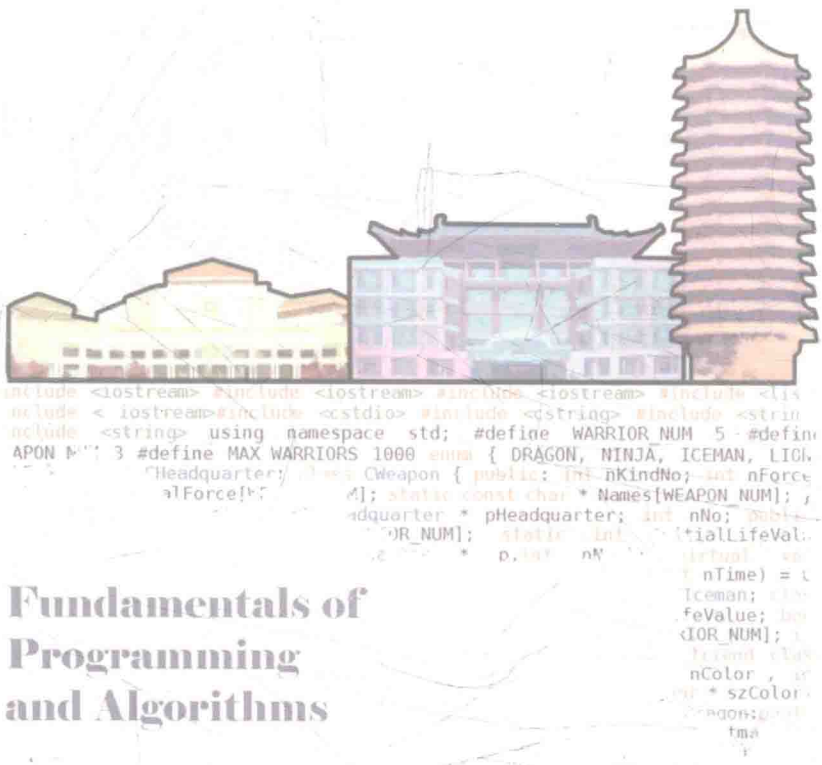
北京大学“程序设计与算法”专项课程系列教材

算法基础与在线实践

刘家瑛　郭　炜　李文新　编著



高等教育出版社

北京大学“程序设计与算法”专项课程系列教材

算法基础与在线实践

刘家瑛　郭　炜　李文新　编著

Fundamentals of Programming and Algorithms

高等教育出版社·北京

内容提要

算法是程序设计的灵魂，代表着用系统的方法描述解决问题的策略与机制。本书将介绍简单模拟、枚举、递归、二分、贪心、动态规划和搜索等经典算法，带领读者体会它们巧妙的构思，感受利用它们解决问题的独特魅力。本书不仅讲解这些算法的基本原理思想，还通过具体例题对这些算法进行灵活、有效的展开和准确实现。本书中涉及的编程任务将充分训练读者的思维能力和动手能力，促成全面、缜密思考问题的习惯。

本书可作为高等学校计算机等相关专业算法设计类课程的教材，也可供对算法设计、程序设计竞赛感兴趣的读者自学使用。

图书在版编目（CIP）数据

算法基础与在线实践／刘家瑛，郭炜，李文新编著. --北京：高等教育出版社，2017.3

ISBN 978-7-04-047300-1

Ⅰ.①算… Ⅱ.①刘… ②郭… ③李… Ⅲ.①电子计算机－算法理论－高等学校－教材 Ⅳ.①TP301.6

中国版本图书馆 CIP 数据核字（2017）第 020327 号

算法基础与在线实践

Suanfa Jichu yu Zaixian Shijian

策划编辑 时 阳	责任编辑 时 阳	封面设计 张 志	版式设计 马 云
插图绘制 杜晓丹	责任校对 李大鹏	责任印制 耿 轩	

出版发行	高等教育出版社	网 址	http://www.hep.edu.cn
社 址	北京市西城区德外大街 4 号		http://www.hep.com.cn
邮政编码	100120	网上订购	http://www.hepmall.com.cn
印 刷	北京七色印务有限公司		http://www.hepmall.com
开 本	787mm×1092mm 1/16		http://www.hepmall.cn
印 张	15.75		
字 数	350 千字	版 次	2017年3月第1版
购书热线	010-58581118	印 次	2017年3月第1次印刷
咨询电话	400-810-0598	定 价	30.40 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

物 料 号 47300-00

前　言

计算机学科是实践性学科，通过编程解决实际工作生活中的问题是该学科的基础，也是训练计算机相关专业学生的基本技能。编写优雅的程序不仅是指熟练运用程序设计语言，更是利用设计精巧的算法高效地解决实际问题。

使用计算机程序解决实际问题，首先要能够将一个具体问题抽象成一个可计算的问题或模型，并设计出一套可行的计算过程。这个构建计算的过程，其实对应的就是算法设计。简捷、高效的算法是计算机科学的核心和精髓。使用算法进行问题求解的例子存在于生活中的方方面面，既可以简单有效，例如最常见的学生成绩信息管理系统、数字排序算法等；也可以非常复杂，例如 Google 公司旗下的 DeepMind 公司为 AlaphGo 程序研发的基于深度学习的人工智能算法等。

算法的本质是取一组值作为输入，经过一系列计算步骤，产生符合要求的输出结果。对于排序算法，输入就是待排序的若干个数，输出就是排好序的数列。对于指纹比对算法，输入就是两个指纹的图像数据，输出就是一个表示相似程度的数值。对于 AlaphGo 的算法，输入就是一个棋局的描述（棋盘上所有棋子的位置），输出就是一个坐标，即落子位置。实际上，算法所研究的不仅是如何得到正确的结果，更重要的是如何尽可能快速地得到正确的结果。试想如果 AlaphGo 下一步棋要计算一天，那李世石还会愿意与它比赛吗？

算法设计又体现出一种计算思维的思想。编写程序的目的是为了将算法思想变为计算机能够执行的指令序列。算法运用不好的程序员，很难说是一个好程序员。作为计算机专业人才，理应具有一定的算法功底，并且应该具备将算法准确实现为程序的能力。因此本书特别强调编程实践，通过具体的例题、样例程序来讲解算法设计的思路和具体实现，并配有大量的课后习题以供练习。本书的作者均在北京大学信息科学技术学院多年讲授计算机专业主干课程“程序设计实习”（通常为本科生第二学期必修课程），课程中长期积累、精挑细选的例

题和习题构成了本书的主要内容。

本书从最简单的算法入手，依次讲述了模拟、枚举、递归、二分查找、贪心算法、动态规划和搜索的基本思想，步步深入，系统地对基础算法进行全面的讲述，并在各章节辅以大量例题对相关算法内容进行有效的补充和深入。通过全面的设计思路分析与详细的程序设计描述展示，有效地促进学生全面、细致地思考问题，提高编程的准确性，增强程序查错、调试的能力。通过训练，学生能够打下较为坚实的程序设计基础，为进一步学习其他计算机专业课程，或在其他专业领域运用计算机编程解决问题创造良好的条件。

本书的最大特点是依托于北京大学在线程序评测系统（简称为 POJ，网址为 http://openjudge.cn），所有例题和习题都可以在线提交和反复练习。POJ 是国内历史最久、访问量最大的在线程序评测系统之一。在线程序评测系统目前已成为程序设计教学中不可或缺的先进手段。学生按照题目要求编写程序并提交源代码，评测系统编译运行程序。通过给定的输入数据，由提交程序进行处理，产生输出结果，系统将该输出结果与标准的输出结果进行比对，必须毫无差别才算正确。由于输入数据往往涵盖各种边界条件、特殊情况，且对提交的学生不可见，因此这种编程练习对学生的要求较高，需要学生心思缜密、考虑周到。目前，北京大学信息科学技术学院的编程类课程的作业和考试几乎都是在 POJ 上完成的，既减轻了教师的工作量，又提高了对学生的要求，强化了对学生的训练。POJ 系统中有数千道习题，大多来源于各类信息学竞赛试题，本书只精选了其中一小部分，这些题目极具典型性，对于程序设计和算法的入门学习能起到举一反三、事半功倍的效果。

同时，本书还配套有北京大学慕课（Massive Open Online Courses，MOOC）课程“算法基础”。该课程也是在 Coursera 平台（http://www.coursera.org）上开设的“程序设计与算法”慕课专项课程（Specialization，又称为微专业）之一。该微专业包含六门课程：计算导论与 C 语言基础、C 程序设计进阶、C++程序设计、算法基础、数据结构基础和高级数据结构与算法，以及程序开发项目实践。“算法基础”作为承上启下的课程，为程序设计语言和数据结构与算法之间构建了桥梁。

最后，感谢北京大学信息科学技术学院“程序设计实习”教学组的各位教师，以及在本书撰写和出版过程中给予帮助的人，包括研究生、杨帅、杨文瀚、李马丁、厉扬豪、杨撒博雅和刘春晖等。本书也得到了高等教育出版社的大力支持。

由于作者水平所限，书中疏漏之处在所难免，敬请广大读者和同行批评指正。作者的电子邮件地址为 liujiaying@pku.edu.cn。

编者　于燕园

2016 年 12 月 24 日

目　录

第1章 绪 论

1.1 什么是算法

什么是算法？通俗地讲，算法就是做事的方法。

在日常生活中，做事的方法有正确和错误之分。正确的方法可以圆满完成任务，错误的方法则不能。而同是正确的方法，效率、性能也有高低之分。以下通过几个具体的例子来初步体验算法在生活中的智慧。

第一，从1加到100。高斯的老师大概只知道1+2+3+…+100这种依次累加的笨办法。但是小高斯却在七岁的时候就想到了(1+100)×(100/2)这样的妙招。老师需要做99次加法，而高斯只要做加法、乘法和除法各一次。高斯的计算效率比他的老师高30倍以上。如果是从1加到1 000 000 000，老师用他这个理论上正确的方法，在有生之年根本不可能得到答案，而高斯依然只需要做加法、乘法和除法各一次就能精准地算出结果。

第二，烤面包。三片面包，每片的两面都需要烤，烤一面需要一分钟，烤炉上只能同时放两片面包。如何在最短时间内烤完三片面包？最基本的方法能花四分钟烤好这些面包，但是不够智慧。经过一定的设计，就可以在三分钟内吃到这三片面包：先烤第一片和第二片的正面，然后烤第一片的反面和第三片的正面，最后再烤第二片和第三片的反面。

第三，猜数。女友心里想一个1～1000之间的数让你猜，考查你有多懂她。你可以问问题，她只回答是或不是。如果你傻傻地问："是1吗？""不！""那是2吗？是3吗？……"等你猜出来，女友就和你分手了。实际上可以这样问："大于500吗？""不！""那大于250吗？""是！""那小于375吗？……"每次把猜测的范围缩小一半，这样最多只要10次就能猜中女友的心思。

第四，在词典里查单词。一页一页依次翻看字典虽然也能查到单词，但是没有人会这么做。大家都是跳着翻页，开始一下跳过数百页，然后再每次跳着翻几十页、十几页、几页，一般翻七八次就能找到单词。如果在每个字母的首页贴上一个表示该字母的小标签（有的词典本身就带有这种标签），这样查起来就会更快。

计算机可以使人类做某些事的效率提高成千上万倍。但是到目前为止，计算机并没有思考能力，它们不曾做过任何一件人类不知该怎么做的事情。计算机能够完成的任何任务，都是人类知道该怎么做，并且理论上只要有足够的时间和人手就能完成的。人类想好做一件事的每个步骤，用编写程序的方式告诉计算机，计算机才能替人类完成工作。即便是战胜国际

象棋特级大师卡斯帕罗夫的“深蓝”计算机也不会思考，它只是机械地贯彻了程序设计者想出的下棋方法而已。理论上来说，一个不懂下棋的人，给他足够的时间，让他按照“深蓝”程序体现的方法一个步骤一个步骤地做，也能战胜卡斯帕罗夫。只不过，按照这些步骤应对卡斯帕罗夫的一步棋，这个棋手恐怕要不停地做上几百年。

计算机程序中表达的方法，就是通常所说的“算法”。从本质上来说，算法只是做事的方法，因此一个不怎么会写程序的数学家也可以是算法领域的顶尖高手。实际上，不论有没有计算机，“算法”都可以存在并有其价值。前面列举的四件事，实际上就是人们用好的算法做事，效率可以比用糟糕的算法显著提高的例子。

你一定不希望在搜索引擎中查找一个关键词，10 分钟后才有结果；也一定不希望在导航仪中输入目的地，等到达了目的地导航仪还没有算出最短的路径；你一定希望用4G 看手机视频时，图像要清晰不卡顿，所耗费的流量还要尽可能少……这一切都离不开高效的算法。

如果计算机的运行速度可以无限快，内存有无限大，那么“算法”这个领域的研究价值就会急剧下降。因为，通常情况下，想出能完成任务的算法并不难，难的是想出高效率的算法。以对一个无序数列进行从小到大排序的问题为例，小学生都能想出的“插入排序”算法（大多数人打牌时，摸牌的同时，让手上的同花色牌保持有序，用的就是这种方法），和精巧设计的“快速排序”算法相比，速度要慢大约 $n/\log_2 n$ 倍（n 是待排序数列的元素个数）。可以想象，n 很小时，差距不大——所以最优秀的程序员打牌时也不会用快速排序的办法整理牌。但是 n 越大，差距就越明显，当 n 达到 1 亿时，两个算法所花的时间会相差约 300 万倍。由于排序算法是极其重要的基本算法，因此几行程序就能实现的“快速排序”被评为二十世纪最伟大的十大算法之一。前几年，当北京大学“程序设计实习”课程还没有推行机考时，几位任课教师在面对期末 300 多份考卷需要按学号进行手工排序时，虽然不会用快速排序，但也会在某种程度上使用“归并排序”算法来达到比单纯的插入排序更快的速度。

1.2 算法的时间复杂度

既然算法的效率如此重要，那么必然需要对其进行量化的分析。通常用“时间复杂度”（简称“复杂度”）来表示算法的快慢。时间复杂度常用大写字母 O 和小写字母 n 来表示，如 $O(n)$、$O(n^2)$等。n 代表问题的规模。复杂度是用算法运行过程中，某些时间固定的操作需要被执行的次数和 n 的关系来度量的。至于这些时间固定的操作每次需要多少时间并不重要。例如在未排序的数组中查找某个给定值，只能从头到尾将数组元素依次与待查找的值比较，那么这个“时间固定的操作”就是比较——这个操作每次所花的时间显然和 n 的大小无关，是固定的。这样的操作，运气好的话，执行一次就能找到给定值；运气坏的话，要执行 n 次才能找到，或断定数组中不包含给定值。如果多次查找不同的值，平均的情况是需要执行 $n/2$ 次比较。因此可以说顺序查找的时间复杂度是 $n/2$。一般来说，标记算法的复杂度时不关心系

数，所以复杂度是 $n/2$ 的算法和复杂度是 $1000 \times n$ 的算法一样，都称其复杂度是 $O(n)$ 的，即和 n 成正比。$O(n)$ 的复杂度也称为线性复杂度。

在排好序的 n 个元素的数组中进行二分查找（类似于猜女友心中所想的数的方法），最多只需进行 $\lceil \log_2 n \rceil$（向上取整）次比较，因此其复杂度就是 $\log_2 n$。$\log_2 n$ 和 $\log_{10} n$ 或 $\log_{100} n$ 的差别只是一个系数，所以它们都可以被记为 $O(\log n)$。$O(\log n)$ 的复杂度也称为对数复杂度。

有时候提到一个问题的复杂度，实际上说的是用已知的最快算法解决这个问题时该算法的复杂度。例如，平面上有 n 个点，要求出任意两点之间的距离。在这个问题中，“求给定两点的距离”就是时间固定的操作，一共有 $n(n-1)/2$ 个点对，所以不论用什么好算法（其实也没什么好算法），这个问题的复杂度都是 $n^2/2 - n/2$。由于 $n^2/2$ 的增长速度要快于 $n/2$，当 n 比较大的时候，$n/2$ 可以忽略，所以这个问题的复杂度就记为 $O(n^2)$。由此例可以看出，在讨论某个问题或算法的复杂度时，只关心随着 n 增长而增长得最快的那一项。例如，如果复杂度是 $2^n - n^3$，就记为 $O(2^n)$；如果复杂度是 $n! + 3^n$，则记为 $O(n!)$。

有的问题是指数复杂度的。例如，一个有 n 个元素的集合，其子集一共有 2^n 个，因此求一个集合的所有子集的复杂度就是 $O(2^n)$。$O(a^n)$ 这样的复杂度（a 是与 n 无关的常数）称为指数复杂度。当有指数复杂度问题的规模达到近百时，基本上就没法解决了，只能寻求近似的解法。

此外还有阶乘复杂度。求 n 个字母的全排列这个问题的复杂度就是 $O(n!)$ 的。阶乘复杂度的算法比指数复杂度的效率更低。

计算复杂度的时候，只统计问题规模足够大时，执行次数最多的那些固定操作的次数。例如，某个算法需要执行加法 n^2 次，除法 n 次，那么就记其复杂度是 $O(n^2)$ 的。

如果一个问题所花费的时间和规模无关，是一个常数，就记其复杂度为 $O(1)$，也称为常数复杂度。例如，取一个排好序的序列中的最大元素的复杂度就是 $O(1)$ 的，因为它和序列中有多少个元素无关。

有时复杂度不能仅由一个 n 来表示。例如矩阵乘法，一个 $m \times n$ 的矩阵和一个 $n \times k$ 的矩阵相乘，如果用最原始和简单的方法计算（实际上有更好的算法），那么需要做 $m \times n \times k$ 次乘法，即复杂度是 $O(m \times n \times k)$。

复杂度还有最坏情况下的复杂度和平均复杂度之分，但在许多情况下最坏复杂度和平均复杂度并没有区别。最常用的快速排序算法一般情况下的复杂度是 $O(n \times \log n)$ 的，但是在待排序的序列本来就是几乎有序的情况下，其复杂度会变成接近 $O(n^2)$。如果实在不能容忍出现最坏情况，那么在快速排序前可以先将序列随机打乱一下。实际应用中，如果最坏情况只有很小概率出现，那么一般只要考虑平均复杂度即可。但是如果最坏情况一旦出现结果就不能容忍，则还是要考虑最坏复杂度。

参加程序设计竞赛或者在各种在线程序评测平台（简称 OJ）上刷题，一般要考虑最坏情况的复杂度。因为题目的设计者可能会有意针对一些算法给出最坏的数据，以使这些算法运行过慢而被判不能通过，这就是通常所说的超时（Time Limit Exceeded）。一般来说，如果复

杂度是10亿的量级，肯定会超时；1亿量级，很危险；千万量级，大可一试；百万量级，基本安全。

1.3 算法时间复杂度分析示例

下面来看一个选择排序的函数，分析其复杂度。

选择排序的基本思想是：如果有 N 个元素需要排序，那么首先从 N 个元素中找到最小的那个（称为第0小的）放在第0个位置上（和原来的第0个位置上的元素交换位置），然后再从剩下的 $N-1$ 个元素中找到最小的放在第1个位置上，之后再从剩下的 $N-2$ 个元素中找到最小的放在第2个位置上……直到所有的元素都就位。具体程序如下：

```
1.  void SelectionSort(int a[], int size)
2.  {
3.      for(int i = 0; i < size - 1; ++i){
4.      //每次循环后将第 i 小的元素放好
5.          int tmpMin = i;
6.      //记录第 i 个到第 size - 1 个元素中，最小的元素的下标
7.          for(int j = i + 1; j < size; ++j){
8.              if(a[j] < a[tmpMin])
9.                  tmpMin = j;
10.         }
11.     //下面将第 i 小的元素放在第 i 个位置上，并将原来第 i 个位置的元素挪到后面
12.         int tmp = a[i];
13.         a[i] = a[tmpMin];
14.         a[tmpMin] = tmp;
15.     }
16. }
```

外重循环的循环体执行（size - 1）次。对于每次外重循环体的执行，第7行内重循环中的 j < size 这个判断需要执行（size - i）次。由于 i 从0变到（size - 2），因此 j < size 这个判断一共需要执行 2 + 3 + … + size 次，即（size + 1）× size/2 - 1 次。因此该函数的时间复杂度就是 $O(\text{size}^2)$ 的。当然，也可以说第7行的 j < size 这个判断要执行（size + 1）× size/2 次，因此复杂度就是 $O(\text{size}^2)$。

再看插入排序的复杂度分析。

插入排序的基本思想是：将整个数组 a 分为有序的和无序的两个部分。前者在左边，后者在右边。开始有序的部分只有 a[0]，其余都属于无序的部分。每次取出无序部分的第一个（最左边）元素，把它加入有序部分。假设插入合适的位置 p，则原 p 位置及其后面的有序部

分元素都向右移动一个位置，有序的部分即增加了一个元素。一直做下去，直到无序的部分没有元素。程序如下：

```
1.  void InsertionSort(int a[], int size)
2.  {
3.      for(int i = 1; i < size; ++i){
4.          //a[i]是最左边的无序元素，每次循环将 a[i]放到合适位置
5.          for(int j = 0; j < i; ++j)
6.              if(a[j] > a[i]){
7.                  //要把 a[i]放到位置 j，原下标 j 到 i-1 的元素都往后移一位
8.                  int tmp = a[i];
9.                  for(int k = i; k > j; --k)
10.                     a[k] = a[k-1];
11.                 a[j] = tmp;
12.                 break;
13.             }
14.         }
15. }
```

外重循环需要执行（size-1）次。每次外重循环执行时，假设在 j=n 时执行了第 12 行的 break 语句，则此时第 6 行的 a[j]>a[i]已经被执行了 n+1 次。当 j=n 时，会执行第 9 行的循环，该循环的 k>j 这个判断会被执行 i-j+1 次，即 i-n+1 次。于是，每次外重循环执行时，两种操作合计执行 i+2 次。也可以定义一种操作，叫作“执行 a[j]>a[i]或 k>j”，则每次执行外重循环时，该操作一共被执行 i+2 次。i 从 1 变到 size-1，因此合计执行次数为 $3+4+\cdots+(\text{size}+1)$，这个复杂度也是 $O(\text{size}^2)$的。

写程序时对复杂度进行分析是十分必要的。不过很多情况下程序的复杂度并不会像上面的例子那么容易分析、一目了然。

1.4 PKU OpenJudge 在线评测系统

“北京大学程序在线评测系统”（Peking University Online Judge System，POJ）最早于 2003 年由北京大学人工智能实验室开发上线，主要目的是为北京大学 ACM 队训练提供评测系统，但同时系统也是对外开放的，面向全球编程爱好者。POJ 目前已经有超过 65 万的用户、3000 道题目以及超过 1500 万的程序提交。

2005 年 9 月，根据北京大学信息学主干平台课程（计算概论、程序设计实习、数据结构与算法等）的需求，从原有 POJ 系统中独立出一个分支，主要面向程序设计语言教学，这个平台就是百练。相比于原 POJ，百练系统最大的变化有两个：一是系统界面的语言由英文更换

成了中文；二是加入了更多符合教学难度的题目，原有系统中大量竞赛题目并不太适合教学。

随着百练平台在程序设计教学中影响的增大，原先只面向北京大学信息学院教学而使用的百练平台的角色发生着重要的改变，逐步面向其他院系、其他高校，甚至其他中学。随着使用者的增多，问题随之而来，各门课程的用户、作业都混在一起，给教师、助教对于课程的管理带来了很大的影响。

为了解决上述问题，2009 年，POJ 团队对百练平台进行了重构，形成了现在的 OpenJudge（域名为 openjudge. cn）。OpenJudge 最大的变化在于实现了小组化的管理，每个小组相当于原先的一个百练平台，这样一来，不同的课程使用不同的小组，每个小组的管理都是独立的，教师或者助教拥有管理员权限，可以自由地创建题目、作业，而不影响其他小组的使用。

OpenJudge 目前已经拥有近 370 个小组，不仅为北大的课程教学提供服务，也为全国各地的中小学（超过 100 所）、高等学校（超过 150 所）提供服务（详情可以查看 http://openjudge. cn/groups）。目前 OpenJudge 的用户已经超过 16 万人，题目数量超过 10 000 道，提交代码超过 620 万次，支持的编程语言有 C、C ++ 、Java、Pascal、C#、Racket、Python。

用户可以针对某个题目编写程序并提交，POJ 会自动判定程序的对错。教材与课程中使用的大部分例题和课后练习题都精选或改编自 POJ 题库。使用者在进行练习时，可以将自己的程序提交给 POJ，几秒之内即可知道所编写程序的正确性。

每个学生都可以在 POJ 上建立自己的账号，教师布置习题后即刻就能看到学生是否已经完成，这几乎将教师评判学生作业的工作量减少到零。POJ 对于程序的评判是极为严格的，学生的程序根据 POJ 给出的输入数据进行计算并输出结果，POJ 在服务器端编译、运行被提交的程序，将取得的输出结果和标准答案对比，程序必须一个字节都不差才能通过。这对于培养严谨、周密的程序设计作风极为有效，学生必须考虑到每一个细节和特殊的边界条件，而不是大体上正确就能通过。传统的人工评判是难以做到这一点的。

1.5 本章小结

算法有的简单有的复杂，简单的算法如排序，几行代码就能写完；复杂的算法如视频编码压缩、数据挖掘、机器学习等，有时需要团队协作才能实现。本书所介绍的都是基本的、常用的算法思想，如模拟、递归、搜索、动态规划、二分和贪心算法。这些基本的算法思想是构造复杂算法的基石。

总之，算法可以说是程序的灵魂。算法水平的高低很大程度上决定了程序员的层次。没有算法功底的程序员俗称为码农，大多只能从事技术含量低、机械的编程工作。一般的 IT（信息技术）培训机构不教算法，培养的就是此类人员。国内外的 IT 巨头在招聘程序员时，几乎都会考查算法能力，考查的题目常常就是 ACM 国际大学生程序设计竞赛的赛题，类似于本书中的例题。学好算法，可以从本书开始！

第2章　简单计算与模拟

2.1　基本思想

用模拟法解决问题的基本思想是对事物进行抽象，将现实世界的事物映射成计算机所能识别的代码符号，而将现实事物之间的关系映射成运算或逻辑控制流。抽象思想的一个简单例子就是整数的产生。数字取代了具体的事物用于对数量进行度量。人类从一只羊、一头狼、一条鱼、一棵树等事物中看到了共通的单位性，从感知中产生出了抽象的数字，而对这些数字进行运算的过程，则代表了这些事物的累加、增加或减少。从此，古人类对数的把握从具体的事务中解放出来。

本节介绍的内容就类似于古代人类建立数字观念的起步过程——将真实世界的事物变成代码中的变量，并把理解的逻辑转换成代码的运算和逻辑流。模拟法的题目通常是对某一类事件进行描述，然后经过一些简单运算给出符合要求的输出结果。读者只需认真读懂、搞明白问题的要求，并能将题目所反映的内容映射为程序逻辑，基本就可以解决问题了。

模拟法是编程的基本功，没有复杂的公式和技巧，只需要读懂题目，照着逻辑，理对代码，基本都可以完成。这些题目非常有利于编程的初学者建立基本的代码感觉。需要提醒的是，有些模拟法的题目背景可能设定得错综复杂，稍微有所大意或者逻辑没有整理清楚就可能步入歧途，简单的问题也可能纠缠半天，不必要的复杂逻辑使得查错无处可循。

模拟法对于一些没有数值解法的题目来讲，是唯一求得精确解的方案，如约瑟夫问题。因此，作为学习编程的第一步，模拟法需要受到足够的重视。

2.2　例题：鸡兔同笼（POJ 3237）

问题描述

一个笼子里面关了若干只鸡和兔子（鸡有2只脚，兔子有4只脚，没有例外）。已经知道了笼子里面脚的总数 a，则笼子里面至少有多少只动物，至多有多少只动物？

输入数据

第1行输入一个正整数 $n(n\leq 1\,000)$，表示测试数据的组数 n，接下来的 n 组测试数据每