算法艺术与信息学竞赛

- 《算经竞赛入门经典》的练习伴侣
- 实用编程技巧与C++11应用介绍
- 简洁、清晰、高效的C++示例代码
- 近期高质量竞赛真题分类选解
- 近期高质量竞赛真题选译

算法竞赛入门经典

习题与解答

陈　锋◎编著

清华大学出版社

算法竞赛入门经典

习题与解答

陈书锋◎编著

清华大学出版社
北京

内 容 简 介

本书是在《算法竞赛入门经典（第2版）》的基础上，延伸出来的一本习题与解答图书，它把C++语言、算法和解题有机地结合在一起，淡化理论，注重学习方法和实践技巧，是一本算法竞赛的入门和提高教材。

本书分为5章。第1章是各种编程训练技巧以及C++11语法特性的简单介绍。第2章精选了一部分《算法竞赛入门经典（第2版）》的习题进行分析、解答。第3章是ACM/ICPC比赛真题分类选解，挑选了近些年ACM/ICPC比赛中较有价值的题目进行分析并解答。第4～5章是比赛真题选译，整理并翻译了近几年来各大区域比赛中笔者认为值得学习训练的比赛真题。

如果你对算法感兴趣，如果你是一名程序员或即将成为一名程序员，如果你想大幅提升自己的算法思维能力，如果你有志于参加ACM/ICPC、NOIP、NOI等竞赛，那就来吧！本书将为你推开一扇算法世界的大门！

图书在版编目（CIP）数据

算法竞赛入门经典——习题与解答/陈锋编著. —北京：清华大学出版社，2018（2018.2重印）
（算法艺术与信息学竞赛）
ISBN 978-7-302-47658-0

Ⅰ. ①算… Ⅱ. ①陈… Ⅲ. ①计算机算法-题解 Ⅳ. ①TP301.6-44

中国版本图书馆CIP数据核字（2017）第154508号

责任编辑：贾小红
封面设计：刘　超
版式设计：刘艳庆
责任校对：马子杰
责任印制：沈　露

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 装 者：清华大学印刷厂
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：28　　**字　　数**：690千字
版　　次：2018年1月第1版　　**印　　次**：2018年2月第2次印刷
印　　数：3501～8500
定　　价：69.80元

产品编号：066737-01

推 荐 序

《算法竞赛入门经典（第 2 版）》问世时，我的心里终于放下了一块大石头。两年多来，因为工作繁忙，我几乎没有再碰及算法竞赛，各种国内外赛事的命题、培训邀请，我都一一回绝。然而，初心未变，我很希望这套丛书能够继续下去，能帮助到更多的人。所以，当这本《算法竞赛入门经典——习题与解答》终于完稿时，我的心情比许多新老读者还要激动。

这无疑是一本很特别的书！陈锋之前虽然参与了《算法竞赛入门经典——训练指南》的编写工作，但大家肯定不知道，他竟然是一名从没有参加过 NOIP 或者 ACM/ICPC 的“非职业选手”，甚至连计算机编程，他都是在大学毕业之后凭着一腔热情和执着自学完成的。正因为如此，要独立编写一本算法竞赛的书籍，对陈锋来说是一项巨大的挑战。令人高兴的是，他做到了，而且做得很棒！

本书的问世，更让我确信了两件事：

第一，“半路出家”的算法爱好者也可以通过自己的努力变得很出色，并不一定要从小接受严格的教育和训练。你看看书里的题目，有些可是顶级选手也不一定敢在比赛中挑战的。

第二，算法竞赛并不是脱离现实的“高级应试教育”。不然的话，一个看似完全不需要和算法打交道的软件工程师，干嘛要花费那么大的精力去学习算法、做题呢？

陈锋的成长轨迹很有代表性。从某种意义上说，他写的东西更能引起读者的共鸣。而且他为人热情、诚恳，可以比我有更多时间和精力与读者交流。事实上，他在写作期间已经与多位中学、大学选手和教师讨论了，目前负责维护丛书的 github 仓库和 wiki。如果你正在学习（或者刚学完）《算法竞赛入门经典（第 2 版）》，相信本书不会让你失望！

刘汝佳

前　言

“请问《算法竞赛入门经典（第 2 版）》有没有配套题解啊？很多练习题好难，真希望能有一本简单、易懂的参考解答！”经常有读者追问类似的问题。笔者在进行训练学习时，也经常会有这样的想法。虽然很多题目可以在网上搜到对应题解，但这些题解多数是解题者为方便自己做题而随手记录的，解答过程未必严密、系统，语言表达上也比较随意，初学者理解起来就有一定的难度。

多年之前，笔者曾有幸参与了《算法竞赛入门经典——训练指南》一书的编写工作，收获颇大。也正是那次，我深刻感受到了自己在算法领域的不足，以及思维能力的亟待提升。私下里，我曾和刘汝佳老师商量，就以《算法竞赛入门经典（第 2 版）》的习题为训练题目，强迫自己在解出每道题之后，再对自己的思路进行严密、仔细的剖析，通过大量的训练，使自己得到一次系统的训练和提升。这次训练，使我记了厚厚一大本的笔记，而这本笔记就是本书的缘起。

希望本书能帮助更多跟我一样迫切需要提升算法思维能力的初学者！

算法有什么用

我大学学的是机械专业，但由于对数学非常热爱，加之毕业后发现软件行业貌似比较好“混”，且工资待遇比其他行业高些，所以就进入了开发领域。经过一段时间的工作后，我发现自己经常会遇到以下一些问题：

- 程序稍微复杂一些，代码就会写的很乱。
- 程序出了问题，不知道该如何调试，只会到处修改，然后再看效果。
- 用户需求稍作改变，就想骂街。
- 特别重要的一点是，如果你想跳到外企去工作，面试时肯定会让你编一些很难的算法程序。

后来，我进入到了微软上海全球技术支持中心做外包技术支持，接触到了许多严谨、求是、好学的工程师前辈。从他们身上，我学到了一些非常有效的解决问题的思路，以及那种“活到老学到老”的人生态度。

我逐渐明白：程序是要设计的。为了设计得清晰，需要学习数据结构、操作系统原理等非常多的基础知识，而这些体系本质上是前辈人思维方法的结晶。

另外，令很多程序员头疼的调试过程，给我印象最深的是一句话：调试的本质实际上就是在定位。大多数时候，调试的过程（并发程序的调试可能就更复杂些）其实就是一个二分查找：假如有 100 行程序结果不对，就可以在第 50 行看看结果是否符合预期，如果 OK，说明问题出在后 50 行，否则前 50 行一定有问题。如此递归下去，很快就能精准定位到有问题的代码。了解二分查找的朋友都知道，这个算法复杂度是 O(logn)。

用C#开发服务端程序时，我经常会遇到内存问题，需要对垃圾收集（GC）的过程进行分析调试。深入学习之后我发现，其实GC模型的本质就是有向图。抱着这个思路再来分析解决内存问题，思路瞬间清晰了很多。

这样的例子还有很多。

在不断解决各类问题的过程中，我逐渐明白了——算法在本质上是诸多计算机学术以及实践领域积累下来的分析解决各种问题的思维方法。它不是象牙塔内的纯学术研究，更不是一堆仅能用来解决特定领域性能问题的高精尖技术。这个行业的技术人员，本质上正是以这些思维方法为武器，高效解决着不同行业领域不断涌现出的各类纷繁问题和挑战。

说到这里，我想到其他很多行业：京剧艺人每天早上要练嗓子，相声演员每天要练贯口，军人在战斗之余要进行大量训练，中医在繁忙之余要天天钻研《伤寒论》《黄帝内经》等经典……类似这样，需要认真对待并把基本功训练作为生活一部分的行业还有很多。对于笔者来说，算法思维就是IT相关行业的技术人员需要用同样态度持续不断进行训练的一项基本功。

所以，就有了这些年的学习过程，以及以本书作为省察的一个小小总结。

内容安排

本书内容分为以下5章。

第1章是各种编程训练技巧以及C++11语法特性的简单介绍。

第2章精选了一部分《算法竞赛入门经典（第2版）》的习题进行分析、解答，主要是读者反映较多的第3~11章的课后习题部分。

第3章是ACM/ICPC比赛真题分类选解，挑选了近些年ACM/ICPC比赛中较有价值的题目进行分析并解答。

第4章是比赛真题选译，整理并翻译了近几年来各大区域比赛中笔者认为值得学习训练的比赛真题。

第5章是比赛难题选译，内容类似于第4章，只是题目难度更上一个台阶。

关于C++语言的使用

本书在解答各类算法题目时，使用C++作为主要的编程语言，尽量使用STL中提供的现成数据结构，同时也尽量使用C++11的新特性。因为笔者认为，算法训练最关键的是训练解决问题的思维能力，包括抽象能力、分析能力、调试能力等，应该充分利用语言提供的语法特性使得程序更加简洁清晰，从而使解题者更专注于问题的抽象和分析本身。

关于题目代码

本书中的所有题目，笔者都是先完成代码并在线提交AC（Accepted），然后才开始编写对应的分析题解。有些需要附上代码的题目，笔者会尽可能把代码的主要部分（去掉模板代码以及C++的namespace导入部分）附在题目后面，但由于篇幅原因，实在无法全部

放入书中。还有些题目，虽然已经在线提交 AC，但由于无法严格证明题目的正确性，也没有在书中提供题解。

书中的所有代码，读者朋友们如有需要，可以通过如下网站进行下载：https://github.com/sukhoeing/aoapc-bac2nd-keys。

勘误和支持

虽然笔者已竭尽全力，力求减少纰漏，但由于水平有限，书中难免仍存在错漏之处，恳请广大读者朋友们批评指正。欢迎您将学习过程中遇到的各类问题、您对本书的想法以及宝贵意见，通过本书网站的 issues 部分一起交流。

致谢

首先要感谢刘汝佳老师，是他把我带进了算法艺术的大门，并且在工作极其繁忙的情况下一直耐心地指导着我的算法学习。

从小父亲就告诉我，对的事情一定要坚持。这句话支撑着我渡过了很多艰难的日子。同时，也要感谢我的太太梁明珠和女儿陈婉之。这三年来，我牺牲了大量本该陪伴他们的时间，投入到了本书的创作中。没有你们的支持和包容，我不可能完成这本书。

还要感谢微软工作期间经常指导我的老师张羿，从他那里，我第一次知道了世界上还有 ACM/ICPC 这回事。在如何做好一个程序员这件事上，他给了我非常多有价值的指导和帮助。

本书初稿完成之后，许多同学和朋友踊跃参与到了本书的试读中，并且提出了许多有价值的意见和反馈，他们的名字（排名不分先后）是陈飞、崔晨、杨恒杰、林永康、陈坤泽、孙博昊等。另外，书中的部分题目也参考了许多网友的在线题解，在此一并表示感谢。

最后要感谢清华大学出版社的贾小红编辑，用极大的耐心容忍着我把交稿时间一拖再拖，希望本书不会让您失望。

陈锋

目　录

第 1 章　编程技巧与 C++11 语法特性介绍

对编程技巧和编程语言语法的熟练掌握有助于提高编码速度和准确率。本章介绍一些笔者在训练过程中总结出来的编程技巧和代码片段，以及 C++11 的语法新特性，希望能够对读者有所帮助。

1.1　编 程 技 巧

本节介绍一些在使用 C++语言进行代码编写以及调试时可能用到的技巧以及常见问题。

1.1.1　排序性能问题

相对于 C 语言内置的 qsort 函数，C++中提供的 sort 函数使用起来更加方便，不需要做指针类型转换。sort 有两种用法：第一种是传入一个 functor 对象，另外一种是直接传入一个排序函数，而笔者发现这两种用法语义上都是正确的，但是笔者实际测试发现使用 functor 的版本比直接使用函数的版本快不少，测试代码如下：

```
using namespace std;
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
const int N = 10000000;
struct TS{
    int a, b, c;
};
inline bool cmp (const TS& t1, const TS& t2) {
    if(t1.a != t2.a) return t1.a < t2.a;
    if(t1.b != t2.b) return t1.b < t2.b;
    return t1.c <= t2.c;
}

int cmp4qsort(const void * a, const void * b) {
    TS *t1 = (TS*)a, *t2 = (TS*)b;
    if(t1->a != t2->a) return t1->a - t2->a;
    if(t1->b != t2->b) return t1->b - t2->b;
    return t1->c - t2->c;
}
```

```
struct cmpFunctor {
  inline bool operator() (const TS& t1, const TS& t2) {
    if(t1.a != t2.a) return t1.a < t2.a;
    if(t1.b != t2.b) return t1.b < t2.b;
    return t1.c <= t2.c;
  }
};

TS tss[N];

void genData(){
    _for(i, 0, N) {
        tss[i].a = rand();
        tss[i].b = rand();
        tss[i].c = rand();
    }
}

int main()
{
    srand(time(NULL));

    genData();
    clock_t start = clock();
    sort(tss, tss+N, cmp);
    printf("sort by funtion pointer : %ld\n", clock() - start);

    genData();
    start = clock();
    sort(tss, tss+N, cmpFunctor());
    printf("sort by functor : %ld\n", clock() - start);

    genData();
    start = clock();
    qsort(tss, N, sizeof(TS), cmp4qsort);
    printf("qsort by funtion pointer : %ld\n", clock() - start);

    return 0;
}
/*
    g++ 4.8.0 result: 编译参数 -O2
    sort by funtion pointer : 36732
```

```
    sort by functor : 6324
    qsort by function : 15996
*/
```

笔者的机器上测试发现，STL 的 sort 使用 functor 的版本是最快的，比 qsort 都快一倍多。而使用 sort 传入函数指针的版本速度是最慢的，相对于前两者有大约 6 倍和 3 倍的差距，会在一些对排序性能要求很高的题目中形成比较明显的瓶颈，提醒读者注意。

1.1.2　整数输入

最经常输入的数据类型就是 int，经常需要输入之后直接插入到一个集合或者数组中，一般的做法是建立一个临时变量，使用 cin 或者 scanf 输入之后，再将这个临时变量插入到集合中。这样稍显烦琐。可以封装读取的函数并且这样调用：

```
int readint(){
    int x; scanf("%d", &x); return x; //此处 scanf 也可以根据需要换成 cin>>x
}
vector<int> vc;
vc.push_back(readint());
```

1.1.3　循环宏定义

算法比赛中，写得最多的代码就是像这样的循环代码：

```
for(int i = 0; i < N; i++) {}
```

这里 N 也可能是一个 STL 中集合的大小，如 vector.size 之类的。许多竞赛选手习惯使用大量的宏定义来简化代码，笔者最常用的宏定义是简化这个循环的：

```
#define _for(i,a,b) for( int i=(a); i<(b); ++i)
```

这样写循环时，就会简化成_for(i, 0, N)，这里的 a、b 两个参数都可传入表达式，例如：

```
vector b;
_for(i, 1, a.size()){...}
```

宏使用得当，可以大量简化代码，最典型的例子是本书习题 9-18 中有一个五维的 DP，里面有一个 5 层 for 循环，使用宏之后，可精简的代码非常可观。

另外一个比较有用的是：

```
#define _rep(i,a,b) for(int i=(a); i<=(b); ++i)
```

1.1.4　STL 容器内容调试输出

比赛中经常用到 STL 中的容器类，如 vector 和 set，而且在调试过程中经常需要输出这些容器的内容，每次都要写循环来输出，非常烦琐。笔者封装了两个泛型函数使用 C++的

IO 流对集合进行输出：

```
template<typename T>
ostream& operator<<(ostream& os, const vector<T>& v) {
    for(int i = 0; i < v.size(); i++) os<<v[i]<<" ";
    return os;
}

template<typename T>
ostream& operator<<(ostream& os, const set<T>& v) {
    for(typename set<T>::iterator it = v.begin(); it != v.end(); it++)
os<<*it<<" ";
    return os;
}
```

使用方法如下：

```
vector<int> a; a.push_back(1); a.push_back(2); a.push_back(3);
cout<<a;                    //输出 1 2 3

set<string> b; b.insert("1"); b.insert("2"); b.insert("3");
cout<<b;                    //输出 1 2 3
```

1.1.5 二维几何运算类

在许多牵涉位置计算的题目（如本书习题 3-5）中，需要模拟物体位置并且进行移动和转向，如果每次都直接用 x 和 y 坐标分别计算，非常烦琐，其实可以使用《算法竞赛入门经典——训练指南》一书第 4 章中的几何操作类，复用向量的移动、旋转等逻辑，详细代码请参考相关章节。

```
struct Point {
  int x, y;
  Point(int x=0, int y=0):x(x),y(y) {}
  Point& operator=(Point& p) : { x = p.x; y = p.y; return *this; }
};
typedef Point Vector;

Vector operator+ (const Vector& A, const Vector& B) { return Vector(A.x+B.x,
A.y+B.y); }
Vector operator- (const Point& A, const Point& B) { return Vector(A.x-B.x,
A.y-B.y); }
Vector operator* (const Vector& A, int p) { return Vector(A.x*p, A.y*p); }
bool operator== (const Point& a, const Point &b) { return a.x == b.x && a.y
== b.y; }
```

```
    bool operator< (const Point& p1, const Point& p2) { return p1.x < p2.x ||
(p1.x == p2.x && p1.y < p2.y); }
    istream& operator>>(istream& is, Point& p) { return is>>p.x>>p.y; }
```

1.1.6　内存池

在一些题目中，需要动态分配对象。例如，表达式解析时需要动态分配语法树的结点对象。一般的做法是直接用数组开辟空间，但是未必容易事先估计出需要开辟的空间大小，在逻辑控制中还要维护一个变量进行分配和释放，如果是多种对象都要动态分配，则更加烦琐。笔者基于 vector 容器和 C++的内存分配机制，编写了一个内存池：

```
template<typename T>
struct Pool {
    vector<T*> buf;
    T* createNew() {
        buf.push_back(new T());
        return buf.back();
    }

    void dispose() {
        for(int i = 0; i < buf.size(); i++) delete buf[i];
        buf.clear();
    }
};
```

使用方法如下：

```
struct Node{...};
struct Node2{...}

Pool <Node> n1Pool;
Pool <Node2> n2Pool;
//要分配内存构造新对象时：直接就是 Node *p = n1Pool.createNew();
Node2 *p2 = n2Pool.createNew();
```

然后在每次需要释放时直接调用 dispose 方法即可，不需要再维护各种中间变量。

1.1.7　泛型参数的使用

入门经典中很多算法的封装都会在某个结构体内部开一个数组，并且使用一个类似于 MAXSIZE 的结构来全局定义这个数组的大小，典型的如图论中的 Dijkstra 等算法：

```
const int MAXSIZE;
struct Dijkstra{
```

```
    int n, m, d[MAXSIZE], p[MAXSIZE];
...
}
```

如果同一个题目（如《算法竞赛入门经典——训练指南》中的习题 UVa10269 Adventure of Super Mario）中需要在两个不同的部分都用到 Dijkstra 算法怎么办？这个时候一般的做法就是定义多个 MAXSIZE 变量，但是会比较烦琐，也容易出错。

其实可以引入 C++的泛型参数来解决这个问题：

```
template<int MAXSIZE>
struct Dijkstra{
int n, m, d[MAXSIZE], p[MAXSIZE];
...
```

使用时，就可以通过下面的方式来指定不同的 MAXSIZE：

```
Dijkstra<MAXK * MAXN> sd;
Dijkstra<MAXN> pd;
```

具体使用可以参考训练指南中 UVa10269 的实现代码。

1.1.8 位运算操作封装

在使用位向量表示集合或进行状态压缩时，有个常用操作就是取得一个整数中某一位或者连续几位对应的 int 值。这些代码写起来较为烦琐，如果一个题目中多处调用，会增加出错的可能，笔者针对这种情况封装了一个位运算的操作类：

```
template<typename TI> //TI 可以是支持位操作的任何类型，一般是 int/long long
struct BitOp{
    //反转 pos 开始，长度为 len 的区域
    inline TI flip(TI op, size_t pos, size_t len = 1) { return op ^ (((1<<len)-1)
<< pos); }

    //取得从 pos 开始，长度为 len 的区域对应的整数值
    inline TI& set(TI& op, size_t pos, int v, size_t len = 1) {
        int o = ((1<<len)-1);
        return op = (op&(~(o << pos))) | ((v&o) << pos);
    }

    //取得从 pos 开始，长度为 len 的区域对应的整数值
    inline int get(TI op, size_t pos, size_t len = 1) { return (op >> pos)
& ((1<<len)-1); }

    //输出整数的二进制表示
    ostream& outBits(ostream& os, TI i) {
```

```
        if (i) outBits(os, (i >> 1)) << (i & 1);
        return os;
    }
};
```

如果是 32 位整数位运算可以使用 BitOp<long>来调用，64 位可以使用 BitOp<long long>来调用。

1.1.9　编译脚本

一般都是使用 g++编译然后在命令行运行，每次编译都要输入一堆命令，效率较低，所以笔者使用 Windows 命令行开发了两个脚本。

（1）编译脚本（ojc.bat）：这里假设 ojc.bat 以及 g++.exe 所在的目录已经加入到系统 PATH 环境变量中：

```
cls
g++ "%1" -lm -O2 -pipe -o"%~n1.exe"
```

使用方法如下：

```
ojc UVa100.cc
```

（2）编译并且直接运行（ojr.bat）：这里同样假设 ojr.bat 以及 g++.exe 所在的目录已经加入到系统 PATH 环境变量中。ojr.bat 的内容如下：

```
cls
echo 编译
del %~n1.exe
@g++ "%1" -lm -O2 -pipe -o"%~n1.exe"
@%~n1.exe<%~n1.in
```

以下命令会直接编译源文件，然后直接从 UVa100.in 读入数据运行：

```
ojr UVa100.cc
```

1.2　C++11 语言特性介绍

笔者写作本书时主流的算法比赛以及在线 OJ 平台均已支持最新的 C++11 语言标准。从开发者的角度来看，新标准中提供了不少能提高开发效率的新特性。本节选择了一些在算法比赛中常用特性进行介绍，希望读者通过练习掌握这些语言特性，提高在比赛中的编码速度和正确率。本书后文中的题解代码也有较多使用 C++11 的案例，请读者参考。

需要注意的是，如果是使用 g++编译，编译器需要加命令行参数-std=c++11。如果用 Visual Studio，则至少需要 2013 版本。在各大 OJ 在线提交时，也要选择 C++11。