算法艺术与信息学竞赛

- 适合具有一定算法基础的读者
- 大量经典算法的详细讲解与完整代码
- 超过150道的竞赛真题讲解
- 包含近千道题目的分类训练参考

算法竞赛入门经典训练指南

刘汝佳　陈　锋◎编著

清华大学出版社

刘汝佳，1982年12月生，高中毕业于重庆市外国语学校。

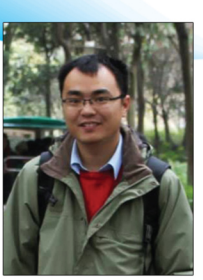
2000年3月获得NOI2000全国青少年信息学奥林匹克竞赛一等奖第四名，进入国家集训队，并因此保送到清华大学计算机科学与技术系。大一时获2001年ACM/ICPC国际大学生程序设计竞赛亚洲–上海赛区冠军和2002年世界总决赛银牌（世界第四），2005年获学士学位，2008年获硕士学位。

学生时代曾为中国计算机学会NOI科学委员会学生委员，担任IOI2002–2008中国国家队教练，并为NOI系列比赛命题十余道。现为NOI竞赛委员会委员，并在NOI 25周年时获得中国计算机学会颁发的“特别贡献奖”。

2004年至今共为ACM/ICPC亚洲赛区命题二十余道，担任6次裁判和2次命题总监，并应邀参加IOI和ACM/ICPC相关国际研讨会，发表论文两篇。

2004年初作为第一作者出版专著《算法艺术与信息学竞赛》，2009年出版译著《编程挑战》。

多年来在全国二十余个城市进行中学生竞赛培训工作，为北京、上海、吉隆坡等地的著名高校授课与宣讲，并多次与TopCoder、百度和网易有道等知名企业合作举办比赛，让更多的IT人才获得展示自我的平台。



陈锋，1982年9月生。毕业于华北水利水电学院机械设计专业。曾就职于微软全球技术支持中心，负责.net虚拟机以及Visual Studio开发技术支持。后进入金融IT行业，专注于银行网点平台的产品研发，曾分别负责基于.net和Eclipse的两代网点平台产品的开发以及架构设计。现就职于北京宇信易诚科技，任前端产品技术经理及架构师。

算法艺术与信息学竞赛

算法竞赛入门经典——训练指南

刘汝佳　陈　锋　编著

清华大学出版社

北　京

内 容 简 介

本书是《算法竞赛入门经典》的重要补充，旨在补充原书中没有涉及或者讲解得不够详细的内容，从而构建一个较完整的知识体系，并且用大量有针对性的题目，让抽象复杂的算法和数学具体化、实用化。

本书共 6 章，分别为算法设计基础、数学基础、实用数据结构、几何问题、图论算法与模型和更多算法专题，全书通过近 200 道例题深入浅出地介绍了上述领域的各个知识点、经典思维方式以及程序实现的常见方法和技巧，并在章末和附录中给出了丰富的分类习题，供读者查漏补缺和强化学习效果。

本书题目多选自近年来 ACM/ICPC 区域赛和总决赛真题，内容全面，信息量大，覆盖了常见算法竞赛中的大多数细分知识点。书中还给出了所有重要的经典算法的完整程序，以及重要例题的核心代码，既适合选手自学，也方便教练组织学习和训练。

图书在版编目（CIP）数据

算法竞赛入门经典——训练指南/刘汝佳，陈锋编著．—北京：清华大学出版社，2012.9
（算法艺术与信息学竞赛）

ISBN 978-7-302-29107-7

I. ①算… II. ①刘… ②陈… III. ①电子计算机-算法理论-教学参考资料 ②C 语言-程序设计-教学参考资料 IV. ①TP301.6 ②TP312

中国版本图书馆 CIP 数据核字（2012）第 130853 号

责任编辑：朱英彪
封面设计：刘　超
版式设计：文森时代
责任校对：张兴旺
责任印制：
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：33　**字　　数**：762 千字
版　　次：2012 年 9 月第 1 版　**印　　次**：2012 年 9 月第 1 次印刷
印　　数：1～5000
定　　价：52.80 元

产品编号：041622-01

序

翻了翻 UVa（Valladolid 大学）在线评测系统的数据库，关于汝佳的第一条记录是在 2000 年 11 月 15 日，UTC 时间是 13 点 58 分 00 秒——他在那时刚刚注册。这个时间在中国已经不算早了，但对于汝佳来说这完全不是问题，因为据我所知他随时都在工作。在那个即将结束的一天中，他提交了 10 道题目的 22 份 C++代码，其中第一份代码是在注册后的短短数分钟之内提交的，并且获得了 AC（正确），通过了题目 264（Count on Cantor）。

当然，他也会犯一些错，不过当天结束时他已经搞定了 3 道题目，并且第二天就解决了剩下的全部题目。在年底之前（其实只有 45 天），汝佳提交了 90 道题目的 307 份代码，其中通过了 59 道。那个时候他还很年轻（差不多就是个孩子，就像他现在的模样），但他的思路已经很清晰了——那就是勤加练习的结果。

尽管上面讲的这些已经是陈年旧事了，但从中我可以断定：你们手中的这本书是一个非比寻常的伴侣，将带领你们走进计算机编程世界，尤其是编程竞赛世界。遗憾的是，书是用中文写成的，因此唯有懂中文的人才能享受到汝佳创造的这份魔法。幸运的是，他用 UVa 在线评测系统作为书中题目的来源，这使得我们这些不懂中文的人也可以透过题目了解到本书的知识体系和教学计划。他的这一举措是我们的荣幸，因为这本书给 UVa 网站增加了一份特殊的价值。汝佳很爱学习，也很乐意把自身所学倾囊相授。有了这本书，我们的工作有了更大的意义。

对我们来说，这也是一项挑战，因为我们必须更加努力地完善评测系统和网站，以回馈那些因为本书而对 UVa 网站产生兴趣的读者。我们也乐于接受这个挑战，因为我们很高兴看到 UVa 网站除了可以用来备战竞赛之外，还可以用来帮助人们学习算法。事实上，我们已经开始着手开发一个新的评测系统，挖掘学生群体中新的需求。我们希望能有更多的人参与，比如你。

最后，我想向所有的中国读者表达一份特殊的欢迎——欢迎你们通过本书认识和了解 UVa 在线评测系统（如果能顺便了解一下 Valladolid 这座城市就更好了），同时感谢汝佳邀请我用这些平凡的文字为这本不平凡的书作序。

Miguel A. Revilla
UVa 在线评测系统创始人
ACM-ICPC 国际指导委员会成员、题目归档专家
Valladolid 大学/西班牙
http://uva.onlinejudge.org
http://livearchive.onlinejudge.org

前　言

“请问新书《算法实践手册》什么时候出版？望眼欲穿啊……”

自《算法竞赛入门经典》（以下简称《入门经典》）出版以来，我收到的这样的来信已经不计其数。

不过，我心里有着自己的打算。《入门经典》的出版固然为广大算法爱好者提供了一些帮助，但其中的缺憾也是很明显的，如例题太少，习题没有中文翻译，而且限于篇幅，基础知识还没讲完……这样看来，《算法实践手册》的出版时机尚未成熟，还需要一本书来铺垫，弥补上述缺憾。可惜的是，由于创业的繁忙，这个想法一直未能实现。

2010 年 8 月底，我收到了一封读者的 E-mail，和我探讨《入门经典》中的一些问题，从此和本书的第二位作者陈锋相识。我万万没有想到，这位来自银行业的软件工程师、产品架构师学习编程的时间只有两三年，他对算法的热爱、严谨求实的态度和认真刻苦的专业精神绝不亚于有着多年算法和软件工程经验的行家。在与陈锋的交流过程中，我重新开始了对这本新书的构思。

事实上，在《入门经典》的写作过程中，完成的书稿远远不止印刷出来的 220 多页，只是因为篇幅和内容限制没有用到该书中。如果好好地把这些书稿加以整理，再算上笔者多年来外出讲课时制作的课件、题目翻译，那么本书的轮廓已经呼之欲出。这些东西对我来说已经是老生常谈，但接触算法不久的陈锋却觉得很新鲜。这样，我萌生出了一个有趣的念头：和陈锋一起合著一本书，我提供资料和总揽全局，而陈锋一边学习以前没有接触过的新知识，一边把这些东西按照适合初学者的方式重新进行组织和细化，并以“没参加过算法竞赛的软件工程师”这样一个独特的视角提出各种意见和建议。

细水长流了一年多之后，这个构想终于成为了现实。虽然大量的业余时间都奉献给了这本书，但我相信这是值得的。

参与本书编写和校对工作的还有中国人民大学的陈卓华和陈怡、北京大学的鲍由之（绘制了书中的几乎全部插图）等；一位不愿透露姓名的台湾朋友阅读了几乎全部书稿并给出了非常详细的修改意见。为了更好地配合本书，我在 UVa 上举办了 3 场专题比赛（数据结构、几何、实用程序），并将其中的典型题目收录在了正文例题或者习题当中。由于题目难度颇大，如果没有李益明、梁盾、沈业基、李耀、周而进、陈卓华、陈怡、唐迪、李晔晨、肖刘明镜、鲍由之等朋友的鼎力相助，这些比赛几乎不可能取得圆满成功。

另外，我还要感谢 CCF（中国计算机学会）的杜子德秘书长、吴文虎教授、王宏教授等，还有 NOI 科学委员会及竞赛委员会的专家们，以及 ACM/ICPC 亚洲区主席黄金雄教授和中国指导委员会秘书长周维民教授。他们都是我的良师益友，在我接触 NOI 和 ACM/ICPC 以来的十多年里让我学到了很多东西。

感谢清华大学出版社辛勤劳动的编辑们，尤其是与我合作多年的朱英彪老师，在本丛书的编写、出版和推广方面都做了大量的工作，是一位真心实意为读者着想的好编辑。

最后，感谢我的爸爸妈妈。不管我做什么，不管别人怎么说怎么看，你们总是那样的

支持我，让我可以全身心的投入自己喜爱的事业，没有半点后顾之忧。你们教给我的善良、感恩和奉献，正是我多年来坚持写书的源动力。

刘汝佳

许多计算机相关专业的人在毕业之后除了为应付面试外基本都很少再去碰算法，而在实际的产品或者项目开发过程中，大多数人也没有必要亲自去实现复杂的算法。因此，算法渐渐淡出程序员的日常生活。同时，在现实生活中有另外一种声音：程序员的生活太纠结，coding 的速度永远跟不上需求变化的速度，提需求的客户似乎成了程序员的“天敌”，成了他们“苦逼”生活的罪魁祸首。

那么，一本讲算法比赛的书跟这又有多少关系？就从我自身的经历说起吧。我不是计算机科班出身，但因种种原因进入了这个行业，而且是从一个很低的起点进入。于是我像所有人一样，平时很难静下心来学习算法，有的面试就去临时抱本书突击一下。终于有一天我受不了这种循环，问自己难道一定是为了一个急功近利的目的才去付出自己的时间吗？佛家有句话叫“凡夫求果，菩萨求因”，我就想，既然成不了圣人，就学一回圣人吧。

因缘际会，我接触到了《入门经典》及其作者刘汝佳，于是一发不可收拾，写这本书的过程也变成了修行与学习的过程。慢慢地，我发现算法对于实际工作的人而言，有着比应付面试更大的价值。所谓的算法、组件、模式，就像是一些基础的原材料，对于优秀的建筑师来说，需要透彻的理解（不一定写得很熟练）它们的关键性。因为一个错误的设计，对于系统来说，所要付出的代价远比一般的程序 bug 要高得多。更进一步说，现在做软件的为啥苦，为啥抱怨需求变化快？因为解决问题的思维有偏差。需求分析绝对不是简单地拿着需求，直接翻译成代码——这是最低层次的。算法分析的意义，更多地不在于性能，不在于那些脑筋急转弯，而在于发现纷繁复杂的问题背后的不变式，而这正是本书要着力与大家分享的。

没有大家的支持和帮助，这本书几乎是不可能写好的。尤其要感谢我的家人和朋友，为了这本书的写作，他们投入了大量的时间和精力。这里尤其是在我工作非常繁忙的情况下，写作占用了大量跟家人在一起的时间，没有妻子梁明珠和女儿婉之的支持，我几乎不可能集中精力参与本书的写作过程。

另外，也感谢我的同事徐海波、张大用、周洲、王洪桥、朱宗耀、朱洁晨等，他们不仅在工作上给了我非常大的支持和帮助，也对我参与写作的章节提出了很多非常好的建议。

陈锋

声明：书中用到了大量的例题和习题，感谢这些命题者和竞赛组织方的辛勤劳动。笔者已经尽可能地找到他们并征求了题目的使用权，但如果你认为本书侵犯了你的权益，请与出版社和作者取得联系。在 UVa 网站上可以找到本书大多数例题/习题的作者。

阅 读 说 明

如何阅读本书

欢迎大家阅读本书！为了最大限度地发挥本书的作用，强烈建议您在阅读正文之前仔细地读完《阅读说明》，还要时不时地在学习过程中重读这些文字，以确保自己不会因为纠缠于书中的一些细节而变得舍本逐末，甚至偏离了预定的学习航线。切记！

“覆盖面广，点到为止，注重代码”是本书的最大特点，而这 3 个特点都是为了向业界靠拢，注重广度而非深度。为了方便练习，书中所有题目来自于 UVa 在线评测系统和 ACM/ICPC 真题库（Live Archive, LA）。虽然这两个题库的题目不能包罗万象，但对于“点到为止”来说绰绰有余。书中的代码更多是为了提供一个容易理解、适合比赛的参考实例，并不推荐读者直接使用甚至背诵它们（关于这一点，在稍后还会有详细叙述）。

关于知识点

本书不是一本算法竞赛入门类图书，因此并不从程序设计语言、算法的概念、基础数据结构、渐进时间复杂度分析这些内容讲起。如果你是一个新手，建议先阅读《算法竞赛入门经典》（和本书搭配着阅读也可以）。

聪明的选手都是善于利用网络资源的，如在网上交流讨论、阅读牛人的博客，寻找解题报告和测试数据等。这些方法也是阅读本书的重要辅助手段。由于算法竞赛涉及的知识点非常广，并且读者所具备的知识水平参差不齐[①]，因此，不同读者在阅读本书时会遇到不同的困难。这些困难有时并不是读者的问题，而且也无法靠修改书稿来避免[②]，因此，最好的办法就是上网搜索！

本书是完全的题目导向，掌握了书中的所有例题，也就掌握了书中的主要知识点和方法、技巧。在习题中也有少量相对不那么重要、例题没有涉及的东西，在每章的小结部分会明确指出。换句话说，对于每个知识点，最好的方法是结合例题分析和代码去理解，上机实践，最终达到能独立解决同类问题的目的。

本书的学习顺序不是单一的，但最好先阅读第 1 章，因为其中不少思想和编码技巧适用于全书。接下来，数学（第 2 章）、数据结构（第 3 章）、几何（第 4 章）、图论（第 5 章）等内容可以根据读者需要按任意顺序进行学习。事实上，笔者建议大家先学习这些章节的重要内容，等对全书有了一个整体认识之后再精读精练，千万不要过早地陷入难懂的细节（每章都有一些相对难懂的内容）。第 6 章主要是一些零散的知识点和技巧、方法，

---

① 笔者认为，本书的读者至少会覆盖从小学高年级学生到从事 IT 工作多年的工程师。

② 比如，有些术语在学术界并没有统一的叫法，或者在不同文献中的叫法不同，如果本书采用的名称和你所熟知的不同，反而会产生误导。

可以与其他章节穿插阅读。

请重视每章后面的小结与习题部分。虽然每章前面的文字一视同仁地对所有内容进行了逐一讲解，但在算法竞赛中，这些内容并不是同等重要的，思维训练和编程训练的内容和比重也不尽相同。当你不知道接下来应该学些什么、练些什么时，这些内容会是你最好的帮手。比如，如果关于某个知识点的习题特别少，这通常意味着该知识点很少在竞赛中出现，或者只需要很少的练习就能掌握到其精髓[①]。

如何做题

书中的重要题目（包括例题和习题）均配有完整代码（限于篇幅，这些代码不一定会在书中出现）和测试数据，以方便读者学习，而其他题目也大都附有中文翻译，并且指明了题号、提交人数和通过比例等统计数据，以供读者训练参考。

书中题目数量不少，因此做题时必然要有一个先后顺序，建议初学者优先解决那些通过人数较多的题目，而已经开始专项训练的读者则可以根据需要选择难一些的题目。

考虑到很多读者并不是“久经沙场”的老手，这里以“蚂蚁”例题为例介绍做题的一般步骤。

蚂蚁(Piotr's Ants, UVa 10881)

一根长度为 L 厘米的木棍上有 n 只蚂蚁，每只蚂蚁要么朝左爬，要么朝右爬，速度为 1 厘米/秒。当两只蚂蚁相撞时，二者同时掉头（掉头时间忽略不计）。给出每只蚂蚁的初始位置和朝向，计算 T 秒之后每只蚂蚁的位置。

【输入格式】

输入第一行为数据组数。每组数据的第一行为 3 个正整数 L, T, n（$0 \leqslant n \leqslant 10\,000$），以下 n 行每行描述一只蚂蚁的初始位置。其中，整数 x 为它距离木棍左端的距离（单位：厘米），字母表示初始朝向（L 表示朝左，R 表示朝右）。

【输出格式】

对于每组数据，输出 n 行，按输入顺序给出每只蚂蚁的位置和朝向（Turning 表示正在碰撞）。在第 T 秒之前已经掉下木棍的蚂蚁（正好爬到木棍边缘的不算）输出 Fell off。

【样例输入】

```
2
10 1 4
1 R
5 R
3 L
10 R
10 2 3
4 R
```

[①] 这两个原因并不是孤立的。一般来说，竞赛会偏爱那些精巧、灵活的内容，而非那些很难变形、扩展的死板东西。

```
5 L
8 R
```

【样例输出】

```
Case #1:
2 Turning
6 R
2 Turning
Fell off

Case #2:
3 L
6 R
10 R
```

这是第 1 章中的例题 5，是一道很有意思的题目。题目正文只有 3 句话，并不难理解，但正文并不是题目的全部。可以看到，一道完整的题目至少包含 3 个部分：题目描述、输入输出格式和样例输入输出，缺一不可[①]。

题目描述可以让你了解到你需要解决一个什么样的问题，但有些细节可能不会涉及。在第一次阅读时可以忽略掉不明白的地方，直接看输入输出格式，以了解具体的输入输出方法。对照输入输出格式和题目描述，可以更清楚地知道这道题目已知什么，要求什么。

输入输出格式往往有规律可循。例如，本书的题目全部采用 ACM/ICPC 比赛题目的格式，采用多数据输入输出。最常见的格式有以下两种。

格式一：输入第一行包含数据组数 T。每组数据的第一行包含……主程序通常这样编写：

```
int main() {
  scanf("%d", &T);
  while(T--) {
    input();     //读取单组数据
    solve();     //求解问题
    output();    //输出
  }
  return 0;
}
```

格式二：输入包含多组数据。每组数据的第一行包含一个整数 n，输入结束标志为 n=0。主程序通常这样编写：

```
int main() {
  while(scanf("%d", &n) == 1) {
    if(n == 0) break;
```

[①] 有的题目还包含其他部分，比如背景介绍、样例解释，甚至解题提示，但这些都不是必需的。

```
    //读取其他数据，求解并且输出
    ...
  }
  return 0;
}
```

注意，上述代码用到了 scanf 的返回值，它返回的是成功读取的元素数目。换句话说，即使真实数据的最后并没有 n=0 的“标志”，上述程序也会在文件结束后退出主循环。当然，正常情况下命题者不会忘记在数据末尾加上这个“标志”。但人非圣贤，孰能无过，即使在 ACM/ICPC 世界总决赛这样的重量级比赛中，也曾出现过数据不符合题目描述的情况。因此，作为选手来说，最好的方法是编写一个尽量鲁棒的程序，即使在数据有瑕疵的情况下仍然能够通过数据[①]。

输入输出格式的另一个作用是告知数据范围和约束。比如上述题目中，“n≤10 000”这个约束实际上是在警告选手：$O(n^2)$时间复杂度的算法也许会超时[②]。另一些重要的约束包括“输出保证不超过 64 位无符号整数的范围”、“输入文件大小不超过 8MB”。前者通常意味着我们不必使用高精度整数[③]，而后者通常意味着我们需要注意输入数据的时间，不要采用太慢的输入方法（这个问题会在第 1 章中详细讨论）。

在很多题目中，正确的输出并不是唯一的。在这种情况下，题目要么会明确指出多解时任意输出一个解即可（此时会有一个称为 special judge 的程序来判断你的解是否正确），要么会告诉你输出哪一组解，以确保输出的唯一性。多数情况都是输出字典序最小或者最大的解。

看完题目描述和输入输出格式之后，还有一件重要的事要做，那就是阅读样例。很多题目的样例都是“可读”的，即至少包含一个简单到能够手算出来的例子。在这种情况下，强烈建议读者手算一下这个样例，因为这不仅可以帮助你理解题意、消除潜在的歧义，还能启发你的思路，使你从手算的过程中概括整理出本题的通用解法。如果有疑问，还可以很方便地向主办方提出[④]。

接下来，就可以设计算法并编写程序了。如何设计算法？如何编写程序？这正是本书的主题，所以在这里不再赘述，假定你已经顺利地写完程序。

接下来应当测试，看看你的程序是不是正确。编程的一大特点是“失之毫厘，谬以千里”，因此，哪怕只是写错了一个运算符，程序的结果也可能完全不对。所以，在提交程序之前应当好好测试一下。测试什么数据呢？最现成的当然就是样例了，但即使这样也远远不够。很多时候，样例并不具有典型性，通过样例也许只是碰巧；如果题目很难，就算样例很典型，也未必能覆盖到所有情况。也许你的程序几乎是完美的，但在一些特殊情况下也会出错，而样例并不包含这些特殊情况。因此，进行全面的测试是很有必要的。

一旦测试出错，就需要调试（debug）你的程序。调试的方法有很多种，如跟踪调试，即利用 IDE 的单步、断点、watch 等功能，动态地检查程序的执行过程是否和预想中的一样。这样的技巧固然有用，但在算法竞赛中更常见的还是“断言+输出中间结果”的方法，这在

① 书中多次强调的“把数组开大一些”也是基于这个考虑。
② 多数情况如此，但如果常数特别小，$O(n^2)$时间算法也有可能通过 n=10000 的数据。
③ 但如果算法糟糕或者实现不好的话，中间结果有可能溢出！
④ 如果你问：“这题是什么意思？”，通常不会有人回答你；如果你问“这个样例为什么输出 1”，常常都能得到满意的回答。

《算法竞赛入门经典》中已有详细叙述。需要特别注意的是，修改程序后最好测试一下以前测过的数据，以免“拆东墙补西墙”，修改了老 bug 的同时又引入了新 bug。

不管是参加什么样的比赛，都有一个“提交代码”的过程[①]。为了尽可能地避免一些“非正常因素”的干扰，进行一些提交前的检查还是很有必要的。主要包括检查输入输出渠道是否正确（比如，文件输入还是标准输入，有没有忘记关闭文件）、调试用的中间结果是否已被屏蔽、数组有没有开得足够大、输出中的常量字符串有没有打错（一般来说，Yes 打成 yes 将被判错）等。

本书的题目风格均为 ACM/ICPC 式的，因此每个程序只有“对”和“不对”两种结果（当然，还会告知“不对”的原因，详见《算法竞赛入门经典》），而没有“半对半错”之说。好在本书的所有题目均选自 UVa 在线评测系统和 ACM/ICPC 真题库（Live Archive，简称 LA），这两个在线题库的操作几乎一样，都可以很方便地提交题目（只需要知道题号，通过网站左下方菜单中的“Quick Submit”命令即可提交）。在完成书后习题时，常常需要参考原题（如看样例或者输入输出格式），这里介绍两个“快捷方式”（以题目 12345 和 2345 为例）：

UVa 题目链接：http://uva.onlinejudge.org/external/123/12345.html

LA 题目链接：http://livearchive.onlinejudge.org/external/23/2345.html

当然，还可以直接用搜索引擎搜索“UVa 12345”或者“livearchive 2345”。

为了方便读者，书中给出了所有例题和习题的提交统计信息，比如 511/80%是指有 511 个用户提交，其中约 80%的用户通过。

请注意，题号越大，说明加入题库的时间越晚。新加入题库的题目，其提交量不会很大，因此统计数据并不一定能客观反映其真实难度。

很多选手都有过“一道题折腾了一个星期，交了 100 次才过”的经历，可见，在算法竞赛中，坚韧不拔的精神是多么地重要。当然了，笔者并不建议读者每道题目都错上 100 次以后才罢手，因此提供了重要题目的数据和代码。毕竟，在初学阶段，学习他人的代码以及“对着评测数据调试”都是重要的学习方法。但在达到一定水平之后，最好是独立编写程序，并且不借助评测数据——要知道，在真实比赛时，你能拿到的所有数据仅仅是样例。

关于范例代码

本书中有很多代码，如果善加利用，这些代码可以帮你很大的忙；但如果滥用，非但不能发挥它们的最大好处，还可能会起到不好的作用。

关于书中的范例代码，笔者的建议只有一点：不要直接使用。理由有如下 3 点。

第一，不理解的代码不好用。所有代码都有它自身的适用范围，如果不理解其中的原理，不但有可能错误地使用这些代码，而且一旦需要对这些代码进行一定的修改才能符合题目要求时，你就会束手无策。解决方法很简单：想用的代码，事先重写一遍。这样不仅能加深你的理解，用的时候也更放心。

第二，代码习惯因人而异。书中的代码符合笔者的思维习惯和编码风格，却不一定适合你。比如，书中的代码把某个东西从 0 开始编号，而你习惯把任何东西从 1 开始编号，则当你

[①] 有的比赛是上机结束后由机器自动收取，没有明显的“提交”过程，在此情况下，请把“提交”理解成“比赛结束”。

混用自己的代码和书中的代码时，不仅编程时必须小心翼翼，而且很容易出现一些难以找到的 bug。

第三，代码风格与传统的工程代码有冲突。作为从事软件开发工作多年的工程师，笔者深知算法竞赛代码和工程代码的差异。在算法竞赛中，由于时间紧迫，很多东西都“从简”了，很多“规矩”也都被无视了。比如，算法竞赛中经常使用全局变量、内存往往是事先分配一大堆而不是按需分配、很少使用 OO 特性（顶多用几个带有成员函数的 struct，所有成员都是 public 的，并且很少用继承）、单个函数通常比工程代码长，但代码总长度通常更短，标识符通常也更短……如果本书采用传统的工程风格来编写代码，不仅会让书的厚度成倍增加，还会让读者在阅读了大量“无关紧要”的代码之后仍然不得要领，找不到最关键、最核心的地方。因此，笔者宁可让代码紧凑些，一方面让读者很容易抓住问题的核心，另一方面也锻炼了读者对程序整体逻辑（而非表达方式）的“感觉”——这恰恰是很多程序员所缺乏的[①]。一旦真正理解了这些“紧凑代码”，将其改成工程代码并不是难事。如前所述，这些代码会更好用，更符合你的需要。

不过，笔者有一点需要澄清：虽然和传统的工程代码有如此多的差异，但这并不是说本书的代码风格完全不适合工程项目。相反，本书的写作目的之一是希望读者能把两种风格有机地结合起来。比如，算法竞赛的选手写出来的代码通常比较简洁，这使程序具有更好的可读性和易维护性。笔者不主张把可读性肤浅地理解成“变量名取得长且有意义”、“注释足够多”、“每个函数都不超过 10 行”以及“拥有一个看上去很棒的类层次结构”，而应该更加看重程序的逻辑关系和执行流程。例如，对于下面这一段代码：

```
for(int i = 0; i < n; i++)
for(int j = i+1; j < n; j++) if(a[i]>a[j]) { int t = a[i]; a[i] = a[j]; a[j] = t; }
```

任何一个具备一定算法基础的程序员都能够毫不犹豫地说出它的作用，不需要任何注释，也不需要给任何变量改一个“更有意义”的名字；相反，很多工整、干净甚至看起来很优美的“工程型”的算法代码，却用了 1 000 行来完成 100 行就能搞定的功能[②]。随着代码长度的增加，潜在的 bug 数量、配套的单元测试数量以及人力成本等都会随之增长，而且增速通常快于线性函数。在这样的情况下，采用竞赛式的风格编写工程项目中的算法代码不仅是可行的，也是值得提倡的[③]。如果认真体会本书中的代码，你会发现它们比传统工程代码短的主要原因并不是变量名短、函数分得不细，而是因为逻辑更简单、清楚、直接。

考虑到参赛语言限制（ACM/ICPC 只能使用 C、C++和 Java，NOI 支持 Pascal，但不支持 Java），本书的正文选择了两种竞赛都支持的 C++。但由于笔者在工作中还使用了大量的非 C++语言（包括 Java、C#、Python、JavaScript/CoffeeScript、ActionScript、Erlang、Scala 等），因此，我们特别编写了一个附录来简单介绍其他 3 种语言——Java、C#和 Python。强烈建议读者反复阅读这个特别的附录，它不仅能开阔你的眼界，还能帮你在竞赛和真实的软件开发之间架起一座桥梁。算法是软件开发的基石，但不是全部。

[①] 如果你是在进行逆向工程，面对的是大量二进制的机器码，不仅没有注释，连变量名都没了，所能把握的就只有逻辑了。
[②] 笔者一点都没有夸张。经验表明，10 倍的比例是很正常的。
[③] 也许最大的问题是：如果仍然以 LOC（代码行数）来衡量工作量，习惯于编写紧凑代码的程序员往往很吃亏。

目　　录