# 编程之法

The Method of Programming

## 面试和算法心得

July 著

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

# 编程之法

The Method of Programming

## 面试和算法心得

July 著



人民邮电出版社
北京

**图书在版编目（CIP）数据**

编程之法 ：面试和算法心得 / July著. -- 北京 ：人民邮电出版社，2015.10
ISBN 978-7-115-38161-3

Ⅰ. ①编… Ⅱ. ①J… Ⅲ. ①程序设计 Ⅳ. ①TP311.1

中国版本图书馆CIP数据核字(2015)第218684号

## 内 容 提 要

本书涉及面试、算法、机器学习三个主题。书中的每道编程题目都给出了多种思路、多种解法，不断优化、逐层递进。本书第1章至第6章分别阐述字符串、数组、树、查找、动态规划、海量数据处理等相关的编程面试题和算法，第7章介绍机器学习的两个算法——K近邻和SVM。此外，每一章都有“举一反三”和“习题”，以便读者及时运用所学的方法解决相似的问题，且在附录中收录了语言、链表、概率等其他题型。书中的每一道题都是面试的高频题目，反复出现在最近5年各大公司的笔试和面试中，对面试备考有着极强的参考价值。

全书逻辑清晰、通俗易懂，适合热爱编程、算法、机器学习，以及准备IT笔试和面试，即将求职、找工作的读者阅读。

◆ 著　　July
责任编辑　杨海玲
责任印制　张佳莹　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：720×960　1/16
印张：17.25
字数：333千字　　2015年10月第1版
印数：1－5 000册　　2015年10月河北第1次印刷

定价：49.00元

**读者服务热线：(010)81055410　印装质量热线：(010)81055316**
**反盗版热线：(010)81055315**

# 序一

如果说计算机的出现推动了现代科技的发展，那么算法的出现则扩大了现代计算机的应用范围。硬件是计算机的基础，而算法则是计算机的灵魂。作为一名计算机爱好者、程序员抑或计算机科学家，如果不了解算法，就不能更好地理解和使用计算机。著名的计算机科学家尼古拉斯·沃斯（Nicklaus Wirth）很早就给出了著名的等式：数据结构+算法=程序。

计算机科学发展至今，虽然出现了很多新的理念和技术（如软件工程、设计模式、面向对象、极限编程、函数式编程……），但万变不离其宗，算法仍然是计算机科学的重要基石。

在各大 IT 公司的笔试和面试中，与算法相关的题目层出不穷。我想原因就是，算法是各大计算机分支的必修课，无论学哪个分支，算法都是一定要学的内容。此外，在短时间内检查一个人的逻辑思维能力，算法是最好的“试金石”。

作为一名参加过 ACM 大学生程序设计竞赛的读者，我个人对算法读物的要求也相对较高。初见《编程之法》，我想这或许也是众多“大部头”算法教材中的一种，难免是老生常谈、落入俗套。然而，我读了几章后，它却令我耳目一新。拍案叫绝之余，我总结了本书作为一本不可多得的与算法面试相关的读物所具有的几个特点。

第一，面向不同层次的读者。无论是对算法一窍不通、渴望入门的读者，还是有一定基础、希望进一步了解算法的精妙的读者，本书都有所考虑。全书结构脉络清晰，从面试、算法到机器学习，层次分明，层层深入。从最基本的数据结构到常用的算法设计思想，从基本的分治思想到现代工业界的 MapReduce、simhash，本书都有所涉及。对不同水平的读者，都会有所裨益。

第二，内容详略得当。大部头的书通常让人望而生畏，感觉可望而不可及。本书短小精悍，对于大家熟知的内容，如复杂度的基本定义，作者惜墨如金；对于复杂难懂的重点和难点，如动态规划，作者又毫不吝惜笔墨。文武之道，一张一弛。本书张弛有度，对简单问题一带而过，而对复杂问题的分析又鞭辟入里，入木三分。

第三，语言简单明了。在不影响读者理解的前提下，本书中尽量避免出现复杂的数学公式和符号。“批阅十载，增删五次”，书中的每个字、词甚至标点符号都经过仔细推敲，都凝聚了作者的心血，真可谓“十年磨一剑”。全文行云流水，读之可谓酣畅淋漓。

第四，分析深入到位。对于重要的内容，如动态规划和 simhash 算法，作者深入分析，让读者了解其产生的背景和具体过程，甚至时间和空间复杂度，让读者真正做到“知其然，知其所以然”。只有这样，理解才能更加深入，应用才能更加灵活。

第五，理论联系实际。除了理论分析外，本书几乎每一节都配有“举一反三”的习题，这些习题通常来自现实生活中的工程应用或者是著名 IT 公司的笔试和面试题，引发读者思考。让读者在领略“这些知识如何使用”的基础上真正意识到“有什么用”。“授之以鱼，不如授之以渔”。书本中的知识毕竟是有限的，而人类的思想是无限的，读者只有仔细思考，才能扩展学到的知识。不然，书本始终是书本，理论永远是理论。

我建议读者阅读本书的同时要积极思考，做一些读书笔记，以帮助理解：一方面，对于书中没有介绍的、自己不熟悉的基础知识一定要厘清，充实自己，这就是所谓的“把书读厚”；另一方面，算法没有那么神秘，不要怕算法难学、难懂，只要肯下苦功夫，掌握算法也不是难事，这就是所谓的“把书读薄”。书中的例题、习题可以亲自实现，“纸上得来终觉浅，绝知此事要躬行”，千万不要眼高手低，身入宝山而空返。

很荣幸可以为本书作序，在本书即将付梓之时，甚为激动。让我们共勉，携手共同揭开算法的神秘面纱。

曹鹏

2015 年 7 月 18 日于美国加州

# 序二

以平实的笔触来探讨算法，以初学的心态来推演公式，这是我读完全书后的第一感触。市面上关于算法的书籍并不算少了，但 July 通过分析、总结各种问题——尤其是程序员面试中的常见试题——帮助数以十万计的莘莘学子找到了期望的工作，帮助更多的人在工作中提升了自己的算法素养，得到更合适的职位。我有时会想："July 的与众不同在哪里？"答曰：心态和视角。

July 早在大二期间就深入思考并细致整理了 Dijkstra 最短路径算法、信息熵、模拟退火算法、遗传算法等问题，并且发表在网上供人自由评论。这种愿意将自己的思考历程公开分享的"开源精神"，客观上也促进 July 逐渐形成了平实、亲切的写作风格，加上后来工作中继续分享自己在算法、数据挖掘、机器学习等领域的诸多心得，日积月累，删繁就简，吐故纳新，遂成此书。

书中对诸多算法和与之相适应的数据结构的介绍完全站在初学者的视角，娓娓道来，循序渐进，层层深入，最终入木三分。这其中包括 KMP 模式串匹配问题、Manacher 最长回文子串问题、LCS 最长递增子序列问题、字符串编辑距离问题、完美洗牌问题等诸多算法，以及红黑树（2-3-4 树、STL 中的 set/map 等），布隆过滤器、hash/simhash 等海量数据问题及诸多数据结构的分析。其中，对 KMP 的分析过程完整清晰，鞭辟入里，堪称国内最好的关于 KMP 算法的论述，值得细细品读，用心琢磨。不仅如此，书中对支持向量机（SVM）的分析过程更是几易其稿、迭代精修的心力之作。与传统艰涩的公式推导甚至严密到压抑的论证方法不同，July 站在初学者的角度，将支持向量机创造性地划分为相对独立的三个理解阶段，使读者能够以更加平缓的思维梯度深刻理解以支持向量为主导的这一机器学习领域中的重要分类器。

能够以讲故事的口吻描述算法，并一直保持初学者的谦虚心态讲述每一个问题的来龙去脉，我想，这就是 July 的博客长期保持 CSDN 超高人气的重要原因。

相信本书的出版，必将帮到更多的人。

邹伟

2015 年 7 月 23 日

# 前言

## 起因

2010 年 10 月 11 日，我开始在 CSDN 上写技术博客。2011 年 1 月，在学校的时候，就有第一家出版社联系我出书，我以“时机未到，尚需积累”的理由婉拒。随后第二家、第三家出版社陆续联系我，因总感觉自己写书的时机还没到，所以都一一拒绝了。2011 年 10 月，杨海玲老师（当时她还在人民邮电出版社图灵公司工作，现在她已在人民邮电出版社信息技术分社工作）再度联系我出书，我依然觉得“时机未到”。

2014 年 1 月 18 日，我想通了一件事：如果什么都不去尝试，那么将年年一事无成。所以元旦一过，我便正式确认 2014 年之内要把拖了近 3 年之久的书出版出来（你没看错，当时的确是计划 2014 年出版，只是后来实际进展与预期计划出入很大）。

## 进展

说干就干。我花了 3 个多月的时间把博客里值得出版的文章全部移到 GitHub 上，然后通读全部文章，修正各类错误，并邀请部分朋友帮忙审阅 GitHub 上的全部文章，接着精简篇幅，调整目录。

2014 年 4 月 25 日，跟人民邮电出版社信息技术分社签订出版合同。当时很美好地觉得，只要再整理 2 个多月便可以交稿了。

干劲满满。我开始逐章、逐节、逐行、逐字优化文字描述，测试重写优化每段、每行代码，确定代码的基本风格，同时再次压缩篇幅，宁愿量少，但求质精。

但是，到了与出版社约定的交稿日期 2014 年 6 月 30 日，书稿却延期了，原因是当时的版本不是所能做到的最好版本。

想把书写好，先把自己逼疯。2014 年 7 月至 9 月，我邀请一些好友跟我一起，开始一轮一轮地审稿，包括优化语言描述、重绘图片、公式，以及重点修订 KMP、SVM 等内容。

2014 年 10 月 8 日，开始一章一章给出版社提交稿件。不过，没想到这一章一

章交稿的过程竟持续了半年多！书稿就这样反复修改，反复优化，反复确认，直到2015年4月底，我才交完全部7章书稿的初稿。

2015年5月，开始陆续收到杨海玲老师发给我的书稿反馈，并不断与她一起讨论、优化。杨老师非常认真负责，给出了大量细致、详尽的修改建议，包括标点符号、术语规范、文字表述、语句逻辑、图片、代码等一切细节。

直至2015年7月20日，经过反复修改、确认，书稿终于基本定稿。

# 内容

本书涉及面试、算法和机器学习三个主题，但主要是面试和算法方面的内容，与机器学习相关的内容相对较少。

书中的很多编程题目都给出多种思路和多种解法，在解决一道道编程问题的过程中，通过更好的算法不断优化解法、逐层深入，注重提高广大初学者的编程能力、思考能力，以及运用编程技巧和高效的算法解决实际应用问题的能力。

第1章阐述与字符串相关的面试题，第2章阐述与数组相关的面试题，第3章阐述与树相关的面试题和数据结构（红黑树、B树等），第4章阐述与查找相关的面试题（重点介绍KMP），第5章阐述与动态规划相关的面试题，第6章阐述与海量数据处理相关的面试题，第7章介绍机器学习的两个算法——K近邻、SVM。此外，每一章都有“举一反三”和“习题”，以便读者及时运用所学的方法解决相似的问题，并且在附录中收录了语言、链表、矩阵、栈、队列、图搜索、概率统计、系统设计等其他题型。

书中的每一道题都是面试的高频题目，反复出现在最近5年各大互联网公司的笔试和面试中，对面试、备考有着极强的参考价值。

全书逻辑清晰、通俗易懂，非常适合热爱编程、算法、机器学习，以及准备IT笔试和面试，即将求职、找工作的读者阅读。

# 致谢

感谢我博客上所有读者的访问、浏览、关注、支持、留言、评论、批评、指正，仅以本书献给我博客的所有读者。

感谢Boshen、sallen450、marchtea、nateriver520等朋友帮我把博客上的部分经典文章移到GitHub上。

感谢 zhou1989、qiwsir、DogK、x140yu、ericxk、zhanglin0129、idouba.net、gaohua、kelvinkuo 等朋友帮我把 GitHub 上的文章转为 Word 文件。

感谢顾运、mastermay、丰俊丙、陈友和等朋友帮忙重绘书中的部分图和重录书中的部分公式。

感谢 cherry、王威扬、邬勇、高增琪、武博文、杨忠宝、葛立娜、林奔、王婷、何欢、许利杰、王亮、陈赢、李祥老师、litaoye、李元超、刘琪、weedge、Frankie 等众多朋友帮忙审校书稿。

特别感谢曹鹏、邹伟两位博士。感谢他们非常认真细致地看完了全部书稿，给出了非常多的建设性意见，并为本书作序。

最后，再次感谢杨海玲老师以及出版社的编辑们。感谢杨海玲老师给出了大量细致的修改建议，并且非常耐心地与我一轮一轮讨论和修改书稿。

感谢以上诸位，正因为他们的帮助，本书的质量才不断提升，从而给广大读者呈现的是更好的作品。

July

2015 年 7 月 27 日

# 目 录

# 第1章　字符串

与字符串相关的问题在各大互联网公司的笔试和面试中出现的频率极高。例如，网上广为流传的一道单词翻转题：输入“I am a student.”，要求输出“student. a am I”。

本章重点介绍 6 个典型的字符串问题，分别是字符串的旋转、字符串的包含、字符串的全排列、字符串转换成整数、回文判断、最长回文子串。这 6 个问题中，除了“将字符串转换成整数”这个问题需要特别注意细节之外，其他 5 个问题都有多种思路和多种解法，比如先从蛮力解法入手，然后考虑是否能逐步优化。

读完本章后读者会发现，好的思路都是在充分考虑到问题本身的特征的前提下，或巧用合适的数据结构，或选择合适的算法降低时间复杂度，或选用效率更高的算法。

# 1.1　字符串的旋转

## 题目描述

给定一个字符串，要求将字符串前面的若干个字符移到字符串的尾部。例如，将字符串"abcdef"的前 3 个字符'a'、'b'和'c'移到字符串的尾部，那么原字符串将变成"defabc"。请写一个函数实现此功能。

## 分析与解法

### 解法一：蛮力移位

初看此题，可能最先想到的方法是将需要移动的字符一个一个地移到字符串的尾部。

如果定义指向该字符串的一个指针 *s*，然后设该字符串的长度为 *n*，那么，可以先编写一个函数 *LeftShiftOne*(char* *s*, int *n*)，以完成将一个字符移到字符串尾部的功能：

```
void LeftShiftOne(char* s, int n)
{
    // 保存第一个字符
    char t = s[0];
    for (int i = 1; i < n; i++)
    {
        s[i - 1] = s[i];
    }
    s[n - 1] = t;
}
```

然后再调用 *m* 次 *LeftShiftOne* 函数，使得字符串开头的 *m* 个字符移到字符串的尾部：

```
void LeftRotateString(char* s, int n, int m)
{
    while (m--)
    {
        LeftShiftOne(s, n);
    }
}
```

这样就完成了将若干个字符移到字符串尾部的要求。

下面来分析一下这种方法的时间复杂度和空间复杂度。针对长度为 $n$ 的字符串来说，假设需要移动 $m$ 个字符到字符串的尾部，那么总共需要 $m \times n$ 次操作。同时设立一个变量保存第一个字符。因此，时间复杂度为 $O(mn)$，空间复杂度为 $O(1)$。

有没有更好的办法来降低时间复杂度呢？

**解法二：三步反转**

对于这个问题，换一个角度思考一下。既然题目要求将字符串前面的那部分原封不动地移到字符串的尾部，那么是否可以把需要移动的部分跟不需要移动的部分分开处理呢？例如，可以先将一个字符串分割成两个部分，然后将这两个部分的字符串分别反转，最后再对整个字符串进行整体反转，即可解决字符串旋转的问题。

拿题目中的例子来说，给定字符串"abcdef"，若要将"def"移动到"abc"的前面，只需要按照下述 3 个步骤操作即可。

（1）将原字符串分为 $X$ 和 $Y$ 两个部分，其中 $X$ 为"abc"，$Y$ 为"def"。

（2）将 $X$ 的所有字符反转，即相当于反转"abc"得到"cba"；再将 $Y$ 的所有字符也反转，即相当于反转"def"得到"fed"。

（3）最后，将上述步骤得到的结果再给予整体反转，即整体反转"cbafed"得到"defabc"，这样，就实现了字符串的反转。

参考代码如下：

```
void ReverseString(char* s, int from, int to)
{
    while (from < to)
    {
        char t = s[from];
        s[from++] = s[to];
        s[to--] = t;
    }
}

void LeftRotateString(char* s, int n, int m)
{
    // 若要左移动大于 n 位，那么与%n 是等价的
    m %= n;
    ReverseString(s, 0, m - 1);
    ReverseString(s, m, n - 1);
    ReverseString(s, 0, n - 1);
}
```

这种把字符串先分为两个部分，各自反转，最后整体反转的方法，俗称“三步反转”法，其时间复杂度为 $O(n)$，空间复杂度为 $O(1)$。

## 举一反三

**单词翻转**

输入一个英文句子，翻转句子中单词的顺序。要求单词内字符的顺序不变，句子中单词以空格符隔开。为简单起见，标点符号和普通字母一样处理。例如，若输入“I am a student.”，则输出“student. a am I”。

# 1.2 字符串的包含

## 题目描述

给定一长字符串 *a* 和一短字符串 *b*。请问，如何最快地判断出短字符串 *b* 中的所有字符是否都在长字符串 *a* 中？请编写函数 bool *StringContain*(string &*a*, string &*b*) 实现此功能。

为简单起见，假设输入的字符串只包含大写英文字母。下面举几个例子。

- 如果字符串 *a* 是"ABCD"，字符串 *b* 是"BAD"，答案是 true，因为字符串 *b* 中的字母都在字符串 *a* 中，或者说 *b* 是 *a* 的真子集。
- 如果字符串 *a* 是"ABCD"，字符串 *b* 是"BCE"，答案是 false，因为字符串 *b* 中的字母 E 不在字符串 *a* 中。
- 如果字符串 *a* 是"ABCD"，字符串 *b* 是"AA"，答案是 true，因为字符串 *b* 中的字母 A 包含在字符串 *a* 中。

## 分析与解法

此题初看似乎很简单，但要高效地实现并不轻松。而且，如果面试官步步紧逼，一个一个否决你想到的方法，要求你给出更快、更好的方案，恐怕就要费不少脑筋了。

### 解法一：蛮力轮询

判断短字符串 *b* 中的字符是否都在长字符串 *a* 中，最直观也是最简单的思路则是：轮询短字符串 *b* 中的每一个字符，逐个与长字符串 *a* 中的每个字符进行比较，看是否都在字符串 *a* 中。

参考代码如下：

```
bool StringContain(string &a, string &b)
{
    for (int i = 0; i < b.length(); ++i)
    {
        int j;
        for (j = 0; (j < a.length()) && (a[j] != b[i]); ++j)
            ;
        if (j >= a.length())
        {
```