;
5.     min = MAX_FLOAT_NUM;
6.     while (i <= n!) {
7.         产生 n 个城市的第 i 个排列于 p;
8.         cost =  路线 p 的费用;
9.         if (cost < min) {
10.            把数组 p 的内容复制到数组 t;
11.            min = cost;
12.        }
13.        i++;
14.    }
15. }
```

这个算法的执行时间取决于第 6 行开始的 while 循环，它产生一个路线的城市排列，并计算该路线所需要的时间。这个循环的循环体共需执行$n!$次。假定每执行一次，需要 1μs 时间，则整个算法的执行时间随n的增长而增长的情况，如表 1.1 所示。从表中看到，当$n=10$时，运行时间是 3.62s，算法是可行的；当$n=13$时，运行时间是 1.72 小时，还可以接受；当$n=16$时，运行时间是 242 天，就不实用了；当$n=20$时，运行时间是 7 万 7 千多年，这样的算法就不可取了。

表 1.1　算法 1.4 的执行时间随 n 的增长而增长的情况

n	$n!$	n	$n!$	n	$n!$	n	$n!$
5	120μs	9	362ms	13	1.72h	17	11.27year
6	720μs	10	3.62s	14	24h	18	203year
7	5.04ms	11	39.9s	15	15day	19	3857year
8	40.3ms	12	479.0s	16	242day	20	77146year

1.1.3　算法的复杂性分析

上面的第一个例子，说明了改进算法的设计方法对提高算法性能的重要性。第二个例子，说明了对一个不太大的n，采用穷举法来解诸如货郎担一类的问题，都是行不通的。可以采

用哪些算法设计方法，来有效地解决现实生活中的问题，就是本书所要叙述的一个问题。但是，如果不是通过上面的简单分析，就不知道改进版的百鸡问题算法比原来的算法效率高很多；更不会知道用穷举法来解诸如货郎担一类的问题，甚至对一个不太大的n，都是不可行的。把这个问题再引申一下，对所设计的算法，如何说明它是有效的；或者，如果有两个解同样问题的算法，如何知道这一个算法比那一个算法更有效？这就提出了另一个问题，如何分析算法的效率，这是本书所要叙述的另一个问题。

一般来说，可以用算法的复杂性来衡量一个算法的效率。对所设计的算法，普遍关心的是算法的执行时间和算法所需要的存储空间。因此，也把算法的复杂性，划分为算法的时间复杂性和算法的空间复杂性。算法的时间复杂性越高，算法的执行时间越长；反之，执行时间越短。算法的空间复杂性越高，算法所需的存储空间越多；反之越少。在算法的复杂性分析中，对时间复杂性的分析考虑得更多。

1.2 算法的时间复杂性

在讨论算法的时间复杂性时，要解决两个问题：用什么来度量算法的复杂性？如何分析和计算算法的复杂性？下面就来讨论这两个问题。

1.2.1 算法的输入规模和运行时间的阶

假定百鸡问题的第一个算法，其最内部的循环体每执行一次，需 1μs 时间。当$n=100$时，第一个算法的这个循环体共需执行 100 万次，约需执行 1s 时间。第二个算法最内部的循环体每执行一次，也需 1μs 时间。当$n=100$时，第二个算法的这个循环体共需执行 714 次，约需 714μs。当n的规模不大时，两个算法运行时间的差别不太明显。

如果把n的大小改为 10000，即 10000 元买 10000 只鸡，以同样的计算机速度执行第一个算法，约需 11 天零 13 小时；而用第二种算法，则需$(10000/5+1)(10000/3+1)\mu s$，约为 6.7s。当$n$的规模变大时，两个算法运行时间的差别就很悬殊。

从上面的分析以及对货郎担问题运行时间的分析，可以看到下面两点事实：第一，算法的执行时间随问题规模的增大而增长，增长的速度随不同的算法而不同。当问题规模较小时，不同增长速度的两个算法，其执行时间的差别或许并不明显。而当规模较大时，这种差别则非常大，甚至令人不能接受。第二，没有一个方法能准确地计算算法的具体执行时间。这一事实是基于如下原因的：算法的执行时间，不但取决于算法是怎样实现的，也取决于算法是用什么语言编写的，用什么编译系统实现的，编译系统所生成代码的质量如何，在什么样的计算机上执行的，而不同计算机的性能、速度都不相同，致使在同一台计算机上执行，加法指令和乘法指令的执行时间差别就很大。人们无法对所有这些都作出准确的统计，致使不能对某台特定计算机的执行性能作出准确的统计，由于软件规模很大，结构复杂，运行状态变化很大。每一种运行状态都有各种各样的条件判断，依据条件判断而确定是否需要执行的各种操作，其执行时间也难以控制。