

# 浅析“最小表示法”思想在字符串循环同构问题中的应用

安徽省芜湖市第一中学

周源

## 【目录】

浅析“最小表示法”思想在字符串循环同构问题中的应用.....	1
【目录】 .....	1
【摘要】 .....	3
【关键字】 .....	3
【正文】 .....	4
一、 问题引入.....	4
1. 明确几个记号和概念.....	4
2. 问题.....	4
二、 枚举算法和匹配算法.....	4
1. 枚举算法.....	4
2. 匹配算法.....	5
3. 小结.....	5
三、 最小表示法思想.....	5

1. “最小表示法”思想的提出.....	6
2. “最小表示法”思想的定义.....	6
3. “最小表示法”在本题的应用.....	7
4. 模拟算法执行.....	10
5. 小结.....	11
四、 总结.....	12

## 【摘要】

最小表示法在搜索判重、判断图的同构等很多问题中有着重要的应用。本文就围绕字符串循环同构的判断这个问题，在很容易找到  $O(N)$  的匹配后，本文引进的“最小表示法”思想，并系统的对其下了定义，最后利用“最小表示法”思想构造出了更优秀，更自然的算法。

无论是增加“最小表示法”思想这方面的知识，提高增加竞赛中的综合素质，相信本文对同学们还是有所裨益的。

## 【关键字】

字符串 循环同构 匹配 最小表示法

## 【正文】

### 一、 问题引入

#### 1. 明确几个记号和概念

由于本篇论文主要讨论与字符串有关的算法，所以在本文中，一切未经说明的以  $s$  开头的变量均表示字符串。

(1) .  $|s| = \text{length}(s)$ ，即  $s$  的长度。

(2) .  $s[i]$  为  $s$  的第  $i$  个字符。这里  $1 \leq i \leq |s|$ 。

(3) .  $s[i \rightarrow j] = \text{copy}(s, i, j - i + 1)$ ，即截取出  $s$  的第  $i$  个字符到第  $j$  个字符的子串。这里  $1 \leq i \leq j \leq |s|$ 。特别的， $s[i \rightarrow i] = s[i]$ 。

(4) . 定义  $s$  的一次循环  $s^{(1)} = s[2 \rightarrow |s|] + s[1]$ ；而  $s$  的  $k(k > 1)$  次循环  $s^{(k)} = s^{(k-1)(1)}$ ， $s$  的零次循环  $s^{(0)} = s$ 。

(5) . 如果字符串  $s_1$  可以经过有限次循环得到  $s_2$ ，即有  $s_2 = s_1^{(k)} (k \in N)$ ，则称  $s_1$  和  $s_2$  是循环同构的。

(6) . 设有两个映射  $f_1, f_2: A \rightarrow A$ ，定义  $f_1$  和  $f_2$  的连接

$f_1 \bullet f_2(x) = f_1(f_2(x))$ ，这里  $x \in A$ 。——这个定义用于后文算法描述中。

#### 2. 问题

给定两个字符串  $s_1$  和  $s_2$ ， $|s_1| = |s_2|$ ，判断他们是否循环同构。

### 二、 枚举算法和匹配算法

#### 1. 枚举算法

很容易知道， $s_1$  的不同的循环串最多只有  $|s_1|$  个，即  $s_1^{(0)}, s_1^{(1)}, \dots, s_1^{(|s_1|-1)}$ ，所以只需要把他们一一枚举，然后分别与  $s_2$  比较即可。

枚举算法思维简单，易于实现，而它的时间复杂度是 $O(N^2)$ 级<sup>1</sup>，已经可以胜任大多数问题的要求了。然而如果 $N$ 大至几十万，几百万，枚举算法就无能为力了，有没有更优秀的算法呢？

## 2. 匹配算法

从枚举算法执行过程中很容易发现，枚举算法的本质就是在一个可以循环的字符串 $s1$ 中寻找 $s2$ 的匹配，于是联想到模式匹配的改进算法是 $O(N)$ 级的，那么在循环串中寻找匹配是不是也有线性的算法呢？回答是肯定的：

由于循环串与一般的字符串本质的区别就是前者是“循环”的，如果能去掉“循环”这个限制，那么就可以直接套用一般字符串的模式匹配算法了！显然，将 $s1$ 复制两次： $S = s1 + s1$ 做为主串，则任何与 $s1$ 循环同构的字符串至少都可以在 $S$ 中出现一次，于是可以说 $S$ 就是循环串 $s1$ 的一般字符串形式！问题成功转化为求 $s2$ 在 $S$ 中的模式匹配。——这完全可以在 $O(N)$ 级时间内解决。

## 3. 小结

很容易得到的枚举算法显然不能满足大数据的要求，于是我们从算法的执行过程入手，探查到了枚举算法的实质：模式匹配。最后，通过巧妙的构造、转换模型，直接套用模式匹配算法，得到了 $O(N)$ 级别的算法。

但是问题是否已经完美解决了呢？也许你会说：以KMP算法为首的模式匹配改进算法，都是以难理解，难记忆著称的！这的确是KMP算法的缺点，而且其next数组繁琐的计算严重制约着算法的可扩展性，看来是有必要寻求更简洁的算法了。

## 三、 最小表示法思想

---

<sup>1</sup>这里 $N=|s1|+|s2|$ 。

## 1. “最小表示法”思想的提出

首先来看一个引例：

**[引例]**有两列数， $a_1, a_2, \dots, a_n$ 和 $b_1, b_2, \dots, b_n$ ，不记顺序，判断它们是否相同。

**[分析]**由于题目要求“不记顺序”，因此每一列数的不同形式高达 $n!$ 种之多！如果要一一枚举，显然是不科学的。于是一种新的思想提出了：如果两列数是相同的，那么将它们排序之后得到的数列一定也是相同的。于是，算法复杂度迅速降为 $O(N \log_2 N)$ 级。

这道题虽然简单，却给了我们一个重要的启示：当某两个对象有多种表达形式，且需要判断它们在某种变化规则下是否能够达到一个相同的形式时，可以将它们都按一定规则变化成其所有表达形式中的最小者，然后只需要比较两个“最小者”是否相等即可！

下面我们系统的给出“最小表示法”思想的定义。

## 2. “最小表示法”思想的定义

设有事物集合 $T = \{t_1, t_2, \dots, t_n\}$ 和映射集合 $F = \{f_1, f_2, \dots, f_m\}$ ，其中 $f_i (1 \leq i \leq m)$ 是 $T$ 到 $T$ 的映射： $f_i: T \rightarrow T$ 。如果两个事物 $s, t \in T$ ，有一系列 $F$ 映射的连接使 $f_{i_1} \bullet f_{i_2} \bullet \dots \bullet f_{i_k}(t) = s$ ，则说 $s$ 和 $t$ 是 $F$ 本质相同的。

这里 $F$ 满足：

(1). 任意 $t \in T$ ，一定能在 $F$ 中一系列映射的连接的作用下，仍被映射至 $t$ 。

(2). 任意  $s, t \in T$  , 若有  $f \in F$  使  $f(s) = t$  , 则一定存在一个或一系列映射

$$f_{i_1}, f_{i_2}, \dots, f_{i_k} \in F, \text{ 他们的连接 } f_{i_1} \bullet f_{i_2} \bullet \dots \bullet f_{i_k}(t) = s.$$

由  $F$  的性质(1)可知,  $s$  和  $s$  是  $F$  本质相同的, 即“本质相同”这个概念具有自反性。

从性质(2)可知, 如果  $s$  和  $t$  是  $F$  本质相同的, 那么  $t$  和  $s$  也一定是  $F$  本质相同的 ( $s, t \in T$ )。即“本质相同”这个概念具有对称性。

另外, 根据“本质相同”概念的定义很容易知道, “本质相同”这个概念具有传递性。即若  $t_1$  和  $t_2$  是  $F$  本质相同,  $t_2$  和  $t_3$  是  $F$  本质相同, 那么一定有  $t_1$  和  $t_3$  是  $F$  本质相同的。

给定  $T$  和  $F$  , 如何判断  $T$  中的两个事物  $s$  和  $t$  是否互为  $F$  本质相同的呢? “最小表示法”就是可以应用于此类题目的一种思想。它规定  $T$  中的所有事物均有一种特殊的大小关系。然后, 根据  $F$  中的变化规则, 将  $s$  和  $t$  均化为规定大小关系中的最小者  $m_1$  和  $m_2$  , 如果两者相同, 则易知,  $s$  和  $m_1$  本质相同,  $m_1$  和  $m_2$  本质相同,  $m_2$  和  $t$  本质相同, 所以  $s$  和  $t$  是本质相同的。否则, 可以证明,  $s$  和  $t$  不是本质相同的。

### 3. “最小表示法”在本题的应用

在本题中, 事物集合  $T$  表示的是不同的字符串, 而映射集合  $F$  则表示字符串的循环法则, “事物中的大小关系”就是字符串间的大小关系。

那么, 如何将最小表示法应用于本题呢? 最简单的方法就是根据上文, 分别求出  $s_1$  和  $s_2$  的最小表示, 比较它们是否相同。如果要是很简单的这么做, 问题就非常麻烦了: 求字符串的最小表示法虽然有  $O(N)$  级算法, 但是思路十分复

杂，还不如匹配算法——如果单纯得这么做，就违背了我们寻找更好算法的初衷。这样，看上去“最小表示法”是无能为力了。

然而我们换一种思路：

和匹配算法相似，将  $s^1$  和  $s^2$  各复制一次： $u = s^1 + s^1$ ， $w = s^2 + s^2$ ；并设两个指针  $i$  和  $j$  分别指向  $u$  和  $w$  的第一个字符。

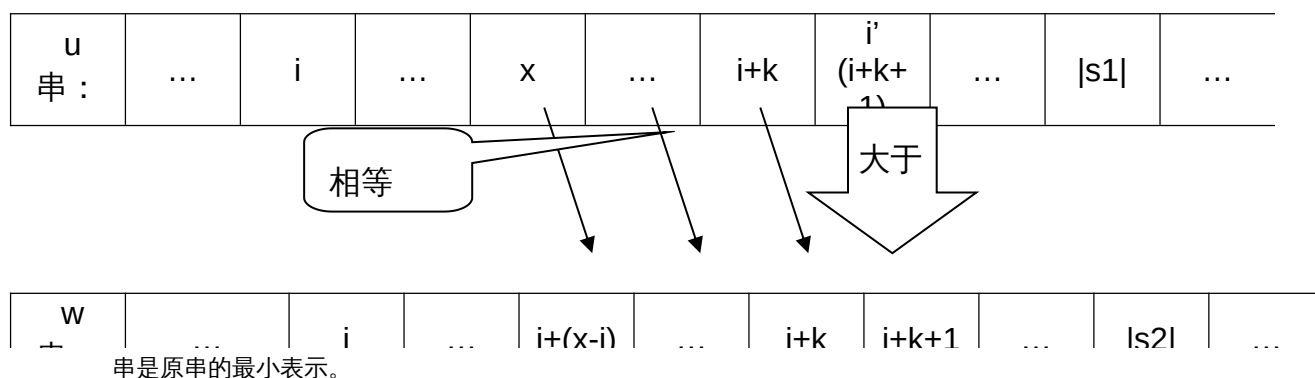
设  $M(s) = \min_{1 \leq k \leq |s|} \{ \text{任意 } 0 \leq i < |s|, \text{ 均有 } s^{(k-1)} \geq s^{(i)} \}$ ，也就是说函数  $M(s)$  返回的是  $s$  最小表示串的第一个字符在原串中的位置<sup>2</sup>，如果有多个最小表示串，

则取在原串中位置最小的一个。

显然如果  $s^1$  和  $s^2$  是循环同构的，且当前两指针  $i = M(s^1)$  且  $j = M(s^2)$  的时候，一定可以得到  $u[i \rightarrow i + |s^1| - 1] = w[j \rightarrow j + |s^2| - 1]$ ，迅速得到  $s^1$  和  $s^2$  是循环同构的。

当  $i \leq M(s^1)$  且  $j \leq M(s^2)$  时，两指针仍然有机会达到  $i = M(s^1)$ ， $j = M(s^2)$  这个状态。于是问题转化成：仍然设  $s^1$  和  $s^2$  是循环同构的，当前两指针分别向后滑动比较，如果发现比较失败，有  $u[i+k] \neq w[j+k] (k \geq 0)$ ，如何向后滑动指针，使新指针仍然满足  $i'$  和  $j'$  仍然满足  $i' \leq M(s^1)$ ， $j' \leq M(s^2)$ 。

从不相等的  $u[i+k]$  和  $w[j+k]$  下手：如果  $u[i+k] > w[j+k]$ ，那么任意整数  $x \in [i, i+k]$ ，从  $s^1$  第  $x$  个字符开头的循环串是  $s^{1(x-1)}$ ，如图， $s^{1(x-1)}$  的前  $(i'-x)$





个字符是  $u[x \rightarrow i'-1]$ ，当然，也可以表示为  $u[x \rightarrow i+k]$ ，对称的，可以在  $w$  串中找到一段字符  $w[i+(x-i) \rightarrow j+k]$ ，这两段字符串长度相等，而且根据上文假设的扫描过程可以得到  $u[x \rightarrow i+k-1] = w[i+(x-i) \rightarrow j+k-1]$ 。而  $u[i+k] > w[j+k]$ ，所以得到  $u[x \rightarrow i+k] > w[i+(x-i) \rightarrow j+k]$ ，即  $s1^{(x-1)}[1 \rightarrow i'-x+1] > w[i+(x-i) \rightarrow j+k]$ 。而根据假设  $s1$  和  $s2$  是循环同构的，那么一定能在  $s1$  的所有循环串中找到一个字符串  $s1'$ ，满足它的前  $(i'-x+1)$  个字符是  $w[i+(x-i) \rightarrow j+k]$ 。于是有  $s1^{(x-1)} > s1'$ ，得  $s1^{(x-1)}$  一定不是  $s1$  的最小表示。所以  $M(s2)$  不可能在  $[i, i+k]$  一段中，也就可以将指针  $i$  向后滑动  $(k+1)$  个字符： $i \leftarrow i+k+1$ 。

同理得到如果  $u[i+k] < w[j+k]$ ，可将指针  $j$  向后滑动  $(k+1)$  个字符：

$j \leftarrow j+k+1$ 。仍满足  $i \leq M(s1)$ ， $j \leq M(s2)$ 。

于是设计出算法：

(1) . 将  $s1$  和  $s2$  各复制一次： $u = s1 + s1$ ， $w = s2 + s2$ ；并设两个指针  $i$  和  $j$  分别指向  $u$  和  $w$  的第一个字符。

(2) . 如果  $i$  和  $j$  均中至少一个大于  $|s1|$ ，

则可以肯定  $s1$  和  $s2$  不是循环同构的，返回 *false*，算法结束；

否则找到最小的  $k \geq 0$  使  $u[i+k] \neq w[j+k]$ ，如果这样的  $k$  不存在或

$k \geq |s1|$ ，

则说明  $u[i \rightarrow i+|s1|-1] = w[j \rightarrow j+|s1|-1]$ ，即  $u$  和  $w$  各有一段长

为  $|s1|$  的子串相等， $s1$  和  $s2$  是循环同构的，返回 *true*，算法结

束；否则转第(3)步。

(3). 如果  $u[i+k] > w[j+k]$ , 则将指针  $i$  向后滑动  $(k+1)$  个字符:

$i \leftarrow i+k+1$ ; 否则说明  $u[i+k] < w[j+k]$ , 就将指针  $j$  向后滑动  $(k+1)$  个字符:  $j \leftarrow j+k+1$ 。继续执行第(2)步。

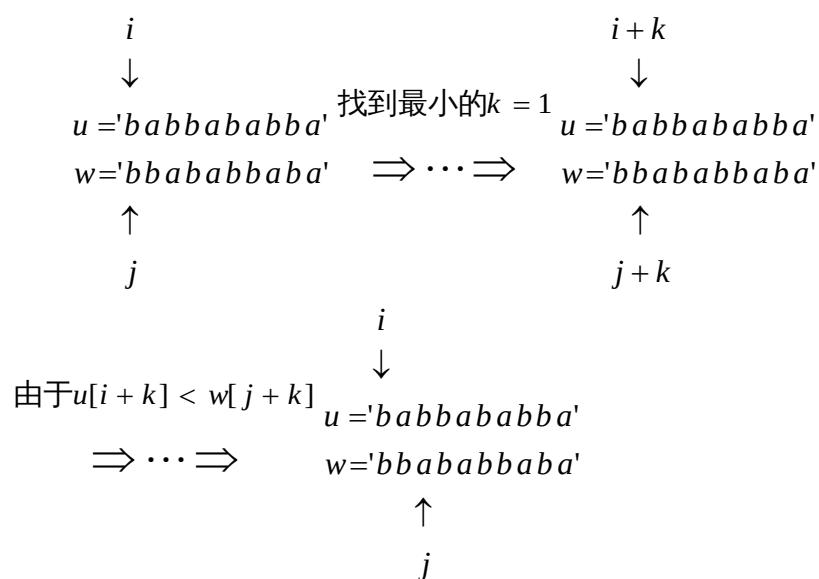
容易得出, 本算法的时空复杂度也是均为  $O(N)$  级。

更清楚一点, 我们附上一个例子:

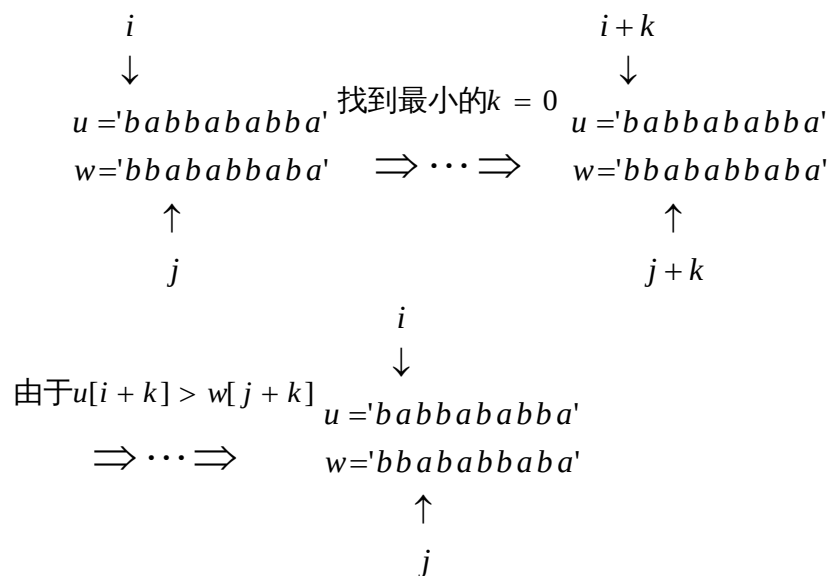
#### 4. 模拟算法执行

设  $s1 = 'babba'$ ,  $s2 = 'bbaba'$ , 显然它们是循环同构的。模拟执行算法是这样的: 首先有  $u = s1 + s1 = 'babbababba'$ ,  $w = s2 + s2 = 'bbababbaba'$ , 并设指针  $i=1, j=1$ 。

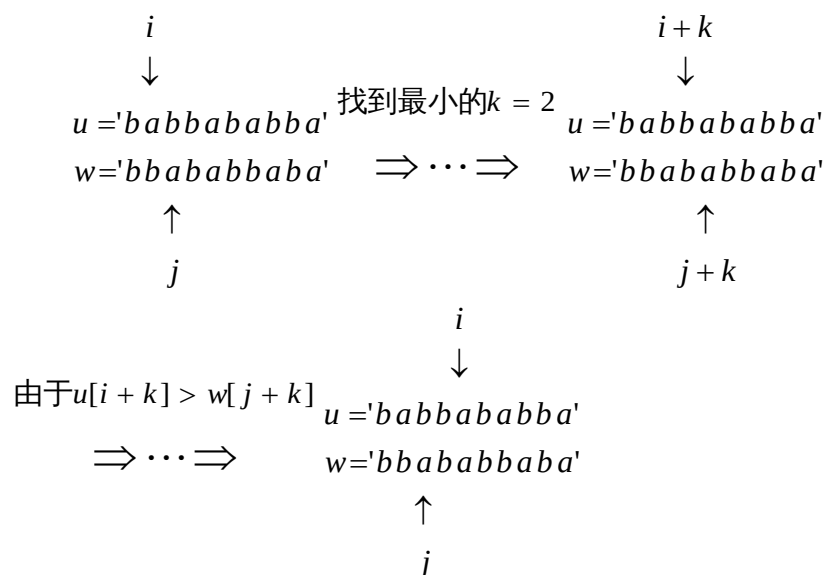
第一次执行(2)、(3)两步:



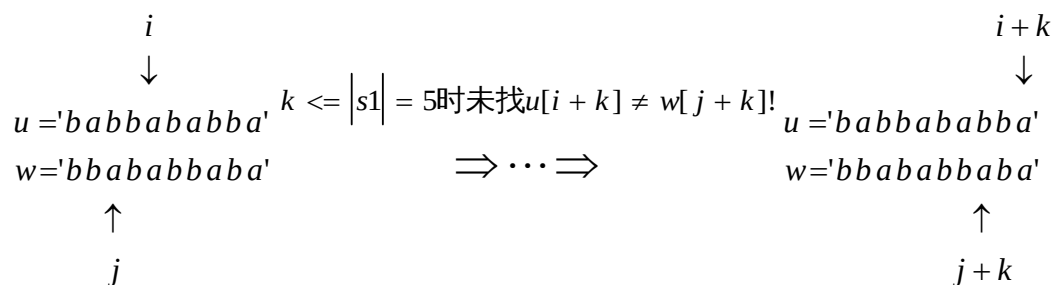
第一次执行完毕后得到  $i=1, j=3$ , 下面第二次执行:



现在结果是  $i=2, j=3$ ，同理执行第三次：



此时已经是  $i=5, j=3$ ，执行第四次：



这时发现  $u[5 \rightarrow 9] = w[3 \rightarrow 7]$ ！成功得出  $s_1$  和  $s_2$  是循环同构的，算法返回值

为真。

## 5. 小结

经过一番努力，我们终于得到了一个与匹配算法本质不同的线性算法。在这个问题中，“最小表示法”思想引导我们从问题的另一方面分析，进而构造出一个全新的算法。比起匹配算法，它容易理解，便于记忆，实现起来，也不过是短短的几重循环，非常方便，便于应用于扩展问题。

#### 四、 总结

“最小表示法”是判断两种事物本质是否相同的一种常见思想，它的通用性也是被人们认可的——无论是搜索中判重技术，还是判断图的同构之类复杂的问题，它都有着无可替代的作用。深入分析不难得出，该思想的精华在于引入了“序”这个概念，将待比较两个事物化为规定顺序中最小的（也可能是最大的）再加以比较。

然而值得注意的是，在如今的信息学竞赛中，试题纷繁复杂，使用的算法也不再拘泥于几个经典的算法，改造经典算法或是将多种算法组合是常用的方法之一。正如本文讨论的问题，单纯的寻求字符串的最小表示显得得不偿失，但利用“最小表示法”的思想，和字符串的最小表示这个客观存在的事物，我们却找到了一个简单、优秀的算法。

因此，在解决实际问题时，只有深入分析，敢于创新，才能将问题

化纷繁为简洁，  
化无序为有序。

# 论对题目中算法的选择

## 【关键词】

非最优算法，编程复杂度

**【摘要】** 本文介绍了对于一些信息学题目的次好或者差的算法，通过各种举例来说明做题时并不要一味追求一个完全理想的算法，相对应比较差的算法在实际编程上对于那些好的算法有很大的优势。最后总结出在平常的练习之中，我们需要对一道题目进行多方面的思考，不能抱有知道算法就完事的一种心态，对一道题目，要多考虑新的算法，这样面对形式未见过的题型的时候，就会有更多的思路。

## 【正文】

### 一、引言

计算机竞赛是一项对选手计算机知识、编程能力的综合测试，在平时的训练之中，我们一般都会精益求精，设法想出最好的算法，但是在竞赛的时候，时间和心理状态都是不一样的，如何在竞赛中编出能得分尽量多的程序，是很重要的，所以在平常的练习之中，要有一题多解的概念，把各种算法尝试一下，就会在看似简单的做法中推想出更好的算法。在学习汇编语言的时候，老师这样说，汇编语言编译出的程序是运行速度最快的，为什么，对于一类题目，或者要实现一个目标，人编程用的算法所需要的语句得越多，则机器运行程序速度也就越快，反之人编程用的精力越少，机器运行速度也会越慢。若能找到一个编程难度较低的算法，使程序仍然能在时间限制内运行完成，这样即减轻了人的劳动，又能编写出正确的程序，而且编写的时候错误率也会降低，所以在竞赛中，次好或者差的算法也是可以考虑的。

### 二、算法的复杂度

我在这里所说的复杂度，包括编程复杂度，时间复杂度，空间复杂度，因为在竞赛几个小时的时间里，编程复杂度就不容忽视，知道算法但是程序没完成，这是最不理想的情况，对源程序的最基本的质量要求是正确性和可靠性，只要保证在这两点的情况下，各种算法都是好算法。但编程复杂度取决于程序的易理解性，易测试性和易修改性，程序不可能都是第一遍就能达到理想的状态，人总有失误的地方，如果有错，不易测试以至难以发现错误，或者知道错误而难以修改，就会浪费不必要的时间。或许，自己对知道的算法还不太熟悉，因为我们不可能对每种算法了解都很透彻，在这种情况下编程序是比较吃亏的，因要花许多时间去了解这种方法。所以在这三个复杂度都考虑的情况下，找到最适宜的算法是必要的。

### 三、对于一些题目的思路

在我们做题的时候，我们先想出的算法不一定是最好的算法，但不是说明它是不好的算法，所以，在我们想出一种算法的时候，要考虑一下它的可行性，如果可行的话，不如自己先把该算法编一编，可以锻炼自己编那些非标准算法的能力，这样对自己面对新颖题型的时候是会有好处的。

例 1 .

[问题描述]

输入：

一个正整数  $n$ ，以及整数数列  $A_1, A_2, A_3, \dots, A_n$ 。

一个正整数  $m$ ，以及整数数列  $B_1, B_2, B_3, \dots, B_m$ 。

其中 ( $1 \leq n \leq 10^6, 1 \leq A_i \leq 10^6, 1 \leq m \leq 1000, 1 \leq B_i \leq 10^6$ )

输出

一共  $m$  行，每行一个整数，第  $i$  行所输出的数表示数列  $\{A\}$  中小于等于  $B_i$  的数的数目。

[算法分析]

该题是 IOI 中移动电话的退化。标准算法是利用线段树来解决，线段树的基本知识可以参阅相关书籍。

但我要说的做法是这样的，首先，我们便注意到了  $n$  与  $m$  的范围的差距。于是，我们采用这样的做法。

其中的记录数组如下。

$L[1..1000000]$  记录直至当前，其中  $A_i$  中取值为  $k$  的数有  $L[k]$  个

$V[1..1000]$  记录直至当前，其中  $A_i$  中取值在  $(k-1)*1000+1$  到  $k*1000$  的数总共有  $V[k]$  个。

程序流程如下：

对  $L, V$  数组进行清零。

(1) 读入  $n$ ，依次读入  $A_i$ 。对于每个  $A_i$ ，设  $(k-1)*1000+1 \leq A_i \leq k*1000$  ( $k$  为整数)

$L[A_i]++$ ,  $V[k]++$

(2) 读入  $m$ ，依次读入  $B_i$ 。对于每个  $B_i$ ，设  $(k-1)*1000+1 \leq B_i \leq k*1000$  ( $k$  为整数)

$$\text{设}\{A\}\text{中小于} B_i \text{的数有} S \text{个，易知} S = \sum_{i=1}^{k-1} V[i] + \sum_{i=(k-1)*1000+1}^{B[i]} L[i]$$

输出  $S$

[复杂度分析]

易知计算每个  $S$  的值最多需要运算操作  $1000+1000=2000$  次。

每次将一个  $A_i$  记录的操作有 2 次

利用数据分布的特点，实际上的运算次数小于  $2000 \times m$ 。

这是分治法，是一个过渡阶段的算法，是一个不完全的算法，本办法是分成  $1000 \times 1000$ ，很容易联想到分成  $100 \times 100 \times 100$ ，既而逐渐推想到线段树的做法。

让我们看一下两种算法的复杂度比较

线段树	本算法	(最坏情况)
$(n+m) \times \log_2 G$	$n \times 2 + m \times 2000$	(其中 $G=10^6$ )

所以，光凭借直觉，我们不能判断两算法时间效率的优越。虽然在大多数情况下，我们并不确定到  $n$  与  $m$  的大小，但是如果能确定两种算法都可行的情况下，用本算法就降低了编程复杂度，毕竟程序是人编的，以后要修改，理解起来也方便一点。

例 2 . 营业额统计(湖南选拔赛)

[问题描述]

公司的账本上记录了公司成立以来每天的营业额。分析营业情况是一项相当复杂的工作,营业额会出现一定的波动，当然一定的波动是能够接受的，但是在某些时候营业额突变得很高或是很低，这就证明公司此时的经营状况出现了问题。经济管理学上定义了一种**最小波动值**来衡量这种情况：

该天的最小波动值 =  $\min \{ | \text{该天以前某一天的营业额} - \text{该天营业额} | \}$

当最小波动值越大时，就说明营业情况越不稳定。

而分析整个公司的从成立到现在营业情况是否稳定，只需要把每一天的最小波动值加起来就可以了。你的任务就是编写一个程序来计算这一个值。第一天的最小波动值为第一天的营业额。

输入文件第一行为正整数  $n(n \leq 32767)$ ，表示该公司从成立一直到现在的天数，接下来的  $n$  行每行有一个正整数  $a_i(|a_i| \leq 100000000)$ ，表示第  $i$  天公司的营业额。

输出文件仅有一个正整数，即  $\sum$  每一天的最小波动值。结果小于  $2^{31}$ 。

输入输出样例

6 5 1 2 5 4 6	12
---------------------------------	----

[算法分析]

本题题意明了，关键是读入一个数，找到前面已经输入的与此数相差最小的数。

题目的标准算法是通过平衡树来解决的，平衡树，就是一种能自己进行适应，使树的深度达到尽量小的一种检索树，详细的情况可以参考相关书籍。

(1) 预处理：

- 把全部数据读入，将{a}从小到大排序，这样得到一个序列  $B_1, B_2, \dots, B_n$ 。
- 累加器 S 赋值为 0

(2) 主过程：读入将要处理的数据，对每天的营业额进行处理，求出该天的最小波动值。

(这里的 V, L 与例 1 中的定义一样。

但这次定义  $L[1..32767]$ ,  $V[1..327]$  也就是范围改了一改，以 100 为长度单位分隔区间)

- 读入当天的营业额 P
- 用二分查找法在数列{B}中找到  $B_q = P$ 
  - 设  $(k-1)*100+1 \leq q \leq k*100$ ，若  $V[k] > 0$  即该区间内已经记录过数据，在该区间内找到大于等于 q 最小的数  $g_a$ ，且  $L[g_a] > 0$ ，及小于等于 q 最大的数  $g_b$ ，且  $L[g_b] > 0$
  - 若找不到  $g_a$  或  $g_b$ ，现在说一下找不到  $g_b$  的情况 ( $g_a$  的情况相同)
  - 依次查询  $V[k-1], V[k-2]$  直至  $V[1]$  找到有  $V[t] > 0$
  - 若找到这样的 t，则在区间  $((t-1)*100+1, t*100)$  内找到最大的数 h，使  $L[h] > 0$ ，即  $g_b = h$
- 计算当天的最小波动值
  - 若找到这样的  $g_a$  或  $g_b$ ，则比较 P 与  $B_{g_a}$  的差或 P 与  $B_{g_b}$  的差，取较小的值，即为当天的最小波动值。
  - 若同时找不到  $g_a$  与  $g_b$ ，证明这是第一天，最小波动值即为 P。
- 将当天的最小波动值加入累加器 S
- 将元素 q 加入数组，即  $L[q]++$ ,  $V[k]++$

(3) 最后收尾：输出 S

(这里的 V, L 与例 1 中的定义一样。

但这次定义  $L[1..32767]$ ,  $V[1..327]$  也就是范围改了一改，以 100 为长度单位分隔区间)

详细程序参见 myturn.pas (稍有一些改进)

标准程序参见 Turnover.pas

虽然，程序的长度大致相同，但是，本算法的测试与修改却比平衡树容易的多，因为它是分两步进行，可以分步进行测试而且每步的要求技术都不高。

#### 四、总结

每个算法都是有它自己的优势，例如，我们对于排序，对于小数据我们就用冒泡，选择等时间复杂度高，但编程复杂度低的算法，反之，我们则会使用堆排序，快速排序，归并排序等，每种算法的效果在不同的场合是不一样的，在编程的过程中，在平常的练习之中，我们就需要对一道题目进行多方面的思考，不能抱有知道算法就完事的一种心态，



对一到题目，要多考虑新的算法，这样便能发现每种算法使用的场合，这样面对形式未见过的题型的时候，就会有更多的思路。

### 【参考书目】

- |                 |         |     |    |
|-----------------|---------|-----|----|
| 1. 算法与数据结构      | 傅清祥     | 王晓东 | 编著 |
|                 | 电子工业出版社 |     |    |
| 2. 实用算法的分析与程序设计 | 吴文虎     | 王建德 | 编著 |
|                 | 电子工业出版社 |     |    |

# 猜数问题的研究

上海市复旦附中 高二(8) 张宁

## 摘要：

在逻辑推理中有一类比较特殊的问题——“思维嵌套”问题，即在 C 的脑海中要考虑 B 是如何思考 A 的想法。对于这种问题通常非常抽象，考虑情况又十分繁多，思想极其复杂，用一般方法分析效果极差。本文以一个典型的“思维嵌套”问题——猜数问题为出发点，用一种新的方法从问题本质入手分析，很好的解决了问题，并阐述了新思路的优越性。由本质入手分析，避免了表面上的“思维嵌套”，且总结出了许多结论，使得解决问题的效率大幅度上升。在新思路的指引下将原问题向更普遍的情形推广，虽然问题变得更为繁琐复杂，但是由于把握住问题的命脉，有效地解决了问题。

**关键字：**推理，思维嵌套，终结情形，一类情形，二类情形，分组

## 正文：

### 一、问题原形

#### 问题描述：

一位逻辑学教授有三名非常善于推理且精于心算的学生 A，B 和 C。有一天，教授给他们三人出了一道题：教授在每个人脑门上贴了一张纸条并告诉他们，每个人的纸条上都写了一个大于 0 的整数，且某两个数的和等于第三个。于是，每个学生都能看见贴在另外两个同学头上的整数，但却看不见自己的数。

教授轮流向 A，B 和 C 发问：是否能够猜出自己头上的数。经过若干次的提问之后，当教授再次询问某人时，此人突然露出了得意的笑容，把贴在自己头上的那个数准确无误的报了出来。

我们的问题就是：证明是否一定有人能够猜出自己头上的数，若有人能够猜出，则计算最早在第几次提问时有人先猜出头上的数，分析整个推理的过程，并总结出结论。

我们先分析一个简单的例子，观察每个人是如何进行推理的。

假设 A, B 和 C 三人，头上的数分别是 1, 2 和 3。

1) 先问 A

这时，A 能看见 B, C 两人头上的数分别是 2, 3。A 会发现自己头上只可能为  $3+2=5$ ，或者  $3-2=1$ 。可到底是 1 还是 5，A 无法判断，所以只能回答“不能”。

2) 再问 B

B 会发现自己头上只可能为  $3+1=4$ ，或者  $3-1=2$ 。可到底是 2 还是 4，B 只能从 A 的回答中入手分析：（以下为 B 脑中的分析）

1. 如果自己头上是 2。则 A 能看见 B, C 两人头上的数分别是 2, 3，A 会发现自己头上只可能为  $3+2=5$ ，或者  $3-2=1$ 。到底是 1 还是 5，A 无法判断，只能回答“不能”。这与 A 实际的回答相同，并不矛盾，所以 B 无法排除这种情况。
2. 如果自己头上是 4。则 A 能看见 B, C 两人头上的数分别是 4, 3，A 会发现自己头上只可能为  $4+3=7$ ，或者  $4-3=1$ 。到底是 1 还是 7，A 无法判断，只能回答“不能”。这也与 A 实际的回答相同，并不矛盾，所以 B 也无法排除这种情况。

B 无法判断，只能回答“不能”。

3) 再问 C

C 会发现自己头上只可能为  $2+1=3$ ，或者  $2-1=1$ 。可到底是 1 还是 3，C 只能从 A 或 B 的回答中入手分析：（以下为 C 脑中的分析）

1. 如果自己头上是 1。
  - a) A 会发现自己头上只可能为  $2+1=3$ ，或者  $2-1=1$ 。可到底是 1 还是 3，是无法判断的，只能回答“不能”。这与 A 实际的回答相同，并不矛盾。
  - b) B 会发现自己头上只可能为  $1+1=2$ （因为 B 头上是大于 0 的整数，所以 B 头上不能是  $1-1=0$ ）。B 应回答“能”。但这与 B 实际的回答矛盾。C 能以此排除头上是 1 这种情况。
2. 如果继续分析 C 头上是 3 这种情况会发现毫无矛盾（因为与实际情况相符）。

C 将准确判断头上的数是 3，所以回答“能”。

所以在第三次提问时有人猜出头上的数。

我们从每个人的角度出发，分析了头上数是 1, 2 和 3 的情况。这种方法也是我们解决简单的逻辑推理问题所采用的普遍做法。但如果将问题的规模变大，会发现问题的复杂程度会急剧上升，几乎是多一次推理，问题的复杂程度就要变大一倍。更复杂的例子，限于篇幅就不举了，但复杂程度是可以想见的。

靠如此烦琐的推理是不能很好解决问题的。原因在于有大量的“思维嵌套”，即：在 C 的脑海中要考虑 B 是如何思考 A 的想法。此外，这种方法不能够推导出有普遍意义的结论。让我们换一种思路来解决问题。

**定义：**

下面我们用第一位、第二位、第三位学生分别表示 A, B, C 三人

定义 1：用四元组  $(a_1, a_2, a_3, k)$ ， $a_1, a_2, a_3 \in \mathbb{Z}^+$ ， $k \in \{1, 2, 3\}$  来描述推理过程中的一种情形，其中三个人头上数分别为  $a_1, a_2, a_3$ ，从第一位学生开始提问，且当前的被提问者为第  $k$  位学生。

性质  $F_1$ ：对于四元组  $(a_1, a_2, a_3, k)$ ，第  $k$  位学生可以不依靠别人的回答进行推理，能够直接判断出自己头上数，并称四元组描述的情形为“终结情形”。

性质  $F_2$ ：对于四元组  $(a_1, a_2, a_3, k)$ ， $a_k < \max\{a_1, a_2, a_3\}$ ，并称四元组描述的情形为“一类情形”。

性质  $F_3$ ：对于四元组  $(a_1, a_2, a_3, k)$ ， $a_k = \max\{a_1, a_2, a_3\}$ ，并称四元组描述的情形为“二类情形”。

定义 2：集合  $S_1 = \{(a_1, a_2, a_3, k) | (a_1, a_2, a_3, k) \text{ 满足性质 } F_1\}$ 。

定义 3：集合  $S_2 = \{(a_1, a_2, a_3, k) | (a_1, a_2, a_3, k) \text{ 满足性质 } F_2\}$ 。

定义 4：集合  $S_3 = \{(a_1, a_2, a_3, k) | (a_1, a_2, a_3, k) \text{ 满足性质 } F_3\}$ 。

定义 5：若对于四元组  $(a_1, a_2, a_3, k)$

- 1) 若第  $k$  位学生不能够猜出头上的数，记  $f(a_1, a_2, a_3, k) = +\infty$ 。
- 2) 若第  $k$  位学生能够猜出头上的数，记  $f(a_1, a_2, a_3, k)$  为从第一位学生开始提问直到猜出头上的数的过程中总共的提问次数。

定义 6：对于四元组  $(a_1, a_2, a_3, k)$ ，记  $g(a_1, a_2, a_3, k, R) \in \{T, F\}$ （表示真或假）， $R \in \mathbb{Z}^+$ ，表示当三位学生头上的数分别为  $a_1, a_2, a_3$ ，第  $k$  位学生是否在恰在第  $R$  次提问时最先猜出头上的数为  $a_k$ 。

定义 7：对于四元组  $(a_1, a_2, a_3, k)$ ，记  $h(a_1, a_2, a_3, k, R) \in \{T, F\}$ （表示真或假）， $R \in \mathbb{Z}^+$ ，表示当三位学生头上的数分别为  $a_1, a_2, a_3$ ，第  $k$  位学生是否能够在第  $R$  次提问时排除头上的数为  $a_k$  的情况。

找出问题的前提，即已知条件：

- 1) 某两个数的和等于第三个
- 2) 每个人的纸条上都写了一个大于 0 的整数

每个人头上的数不是另两数的和就是另两数的差。由于不能直接判断一定是其中某个数，就只能采取排除法。由于他们不会猜错，因此在下面只需考虑他们如何能够排除头上数不同于实际的可能情况。

考虑四元组  $(a_1, a_2, a_3, k) \in S_1$ ，设  $i, j$  满足  $\{i, j, k\} = \{1, 2, 3\}$ ，则第  $k$  位学生头上的有两种情况  $a_k = a_i + a_j$  或  $a_k = |a_i - a_j|$ 。其中  $a_i + a_j > 0$ ，不能直接排除。由于  $|a_i - a_j| \geq 0$ ，所以当且仅当  $|a_i - a_j| = 0$  能够直接排除这种情况。即

$\forall (a_1, a_2, a_3, k) \in S_1 \Leftrightarrow a_i = a_j$ ，其中  $\{i, j, k\} = \{1, 2, 3\}$ 。

显然  $a_k = a_i + a_j$ ， $a_k = \max\{a_1, a_2, a_3\}$ ，因此  $S_1 \subseteq S_3$ 。注意到集合  $S_2$  与  $S_3$  的定义，显然  $S_2 \cap S_3 = \emptyset$ ，因此必然有  $S_1 \cap S_2 = \emptyset$ 。

考虑四元组  $(a_1, a_2, a_3, k) \notin S_1$ ，即第  $k$  位学生不能直接猜出自己头上的数，就需要通过推理来排除头上数的两种情况中的一种。当第  $k$  位学生假设自己头上是某数，并通过推理发现别人理应能够在前两次提问中最先猜出头上的数，就可以根据实际上别人并没有猜出这一矛盾来排除其中一种情况。（可以参考前面例子中对 C 思想的分析）

显然三位学生都不可能通过推理排除自己头上数的实际情况（因为他们永远是永远猜错的）。所以下面仅需考虑如何排除头上数不同于实际的另一种情况。

对于四元组  $(a_1, a_2, a_3, k)$ ，为了讨论方便，不妨设  $a_3 = a_1 + a_2$ 。

注：下面的  $\vee$  为析取符号，即  $T \vee T = T, T \vee F = T, F \vee T = T, F \vee F = F$

- 1) 当  $k=1$  时

$$\begin{aligned} g(a_1, a_2, a_3, k, R) &= T \\ \Leftrightarrow g(a_3 - a_2, a_2, a_3, 1, R_1) &= T \\ \Rightarrow h(a_2 + a_3, a_2, a_3, 1, R_1) &= T \\ \Leftrightarrow g(a_2 + a_3, a_2, a_3, 2, R_1 - 2) \vee g(a_2 + a_3, a_2, a_3, 3, R_1 - 1) &= T \end{aligned}$$

- 2) 当  $k=2$  时

$$\begin{aligned} g(a_1, a_2, a_3, k, R) &= T \\ \Leftrightarrow g(a_1, a_3 - a_1, a_3, 2, R_2) &= T \\ \Rightarrow h(a_1, a_1 + a_3, a_3, 2, R_2) &= T \\ \Leftrightarrow g(a_1, a_1 + a_3, a_3, 1, R_2 - 1) \vee g(a_1, a_1 + a_3, a_3, 3, R_2 - 2) &= T \end{aligned}$$

- 3) 当  $k=3$  时

$$\begin{aligned}
&g(a_1, a_2, a_3, k, R) = T \\
&\Leftrightarrow g(a_1, a_2, a_1 + a_2, 3, R_3) = T \\
&\Rightarrow h(a_1, a_2, |a_1 - a_2|, 3, R_3) = T \\
&\Leftrightarrow g(a_1, a_2, |a_1 - a_2|, 1, R_3 - 2) \vee g(a_1, a_2, |a_1 - a_2|, 2, R_3 - 1) = T
\end{aligned}$$

对于四元组  $(a_1, a_2, a_3, k)$  , 设  $i, j$  满足  $\{i, j, k\} = \{1, 2, 3\}$  ,  $a'_i = a_i$  ,  $a'_j = a_j$  。

当  $a_k = a_i + a_j$  时,  $a'_k = |a_i - a_j|$  , 当  $a_k = |a_i - a_j|$  时,  $a'_k = a_i + a_j$  。

1. 当  $k > i$  时, 设  $V_i = V - (k - i)$  , 当  $k < i$  时, 设  $V_i = V - (n + k - i)$  。

2. 当  $k > j$  时, 设  $V_j = V - (k - j)$  , 当  $k < j$  时, 设  $V_j = V - (n + k - j)$  。

由上面定义可得  $g(a_1, a_2, a_3, k, V) = T \Rightarrow g(a'_1, a'_2, a'_3, i, V_i) \vee g(a'_1, a'_2, a'_3, j, V_j) = T$

分别考虑四元组  $(a_1, a_2, a_3, k) \in S_2$  和  $(a_1, a_2, a_3, k) \in S_3$  两种情形：

1) 当  $(a_1, a_2, a_3, k) \in S_2$  , 有  $a_k < \max\{a_1, a_2, a_3\}$  , 则  $a_k = |a_i - a_j|$  。由于  $S_1 \cap S_2 = \emptyset$  , 第  $k$  位学生须通过推理排除头上数为  $a'_k = a_i + a_j$  的情况。

若  $\exists (a_1, a_2, a_3, k) \in S_2$  满足  $g(a_1, a_2, a_3, k, f(a_1, a_2, a_3, k)) = T$  , 记  $V = \min\{f(a_1, a_2, a_3, k) | (a_1, a_2, a_3, k) \in S_2, g(a_1, a_2, a_3, k, f(a_1, a_2, a_3, k)) = T\}$  。  $\exists (a_1, a_2, a_3, k) \in S_2$  满足  $f(a_1, a_2, a_3, k) = V$  ,  $g(a_1, a_2, a_3, k, V) = T$  , 则可以得到  $g(a'_1, a'_2, a'_3, i, V_i) = T$  或  $g(a'_1, a'_2, a'_3, j, V_j) = T$  , 因此  $f(a'_1, a'_2, a'_3, i) = V_i$  或  $f(a'_1, a'_2, a'_3, j) = V_j$  。  $a'_k = a_i + a_j = a'_i + a'_j$  , 因此  $(a'_1, a'_2, a'_3, i) \in S_2$  ,  $(a'_1, a'_2, a'_3, j) \in S_2$  , 而  $V_i < V$  ,  $V_j < V$  , 矛盾。

**结论：**对  $\forall (a_1, a_2, a_3, k) \in S_2$  ,  $g(a_1, a_2, a_3, k, R) = F, R \in \mathbb{Z}^+$  。

2) 当  $(a_1, a_2, a_3, k) \in S_3$  , 且  $(a_1, a_2, a_3, k) \notin S_1$  。有  $a_k = \max\{a_1, a_2, a_3\}$  ,  $a_k = a_i + a_j$  , 第  $k$  位学生须通过推理排除头上数为  $a'_k = |a_i - a_j|$  的情况。由于  $(a_1, a_2, a_3, k) \notin S_1$  , 显然  $a_i \neq a_j$  。

由前面讨论 1) 可推得  $g(a_1, a_2, a_3, k, V) = T \Leftrightarrow h(a'_1, a'_2, a'_3, k, V) = T$

a) 当  $a_i > a_j$  , 即  $a'_i > a'_j$  ,  $(a'_1, a'_2, a'_3, i) \in S_3$  ,  $(a'_1, a'_2, a'_3, j) \in S_2$  。因此  $g(a'_1, a'_2, a'_3, j, V_j) = F$  ,

$$g(a_1, a_2, a_3, k, V) = T \Leftrightarrow g(a'_1, a'_2, a'_3, i, V_i) = T。$$

b) 当  $a_j > a_i$  , 即  $a'_j > a'_i$  ,  $(a'_1, a'_2, a'_3, j) \in S_3$  ,  $(a'_1, a'_2, a'_3, i) \in S_2$  。因此  $g(a'_1, a'_2, a'_3, i, V_i) = F$  ,

$$g(a_1, a_2, a_3, k, V) = T \Leftrightarrow g(a'_1, a'_2, a'_3, j, V_j) = T。$$

我们可以从考虑四元组  $(a_1, a_2, a_3, k) \in S_3$  , 变为考虑四元组

$(a'_1, a'_2, a'_3, i) \in S_3$  或  $(a'_1, a'_2, a'_3, j) \in S_3$  。这样就形成了一个线性的推理方式, 而非先前分支庞大的推理方式。

由于可以只考虑  $(a_1, a_2, a_3, k) \in S_3$ ，若  $(a_1, a_2, a_3, k) \notin S_1$ ，可以转而考虑四元组  $(a'_1, a'_2, a'_3, i) \in S_3$  或  $(a'_1, a'_2, a'_3, j) \in S_3$ 。由于三个数不可能无穷的递减，因此在有限次转化后必然达到“终结情形”。

结论：无论三个数如何变化，无论从谁开始提问，必然是头上数最大的人最先猜出自己头上的数。

由上述结论，对于  $(a_1, a_2, a_3, k)$ ，可以定义  $f(a_1, a_2, a_3, k)$  的递推式：

1) 当  $k=1$  时

1. 当  $a_2 = a_3$  时， $f(a_1, a_2, a_3, 1) = 1$
2. 当  $a_2 > a_3$  时， $f(a_1, a_2, a_3, 1) = f(a_2 - a_3, a_2, a_3, 2) + 2$
3. 当  $a_2 < a_3$  时， $f(a_1, a_2, a_3, 1) = f(a_3 - a_2, a_2, a_3, 3) + 1$

2) 当  $k=2$  时

1. 当  $a_1 = a_3$  时， $f(a_1, a_2, a_3, 2) = 2$
2. 当  $a_1 > a_3$  时， $f(a_1, a_2, a_3, 2) = f(a_1, a_1 - a_3, a_3, 1) + 1$
3. 当  $a_1 < a_3$  时， $f(a_1, a_2, a_3, 2) = f(a_1, a_3 - a_1, a_3, 3) + 2$

3) 当  $k=3$  时

1. 当  $a_1 = a_2$  时， $f(a_1, a_2, a_3, 3) = 3$
2. 当  $a_1 > a_2$  时， $f(a_1, a_2, a_3, 3) = f(a_1, a_2, a_1 - a_2, 1) + 2$
3. 当  $a_1 < a_2$  时， $f(a_1, a_2, a_3, 3) = f(a_1, a_2, a_2 - a_1, 2) + 1$

由于我们只考虑  $(a_1, a_2, a_3, k) \in S_3$ ，因此  $k$  可由  $a_1, a_2, a_3$  三个数直接确定，因此  $f(a_1, a_2, a_3, k)$  可以简化为  $f(a_1, a_2, a_3)$ 。

利用上面的公式，通过计算机编程来解决问题。参见[源程序 1](#)。

由于建立了线性的递推关系，因此避免了问题规模随着提问次数呈指数型增长。有效的解决了问题，其解决方法是建立在对问题的深入分析之上的。现在让我们总结解决问题中思路的主线：

提炼重要的前提条件 → 考虑何种情形为“终结情形” → 对非“终结情形”建立推理的等价关系 → 考虑何种情形能归结到“终结情形” → 分情况讨论并加以证明 → 得出结论并改写等价关系 → 得出公式

整个过程是从分析问题的本质入手，而非一味单纯地从每个人思想出发，并推导出普遍意义的结论。从综观全局的角度分析问题，避免了最烦琐的“思维嵌套”，并且使得问题规模从指数型转变为线性。

## 二、第一种推广

问题描述：

一位逻辑学教授有  $n(n \geq 3)$  名非常善于推理且精于心算的学生。有一天,教授给他们  $n$  人出了一道题:教授在每个人脑门上贴了一张纸条并告诉他们,每个人的纸条上都写了一个大于 0 的整数,且某个数等于其余  $n-1$  个数的和。于是,每个学生都能看见贴在另外  $n-1$  个同学头上的整数,但却看不见自己的数。

教授轮流向  $n$  发问:是否能够猜出自己头上的数。经过若干次的提问之后,当教授再次询问某人时,此人突然露出了得意的笑容,把贴在自己头上的那个数准确无误的报了出来。

我们的问题就是:证明是否一定有人能够猜出自己头上的数,若有人能够猜出,则计算最早在第几次提问时有人先猜出头上的数,分析整个推理的过程,并总结出结论。

我们对  $n=3$  时的定义加以修改:

**定义:**

定义 1: 用  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ ,  $a_1, a_2, \dots, a_n \in \mathbb{Z}^+$ ,  $k \in \{1, 2, \dots, n\}$  来描述推理过程中的一种情形,其中  $n$  个人头上数分别为  $a_1, a_2, \dots, a_n$ , 从第一位学生开始提问,且当前的被提问者为第  $k$  位学生。

性质  $F_1$ : 对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ , 第  $k$  位学生可以不依靠别人的回答进行推理,能够直接判断出自己头上数,并称  $n+1$  元组描述的情形为“终结情形”。

性质  $F_2$ : 对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ ,  $a_k < \max\{a_1, a_2, \dots, a_n\}$ , 并称  $n+1$  元组描述的情形为“一类情形”。

性质  $F_3$ : 对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ ,  $a_k = \max\{a_1, a_2, \dots, a_n\}$ , 并称  $n+1$  元组描述的情形为“二类情形”。

定义 2: 集合  $S_1 = \{(a_1, a_2, \dots, a_n, k) | (a_1, a_2, \dots, a_n, k) \text{ 满足性质 } F_1\}$ 。

定义 3: 集合  $S_2 = \{(a_1, a_2, \dots, a_n, k) | (a_1, a_2, \dots, a_n, k) \text{ 满足性质 } F_2\}$ 。

定义 4: 集合  $S_3 = \{(a_1, a_2, \dots, a_n, k) | (a_1, a_2, \dots, a_n, k) \text{ 满足性质 } F_3\}$ 。

定义 5: 若对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$

- 1) 若第  $k$  位学生不能够猜出头上的数,记  $f(a_1, a_2, \dots, a_n, k) = +\infty$ 。
- 2) 若第  $k$  位学生能够猜出头上的数,记  $f(a_1, a_2, \dots, a_n, k)$  为从第一位学生开始提问直到猜出头上的数的过程中总共的提问次数。

定义 6: 对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ , 记  $g(a_1, a_2, \dots, a_n, k, R) \in \{T, F\}$  (表示真或假),  $R \in \mathbb{Z}^+$ , 表示当三位学生头上的数分别为  $a_1, a_2, \dots, a_n$ , 第  $k$  位学生是否在恰在第  $R$  次提问时最先猜出头上的数为  $a_k$ 。



定义 7：对于  $n+1$  元组  $(a_1, a_2, \dots, a_n, k)$ ，记  $h(a_1, a_2, \dots, a_n, k, R) \in \{T, F\}$ （表示真或假）， $R \in \mathbb{Z}^+$ ，表示当三位学生头上的数分别为  $a_1, a_2, \dots, a_n$ ，第  $k$  位学生是否能够在第  $R$  次提问时排除头上的数为  $a_k$  的情况。

定义 8：对于  $(a_1, a_2, \dots, a_n, k)$ ，记  $M = \sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i$ ，

$$W = \max\{a_1, \dots, a_{k-1}, a_{k+1}, \dots, a_n\}。$$

$$\text{对于 } (a_1, a_2, \dots, a_n, k)，\exists u \in \{1, 2, \dots, n\}, a_u = \sum_{i=1}^{u-1} a_i + \sum_{i=u+1}^n a_i，$$

$\exists v \in \{1, 2, \dots, n\} \setminus \{k\}, a_v = W$ 。 $a_u = \max\{a_1, a_2, \dots, a_n\} = \max\{a_v, a_k\}$ 。所以对第  $k$  位

学生而言，头上的数可能有两种情况： $\sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i = M$ （当  $a_u = a_k$  时）或

$$a_v - (\sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i - a_v) = W - (M - W) = 2W - M \quad (\text{当 } a_u = a_v \text{ 时})。$$

考虑  $(a_1, a_2, \dots, a_n, k) \in S_1$ 。由于  $\sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i = M > 0$ ，不能直接排除。而

当且仅当  $a_v - (\sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i - a_v) = W - (M - W) = 2W - M \leq 0$  时，能够直接排除

这种情况。即  $\forall (a_1, a_2, \dots, a_n, k) \in S_1 \Leftrightarrow 2W - M \leq 0$ 。

对于  $(a_1, a_2, \dots, a_n, k) \notin S_1$ ，即  $2W - M > 0$ 。若有  $a_v = W$ ， $a_{v'} = W$ ，其中  $v, v' \in \{1, 2, \dots, n\} \setminus \{k\}, v \neq v'$ ，则必然有  $2W = a_v + a_{v'} \leq M$ ，矛盾。所以除第  $k$  位学生外剩下学生中仅有第  $v$  位学生头上的数  $a_v = W$ 。

当第  $k$  位学生假设自己头上是某数，并通过推理发现别人理应能够在前  $n-1$  次提问中最先猜出头上的数，就可以根据实际上别人并没有猜出这一矛盾来排除其中一种情况。

1) 当  $k = u$  时，即  $a_k = M$ ，因此第  $k$  位学生需排除头上数是  $2W - M$ 。令

$$\forall i \in \{1, 2, \dots, n\} \setminus \{k\}, a'_i = a_i, a'_k = 2W - M$$

2) 当  $k \neq u$  时，即  $a_k = 2W - M$ ，因此第  $k$  位学生需排除头上数是  $M$ 。令

$$\forall i \in \{1, 2, \dots, n\} \setminus \{k\}, a'_i = a_i, a'_k = M$$

与  $n=3$  时的问题原形类似，可以得到如下式子：

$$\begin{aligned}
& g(a_1, a_2, \dots, a_n, k, R) = T \\
& \Rightarrow h(a'_1, a'_2, \dots, a'_n, k, R) = T \\
& \Leftrightarrow g(a'_1, a'_2, \dots, a'_n, 1, R - (k - 1)) \vee g(a'_1, a'_2, \dots, a'_n, 2, R - (k - 2)) \vee \dots \\
& \quad \vee g(a'_1, a'_2, \dots, a'_n, k - 1, R - 1) \vee g(a'_1, a'_2, \dots, a'_n, k + 1, R - (n - 1)) \vee \dots \\
& \quad \vee g(a'_1, a'_2, \dots, a'_n, n - 1, R - (k + 1)) \vee g(a'_1, a'_2, \dots, a'_n, n, R - k) = T
\end{aligned}$$

注：上面总共  $n-1$  项，其中不含  $g(a'_1, a'_2, \dots, a'_n, k, R)$

分别考虑  $(a_1, a_2, \dots, a_n, k) \in S_2$  和  $(a_1, a_2, \dots, a_n, k) \in S_3$  两种情形：

- 1) 当  $(a_1, a_2, \dots, a_n, k) \in S_2$ ，有  $a_k < \max\{a_1, a_2, \dots, a_n\}$ ，则  $a_k = 2W - M$ 。由于  $S_1 \cap S_2 = \emptyset$ ，第  $k$  位学生须通过推理排除头上数为  $a'_k = M$  的情况。

若  $\exists (a_1, a_2, \dots, a_n, k) \in S_2$  满足  $g(a_1, a_2, \dots, a_n, k, f(a_1, a_2, \dots, a_n, k)) = T$ ，记

$$\begin{aligned}
V &= \min\{f(a_1, a_2, \dots, a_n, k) \mid (a_1, a_2, \dots, a_n, k) \in S_2, g(a_1, a_2, \dots, a_n, k, f(a_1, a_2, \dots, a_n, k)) = T\} \\
& \quad \text{。} \exists (a_1, a_2, \dots, a_n, k) \in S_2 \text{ 满足 } f(a_1, a_2, \dots, a_n, k) = V, \\
& \quad g(a_1, a_2, \dots, a_n, k, V) = T, \exists i \in \{1, 2, \dots, n\} \setminus \{k\}, g(a'_1, a'_2, \dots, a'_n, i, V_i) = T。 \\
& \quad \text{因此 } f(a'_1, a'_2, \dots, a'_n, i) = V_i, \text{ 而 } V_i < V, \text{ 矛盾。}
\end{aligned}$$

**结论：**对  $\forall (a_1, a_2, \dots, a_n, k) \in S_2$ ， $g(a_1, a_2, \dots, a_n, k, R) = F, R \in Z^+$ 。

- 2) 当  $(a_1, a_2, \dots, a_n, k) \in S_3$ ，且  $(a_1, a_2, \dots, a_n, k) \notin S_1$ 。有  $a_k = \max\{a_1, a_2, \dots, a_n\}$ ， $a_k = M$ ，第  $k$  位学生须通过推理排除头上数为  $a'_k = 2W - M$  的情况。此时有  $a'_v = \max\{a'_1, a'_2, \dots, a'_n\}$ ，因此  $(a'_1, a'_2, \dots, a'_n, v, V_v) \in S_3$ 。

由前面结论 1) 推得

$$g(a_1, a_2, \dots, a_n, k, V) = T \Leftrightarrow h(a'_1, a'_2, \dots, a'_n, k, V) = T$$

由于  $\forall (a_1, a_2, \dots, a_n, k) \in S_2$ ， $g(a_1, a_2, \dots, a_n, k, R) = F, R \in Z^+$ 。当  $i \neq v$  时，有  $(a'_1, a'_2, \dots, a'_n, i) \in S_2$ ，由上述结论得  $g(a'_1, a'_2, \dots, a'_n, i, V_i) = F$ 。因此  $g(a_1, a_2, \dots, a_n, k, V) = T \Leftrightarrow g(a'_1, a'_2, \dots, a'_n, v, V_v) = T$ 。

我们可以从考虑  $(a_1, a_2, \dots, a_n, k) \in S_3$ ，变为考虑

$(a'_1, a'_2, \dots, a'_n, v, V_v) \in S_3$ 。这样就形成了一个线性的推理方式，而非先前分支庞大的推理方式。

由于可以只考虑  $(a_1, a_2, \dots, a_n, k) \in S_3$ ，若  $(a_1, a_2, \dots, a_n, k) \notin S_1$ ，可以转而考虑  $(a'_1, a'_2, \dots, a'_n, v, V_v) \in S_3$ ， $a'_k = 2W - M < M = a_k$ 。由于  $n$  个数不可能无穷的递减，因此在有限次转化后必然达到“终结情形”。

**结论：**无论  $n$  个数如何变化，无论从谁开始提问，必然是头上数最大的人最先猜出自己头上的数。

由上述结论，对于  $(a_1, a_2, \dots, a_n, k)$ ，可以定义  $f(a_1, a_2, \dots, a_n, k)$  的递推式：

1) 当  $2W - M \leq 0$  时， $f(a_1, a_2, \dots, a_n, k) = k$

2) 当  $2W - M > 0$  时

设  $a'_i = a_i$ ，其中  $i \neq k$ ， $a'_k = 2W - M$

a) 当  $v < k$  时， $f(a_1, a_2, \dots, a_n, k) = f(a'_1, a'_2, \dots, a'_n, v) + k - v$

b) 当  $v > k$  时， $f(a_1, a_2, \dots, a_n, k) = f(a'_1, a'_2, \dots, a'_n, v) + n - k + v$

由于我们只考虑  $(a_1, a_2, \dots, a_n, k) \in S_3$ ，因此  $k$  可由  $n$  个数直接确定，因此  $f(a_1, a_2, \dots, a_n, k)$  可以简化为  $f(a_1, a_2, \dots, a_n)$ 。

利用上面的公式，通过计算机编程来解决问题。参见[源程序 2](#)。

至此，第一种推广情形就解决了。可以发现  $n=3$  时情形的证明，对解决一般情形提供了很好的对比，使得我们能够较为轻松地解决问题，这其实也是建立在对  $n=3$  时的情形的分析之上的。

### 三、第二种推广

#### 问题描述：

一位逻辑学教授有  $n(n \geq 3)$  名非常善于推理且精于心算的学生。有一天，教授给他们  $n$  人出了一道题：教授在每个人脑门上贴了一张纸条并告诉他们，每个人的纸条上都写了一个大于 0 的整数，并将他们分成了两组（一组学生有  $m$  人，且  $m \geq n/2$ ，且学生并不知道如何分组），且两组学生头上数的和相等。于是，每个学生都能看见贴在另外  $n-1$  个同学头上的整数，但却看不见自己的数。

教授轮流向  $n$  发问：是否能够猜出自己头上的数。经过若干次的提问之后，当教授再次询问某人时，此人突然露出了得意的笑容，把贴在自己头上的那个数准确无误的报了出来。

我们的问题就是：证明是否一定有人能够猜出自己头上的数，若有人能够猜出，则计算最早在第几次提问时有人先猜出头上的数，分析整个推理的过程，并总结出结论。

由于当  $n=3$  时， $m$  只可能为 2，即为问题原形，而对于  $m=n-1$ ，即第一种推广情形。因此在下文只讨论  $n>3$ ， $m<n-1$  时的情形。

对于每个人判断自己头上的数，依据分组情况不同，头上的数就可能不同。

对  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生可以看见除自己外所有学生头上的数，并假设在某种分组情况下，可以计算出与自己不同组的学生头上数的和，由题目条件“两组学生头上数的和相等”，可以计算出自己头上的数。由于有  $C_n^m$  种分组

情况，因此相对应头上的数有  $C_n^m$  种（其中可能也包括了一部分重复的数及非正整数）。

定义：

定义 1：用  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ， $A_1, A_2, \dots, A_n \in Z^+$ ， $k \in \{1, 2, \dots, n\}$  来描述推理过程中的一种情形，其中  $n$  个人头上数分别为  $A_1, A_2, \dots, A_n$ ，从第一位学生开始提问，且当前的被提问者为第  $k$  位学生。

性质  $F_1$ ：对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生可以不依靠别人的回答进行推理，能够直接判断出自己头上数，并称  $n+1$  元组描述的情形为“终结情形”。

性质  $F_2$ ：对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ， $A_k < \max\{A_1, A_2, \dots, A_n\}$ ，并称  $n+1$  元组描述的情形为“一类情形”。

性质  $F_3$ ：对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ， $A_k = \max\{A_1, A_2, \dots, A_n\}$ ，并称  $n+1$  元组描述的情形为“二类情形”。

定义 2：集合  $S_1 = \{(A_1, A_2, \dots, A_n, k) | (A_1, A_2, \dots, A_n, k) \text{ 满足性质 } F_1\}$ 。

定义 3：集合  $S_2 = \{(A_1, A_2, \dots, A_n, k) | (A_1, A_2, \dots, A_n, k) \text{ 满足性质 } F_2\}$ 。

定义 4：集合  $S_3 = \{(A_1, A_2, \dots, A_n, k) | (A_1, A_2, \dots, A_n, k) \text{ 满足性质 } F_3\}$ 。

定义 5：若对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$

- 1) 若第  $k$  位学生不能够猜出头上的数，记  $f(A_1, A_2, \dots, A_n, k) = +\infty$ 。
- 2) 若第  $k$  位学生能够猜出头上的数，记  $f(A_1, A_2, \dots, A_n, k)$  为从第一位学生开始提问直到猜出头上的数的过程中总共的提问次数。

定义 6：对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ，记  $g(A_1, A_2, \dots, A_n, k, R) \in \{T, F\}$ （表示真或假）， $R \in Z^+$ ，表示当三位学生头上的数分别为  $A_1, A_2, \dots, A_n$ ，第  $k$  位学生是否在恰在第  $R$  次提问时最先猜出头上的数为  $a_k$ 。

定义 7：对于  $n+1$  元组  $(A_1, A_2, \dots, A_n, k)$ ，记  $h(A_1, A_2, \dots, A_n, k, R) \in \{T, F\}$ （表示真或假）， $R \in Z^+$ ，表示当三位学生头上的数分别为  $A_1, A_2, \dots, A_n$ ，第  $k$  位学生是否能够在第  $R$  次提问时排除头上的数为  $a_k$  的情况。

定义 8：对于  $(A_1, A_2, \dots, A_n, k)$ ，记  $M = \sum_{i=1}^{k-1} a_i + \sum_{i=k+1}^n a_i$ ，

$W = \max\{A_1, \dots, A_{k-1}, A_{k+1}, \dots, A_n\}$ 。

定义 9：对于  $(A_1, A_2, \dots, A_n, k)$ ，用  $X$  指代第  $k$  位学生推测的任意一种分组情况。定义  $G(X)$  为第  $k$  位学生假设分组  $X$  情况下两组学生头上数的和。

定义 10：对于  $(A_1, A_2, \dots, A_n, k)$ ，当第  $k$  学生在考虑某一可能分组的情况下，记  $A'_k$  为所推测出的头上数的可能值。

定义 11：对于  $(A_1, A_2, \dots, A_n, k)$ ，定义  $At_1 \geq At_2 \geq \dots \geq At_{n-1}$ ，其中  $\{t_1, t_2, \dots, t_{n-1}\} = \{1, 2, \dots, n\} \setminus \{k\}$ 。

定义 12：对于  $(A_1, A_2, \dots, A_n, k)$ ，定义分组  $T$ ：选取第  $t_1, t_2, \dots, t_m$  位学生为一

组，剩下的学生为另一组（第  $k$  位学生在这一组）。则  $G(T) = \sum_{i=1}^m At_i$

对于  $(A_1, A_2, \dots, A_n, k)$ ，考虑分组  $T$  的情况， $G(T) = \sum_{i=1}^m At_i = \sum_{i=m+1}^{n-1} At_i + A'_k$ ，

显然  $A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i \geq At_1$ （第一个求和符号有  $m$  项，第二个求和符号有  $n-1-m$  项，由于  $m \geq n/2 \geq n-m > n-1-m$ ，且  $At_1 \geq At_2 \geq \dots \geq At_{n-1}$ ）。

对于  $(A_1, A_2, \dots, A_n, k)$  任意一个分组  $X$ ，设  $\{x_1, x_2, \dots, x_{n-1}\} = \{1, 2, \dots, n\} \setminus \{k\}$

1) 当与第  $k$  位学生不同组的学生有  $m$  人时，设  $G(X) = \sum_{i=1}^m Ax_i$ （ $m$  项）。

显然有  $G(T) = \sum_{i=1}^m At_i \geq \sum_{i=1}^m Ax_i = G(X)$

$$A'_k = \sum_{i=1}^m Ax_i - \sum_{i=m+1}^{n-1} Ax_i = 2 \sum_{i=1}^m Ax_i - \sum_{i=1}^m Ax_i - \sum_{i=m+1}^{n-1} Ax_i = 2 \sum_{i=1}^m Ax_i - \sum_{i=1}^{n-1} Ax_i = 2G(X) - M$$

2) 当与第  $k$  位学生同组的学生有  $m$  人时，设  $G(X) = \sum_{i=1}^{n-m} Ax_i$ （ $n-m$  项）。

显然有  $G(T) = \sum_{i=1}^m At_i \geq \sum_{i=1}^{n-m} At_i \geq \sum_{i=1}^{n-m} Ax_i = G(X)$

$$A'_k = \sum_{i=1}^{n-m} Ax_i - \sum_{i=n-m+1}^{n-1} Ax_i = 2 \sum_{i=1}^{n-m} Ax_i - \sum_{i=1}^{n-m} Ax_i - \sum_{i=n-m+1}^{n-1} Ax_i = 2 \sum_{i=1}^{n-m} Ax_i - \sum_{i=1}^{n-1} Ax_i = 2G(X) - M$$

考虑  $(A_1, A_2, \dots, A_n, k) \in S_1$  的条件：

I. 当  $(A_1, A_2, \dots, A_n, k) \in S_2$ 。

考虑在分组 T 的情况下， $A_k < \max\{A_1, A_2, \dots, A_n\} = At_1 \leq A'_k$ ，这种情况不同于实际情况，需要进行推理。得到结论： $\forall (A_1, A_2, \dots, A_n, k) \in S_2$ ， $(A_1, A_2, \dots, A_n, k) \notin S_1$ 。

II. 当  $(A_1, A_2, \dots, A_n, k) \in S_3$ 。

考虑实际分组 B 的情况

1) 当与第 k 位学生不同一组的学生有 m 人时，

$$G(B) = \sum_{i=1}^m Ab_i = \sum_{i=m+1}^{n-1} Ab_i + A_k。在分组 T 的情况下，有$$

$$A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i \geq \sum_{i=1}^m Ab_i - \sum_{i=m+1}^{n-1} Ab_i = A_k。$$

2) 当与第 k 位学生同一组的学生有 m 人时，

$$G(B) = \sum_{i=1}^{n-m} Ab_i = \sum_{i=n-m+1}^{n-1} Ab_i + A_k。当 m = n/2 时，即为上面一种情况，因$$

此只考虑  $m > n/2$

$$在分组 T 的情况下，有 A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i > \sum_{i=1}^{n-m} Ab_i - \sum_{i=n-m+1}^{n-1} Ab_i = A_k。$$

要使得  $(A_1, A_2, \dots, A_n, k) \in S_1$ ，必然要满足  $A_k = A'_k$ 。因此实际分组 B

$$只可能是上面第一种情况，且满足 \sum_{i=1}^m Ab_i - \sum_{i=m+1}^{n-1} Ab_i = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i，$$

$$显然 \sum_{i=1}^{n-1} A_i = \sum_{i=1}^m Ab_i + \sum_{i=m+1}^{n-1} Ab_i = \sum_{i=1}^m At_i + \sum_{i=m+1}^{n-1} At_i，因此可得$$

$$\sum_{i=1}^m Ab_i = \sum_{i=1}^m At_i，即 G(B) = G(T)。$$

显然  $G(X) \leq G(T) = G(B)$ ，在此条件下，若存在可能的分组 C 满足  $G(C) < G(B)$ ， $A'_k = 2G(C) - M < 2G(B) - M = A_k$ 。显然由于  $(A_1, A_2, \dots, A_n, k) \in S_1$ ，必然有  $A'_k \leq 0$ ，即  $2G(C) - M \leq 0$ 。

若不存在可能的分组 C 满足  $G(C) < G(B)$ ，即任意分组  $G(X) = G(B)$ ，则  $At_1 = At_2 = \dots = At_{n-1}$ 。

进一步分析对可能的分组 C 满足  $G(C) < G(B)$ ，何时满足  $2G(C) - M \leq 0$ 。分两种情况考虑：

1) 当  $At_2, At_3, \dots, At_{n-1}$  不全相等时，必然有  $At_2 > At_{n-1}$ ，考虑如下分组

C：第  $t_1, t_3, t_4, \dots, t_{m-1}, t_m, t_{n-1}$  位学生为一组，第  $k, t_2, t_{m+1}, t_{m+2}, \dots, t_{n-2}$  位学生为一组，此时显然有

$$G(C) = At_1 + \sum_{i=3}^m At_i + At_{n-1} < \sum_{i=1}^m At_i = G(B)。$$

需满足  $2G(C) - M \leq 0$ ，即  $G(C) \leq M - G(C)$ ，因此

$$At_1 + \sum_{i=3}^m At_i + At_{n-1} = G(C) \leq M - G(C) = At_2 + \sum_{i=m+1}^{n-2} At_i。 另一方$$

面， $At_1 \geq At_2, At_3 \geq At_{m+1}, At_4 \geq At_{m+2}, \dots, At_{n-m} \geq At_{n-2}$ ，而

$m \geq n - m$ ，因此

$$At_1 + \sum_{i=3}^m At_i + At_{n-1} > At_1 + \sum_{i=3}^m At_i \geq At_1 + \sum_{i=3}^{n-m} At_i \geq At_2 + \sum_{i=m+1}^{n-2} At_i， 矛$$

盾。因此，

$\forall (A_1, A_2, \dots, A_n, k)$  满足  $At_2 > At_{n-1}, (A_1, A_2, \dots, A_n, k) \notin S_1。$

2) 当  $At_2, At_3, \dots, At_{n-1}$  全相等时，不妨设， $At_1 = p$ ，

$\forall i \in \{2, 3, \dots, n-1\}, At_i = q。$

考虑分组 C：m 个头上数为 q 的学生为一组，剩下的学生为一组（第 k 位学生在这组中）。

由于  $2G(C) - M \leq 0$ ，即  $2mq - [p + (n-2)q] \leq 0$ ，因此

$$(2m - n + 2)q \leq p。$$

1. 当  $m > n/2$  时，再来考虑分组 D：头上数是 p 的学生与 n-m-1 个头上数是 q 的学生在一组，剩下学生为一组。由于  $m > n/2$ ，因此  $G(D) < G(B)。$

由于  $2G(D) - M \leq 0$ ，因此

$2[p + (n-m-1)q] \leq a + (n-2)b$ ，因此  $p \leq (2m-n)q$ ，观察得到的两个不等式，显然矛盾，因此当  $m > n/2$  时，

$\forall (A_1, A_2, \dots, A_n, k)$  满足  $At_1 > At_{n-1}, (A_1, A_2, \dots, A_n, k) \notin S_1。$

2. 当  $m = n/2$  时，可以得到  $A_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i = At_1 = p$ ，且

$$2q \leq p。$$

因此,  $(A_1, A_2, \dots, A_n, k) \in S_1$  的条件为:

1.  $(A_1, A_2, \dots, A_n, k) \in S_3$
2.  $At_1 = At_2 = \dots = At_{n-1}$  或  
 $m = n/2, p, q \in \mathbb{Z}^+, 2q \leq p, A_k = At_1 = p, \forall i \in \{2, 3, \dots, n-1\}, At_i = q$

分情况进行讨论  $(A_1, A_2, \dots, A_n, k) \notin S_1$  的情况:

I. 对于  $(A_1, A_2, \dots, A_n, k) \in S_2$

若  $\exists (A_1, A_2, \dots, A_n, k) \in S_2$ ,  $g(A_1, A_2, \dots, A_n, k, f(A_1, A_2, \dots, A_n, k)) = T$ ,

记

$$V = \min\{f(A_1, A_2, \dots, A_n, k) \mid (A_1, A_2, \dots, A_n, k) \in S_2, g(A_1, A_2, \dots, A_n, k, f(A_1, A_2, \dots, A_n, k)) = T\}$$

。  $\exists (A_1, A_2, \dots, A_n, k) \in S_2$  满足  $f(A_1, A_2, \dots, A_n, k) = V$ ,

$$g(A_1, A_2, \dots, A_n, k, V) = T。$$

若第  $k$  位学生能够猜出自己头上的数, 则他一定要通过推理排除头上数所有可能情况中与实际不相等的值。

由于  $(A_1, A_2, \dots, A_n, k) \in S_2$ , 即  $A_k < \max\{A_1, A_2, \dots, A_n\}$ , 因此  
 $\max\{A_1, A_2, \dots, A_n\} = \max\{At_1, At_2, \dots, At_{n-1}\}$ 。考虑分组 T 的情况,  
 $A_k < \max\{A_1, A_2, \dots, A_n\} = \max\{At_1, At_2, \dots, At_{n-1}\} = At_1 \leq A'_k$ , 因此需要  
 排除这种情况。令  $A'_i = A_i, i \in \{t_1, t_2, \dots, t_{n-1}\}$ 。

$$g(A_1, A_2, \dots, A_n, k, R) = T$$

$$\Rightarrow h(A'_1, A'_2, \dots, A'_n, k, R) = T$$

$$\Leftrightarrow g(A'_1, A'_2, \dots, A'_n, t_1, Rt_1) \vee g(A'_1, A'_2, \dots, A'_n, t_2, Rt_2) \vee \dots$$

$$\vee g(A'_1, A'_2, \dots, A'_n, t_{n-1}, Rt_{n-1}) = T$$

注: 上面  $Rt_1, Rt_2, \dots, Rt_{n-1} \in \mathbb{Z}^+$  为代定变量, 显然有

$Rt_1, Rt_2, \dots, Rt_{n-1} < R$ , 由于具体值与结论无关, 因此不必讨论。

1) 当  $A'_k > At_1$

则可得  $(A'_1, A'_2, \dots, A'_n, t) \in S_2, t \in \{t_1, t_2, \dots, t_{n-1}\}$ 。

$\exists i \in \{1, 2, \dots, n-1\}$ , 使得  $g(A'_1, A'_2, \dots, A'_n, t_i, Vt_i) = T$ ,

$f(A'_1, A'_2, \dots, A'_n, t_i) = Vt_i$ , 而  $Vt_i < V$ , 矛盾。

因此得到结论: 对于  $\forall (A_1, A_2, \dots, A_n, k) \in S_2$ , 若在分组 T 的情况下满足  $A'_k > At_1$ , 则  $g(A_1, A_2, \dots, A_n, k, R) = F, R \in \mathbb{Z}^+$ 。

2) 当  $A'_k = At_1$

由于, 因此等号成立的条件为  $A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i = At_1$ , 由

于  $At_1 \geq At_2 \geq \dots \geq At_{n-1}$ , 因此仅当  $m = n/2$  存在这种情况, 此时

$$At_2 = At_3 = \dots = At_{n-1}。$$



1. 若  $At_1 = At_2$

考虑实际分组 B 的情况，显然  $Ab_1 = Ab_2 \cdots = Ab_{n-1}$ ，

$$A_k = \sum_{i=1}^m Ab_i - \sum_{i=m+1}^{n-1} Ab_i = Ab_1, \text{ 此时 } (A_1, A_2, \dots, A_n, k) \notin S_2。$$

2. 若  $At_1 > At_2$

显然  $(A'_1, A'_2, \dots, A'_n, t_1) \in S_3$ ，对  $\forall i \in \{2, 3, \dots, n\}$ ，

$(A'_1, A'_2, \dots, A'_n, t_i) \in S_2$ 。

若  $\exists i \in \{2, \dots, n-1\}$ ， $g(A'_1, A'_2, \dots, A'_n, t_i, Vt_i) = T$ ，

$f(a'_1, a'_2, \dots, a'_n, t_i) = Vt_i$ ，而  $Vt_i < V$ ，矛盾。

因此

$$g(A_1, A_2, \dots, A_n, k, V) = T \Rightarrow g(A'_1, A'_2, \dots, A'_n, t_1, Vt_1) = T。$$

设  $At_1 = p$ ， $\forall i \in \{2, 3, \dots, n-1\}$ ， $At_i = q$ 。

由于  $m = n/2$ ，对第 k 位学生只有两种本质不同的分组

情况，即第 k 位学生与第  $t_1$  位学生在同一组或不在同一组，

两种情况对应头上数的可能值分别是：

$$A_k = \sum_{i=m}^{n-1} At_i - \sum_{i=1}^{m-1} At_i = 2q - p \text{ 及 } A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i = p$$

设  $\forall i \in \{1, 2, \dots, n-1\}$ ， $A't_i = At_i$ ，对  $(A'_1, A'_2, \dots, A'_n, t_1)$ ，第  $t_1$

位学生只有两种本质不同的分组情况，两种情况对应头上数

的可能值分别是：

$$A't_1 = p \text{ 及 } A''t_1 = 2q - p$$

设  $\forall i \in \{1, 2, \dots, n\} \setminus \{t_1\}$ ， $A''_i = A'_i$ 。

$$g(A'_1, A'_2, \dots, A'_n, t_1, R) = T$$

$$\Rightarrow h(A''_1, A''_2, \dots, A''_n, t_1, R) = T$$

$$\Leftrightarrow g(A''_1, A''_2, \dots, A''_n, k, Rk) \vee g(A''_1, A''_2, \dots, A''_n, t_2, Rt_2) \vee \cdots \\ \vee g(A''_1, A''_2, \dots, A''_n, t_{n-1}, Rt_{n-1}) = T$$

显然  $(A''_1, A''_2, \dots, A''_n, k) \in S_3$ ，对  $\forall i \in \{2, 3, \dots, n\}$ ，

$(A''_1, A''_2, \dots, A''_n, t_i) \in S_2$ 。

若  $\exists i \in \{2, \dots, n-1\}$ ， $g(A''_1, A''_2, \dots, A''_n, t_i, Vt_i) = T$ ，

$f(A''_1, A''_2, \dots, A''_n, t_i) = Vt_i$ ，而  $Vt_i < V$ ，矛盾。

因此

$$g(A'_1, A'_2, \dots, A'_n, t_1, Vt_1) = T \Rightarrow g(A''_1, A''_2, \dots, A''_n, k, V'k) = T$$

考虑  $(A''_1, A''_2, \dots, A''_n, k)$  ,  $A''t_1 = 2q - p$  ,

$\forall i \in \{2, 3, \dots, n-1\}, A''t_i = q$  , 对第  $k$  位学生只有两种本质不同

的分组情况, 两种情况对应头上数的可能值分别是:

$$A''_k = p \text{ 及 } A'''_k = 2q - p$$

设  $\forall i \in \{1, 2, \dots, n\} \setminus \{k\}$  , 且  $i \neq k$  ,  $A'''_i = A''_i$  .

$$g(A''_1, A''_2, \dots, A''_n, k, V'k) = T$$

$$\Rightarrow g(A'''_1, A'''_2, \dots, A'''_n, t_2, V''t_2) \wedge g(A'''_1, A'''_2, \dots, A'''_n, t_2, V''t_3) \wedge \dots$$

$$\wedge g(A'''_1, A'''_2, \dots, A'''_n, t_{n-1}, V''t_{n-1}) = T$$

定义:

$$V_{\min} = \min\{f(A'''_1, A'''_2, \dots, A'''_n, t_2), f(A'''_1, A'''_2, \dots, A'''_n, t_2), \dots, f(A'''_1, A'''_2, \dots, A'''_n, t_{n-1})\}$$

显然  $f(A''_1, A''_2, \dots, A''_n, k) > V_{\min}$

当只有两种本质不同的分组情况时, 排除其一即可猜出

头上的数, 利用上述结论讨论  $f(A_1, A_2, \dots, A_n, k)$ 。

a) 当  $k > t_1$

$$\begin{aligned} & f(A_1, A_2, \dots, A_n, k) \\ &= f(A'_1, A'_2, \dots, A'_n, t_1) + k - t_1 \\ &= f(A''_1, A''_2, \dots, A''_n, k) + (n - k + t_1) + (k - t_1) \\ &> V_{\min} + n \end{aligned}$$

b) 当  $k < t_1$

$$\begin{aligned} & f(A_1, A_2, \dots, A_n, k) \\ &= f(A'_1, A'_2, \dots, A'_n, t_1) + n + k - t_1 \\ &= f(A''_1, A''_2, \dots, A''_n, k) + (t_1 - k) + (n + k - t_1) \\ &> V_{\min} + n \end{aligned}$$

由于  $f(A_1, A_2, \dots, A_n, k) = V$  ,  $(A_1, A_2, \dots, A_n, t_1) \in S_3$  ,

$\forall i \in \{2, 3, \dots, n\}$  ,  $(A_1, A_2, \dots, A_n, t_i) \in S_2$  , 因此对

$(A_1, A_2, \dots, A_n, k)$ ，只有第  $k$  位学生和第  $t_1$  位学生两人可能最先猜出头上的数。

对第  $t_1$  位学生也只有两种本质不同的分组情况，即第  $k$  位学生与第  $t_1$  位学生在同一组或不在同一组，两种情况对应头上数的可能值分别是：

$$At_1 = p \text{ 及 } A'''t_1 = 2q - p$$

设  $\forall i \in \{1, 2, \dots, n\} \setminus \{k\}$ ， $A_i''' = A_i$ 。

比较后可发现  $\forall i \in \{1, 2, \dots, n\}$ ， $A_i''' = A_i''$

$$h(A_1'', A_2'', \dots, A_n'', t_1, Vt_1) = T$$

$$\Leftrightarrow g(A_1'', A_2'', \dots, A_n'', t_2, V''t_2) \wedge g(A_1'', A_2'', \dots, A_n'', t_2, V''t_3) \wedge \dots$$

$$\wedge g(A_1'', A_2'', \dots, A_n'', t_{n-1}, V''t_{n-1}) = T$$

由于对第  $t_1$  位学生只有两种本质不同的分组情况，因此排除其一即可猜出头上的数，即  $f(A_1, A_2, \dots, A_n, t_1) = Vt_1$ 。

$$\text{因而 } f(A_1, A_2, \dots, A_n, t_1) = Vt_1 < V_{\min} + n$$

$$\text{显然 } f(A_1, A_2, \dots, A_n, t_1) < f(A_1, A_2, \dots, A_n, k)$$

则说明第  $t_1$  位学生将在第  $k$  位学生之前猜出头上的数，

说明  $g(A_1, A_2, \dots, A_n, k, V) = F$ ，矛盾。

得到结论：对于  $\forall (A_1, A_2, \dots, A_n, k) \in S_2$ ，若在分组  $T$  的情况下满足  $A'_k = At_1$ ，则  $g(A_1, A_2, \dots, A_n, k, R) = F, R \in Z^+$ 。

因此综合考虑上述所有情况，得到结论：对于

$$\forall (A_1, A_2, \dots, A_n, k) \in S_2, \quad g(A_1, A_2, \dots, A_n, k, R) = F, R \in Z^+。$$

II. 对于  $(A_1, A_2, \dots, A_n, k) \in S_3$

并非所有  $(A_1, A_2, \dots, A_n, k) \in S_3$ ，第  $k$  位学生都能猜出头上的数，下面举一个例子：

当  $n=6, m=3$  时，考虑  $(1,1,2,3,3,4,6) \in S_3$ （实际分组为 1, 3, 3 为一组，1, 2, 4 为一组），对于第 6 位学生须排除头上数是 6 的可能（分组为 2, 3, 3 为一组，1, 1, 6 为一组），记

$V = f(1,1,2,3,3,4,6)$ ， $g(1,1,2,3,3,4,6,V) = T \Rightarrow h(1,1,2,3,3,6,6,V') = T$ ，由前面结论可得

$$h(1,1,2,3,3,6,6,V) = T \Leftrightarrow g(1,1,2,3,3,6,1,V_1) \vee g(1,1,2,3,3,6,2,V_2) \vee \dots \vee g(1,1,2,3,3,6,5,V_5)$$

，而  $\forall i \in \{1,2,\dots,5\}$ ， $(1,1,2,3,3,6,i) \in S_2$ ，即  $g(1,1,2,3,3,6,i,V_i) = F$ 。因此  $g(1,1,2,3,3,4,6,V) = F$ ，即第 6 位学生不能最先猜出头上的数。而  $\forall i \in \{1,2,\dots,5\}$ ， $(1,1,2,3,3,4,i) \in S_2$ ，考虑到前面讨论 1，因此没有人能猜出头上的数。

由于对于能够猜出头上数的情形，在形式上不具特殊性，因此我们只有在推理过程中不断判定是否有可能猜出头上的数。

对于  $(A_1, A_2, \dots, A_n, k)$ ，若  $A_k < \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i$ ，则分组 T 的情况

$$\text{下， } A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i > A_k = \max\{1,2,\dots,n\}，$$

$$\forall i \in \{1,2,\dots,n-1\}, A't_i = At_i，\text{则 } \forall i \in \{1,2,\dots,n-1\}，$$

$$(A'_1, A'_2, \dots, A'_n, t_i) \in S_2，\text{由前面讨论 1 可得 } g(A_1, A_2, \dots, A_n, k, V) = F。$$

显然有  $A_k \leq \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i$ ，因此判定条件之一为

$$A_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i。$$

可能有多个学生头上的数同为最大数，若有多个学生头上的数同为最大数，则  $At_1 = A_k$ 。下面分情况讨论：

1. 当  $At_2 > At_{n-1}$  或  $m > n/2$

$$\text{则分组 T 的情况下， } A'_k = \sum_{i=1}^m At_i - \sum_{i=m+1}^{n-1} At_i > At_1 = A_k，\text{由前面}$$

的判定条件，在这种情形下，第  $k$  位学生不能够最先猜出头上的数。

当  $At_i = A_k$  , 利用前面结论可知第  $t_i$  位学生不能够最先猜出头上的数。而当  $At_i < A_k$  ,  $(A_1, A_2, \dots, A_n, t_i) \in S_2$ 。因此没有人能猜出头上的数。

2. 当  $At_2 = At_1 = A_k$  ,  $m = n/2$ 
  - a) 若  $At_2 > At_{n-1}$  , 此时即为第 1)种情况。
  - b) 若  $At_2 = At_3 = \dots = At_{n-1}$  , 则所有  $n$  个数都相等, 此时  $(A_1, A_2, \dots, A_n, k) \in S_1$ 。

3. 当  $At_1 > At_2$  ,  $m = n/2$  ,  $n \geq 5$   
显然当  $n$  为奇数时, 必然有  $m > n/2$  , 即第 1)种情况。因此下面只考虑  $n$  为偶数。

- a) 若  $At_2 > At_{n-1}$  , 此时即为第 1)种情况。
- b) 若  $At_2 = At_3 = \dots = At_{n-1}$

对第  $k$  位学生只有两种本质不同的分组, 不妨设

$$A_k = At_1 = p , \forall i \in \{2, 3, \dots, n-1\}, At_i = q。$$

- i. 若  $2q \leq p$  ,  $(A_1, A_2, \dots, A_n, k) \in S_1$ 。
- ii. 若  $2q > p$  , 考虑须排除的情况,  $A'_k = 2q - p$  , 令  $\forall i \in \{1, 2, \dots, n-1\}, A'_i t_i = At_i$ 。

显然  $\forall i \in \{2, 3, \dots, n-1\}$  ,  $(A'_1, A'_2, \dots, A'_n, t_i) \in S_2$

$$g(A_1, A_2, \dots, A_n, k, V) = T \Rightarrow g(A'_1, A'_2, \dots, A'_n, t_1, Vt_1) = T$$

考虑  $(A'_1, A'_2, \dots, A'_n, t_1)$  , 则  $A''t_1 = 2q - p$  , 令

$$\forall i \in \{2, 3, \dots, n-1\}, A''t_i = A'_i t_i。$$

显然  $(A'_1, A'_2, \dots, A'_n, k) \in S_2$  ,

$\forall i \in \{2, 3, \dots, n-1\}, A''t_i = At_i = q$ 。观察  $A'_1, A'_2, \dots, A'_n$  , 可以发现即为第 1)种情况。因此, 第  $k$  位学生不能够最先猜出头上的数。

当  $At_i = A_k$  , 利用前面结论可知第  $t_i$  位学生不能够最先猜出头上的数。而当  $At_i < A_k$  ,  $(A_1, A_2, \dots, A_n, t_i) \in S_2$ 。因此没有人能猜出头上的数。

4. 当  $At_1 > At_2$  , 且  $n = 4$

则必然有  $m = 2$  , 显然可以得到  $At_2 = At_3$  , 不妨设

$A_k = At_1 = p$  ,  $At_2 = At_3 = q$  , 并称这种两两相等的情形为“A 类情形”。

对第  $k$  位学生只有两种本质不同的分组方式, 考虑须排除的情况,  $A'_k = 2q - p, A'_1 t_1 = p, A'_2 t_2 = A'_3 t_3 = q$ 。

显然  $\forall i \in \{2, 3\}$  ,  $(A'_1, A'_2, \dots, A'_n, t_i) \in S_2$

$$g(A_1, A_2, \dots, A_n, k, V) = T \Rightarrow g(A'_1, A'_2, \dots, A'_n, t_1, Vt_1) = T$$

对第  $t_1$  位学生只有两种本质不同的分组方式，考虑须排除的情况， $A''t_1 = 2q - p, A''_k = 2q - p, A''t_2 = A''t_3 = q$ 。又回到了“A 类情形”

又由于四数中始终有一个数在减小，因此有限次推理之后，必然达到“终结情形”，此时满足条件为  $2q \leq p$ 。

因此一定有人能够猜出头上的数。但具体由第  $k$  位学生和第  $t_1$  位学生中哪一人最先猜出头上的数需要具体计算后才能确定。至此，讨论完所有存在两个以上的学生头上为最大数的情形。

可能的分组  $C$  满足  $G(C) < G(B)$ ，进行讨论。

在这种情况下， $A'_k = 2G(C) - M < 2G(B) - M = A_k$ 。令

$$\forall i \in \{1, 2, \dots, n-1\}, A't_i = At_i。$$

1) 当  $A'_k > At_1$  时

$\forall i \in \{1, 2, \dots, n-1\}$ ， $(A'_1, A'_2, \dots, A'_n, t_i) \in S_2$ ，由前面讨论 1 可得  $g(A_1, A_2, \dots, A_n, k, V) = F$ 。因此第  $k$  位学生不能够最先猜出头上的数。由  $At_1 < A'_k < A_k$ ，因此  $\forall i \in \{1, 2, \dots, n-1\}$ ， $(A_1, A_2, \dots, A_n, t_i) \in S_2$ 。因此没有人能够猜出头上的数。

2) 当  $A'_k = At_1$  时

此时  $A't_1 = A'_k = \max\{A'_1, A'_2, \dots, A'_n\}$ ，可以利用前面对多个学生头上数同为最大数时的讨论的结果：

1. 当  $m > n/2$ （讨论多个最大数时的讨论 1）

对  $\forall i \in \{1, 2, \dots, n\}$ ，考虑  $(A'_1, A'_2, \dots, A'_n, i)$ ，第  $i$  位学生不能够最先猜出头上的数。即没人能猜出头上的数。因此，对于  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生不能够最先猜出头上的数。

2. 当  $A't_2 = A't_1 = A'_k$ ， $m = n/2$ （讨论多个最大数时的讨论 2）

a) 当  $A't_2 > A't_{n-1}$

对  $\forall i \in \{1, 2, \dots, n\}$ ，考虑  $(A'_1, A'_2, \dots, A'_n, i)$ ，第  $i$  位学生不能够最先猜出头上的数。即没人能猜出头上的数。因此，对于  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生不能够最先猜出头上的数。

b) 当  $A't_2 = A't_{n-1}$

由于  $A'_k = A't_1 = A't_2 = \dots = A't_{n-1}$ ，因此可以得知

$$A_k = At_1 = At_2 = \dots = At_{n-1}，即 (A_1, A_2, \dots, A_n, k) \in S_1。$$

3. 当  $A't_2 < A't_1$ ， $n \geq 5$ ， $m = n/2$ （讨论多个最大数时的讨论 3）

a) 当  $A't_2 > A't_{n-1}$

对  $\forall i \in \{1, 2, \dots, n\}$ ，考虑  $(A'_1, A'_2, \dots, A'_n, i)$ ，第  $i$  位学生不能够最先猜出头上的数，即没人能猜出头上的数。因此，

对于  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生不能够最先猜出头上的数。

b) 当  $A't_2 = A't_{n-1}$

不妨设  $A'_k = A't_1 = p$ ， $\forall i \in \{2, 3, \dots, n-1\}, A't_i = q$ 。

i. 当  $2q \leq p$

可以由之前讨论得到  $(A'_1, A'_2, \dots, A'_n, k) \in S_1$ ，于是第  $k$  位学生头上的数仅有一种可能情况，不属于讨论的范围。

ii. 当  $2q > p$

对  $\forall i \in \{1, 2, \dots, n\}$ ，考虑  $(A'_1, A'_2, \dots, A'_n, i)$ ，第  $i$  位学生不能够最先猜出头上的数，即没人能猜出头上的数。

因此，对于  $(A_1, A_2, \dots, A_n, k)$ ，第  $k$  位学生不能够最先猜出头上的数。

4. 当  $A't_2 < A't_1$ ，且  $n = 4$ （讨论多个最大数时的讨论 4）

显然可以得到  $A't_2 = A't_3$ ，此时不存在实际划分  $B$ ，使得

$G(C) < G(B)$ ，即这种情况不属于讨论的范围。

3) 当  $A'_k < A't_1$  时

$(A'_1, A'_2, \dots, A'_n, t_1) \in S_3$ ，因此，第  $k$  位可能最先猜出头上的数，因此需要继续推理。

结论：对于  $(A_1, A_2, \dots, A_n, k) \in S_3$ ，第  $k$  位学生可能最先猜出头上的

数，需要继续推理的判定条件为：
$$A_k = \sum_{i=1}^m A't_i - \sum_{i=m+1}^{n-1} A't_i = 2G(T) - M$$

即  $G(B) = G(T)$ ，且对任意可能分组  $C$  符合  $G(C) < G(B)$ ，满足

$$A'_k < A't_1。$$

当  $n = 4, m = 2$ ，考虑  $(A_1, A_2, A_3, A_4, k) \in S_3$

1) 当  $A_k = A't_1$ ，由前面讨论多个最大数时的讨论 4，及，一定有人能够猜出头上的数。

2) 当  $A_k > A't_1$ ，显然  $A_k = A't_1 + A't_2 - A't_3$ ，因此  $A't_1 + A't_2 - A't_3 > A't_1$ ，即  $A't_2 > A't_3$ 。第  $k$  位学生头上数其余两种可能值，

$A'_k = A't_1 + A't_3 - A't_2 < A't_1$ ， $A''_k = A't_2 + A't_3 - A't_1 < A't_1$ 。由前面结论，需要继续推理。

由于仅有这两种情况，即不存在情况使得没有人能够猜出头上的数。且推理时四个数始终在减小，因此经过有限次推理之后，必然达到“终结情形”。

而对于第一种推广情形，即  $n = 4, m = 3$ ，必然有人能猜出自己头上的数。

因此  $n = 4$  时的一切情况，必然有人能猜出自己头上的数。

由于现在的推理在加强判定的情况下，依然可能出现多种考虑情况。所以推理已不是线性的推理，整个推理过程将成为树状结构。

由于分组情况繁多，而且判定方式也比较复杂，因此这时计算  $f(A_1, A_2, \dots, A_n, k)$  的值已经非人力能够解决，但是已经可以编程解决问题了，参见[源程序 3](#)。

## 结束语

本文深入地分析了一个逻辑推理问题，从综观全局的角度来考虑问题的本质联系，而非一味单纯地从每个人思想出发，简化了最烦琐的“思维嵌套”，并在此基础上建立了递推关系，因此避免了问题规模随着推理次数急剧增长，有效地解决了问题。并通过对比将问题推广到更为一般的情形，尤其对于第二种推广情形，存在极为烦琐的讨论，但其讨论问题的核心思想是一致的。对解决逻辑推理的问题提供了一种可以借鉴的方法。

## 参考文献

《CTSC2001 分析》

《算法与数据结构》 傅清祥 王晓东 编著



# 浅谈补集转化思想在统计问题中的应用

## 目录

前言.....2

关键字.....2

摘要.....2

正文.....2

    例一.....3

        题目大意.....3

        初步分析.....3

        深入思考.....3

        补集转化.....4

        小结.....5

    例二.....6

        题目大意.....6

        初步分析.....6

        几个工具.....7

        补集转化.....7

        小结.....8

总结.....9

## 前言

不管是在实际生活中还是在信息学竞赛里，都常常会遇到统计问题，即对满足某些性质的对象进行计数的问题。

解决统计问题有一些常用的方法比如离散化、极大化、二分、事件表等等，利用它们基本上可以解决大多数的统计问题。我们不妨将这些方法称为解决统计问题的常规方法。

然而世上的任何事物都是既有普遍性，又有特殊性的。确实存在一部分统计问题，用常规方法是很难甚至是根本无法解决的。对于这些问题，就应该具体情况具体分析，采用非常规的解法。

本文中将要讨论的就是这些非常规解法中的一种——利用补集转化思想解决统计问题。

## 关键字

统计问题 常规／非常规（统计）方法 枚举 时间／思维／编程复杂度  
补集转化 组合计数 有序化 枚举对象 枚举量

## 摘要

本文通过对两个例题——单色三角形问题（POI9714）和海战游戏（改编自 Ural1212），探讨了补集转化思想在统计问题中的应用，分析比较了补集转化思想在两个例子中的作用效果和价值。

最后得出结论：补集转化思想应用于统计问题中往往有着很好的效果，我们应该注意培养逆向思维，掌握好这种非常规的统计方法。

## 正文

统计问题，我们认为对于满足某些性质的对象进行计数的问题。既然是计数，那么不可避免地，其解法或多或少地建立于枚举之上。在很多情况下，“枚举”往往是低效的代名词。因此，我们通常所见到的统计问题，其最好解法动辄就达到  $O(n^2)$ 、 $O(n^3)$  的时间复杂度，也因此统计问题的规模往往不是很大，一般都在 1000 以内。

如果光是这样也就罢了，更糟糕的是很多统计问题，按照直观的想法设计出来的算法往往具有令人望而生畏的时间复杂度：要么  $n$  的指数过高，要么题目中给的  $n$  就是很大的，或者时间复杂度是关于另外一个很大的量  $M$ （往往是表格的尺寸、图的边数等等）的表达式……总之，这个时候的统计问题实在不那么讨人喜欢。

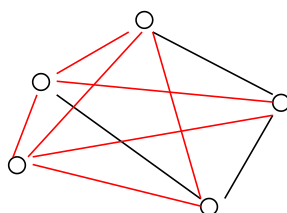
于是我们又需要利用很多技巧来降低复杂度，离散化和极大化思想、二分法、事件表等技术都是非常有用的，在它们的帮助下，我们成功地解决了非常多的刁钻的统计问题，以至于这些技巧也已经成了理所当然的“常规”方法，很多本来棘手的统计问题也成了常规题。然而这些方法显然也有不那么奏效的时候，这时我们就需要一些非常规的方法来解决一些更加棘手的问题。

这些非常规方法之一就是利用补集转化思想来帮助解决统计问题。补集转化思想在很多方面有着广泛的应用，让我们来看看在解决统计问题方面它又有哪些精彩表现吧！

## 例一 单色三角形问题 (POI9714 TRO)

### 题目大意

空间里有  $n$  个点，任意三点不共线。每两点之间都用红色或黑色线段（只有一条，非红即黑！）连接。如果一个三角形的三条边同色，则称这个三角形



图

是单色三角形。对于给定的红色线段的列表，找出单色三角形的个数。例如图一中有 5 个点，10 条边，形成 3 个单色三角形。

输入点数  $n$ 、红色边数  $m$  以及这  $m$  条红色的边所连接的顶点标号，输出单色三角形个数  $R$ 。  $3 \leq n \leq 1000$ ，  $0 \leq m \leq 250000$ 。

### 初步分析

很自然地，我们想到了如下算法：用一个  $1000 \times 1000$  的数组记录每两个点之间边的颜色，然后枚举所有的三角形（这是通过枚举三个顶点实现的），判断它的三条边是否同色，如果同色则累加到总数  $R$  中（当然，初始时  $R$  为 0）。

这个算法怎么样呢？姑且不论它非常奢侈地需要一个 1M 的大数组（POI97 的时候还没有大内存），它的时间复杂度已经高达  $O(C(n,3))$ ，也就是  $O(n^3)$  的级别。对于  $n$  最大达到 1000 的本题来说，实在不能说是个好算法。

那些常规的技巧能不能用到本题上呢？离散化和极大化看起来跟本题毫不沾边。由于本题的限制条件非常少，也不是求最值型的计数问题，所以二分和事件表看来也用不上。

看来，想要循规蹈矩地解决这题是不可能了，我们需要全新的思路！

## 深入思考

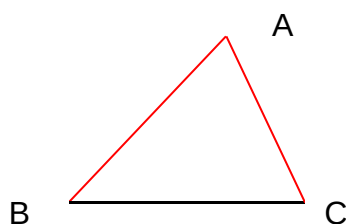
稍稍进一步分析就会发现，本题中单色三角形的个数将是非常多的，所以一切需要枚举出每一个单色三角形的方法都是不可能高效的。

单纯的枚举不可以，那么组合计数是否可行呢？从总体上进行组合计数很难想到，那么我们尝试枚举每一个点，设法找到一个组合公式来计算以这个点为顶点的单色三角形的个数。

这样似乎已经触及到问题的本质了，因为利用组合公式进行计算是非常高效的。但是仔细分析后可以发现，这个组合公式是很难找到的，因为对于枚举确定的点  $A$ ，以  $A$  为一个顶点的单色三角形  $ABC$  不仅要满足边  $AB$  和边  $AC$  同色，而且边  $BC$  也要和  $AB$ 、 $AC$  边同色，于是不可能仅仅通过枚举一个顶点  $A$  就可以确定单色三角形。

经过上面的分析，我们得出枚举 + 组合计数有可能是正确的解法，但是在组合公式的构造上我们遇到了障碍。这个障碍的本质是：

从一个顶点  $A$  出发的两条同色的边  $AB$ 、 $AC$  并不能确定一个单色三角形



图一  
第 45 页 共 205 页

ABC，因为 BC 边有可能不同色。

也就是说，我们无法在从同一个顶点出发的某两条边与所有的单色三角形之间建立一种确定的对应关系。

### 补集转化

让我们换一个角度，从反面来看问题：因为每两点都有边连接，所以每三个点都可以组成一个三角形（单色或非单色的），那么所有的三角形数  $S=C(n,3)=n*(n-1)*(n-2)/6$ 。又因为单色三角形数 R 加上非单色三角形数 T 就等于 S，所以如果我们可以求出 T，那么显然， $R=S-T$ 。于是原问题就等价于：怎样高效地求出 T。

纯枚举的算法想都不用想就被排除，那么在上面分析中夭折的枚举 + 组合计数的算法又怎样呢？这个算法原先的障碍是无法在“某两条边”与“单色三角形”之间建立确定的对应关系。那么有公共顶点的某两条边与非单色三角形之间是否有着确定的关系呢？对了！这种关系是明显的：

非单色三角形的三条边，共有红黑两种颜色，也就是说，只能是两条边同色，另一条边异色。假设同色的两条边顶点为 A，另外两个顶点为 B 和 C，则

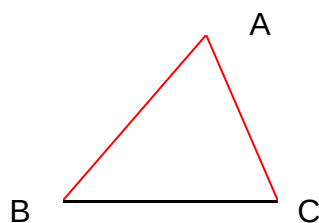


图1

从 B 点一定引出两条不同色的边 BA 和 BC，同样，从 C 点引出两条不同色的边 CA 和 CB。

这样，一个非单色三角形对应着两对“有公共顶点的异色边”。

另一方面，如果从一个顶点 B 引出两条异色的边 BA、BC，则无论 AC 边是何种颜色，三角形 ABC 都只能是一个非单色三角形。也就是说，一对“有公共顶点的异色边”对应着一个非单色三角形。

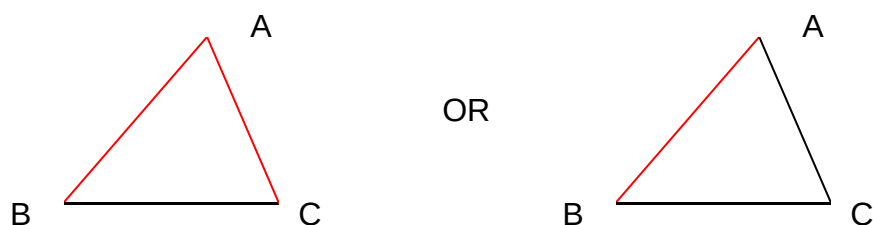


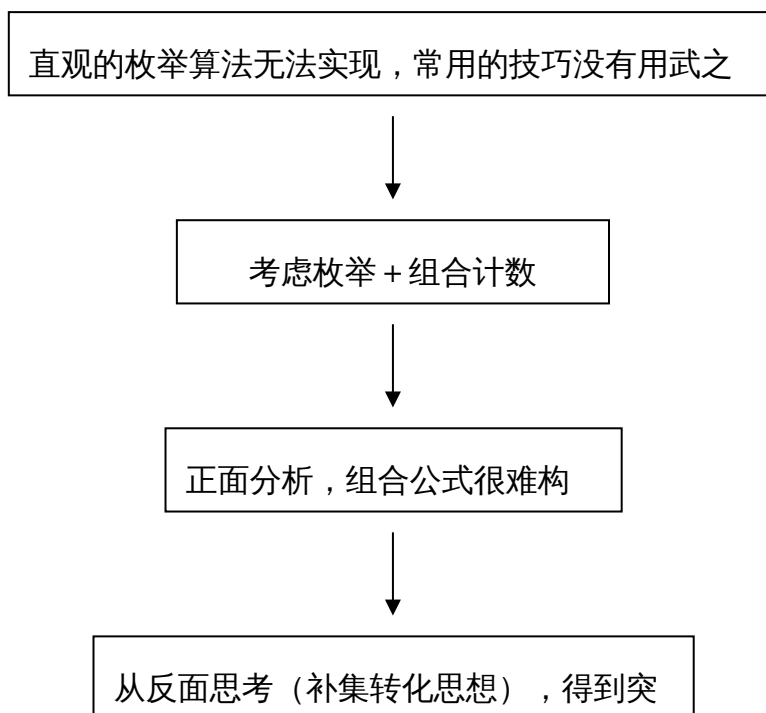
图 1

很明显，我们要求的非单色三角形数  $T$  就等于所有“有公共顶点的异色边”的总对数  $Q$  的一半。而这个总对数是很好求的：每个顶点有  $n-1$  条边，根据输入的信息可以知道每个顶点  $i$  的红边数  $E[i]$ ，那么其黑边数就是  $n-1-E[i]$ 。枚举顶点  $A$ ，则根据乘法原理，以  $A$  为公共顶点的异色边的对数就是  $E[i]*(n-1-$

$E[i])$ 。所以  $Q = \sum_{i=1}^n E[i] * (n-1-E[i])$

求出  $Q$  之后，答案  $R=S-T=n*(n-1)*(n-2)/6-Q/2$ 。这个算法的时间复杂度仅为  $O(m+n)$ ，空间复杂度是  $O(n)$ ，非常优秀。

## 小结



在这个例子中，我们经历了如下的思维过程：

补集转化思想在其中起着至关重要的作用：通过补集转化，我们才能够在原来无法联系起来的“边”和“三角形”之间建立起确定的关系，并以此构造出组合计数的公式。这样，就由单纯的枚举算法改为了枚举+组合计数的算法，大大降低了时间和空间复杂度。在这里，补集转化思想的作用体现在**为找到一个本质上不同的算法创造了条件**。

## 例二 海战游戏（改编自 Ural1212 Sea Battle）

### 题目大意

“海战游戏”是在一个  $N$  行  $M$  列的方格棋盘上摆放“军舰”，一艘军舰是连在一起的  $X$  行  $Y$  列方格，每个方格都全等于棋盘上的格子，于是军舰就可以摆放



在棋盘上，使军舰的每个格子和棋盘的格子重合。但摆放时必须遵守如下规则：任意两艘军舰的任意两个格子不得重合或相邻（指在上、下、左、右、左上、右上、右下、左下八个方向上相邻）。现在已经摆放了  $L$  艘军舰（符合摆放的规则），下一步想要再摆放一个  $P$  行  $Q$  列的军舰，求出共有多少种不同的可能摆放方案。输入  $N$ 、 $M$ 、 $L$ ，已经摆放的  $L$  艘军舰的信息（左上角和右下角的行列坐标），以及下一步要摆放的军舰的大小  $P$ 、 $Q$ ，输出方案数  $R$ 。其中  $2 \leq N, M \leq 30000$ ， $0 \leq L \leq 30$ ， $1 \leq P, Q \leq 5$ 。我们认为所有已摆放的军舰大小都在  $P \times Q$  这样的规模。

例如图五中已经摆放了一个 1 行 2 列的军舰和一个 2 行 1 列的军舰，如果

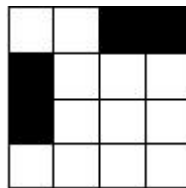


图1

我们要再摆一个 1 行 2 列的军舰，有两种方案。如果要再摆一个 2 行 2 列的军舰，只有一种方案。

## 初步分析

枚举每一种摆放方式，再分别进行判断是否符合规则的方法显然是不行的，因为需要枚举的摆放方式已经在  $O(N \times M)$  的级别，更不要说还需要嵌套一个判断的过程了。

实际上，原题就是给定了一个网格，上面某些矩形区域已经被占用，现在要在里面放入一个新的矩形，不能和已被占用的格子重合或是相邻。这是一种典型的在有障碍点的网格上求摆放方案数的统计问题。对于看起来如此经典的

问题，用常规的方法能否解决呢？实际上，用离散化可以设计出能够接受的算法。由于这不是本文的讨论内容，所以我们不多做研究，只给出初步的结论是：离散化的算法时间复杂度为  $O((M+N)*L)$ ，虽然对于原题勉强可以应付，但是一旦数据规模再稍稍扩大一点，必定超时。而且离散化的算法思考比较复杂，编程比较烦琐。

如果想要圆满地解决这道题，我们需要寻找新的算法。

## 几个工具

在进一步地思考之前，我们先明确几个小问题，以作为下面研究的工具。

- 在一个  $X$  行  $Y$  列的矩形  $A$  中放入一个  $P$  行  $Q$  列的矩形  $B$ ，共有多少种摆放方案？

**结论一：**矩形  $B$  能够放入矩形  $A$  中的充要条件是  $X \geq P$  且  $Y \geq Q$ ，所以如果  $X < P$  或  $Y < Q$ ，方案数为 0。否则矩形  $B$  的左上角可以位于矩形  $A$  的 1 至  $X-P+1$  行，1 至  $Y-Q+1$  列，也就是总共有  $(X-P+1)*(Y-Q+1)$  种摆放方案。

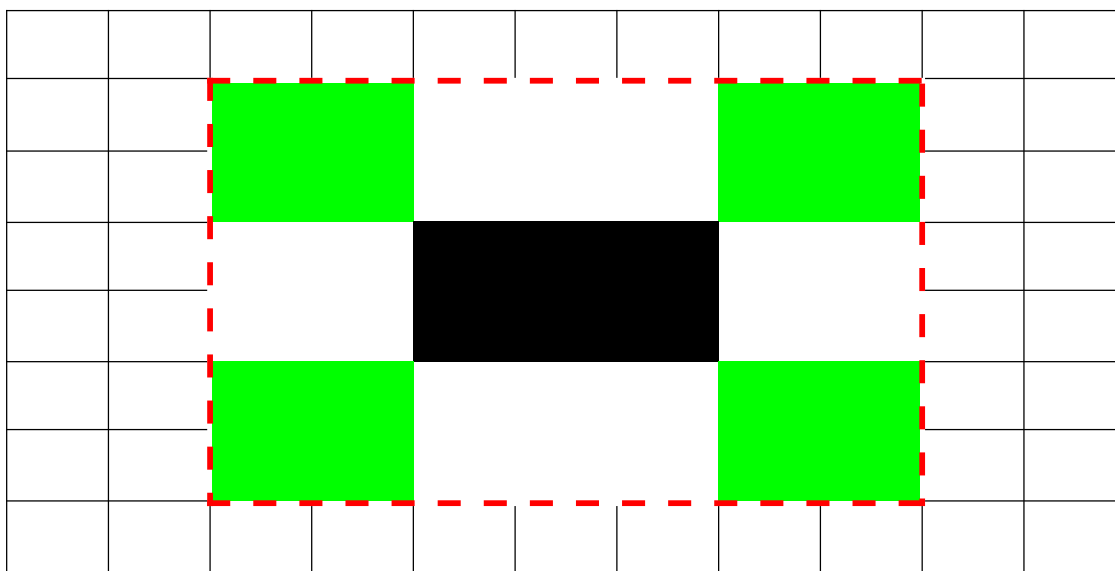
- 矩形  $A$  的左上角为  $(AX1, AY1)$ ，右下角为  $(AX2, AY2)$ ，矩形  $B$  的左上角为  $(BX1, BY1)$ ，右下角为  $(BX2, BY2)$ ，如果存在某两个格子  $a \in A$ ， $b \in B$  且  $a$ 、 $b$  相邻或重合，就称  $A$  和  $B$  “相交”。如何判断  $A$ 、 $B$  是否相交？

这个问题稍稍复杂一点，但是仔细分析各种情况之后可以得出**结论二**： $A$  和  $B$  相接的充要条件是：

$(BX1 \leq AX2+1) \text{ and } (BX2 \geq AX1-1) \text{ and } (BY1 \leq AY2+1) \text{ and } (BY2 \geq AY1-1)$ 。

- 设一个已经摆放的矩形 A 为  $X$  行  $Y$  列，新摆放的矩形 B 为  $P$  行  $Q$  列，矩形 B 怎样摆放才能和矩形 A 相交呢？根据结论二我们直接就可以得出

**结论三：**矩形 B 能够与 A 相交的所有方案位于一个  $2P+X$  行， $2Q+Y$  列的矩形框内。如图六，正中的黑色矩形是矩形 A，四角的绿色矩形代表矩形 B 能够和 A 相交的摆放方式的“边界情况”。当然，这样说还不是太严密，因为这个矩形框有可能超出了棋盘的边界，此时它的边就要调整到棋盘边界内。所以我们把结论三改为：矩形 B 能够与 A 相交的所有方案位于一个最多  $2P+X$  行，最多  $2Q+Y$  列的矩形框内。

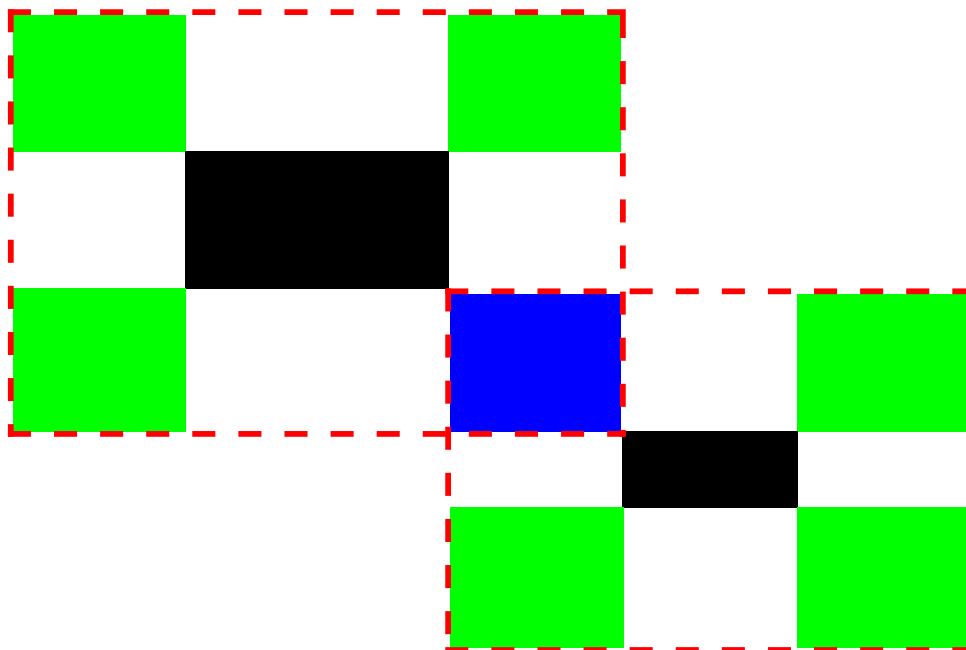


图六

### 补集转化

根据结论一，在给定的棋盘上不加限制摆放一个矩形，其方案数  $S$  是可以根据公式计算出来的。而符合规则的摆放方案数  $R$  + 违反规则的摆放方案数  $T = S$ ，因此  $R = S - T$ 。问题就转化为：如何高效地求出  $T$ 。 $T$  也就是所有与已经摆放的军舰相交的方案数。

在结论三的基础上稍加判断，再结合结论一，我们实际上已经能够直接计算和任意一个已经摆放的矩形相交的方案数。但是我们并不能把和每一个已有矩形相交的方案数累加起来作为  $T$ 。因为有些摆放方案会同时和不止一个矩形相交，例如图七中，蓝色的矩形同时和两个黑色已摆放矩形相交。但是按照上面的算法，这种蓝色的摆放方案会被重复计入  $T$ ，导致  $T$  比实际的要大。



图七

如何排除这种重复计数呢？我们采用一种排除重复的常用方法：有序化。也就是设法对于新矩形的一种摆放方案，只在处理与它相交的编号最小的已有矩形时才允许计入总数  $T$ 。例如图七中，如果我们规定左边的黑色矩形编号较小，则在处理右边的黑色矩形时，与它相交的蓝色摆放方案就不允许计入  $T$ ，因为右边的黑色矩形并不是与蓝色方案相交的编号最小的已摆放矩形。容易证明，这样计数不会出现重复。

为了做到这一点，我们只需采取如下算法：依次处理每个已经摆放的矩形，设当前处理的矩形编号为  $i$ ，一一枚举与它相交的摆放方案，对于每个方

案，再依次枚举编号为  $1, 2, \dots, (i-1)$  的矩形，判断这些矩形能否与当前枚举的方案相交，如果发现有相交的情况，则此方案不能计入  $T$ ，否则就将  $T$  加 1。

根据结论三，与每个已摆放的  $X$  行  $Y$  列的矩形相交的摆放方案位于它周围的一个矩形框内，这个矩形框最多  $2P+X$  行， $2Q+Y$  列，再根据结论一，在其中摆放  $P$  行  $Q$  列的矩形最多只有  $(P+X+1)*(Q+Y+1)$  种方案，由于每个矩形的大小均在  $P*Q$  这样的级别，所以总共需要处理的方案数规模为  $O(P^2Q^2L)$ ，而对于每个方案，最多只需要枚举  $L-1$  个已摆放矩形判断是否与之相交，根据结论二，判断两个矩形是否相交的复杂度为  $O(1)$ ，所以处理每个方案的复杂度为  $O(L)$ ，因此整个算法的复杂度仅为  $O(P^2Q^2L^2)$ ，非常优秀，大大领先于离散化的常规方法。而且，一旦想到了补集转化，则下面的分析、算法设计都非常自然，程序基本上都是循环枚举和简单的判断。可以说，在思维复杂度和编程复杂度上，基于补集转化的算法比离散化的常规算法好得多。

## 小结

在这个例子中，离散化思想能够帮助我们解决原题，但是时间、思维和编程复杂度都很高。但是利用补集转化思想，我们却可以设计出全面超越离散化算法的新算法。

本题从正面考虑，枚举量太大，所以常规的解法是采用离散化技巧来减少枚举量。但是从反面考虑，枚举量就非常小了。补集转化思想在这里起到的作用是帮助我们选择了合适的枚举对象，从而减少了枚举量。

## 总结

本文中的两个例子都是利用补集转化思想解决统计问题，它们在思想上有着明显的相同点：如果目标  $R$  比较难以求解，而它的补集  $T$  以及  $R$  和  $T$  的总和  $S$  相对容易求出，那么不妨从反面考虑，求出  $S$  和  $T$ ，也就间接地求出了  $R$ 。

但是补集转化思想体现在两个具体的例子中，又有所不同：

从作用效果上来看，在例 1 中，补集转化思想指导我们设计出了本质上不同于单纯枚举的算法—枚举 + 组合计数，可以说是“另辟蹊径”；而在例 2 中，它并没有改变算法的本质，而只是通过改变枚举对象减少了枚举量。由此可见，补集转化思想应用于统计问题的形式是多种多样的，可以从解决问题的每个方面帮助我们。

从意义价值上看，在例 1 中，补集转化思想似乎是解决问题的唯一可行方法；而在例 2 中，用离散化也可以解决问题，但补集转化的算法更自然、更优秀，更有普遍性。由此可见，补集转化思想不仅可以应用于一些非常规的统计问题，而且**对于一些常规算法能够解决的问题，应用补集转化思想也许可以做得更好。**

总之，补集转化思想可以比较广泛地应用于统计问题中，是解决统计问题的一种很值得掌握的非常规思想。

补集转化思想，体现了**矛盾对立统一，互相转化**的一种哲学观念。统计问题的本质就是在给定了对象之间的一系列关联、限制（也就是矛盾）的基础上进行计数的问题。所以在统计问题中灵活地应用补集转化思想，往往可以起到“出奇制胜”的效果。我们应该注意培养逆向思维的能力，才能用好、用活补集转化思想。

值得注意的是，利用补集转化思想解决统计问题作为一种非常规的统计方法，它和一些常规的统计方法、技巧之间的关系是辩证的。虽然在本文的例子中，补集转化思想有着很多的闪光之处，但是并不能认为常规方法一定不如非常规方法。大多数的统计问题还是适合用常规方法的，所以

**只有将常规方法和非常规方法都灵活地掌握，并对于具体问题选择合适的方法，才能够游刃有余地解决统计问题。**

# 浅谈用极大化思想解决最大子矩形问题

福州第三中学 王知昆

## 【摘要】

本文针对一类近期经常出现的有关最大（或最优）子矩形及相关变形问题，介绍了极大化思想在这类问题中的应用。分析了两个具有一定通用性的算法。并通过一些例题讲述了这些算法选择和使用时的一些技巧。

**【关键字】** 矩形，障碍点，极大子矩形

## 【正文】

### 一、 问题

最大子矩形问题：在一个给定的矩形网格中有一些障碍点，要找出网格内部不包含任何障碍点，且边界与坐标轴平行的最大子矩形。

这是近期经常出现的问题，例如冬令营 2002 的《奶牛浴场》，就属于最大子矩形问题。

Winter Camp2002,奶牛浴场

题意简述：（原题见论文附件）



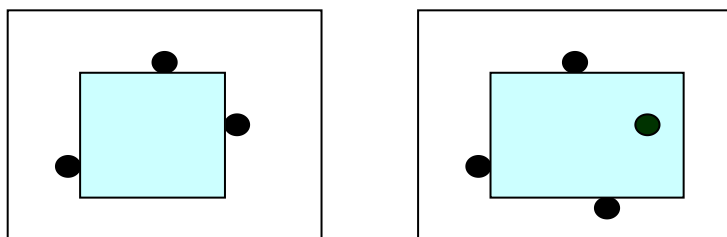
John 要在矩形牛场中建造一个大型浴场，但是这个大型浴场不能包含任何一个奶牛的产奶点，但产奶点可以出在浴场的边界上。John 的牛场和规划的浴场都是矩形，浴场要完全位于牛场之内，并且浴场的轮廓要与牛场的轮廓平行或者重合。要求所求浴场的面积尽可能大。

参数约定：产奶点的个数  $S$  不超过 5000, 牛场的范围  $N \times M$  不超过  $30000 \times 30000$ 。

## 二、 定义和说明

首先明确一些概念。

- 1、 定义**有效子矩形**为内部不包含任何障碍点且边界与坐标轴平行的子矩形。如图所示，第一个是有效子矩形（尽管边界上有障碍点），第二个不是有效子矩形（因为内部含有障碍点）。



- 2、 **极大有效子矩形**：一个有效子矩形，如果不存在包含它且比它大的有效子矩形，就称这个有效子矩形为极大有效子矩形。（为了叙述方便，以下称为**极大子矩形**）
- 3、 定义**最大有效子矩形**为所有有效子矩形中最大的一个（或多个）。以下简称为**最大子矩形**。

### 三、 极大化思想

**【定理 1】** 在一个有障碍点的矩形中的最大子矩形一定是一个极大子矩形。

证明：如果最大子矩形  $A$  不是一个极大子矩形，那么根据极大子矩形的定义，存在一个包含  $A$  且比  $A$  更大的有效子矩形，这与“ $A$  是最大子矩形”矛盾，所以

**【定理 1】** 成立。

### 四、 从问题的特征入手，得到两种常用的算法

定理 1 虽然很显然，但却是很重要的。根据定理 1，我们可以得到这样一个解题思路：通过枚举所有的极大子矩形，就可以找到最大子矩形。下面根据这个思路来设计算法。

**约定：**为了叙述方便，设整个矩形的大小为  $n \times m$ ，其中障碍点个数为  $s$ 。

#### 算法 1

算法的思路是通过枚举所有的极大子矩形找出最大子矩形。根据这个思路可以发现，如果算法中有一次枚举的子矩形不是有效子矩形、或者不是极大子矩形，那么可以肯定这个算法做了“无用功”，这也就是需要优化的地方。怎样保证每次枚举的都是极大子矩形呢，我们先从极大子矩形的特征入手。

**【定理 2】**：一个极大子矩形的四条边一定都不能向外扩展。更进一步地说，一个有效子矩形是极大子矩形的充要条件是这个子矩形的每条边要么覆盖了一个障碍点，要么与整个矩形的边界重合。

定理 2 的正确性很显然，如果一个有效子矩形的某一条边既没有覆盖一个障碍点，又没有与整个矩形的边界重合，那么肯定存在一个包含它的有效子矩形。根据定理 2，我们可以得到一个枚举极大子矩形的算法。为了处理方便，首先在障碍点的集合中加上整个矩形四角上的点。每次枚举子矩形的上下左右边界（枚举覆盖的障碍点），然后判断是否合法（内部是否有包含障碍点）。这样的算法时间复杂度为  $O(S^5)$ ，显然太高了。考虑到极大子矩形不能包含障碍点，因此这样枚举 4 个边界显然会产生大量的无效子矩形。

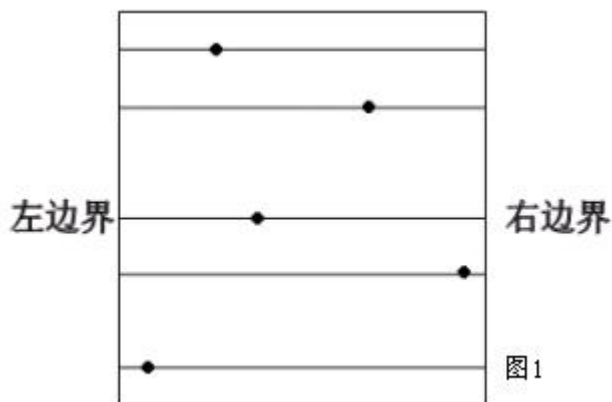


图1

考虑只枚举左右边界的情况。对于已经确定的左右边界，可以将所有处在这个边界内的点按从上到下排序，如图1 中所示，每一格就代表一个有效子矩形。这样做时间复杂度为  $O(S^3)$ 。

由于确保每次得到的矩形都是合法的，所以枚举量比前一种算法小了很多。但需要注意的是，这样做枚举的子矩形虽然是合法的，然而不一定是极大的。所以这个算法还有优化的余地。通过对这个算法不足之处的优化，我们可以得到一个高效的算法。

回顾上面的算法，我们不难发

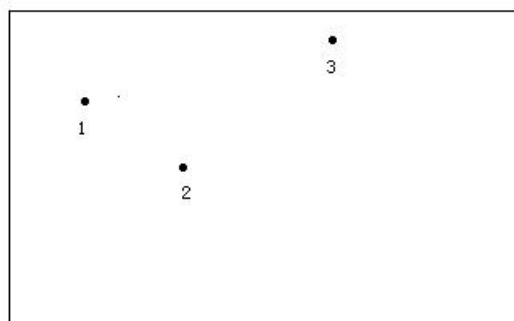


图2

现，所枚举的矩形的上下边界都覆盖了障碍点或者与整个矩形的边界重合，问题就在于左右边界上。只有那些左右边界也覆盖了障碍点或者与整个矩形的边界重合的有效子矩形才是我们需要考察的极大子矩形，所以前面的算法做了不少“无用功”。怎么减少“无用功”呢，这里介绍一种算法（算法 1），它可以用在不少此类题目上。

算法的思路是这样的，先枚举极大子矩形的左边界，然后从左到右依次扫描每一个障碍点，并不断修改可行的上下边界，从而枚举出所有以这个定点为左边界的极大子矩形。考虑如图 2 中的三个点，现在我们要确定所有以 1 号点为左边界的极大矩形。先将 1 号点右边的点按横坐标排序。然后按从左到右的顺序依次扫描 1 号点右边的点，同时记录下当前的可行的上下边界。

开始时令当前的上下边界分别为整个矩形的上下边界。然后开始扫描。第一次遇到 2 号点，以 2 号点作为右边界，结合当前的上下边界，就得到一个极大子矩形（如图 3）。同时，由于所求矩形不能包含 2 号点，

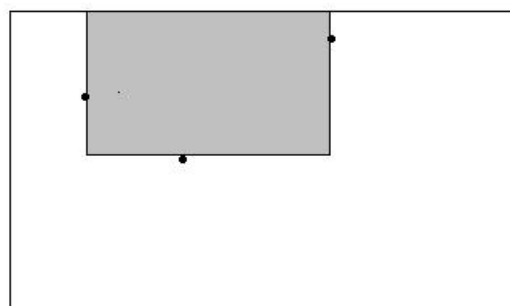


图4



图3

且 2 号点在 1 号点的下方，所以需要修改当前的下边界，即以 2 号点的纵坐标作为新的下边界。第二次遇到 3 号点，这时以 3 号点的横坐标作为右边界又可以得到一个满足性质 1 的矩形（如图 4）。类似的，需要相应地修改上边界。以此类推，如果这个点是在当前点（确定左边界的点）上方，则修改上边界；如果在下方，则修改下边界；如果处在同一行，则可中止搜索（因

为后面的矩形面积都是 0 了)。由于已经在障碍点集合中增加了整个矩形右上角和右下角的两个点, 所以不会遗漏右边界与整个矩形的右边重合的极大子矩形(如图 5)。需要注意的是, 如果扫描到的点不在当前的上下边界内, 那么就不需要对这个点进行处理。

这样做是否将所有的极大子矩形都枚举过了呢? 可以发现, 这样做只考虑到了左边界覆盖一个点的矩形, 因此我们还需要枚举左边界与整个矩形的左边界重合的情况。这还可以分

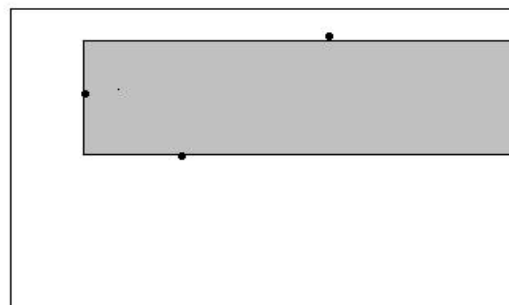


图5

为两类情况。一种是左边界与整个矩形的左边界重合, 而右边界覆盖了一个障碍点的情况, 对于这种情况, 可以用类似的方法从右到左扫描每一个点作为右边界的情况。另一种是左右边界均与整个矩形的左右边界重合的情况, 对于这类情况我们可以在预处理中完成: 先将所有点按纵坐标排序, 然后可以得到以相邻两个点的纵坐标为上下边界, 左右边界与整个矩形的左右边界重合的矩形, 显然这样的矩形也是极大子矩形, 因此也需要被枚举到。

通过前面两步, 可以枚举出所有的极大子矩形。算法 1 的时间复杂度是  $O(S^2)$ 。这样, 可以解决大多数最大子矩形和相关问题了。

虽然以上的算法(算法 1)看起来是比较高效的, 但也有使用的局限性。可以发现, 这个算法的复杂度只与障碍点的个数  $s$  有关。但对于某些问题,  $s$  最大有可能达到  $n \times m$ , 当  $s$  较大时, 这个算法就未必能满足时间上的

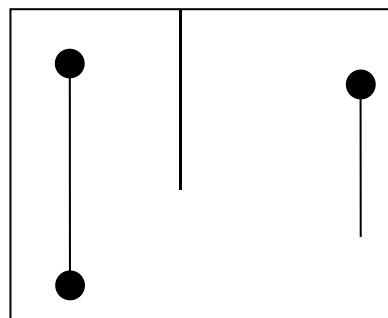
要求了。能否设计出一种依赖于  $n$  和  $m$  的算法呢？这样在算法 1 不能奏效的时候我们还有别的选择。我们再重新从最基本的问题开始研究。

## 算法 2

首先，根据定理 1：最大有效子矩形一定是一个极大子矩形。不过与前一种算法不同的是，我们不再要求每一次枚举的一定是极大子矩形而只要求所有的极大子矩形都被枚举到。看起来这种算法可能比前一种差，其实不然，因为前一种算法并不是完美的：虽然每次考察的都是极大子矩形，但它还是做了一定量的“无用功”。可以发现，当障碍点很密集的时候，前一种算法会做大量没用的比较工作。要解决这个问题，我们必须跳出前面的思路，重新考虑一个新的算法。注意到极大子矩形的个数不会超过矩形内单位方格的个数，因此我们有可能找出一种时间复杂度是  $O(N \times M)$  的算法。

定义：

**有效竖线**：除了两个端点外，不覆盖任何障碍点的竖直线段。



**悬线**：上端点覆盖了一个障碍点或达到整个矩形上端的有效竖线。如图所示的三个有效竖线都是悬线。

对于任何一个极大子矩形，它的上边界上要么有一个障碍点，要么和整个矩形的上边界重合。那么如果把一个极大子矩形按  $x$  坐标不同切割成多个（实

际上是无数个)与  $y$  轴垂直的线段,则其中一定存在一条悬线。而且一条悬线通过尽可能地向左右移动恰好能得到一个子矩形(未必是极大子矩形,但只可能向下扩展)。通过以上的分析,我们可以得到一个重要的定理。

**【定理 3】**: 如果将一个悬线向左右两个方向尽可能移动所得到的有效子矩形称为这个悬线所对应的子矩形,那么所有悬线所对应的有效子矩形的集合一定包含了所有极大子矩形的集合。

定理 3 中的“尽可能”移动指的是移动到一个障碍点或者矩形边界的位置。

根据【定理 3】可以发现,通过枚举所有的悬线,就可以枚举出所有的极大子矩形。由于每个悬线都与它底部的那个点一一对应,所以悬线的个数 =  $(n-1) \times m$  (以矩形中除了顶部的点以外的每个点为底部,都可以得到一个悬线,且没有遗漏)。如果能做到对每个悬线的操作时间都为  $O(1)$ ,那么整个算法的复杂度就是  $O(NM)$ 。这样,我们看到了解决问题的希望。

现在的问题是,怎样在  $O(1)$  的时间内完成对每个悬线的操作。我们知道,每个极大子矩形都可以通过一个悬线左右平移得到。所以,对于每个确定了底部的悬线,我们需要知道有关于它的三个量:顶部、左右最多能移动到的位置。对于底部为  $(i,j)$  的悬线,设它的高为  $height[i,j]$ ,左右最多能移动到的位置为  $left[i,j], right[i,j]$ 。为了充分利用以前得到的信息,我们将这三个函数用递推的形式给出。

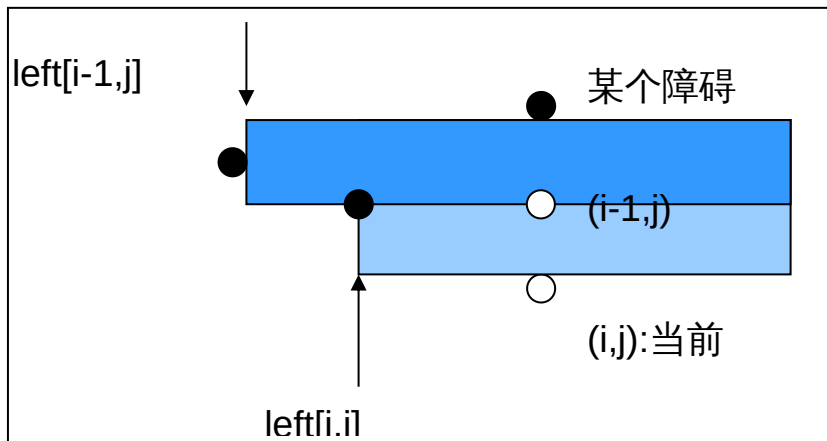
对于以点  $(i,j)$  为底部的悬线:

如果点 $(i-1, j)$ 为障碍点，那么，显然以 $(i, j)$ 为底的悬线高度为 1，而且左右均可以移动到整个矩形的左右边界，即

$$\begin{cases} \text{height}[i, j] = 1 \\ \text{left}[i, j] = 0 \\ \text{right}[i, j] = m \end{cases}$$

如果点 $(i-1, j)$ 不是障碍点，那么，以 $(i, j)$ 为底的悬线就等于以 $(i-1, j)$ 为底的悬线 + 点 $(i, j)$ 到点 $(i-1, j)$ 的线段。因此， $\text{height}[i, j] = \text{height}[i-1, j] + 1$ 。比较麻烦的是左右边界，先考虑  $\text{left}[i, j]$ 。如下图所示， $(i, j)$ 对应的悬线左右能移动的位置要在 $(i-1, j)$ 的基础上变化。

$$\text{即 } \text{left}[i, j] = \max \begin{cases} \text{left}[i-1, j] \\ (i-1, j) \text{ 左边第一个障碍点的位置} \end{cases}$$



$\text{right}[i, j]$ 的求法类似。综合起来，可以得到这三个参数的递推式：



$$\begin{cases} height[i, j] = height[i-1, j] + 1 \\ left[i, j] = \max \begin{cases} left[i-1, j] \\ (i-1, j) \text{ 左边第一个障碍点位置 (边界0也算障碍点)} \end{cases} \\ right[i, j] = \min \begin{cases} right[i-1, j] \\ (i-1, j) \text{ 右边第一个障碍点位置 (边界} m \text{也算障碍点)} \end{cases} \end{cases}$$

这样做充分利用了以前得到的信息，使每个悬线的处理时间复杂度为  $O(1)$ 。对于以点  $(i, j)$  为底的悬线对应的子矩形，它的面积为  $(right[i, j] - left[i, j]) * height[i, j]$ 。

这样最后问题的解就是：

Result =

$$\max \{ (right[i, j] - left[i, j]) * height[i, j] \quad (1 \leq i < n, 1 \leq j \leq m) \}$$

整个算法的时间复杂度为  $O(NM)$ ，空间复杂度是  $O(NM)$ 。

两个算法的对比：

以上说了两种具有一定通用性的处理算法，时间复杂度分别为  $O(S^2)$  和  $O(NM)$ 。两种算法分别适用于不同的情况。从时间复杂度上来看，第一种算法对于障碍点稀疏的情况比较有效，第二种算法则与障碍点个数的多少没有直接的关系（当然，障碍点较少时可以通过对障碍点坐标的离散化来减小处理矩形的面积，不过这样比较麻烦，不如第一种算法好），适用于障碍点密集的情况。

## 五、 例题

将前面提出的两种算法运用于具体的问题。

## 1、 Winter Camp2002,奶牛浴场

分析：

题目的数学模型就是给出一个矩形和矩形中的一些障碍点，要求出矩形内的最大有效子矩形。这正是我们前面所讨论的**最大子矩形问题**，因此前两种算法都适用于这个问题。

下面分析两种算法运用在本题上的优劣：

对于第一种算法，不用加任何的修改就可以直接应用在这道题上，时间复杂度为  $O(S^2)$ ， $S$  为障碍点个数；空间复杂度为  $O(S)$ 。

对于第二种算法，需要先做一定的预处理。由于第二种算法复杂度与牛场的面积有关，而题目中牛场的面积很大（ $30000 \times 30000$ ），因此需要对数据进行离散化处理。离散化后矩形的大小降为  $S \times S$ ，所以时间复杂度为  $O(S^2)$ ，空间复杂度为  $O(S)$ 。说明：需要注意的是，为了保证算法能正确执行，在离散化的时候需要加上  $S$  个点，因此实际需要的时间和空间较大，而且编程较复杂。

从以上的分析来看，无论从时空效率还是编程复杂度的角度来看，这道题采用第一种算法都更优秀。

## 2、 OIBH 模拟赛 1,提高组，Candy

题意简述：（原题见论文附件）

一个被分为  $n*m$  个格子的糖果盒，第  $i$  行第  $j$  列位置的格子里面有  $a[i,j]$  颗糖。但糖果盒的一些格子被老鼠洗劫。现在需要尽快从这个糖果盒里面切割出一个矩形糖果盒，新的糖果盒不能有洞，并且希望保留在新糖果盒内的糖的总数尽量多。

参数约定： $1 \leq n, m \leq 1000$

## 分析

首先需要注意的是：本题的模型是一个矩阵，而不是矩形。在矩阵的情况下，由于点的个数是有限的，所以又产生了一个新的问题：**最大权值子矩阵**。

定义：

**有效子矩阵**为内部不包含任何障碍点的子矩形。与有效子矩形不同，有效子矩阵地边界上也不能包含障碍点。

**有效子矩阵的权值**（只有有效子矩形才有权值）为这个子矩阵包含的所有点的权值和。

**最大权值有效子矩阵**为所有有效子矩阵中权值最大的一个。以下简称为**最大权值子矩阵**。

本题的数学模型就是正权值条件下的最大权值子矩阵问题。再一次利用极大化思想，因为矩阵中的权值都是正的，所以最大权值子矩阵一定是一个极大子矩阵。所以我们只需要枚举所有的极大子矩阵，就能从中找到

最大权值子矩阵。同样，两种算法只需稍加修改就可以解决本题。下面分析两种算法应用在本题上的优劣：

对于第一种算法，由于矩形中障碍点的个数是不确定的，而且最大有可能达到  $N \times M$ ，这样时间复杂度有可能达到  $O(N^2M^2)$ ，空间复杂度为  $O(NM)$ 。此外，由于矩形与矩阵的不同，所以在处理上会有一些小麻烦。

对于第二种算法，稍加变换就可以直接使用，时间复杂度为  $O(NM)$ ，空间复杂度为  $O(NM)$ 。

可以看出，第一种算法并不适合这道题，因此最好还是采用第二种算法。

### 3、 Usaco Training, Section 1.5.4, Big Barn

**题意简述**（原题见论文附件）

Farmer John 想在他的正方形农场上建一个正方形谷仓。由于农场上有一些树，而且 Farmer John 又不想砍这些树，因此要找出最大的一个不包含任何树的一块正方形场地。每棵树都可以看成一个点。

参数约定：牛场为  $N \times N$  的，树的棵数为  $T$ 。  $N \leq 1000, T \leq 10000$ 。

**分析：**

这题是矩形上的问题，但要求的是最大子正方形。首先，明确一些概念。

- 1、 定义**有效子正方形**为内部不包含任何障碍点的子正方形

- 2、 定义**极大有效子正方形**为不能再向外扩展的有效子正方形，一下简称**极大子正方形**
- 3、 定义**最大有效子正方形**为所有有效子正方形中最大的一个（或多个），以下简称**最大子正方形**。

本题的模型有一些特殊，要在一个含有一些障碍点的矩形中求**最大子正方形**。这与前两题的模型是否有相似之处呢？还是从最大子正方形的本质开始分析。

与前面的情况类似，利用极大化思想，我们可以得到一个定理：

**【定理 4】**：在一个有障碍点的矩形中的最大有效子正方形一定是一个极大有效子正方形。

根据**【定理 4】**，我们只需要枚举出所有的极大子正方形，就可以从中找出最大子正方形。极大子正方形有什么特征呢？所谓**极大**，就是不能再向外扩展。如果是极大子矩形，那么不能再向外扩展的充要条件是四条边上都覆盖了障碍点（**【定理 2】**）。类似的，我们可以知道，一个有效子正方形是极大子正方形的充要条件是它任何两条相邻的边上都覆盖了至少一个障碍点。根据这一点，可以得到一个重要的定理。

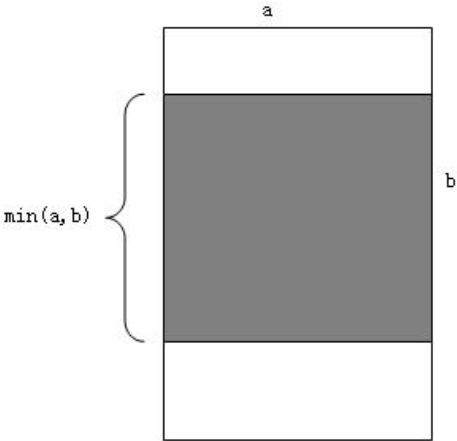
**【定理 5】**：每一个极大子正方形都至少被一个极大子矩形包含。且这个极大子正方形一定有两条不相邻的边与这个包含它的极大子矩形的边重合。

根据【定理 5】,我们只需要枚举所有的极大子矩形，并检查它所包含的极大子正方形（一个极大子矩形包含的极大子正方形都是一样大的）是否是最大的就可以了。这样，问题的实质和前面所说的最大子矩形问题是一样的，同样的，所采用的算法也是一样的。

因为算法 1 和算法 2 都枚举出了所有的极大子矩形，因此，算法 1 和算法 2 都可以用在本题上。具体的处理方法如下：对于每一个枚举出的极大子矩形，如图所示，如果它的边长为  $a$ 、 $b$ ，那么它包含的极大子正方形的边长即为  $\min(a,b)$ 。

考虑到  $N$  和  $T$  的大小不同，所以不同的算法会有不同的效果。下面分析两种算法应用在本题上的优劣。

对于第一种算法，时间复杂度为  $O(T^2)$ ，对于第二种算法，时间复杂度为  $O(N^2)$ 。因为  $N < T$ ，所以从时间复杂度的角度看，第二种算法要比第一种算法好。考虑到两个算法的空间复杂度都可以承受，所以选择第二种算法较好些。



以下是第一种和第二种算法编程实现后在 [USACO Training Program Gateway](#) 上的运行时间。可以看出，在数据较大时，算法 2 的效率比算法 1 高。

算法 1 :	算法 2 :
Test 1: 0.009375	Test 1: 0.009375
Test 2: 0.009375	Test 2: 0.009375

Test 3: 0.009375	Test 3: 0.009375
Test 4: 0.009375	Test 4: 0.009375
Test 5: 0.009375	Test 5: 0.009375
Test 6: 0.009375	Test 6: 0.00625
Test 7: 0.021875	Test 7: 0.009375
Test 8: 0.025	Test 8: 0.009375
Test 9: 0.084375	Test 9: 0.0125
Test 10: 0.3875	Test 10: 0.021875
Test 11: 0.525	Test 11: 0.028125
Test 12: 0.5625	Test 12: 0.03125
Test 13: 0.690625	Test 13: 0.03125
Test 14: 0.71875	Test 14: 0.03125
Test 15: 0.75	Test 15: 0.034375

以上，利用极大化思想和前面设计的两个算法，通过转换模型，解决了三个具有一定代表性的例题。解题的关键就是如何利用极大化思想进行模型转换和如何选择算法。

## 五、小结

设计算法要从问题的基本特征入手，找出解题的突破口。本文介绍了两种适用于大部分最大子矩形问题及相关变型问题的算法，它们设计的突破口就是利用了极大化思想，找到了枚举极大子矩形这种方法。

在效率上，两种算法对于不同的情况各有千秋。一个是针对障碍点来设计的，因此复杂度与障碍点有关；另一个是针对整个矩形来设计的，因此复杂度与矩形的面积有关。虽然两个算法看起来有着巨大的差别，但他们的本

质是相通的，都是利用极大化思想，从枚举所有的极大有效子矩形入手，找出解决问题的方法。

需要注意的是，在解决实际问题时仅靠套用一些现有算法是不够的，还需要对问题进行全面、透彻的分析，找出解题的突破口。

此外，如果采用极大化思想，前面提到的两种算法的复杂度已经不能再降低了，因为极大有效子矩形的个数就是  $O(NM)$  或  $O(S^2)$  的。如果采用其他算法，理论上是有可能会进一步提高算法效率，降低复杂度的。

## 七、附录：

1、几个例题的原题。 [见论文附件.doc](#)

2、例题的程序。 [见论文附件.doc](#)

说明：所有程序均在 Free Pascal IDE for Dos, Version 0.9.2 上编译运行

## 参考书目

1、 信息学奥林匹克 竞赛指导

----1997 ~ 1998 竞赛试题解析

吴文虎 王建德 著

2、 IOI99 中国集训队优秀论文集

3、 信息学奥林匹克（季刊）



4、 《金牌之路 竞赛辅导》

江文哉主编 陕西师范大学出版社出版

## 一、一些例题的原题

### 1、奶牛浴场

#### 奶牛浴场

##### 【问题描述】

由于 John 建造了牛场围栏，激起了奶牛的愤怒，奶牛的产奶量急剧减少。为了讨好奶牛，John 决定在牛场中建造一个大型浴场。但是 John 的奶牛有一个奇怪的习惯，每头奶牛都必须在牛场中的一个固定的位置产奶，而奶牛显然不能在浴场中产奶，于是，John 希望所建造的浴场不覆盖这些产奶点。这回，他又要求助于 Clevow 了。你还能帮助 Clevow 吗？

John 的牛场和规划的浴场都是矩形。浴场要完全位于牛场之内，并且浴场的轮廓要与牛场的轮廓平行或者重合。浴场不能覆盖任何产奶点，但是产奶点可以位于浴场的轮廓上。

Clevow 当然希望浴场的面积尽可能大了，所以你的任务就是帮她计算浴场的最大面积。

##### 【输入文件】

输入文件的第一行包含两个整数  $L$  和  $W$ ，分别表示牛场的长和宽。文件的第二行包含一个整数  $n$ ，表示产奶点的数量。以下  $n$  行每行包含两个整数  $x$  和  $y$ ，表示一个产奶点的坐标。所有产奶点都位于牛场内，即： $0 < x < L$ ， $0 < y < W$ 。

##### 【输出文件】

输出文件仅一行，包含一个整数  $S$ ，表示浴场的最大面积。

## 【输入输出样例】

happy.in	happy.out
10 10 4 1 1 9 1 1 9 9 9	80

## 【参数约定】

 $0 < n < 5000$  $1 < L, W < 30000$ 

## 2、Candy

### 糖果盒 ( Candy Box )

#### 问题描述：

一个被分为  $n*m$  个格子的糖果盒，第  $i$  行第  $j$  列位置的格子里面有  $a[i][j]$  颗糖。本来 tenshi 打算送这盒糖果给某 PPMM 的，但是就在要送出糖果盒的前一天晚上，一只极其可恶的老鼠夜袭糖果盒，有部分格子被洗劫并且穿了洞。tenshi 必须尽快从这个糖果盒里面切割出一个矩形糖果盒，新的糖果盒不能有洞，并且 tenshi 希望保留在新糖果盒内的糖的总数尽量多。

#### 任 务：

请帮 tenshi 设计一个程序 计算一下新糖果盒最多能够保留多少糖果。

输入格式：

从文件 CANDY.INP 读入数据。第一行有两个整数 n、m。第 i + 1 行的第 j 个数表示 a [ i ][ j ]，如果这个数为 0，则表示这个位置的格子被洗劫过。其中：

$$1 \leq n, m \leq 1000$$

$$0 \leq a [ i ][ j ] \leq 255$$

注意：本题提供 16 MB 内存，时间限制为 2 秒。

输出格式：

输出最大糖果数到 CANDY.OUT。

样例

CANDY.INP	CANDY.OUT
3 4  1 2 3 4  5 0 6 3  10 3 4 0	17

注：

10 3 4

这个矩形的糖果数最大

### 3、Big Barn

# Big Barn

A Special Treat

Farmer John wants to place a big square barn on his square farm. He hates to cut down trees on his farm and wants to find a location for his barn that enables him to build it only on land that is already clear of trees. For our purposes, his land is divided into  $N \times N$  parcels. The input contains a list of parcels that contain trees. Your job is to determine and report the largest possible square barn that can be placed on his land without having to clear away trees. The barn sides must be parallel to the horizontal or vertical axis.

## EXAMPLE

Consider the following grid of Farmer John's land where '.' represents a parcel with no trees and '#' represents a parcel with trees:

1 2 3 4 5 6 7 8

1 . . . . . . . .

2 . # . . . # . .

3 . . . . . . .

4 . . . . . . .

5 . . . . . . .

6 . . # . . . . .

7 . . . . . . .

8 . . . . . . .

The largest barn is 5 x 5 and can be placed in either of two locations in the lower right part of the grid.

**PROGRAM NAME: bigbrn**

## INPUT FORMAT

Line 1: Two integers: N ( $1 \leq N \leq 1000$ ), the number of parcels on a side, and T ( $1 \leq T \leq 10,000$ ) the number of parcels with trees

Lines 2..T+1: Two integers ( $1 \leq \text{each integer} \leq N$ ), the row and column of a tree parcel

## SAMPLE INPUT (file bigbrn.in)

8 3

2 2

2 6

6 3

## OUTPUT FORMAT

The output file should consist of exactly one line, the maximum side length of John's barn.

## SAMPLE OUTPUT (file bigbrn.out)

5

## 二、一些例题的程序

### 1、 奶牛浴场用第一种算法解的程序

```
program happy;
var
  f:text;
  x,y:array[1..5002] of longint;
  maxl,n,best,a,b,c,w,l,i,j,high,low:longint;

procedure sort(l,r:longint);
var
  i,j:longint;
begin
  i:=l+random(r-l+1);
  a:=x[i]; b:=y[i]; i:=l; j:=r;
  repeat
    while (x[i]<a) or ((x[i]=a) and (y[i]<b)) do i:=i+1;
    while (x[j]>a) or ((x[j]=a) and (y[j]>b)) do j:=j-1;
    if i<=j then
      begin
        c:=x[i]; x[i]:=x[j]; x[j]:=c;
        c:=y[i]; y[i]:=y[j]; y[j]:=c;
        inc(i); dec(j);
      end;
  until i>j;
```

```
    if j>l then sort(l,j);
    if i<r then sort(i,r);
end;

procedure sort_y(l,r:longint);
var
    i,j:longint;
begin
    a:=y[(l+r) div 2]; i:=l; j:=r;
    repeat
        while (y[i]<a) do i:=i+1;
        while (y[j]>a) do j:=j-1;
        if i<=j then
            begin
                c:=y[i]; y[i]:=y[j]; y[j]:=c;
                inc(i); dec(j);
            end;
        until i>j;
        if j>l then sort_y(l,j);
        if i<r then sort_y(i,r);
    end;

procedure max(a:longint);
begin
    if a>best then best:=a;
end;

begin
    assign(f,'happy.in');
    reset(f);
    readln(f,l,w);
    readln(f,n);
    for i:=1 to n do
        readln(f,x[i],y[i]);
    close(f);
    inc(n); x[n]:=l; y[n]:=0;
    inc(n); x[n]:=0; y[n]:=w;
    sort(1,n);
    best:=0;
    for i:=1 to n do
        begin
            high:=w; low:=0; maxl:=l-x[i];
            for j:=i+1 to n do
                if (y[j]<=high) and (y[j]>=low) then
```



```
begin
  if maxl*(high-low)<=best then break;
  max((x[j]-x[i])*(high-low));
  if y[j]=y[i] then break
  else if y[j]>y[i] then
    if y[j]<high then high:=y[j]
    else
      else if y[j]>low then low:=y[j];
  end;
  high:=w; low:=0; maxl:=l-x[i];
  for j:=i-1 downto 1 do
    if (y[j]<=high) and (y[j]>=low) then
      begin
        if maxl*(high-low)<=best then break;
        max((x[i]-x[j])*(high-low));
        if y[j]=y[i] then break
        else if y[j]>y[i] then
          if y[j]<high then high:=y[j]
          else
            else if y[j]>low then low:=y[j];
        end;
      end;
  end;
  sort_y(1,n);
  for i:=1 to n-1 do
    max((y[i+1]-y[i])*l);
  writeln(best);
end.
```

## 2、 Candy 的程序

```
program candy;
const
  maxn=1000;
var
  left,right,high:array[1..maxn] of longint;
  s:array[0..maxn,0..maxn] of longint;
  now,res,leftmost,rightmost,i,j,k,n,m:longint;
  f:text;
begin
  assign(f,'candy.in');
  reset(f);
  readln(f,n,m);
  fillchar(s,sizeof(s),0);
  for i:=1 to m do
```

```
begin
  left[i]:=1; right[i]:=m; high[i]:=0;
end;
res:=0;
for i:=1 to n do
begin
  k:=0; leftmost:=1;
  for j:=1 to m do
begin
  read(f,now); k:=k+now;
  s[i,j]:=s[i-1,j]+k;
  if now=0 then
begin
  high[j]:=0; left[j]:=1; right[j]:=m;
  leftmost:=j+1;
end
else
begin
  high[j]:=high[j]+1;
  if leftmost>left[j] then left[j]:=leftmost;
end;
end;
rightmost:=m;
for j:=m downto 1 do
begin
  if high[j]=0 then
begin
  rightmost:=j-1;
end
else
begin
  if right[j]>rightmost then right[j]:=rightmost;
  now:=s[i,right[j]]+s[i-high[j],left[j]-1]-s[i-high[j],right[j]]-s[i,left[j]-1];
  if now>res then res:=now;
end;
end;
end;
writeln(res);
end.
```

### 3、 Big Barn 用第二种算法解的程序

```
program BigBarn;
```

```
var
  d:array[1..1000,1..1000] of longint;
  height,left,right:array[1..1000] of longint;
  leftmost,rightmost,res,i,j,k,t,n:longint;
  f:text;
begin
  assign(f,'bigbrn.in');
  reset(f);
  readln(f,n,t);
  fillchar(d,sizeof(d),0);
  for i:=1 to t do
    begin
      readln(f,j,k);
      d[j,k]:=1;
    end;
  close(f);
  for i:=1 to n do
    begin
      height[i]:=0; left[i]:=1; right[i]:=n;
    end;
  res:=0;
  for i:=1 to n do
    begin
      leftmost:=1;
      for j:=1 to n do
        if d[i,j]=1 then
          begin
            height[j]:=0; left[j]:=1; right[j]:=n;
            leftmost:=j+1;
          end
        else
          begin
            height[j]:=height[j]+1;
            if leftmost>left[j] then left[j]:=leftmost;
          end;
      rightmost:=n;
      for j:=n downto 1 do
        if d[i,j]=1 then rightmost:=j-1
      else
        begin
          if rightmost<right[j] then right[j]:=rightmost;
          k:=height[j];
          if right[j]-left[j]+1<k then k:=right[j]-left[j]+1;
          if k>res then res:=k;
```

```
        end;  
    end;  
    assign(f,'bigbrn.out');  
    rewrite(f);  
    writeln(f,res);  
    close(f);  
end.
```

# 数学思想助你一臂之力

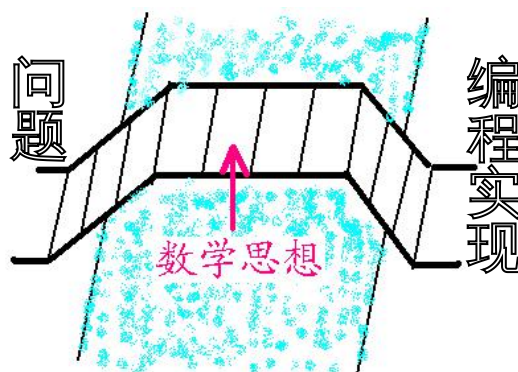
复旦大学附属中学

邵烜程

## [关键字]

数学思想、构造算法

## [摘要]



数学和计算机原本就是密不可分  
的学科。有许多计算机编程问题如果不利用数学思想则很难甚至无法达到预期的效果。

如果把问题和编程实现看成是河的两岸，那么数学思想就是连接河两岸的一座桥梁，有了这座桥，从河的一岸到另一岸便不再是件难事了。

有些问题，利用这座桥可以更方便地往返于河两岸，而还有一些问题，如果不利用这座桥，可能根本无法到达河对岸。也就是说，有些问题利用数学思想可以走捷径（例如NOI2002的“荒岛野人”），而还有一些问题，如果不利用数学思想，就根本无法解决（例如NOI2002的“机器人M号”）。我们将从四个方面探讨利用数学思想提高算法效率，简化问题的例子：

## 一．利用数学思想直接找出解的一般规律。

有些问题，如果直接用动态规划或是图论的方法来解决效率可能会并不理想。这时，我们首先应该想到的是优化，而如果优化无法达到预期的效果，那

我们只有重新寻找算法了。于是，我们就试图找出问题的一般规律、或是该问题所用到的一个小问题的一般规律，这样，时间效率将会大大提高。

我们首先来看一个直接找出原问题的一般规律的例子——

## 例题一 最优分解方案

把正整数  $N$  分解成若干个互不相等的自然数的和，且使这些自然数的乘积最大。

### [输入]

只有一行，包括数  $N$  ( $3 \leq N \leq 1000$ )。

### [输出]

第一行输出最优分解方案，相邻两数之间用单个空隔隔开；第二行输出最大的乘积。

## 算法分析

设把  $N$  分解成  $m$  个互不相等的自然数  $a_1, a_2, \dots, a_m$ ，且满足：  
 $a_1 < a_2 < \dots < a_m$ 。

直觉告诉我们：要使乘积最大，就要使  $a_1, a_2, \dots, a_m$  尽量接近，并且  $m$  尽可能大。不要怀疑自己的感觉，它往往给我们带来了正确的思路，有时甚至是比其它方法简便得多的。而本题正是这样。

让我们先把这个“感觉”表述成数学语言：

先令  $a_i = i + 1 (i = 1, 2, \dots, k)$ ，这里  $k$  满足：

$$\sum_{j=1}^k a_j \leq N < \sum_{j=1}^k a_j + (k+2)$$

设  $rest = N - \sum_{j=1}^k a_j$ , 由于  $rest < k+2$ , 故若  $rest < k+1$ , 则令

$$a_i = a_i + 1 (k - rest + 1 \leq i \leq k)$$

若  $rest = k+1$ , 则令  $a_i = a_i + 1 (1 \leq i \leq k-1)$ ,  $a_k = a_k + 2$ , 这样得到的  $k$  个数  $a_i$  ( $1 \leq i \leq k$ ) 的乘积有最大值。

通俗一点讲, 就是将  $a_1, a_2; \dots, a_m$  首先依次置为 2, 3, 4, 等等。当已经确定的数之和要达到  $N$  时, 将还剩下平均分在前面一些数上。

下面我们来简单证明这个结论: (设  $a_1, a_2; \dots, a_m$  是最优分解方案)

**第一步:** 不存在  $i (1 \leq i < m)$  使得  $a_{i+1} - a_i > 2$

**证明:** 若存在, 使得。则令

$$a_k' = a_k (1 \leq k \leq m, k \neq i, k \neq i+1), a_i' = a_i + 1, a_{i+1}' = a_{i+1} - 1$$

显然

$$\prod_{k=1}^m a_k' > \prod_{k=1}^m a_k$$

矛盾。

**第二步:** 不存在正整数对  $(i, j) (1 \leq i < j < m)$ , 满足  $a_{i+1} - a_i = 2$  且  $a_{j+1} - a_j = 2$ 。

**证明:** 若存在正整数对  $(i, j) (1 \leq i < j < m)$ , 满足  $a_{i+1} - a_i = 2$  且  $a_{j+1} - a_j = 2$ , 则令  $a_k' = a_k (1 \leq k \leq m, k \neq i, k \neq j+1)$ ,  $a_i' = a_i + 1, a_{j+1}' = a_{j+1} - 1$ , 则

$$\prod_{k=1}^m a_k' > \prod_{k=1}^m a_k, \text{ 矛盾。}$$

**第三步:**  $a_1 < 4$   
 $a_1 \geq 4$

**证明:** 假设。

首先我们明确一点: 若  $a_i \geq 5 (1 \leq i \leq m)$ , 则必存在某个  $j (1 \leq j \leq m)$ , 满足  $a_j = a_i - 2$ 。

若不然，则令  $a_k = a_k (1 \leq k \leq m, k \neq i), a_i' = a_i - 2, a_{m+1} = 2$

显然，有：

$$\prod_{k=1}^{m+1} a_k' > \prod_{k=1}^m a_k$$

矛盾。

因此，有：

$$a_1 = 4$$

由于不存在  $i (1 \leq i \leq m)$ ，使得  $a_i = 3$ ，再结合第一步的证明，有： $a_2 = 6$ ；由于不存在  $i (1 \leq i \leq m)$ ，使得  $a_i = 5$ ，再结合第一步的证明，有： $a_3 = 8$  或者  $m = 2$ ，

而无论哪种情况，都产生矛盾。故原命题得证。

第四步：若  $a_1 = 3$  且存在  $i (1 \leq i < m)$ ，使得  $a_i = 2$ ，则必有

$$i = m - 1$$

若存在  $i (1 \leq i < m - 1)$ ，满足  $a_{i+1} - a_i = 2$ ，则由第二步的证明，知： $a_{i+2} = a_{i+1} +$

$1$ ，则若令： $a_{m+1}' = 2, a_{i+2}' = a_{i+2} - 2, a_j' = a_j (1 \leq j \leq m, j \neq i + 2)$ ，则  $\prod_{k=1}^{m+1} a_k' > \prod_{k=1}^m a_k$ ，

矛盾。故命题得证。

**证明：**

由以上几步的证明，我们便可确认：我们的感觉是无误的。这样，这道题似乎变得异常的简单，因为我们接着要做的，就只是计算这个最大的乘积的值了，这里用到了高精度计算。

让我们再回顾一下解题过程，对于较原始的动态规划算法，我们觉得里面显然有许多不必要的计算，从而提出：能否直接导出一般规律？通过大胆的猜想和严密的证明，这一点得到了肯定，而算法的时间复杂度也从动态规划的



$O(N^2)$ ，下降到了现在的 $O(N)$ 。而这一切都应该归功于数学思想的大胆和合理的应用。

## 二．利用数学模型化繁为简。

能够对原题导出数学结论无疑是最直接地运用了数学思想，而在很多情况下，往往没有那么简单。在有些实际应用中，我们需要将原来的实际问题抽象成数学模型，然后加以解决。而在某些程序设计竞赛的试题中，建立数学模型也需要一定的技巧，需要运用一定的数学思想。

下面，我们来考虑一个利用数学思想“建模”的例子——

### 例题二 三角形灯塔

有一个 $N$ 行 ( $0 < N \leq 50$ ) 的三角形灯塔，它的第1行有1个灯，第2行有2个灯，...，第 $N$ 行有 $N$ 个灯。我们用  $(i, j)$  表示从上至下第 $i$ 行，从左至右第 $j$ 个灯。

每个灯有明暗两种状态，第 $i$ 行 ( $1 \leq i < N$ ) 的任一个灯  $(i, j)$  的状态由下一行的两个灯  $(i+1, j)$  和  $(i+1, j+1)$  的状态决定 ( $1 \leq j \leq i$ )。具体的规则如下表所示：

$(i, j)$ 的状态	$(i+1, j)$ 的状态	$(i+1, j+1)$ 的状态
暗	亮	亮
亮	暗	亮
暗	暗	暗
亮	亮	暗

请你编一个程序，从已知的  $P$  ( $P \geq 0$ ) 个灯的状态出发，推出最底一行  $N$  个灯的所有可能的状态总数。

**[输入]**

第一行行首为三角形灯塔的行数  $N$ ，从第 2 行开始每行为一个已知状态的灯的信息，即三个由空格隔开的整数：

其中  $k$  为该灯的状态，由 0, 1 表示（0 表示暗，1 表示亮）。输入数据以满足 的一行结束。

### [输出]

若问题无解，则输出“NO ANSWER！”；否则输出可能的状态总数。

## 算法分析

乍一看，似乎只能用搜索来解决，但搜索的时间效率不尽如人意。因此，我们不得不另觅他路。

我们把注意力集中在灯的亮暗规则上。我们容易发现，如果令  $l[i, j]$  表示灯的亮暗情况（灯亮时为 1，灯暗时为 0），则（我们发觉 + 1）

我们希望利用上面这个特点，得到一个效率更高的算法。为此，我们不希望使用“xor”运算，如果能用加、减、乘或除号来代替“xor”，那么我们就能够用第  $N$  行的灯的状态来用表达式表示出所有灯的状态。这不由使我们想到了“二进制的无进位加法”，即：

为方便起见，这里仅用普通的加号来代表二进制的无进位加法（即  $0+1=1$ ， $1+0=1$ ， $0+0=0$ ， $1+1=0$ ）。

这样，若令

$$x[i] = l[n, i] (1 \leq i \leq n)$$

我们就可以把每个  $l[i, j]$  用  $x[i] (1 \leq i \leq n)$  表示出来，根据所给的已知条件，就可以得到若干个关于  $x[1], x[2], \dots, x[n]$  的方程，构成方程组。我们只要求出该方程组的解的组数即可。

求这个特殊的方程组的解的组数的大致方法是：

先用高斯消元法把方程的个数减少到一个，如果最后剩下的那个方程中含有  $p$  个未知数（ $p$  为自然数），则由于任意地取定其中  $p-1$  个未知数的值，则最后一个未知数的值也随之确定，故得到解的组数为  $2^{p-1}$ 。

但在求解过程中必须考虑一些特殊情况。

这样，我们就找到了一个较为高效的算法。

在本题中，我们先是由灯的亮暗所遵循的规律发觉了一个较为一般的递推关系，而下面一步才是非常关键的——把递推关系中的逻辑运算转换成数学运算，这一步，为我们构造方程组这一数学模型提供了极大的方便，使问题迎刃而解。

数学思想在本题的应用，使我们方便地构造出了数论模型，使问题圆满地解决。

### 三．通过数学分析化未知为已知。

有些构造性地问题本身就是由数学问题衍生而来，但正因为问题的“构造性”，使这类问题的解法让人捉摸不透，很难想到。在这种情况下，我们可以试

着先从数学的角度分析问题，若能得出对问题直接有益的结论则是最好，如果不能，我们也可以从分析问题的过程中启发思维，从而巧妙地构造算法。

下面来看一个具体的例子体会一下数学分析对解题的帮助——

### 例题三 配锁问题

某机要部门安装了电子锁。M 个工作人员每人发一张磁卡，卡上有开锁的密码特征。为了确保安全，规定至少要有 N 个人同时使用各自的磁卡才能将锁打开，并且任意 N 个人在一起都能将锁打开。现在需要你计算一下，电子锁上至少要有多少种特征，每个人的磁卡上至少有几个特征。如果特征的编号用从 1 开始的自然数表示，将每个人的磁卡的特征编号打印出来。要求输出的电子锁的总特征数最少。

为了使问题简单，规定：

$$3 \leq M \leq 7, 1 \leq N \leq 4, N \leq M$$

#### [输入]

只有一行，包括两个由空格隔开的正整数 M，N。

#### [输出]

输出包括 M 行，第 i 行有若干个递增的正整数，表示第 i 个工作人员所持磁卡上的全部特征的编号。

### 算法分析

初看此题，也许你会觉得毫无思路。“至少要有  $N$  个人同时使用各自的磁卡才能将锁打开”这个看似极其关键的条件到底蕴涵了哪些性质呢？我们不妨先从数学的角度来考虑这个问题，来看一看能否估计出电子锁上特征数的最小值和每人磁卡上特征数的下界。

我们可以证明：

**引理：**给  $M$  个工作人员每人发一张具有开锁密码特征的磁卡，规定至少要有  $N$

个人同时使用各自的磁卡才能将锁打开，则：电子锁上特征数  $tot \geq C_m^{m-n+1}$

$$per \geq C_{m-1}^{n-1}$$

每个人的磁卡上的特征数。

**证明：**由于“至少要有  $N$  个人同时使用各自磁卡才能将锁打开”，意即任意  $N-1$  个人在一起，都无法将锁打开，从而必然缺少一种开锁的密码特征  $A$ ；并且在其余的  $M - (N-1)$  个人中，任意一人加入到  $N-1$  个人中，他们就能将锁打开，故这其余的  $M - (N-1)$  个人必同时拥有密码特征  $A$ 。而容易证明：每  $N-1$  个人在一起，他们缺少的一种密码特征  $A$  不能和其他一组  $N-1$  个人一起缺少的密码特征相同（否则，由于这两组至少有  $N$  个不同的人，且他们都缺少密码特征  $A$ ，故这些至少  $N$  个人在一起无法将锁打开，矛盾）。从而电子锁上特征数

另外，对于每一个工作人员  $T$  来说，在其余  $M-1$  个人中，任选  $N-1$  个人在一起，都会因为缺少某中特征而无法开锁，而这缺少的特征必须是  $T$  所具备

的。故每个工作人员的磁卡上的特征数  $per \geq C_{m-1}^{n-1}$ 。

事实上，从后面的分析中，我们会发觉：所求的 tot 和 per 的最小值恰好就是

$$C_m^{m-n+1} \text{ 和 } C_{m-1}^{n-1}。$$

在上面的证明过程中，我们最感兴趣的并不是得到的结论（因为本题要求打印方案，故 tot 和 per 的值完全可以在打印过程中累加），而是在证明的“过程”中用到的：“每  $N-1$  个人都缺少一种密码特征，而**每  $M-(N-1)=M-N+1$  个人都同时具备该密码特征**，并且这个密码特征对从  $M$  个人中选出不同的  $M-N+1$  个人的组合来说是不同的”。这句话为我们打开了思路，由此，一个有效算法便应运而生：

初始时，特征数置为 1，在  $M$  个人中每选取  $M-N+1$  个人的组合，就为组合中每个工作人员配备当前特征，并将特征数+1。这样，枚举出了所有的

$C_m^{m-n+1}$  个组合后，便得出了所有工作人员磁卡上特征的方案了。

在本题中，我们通过数学分析，尽管也得出了结论，但由于本题要求输出所有方案，它的用处并不大，但分析过程中所用到的一个思路却直接导致了算法的形成。如果没有进行数学分析，或者没有考虑到这一思路，本题似乎就很难完成了。

#### 四．利用数学结论优化算法。

在上面的例题中，我们清晰地看到：数学思想的火花孕育了解题的算法。综合上面的三个例题，数学方法、思想的巧妙使用，破解了一个个难题。最

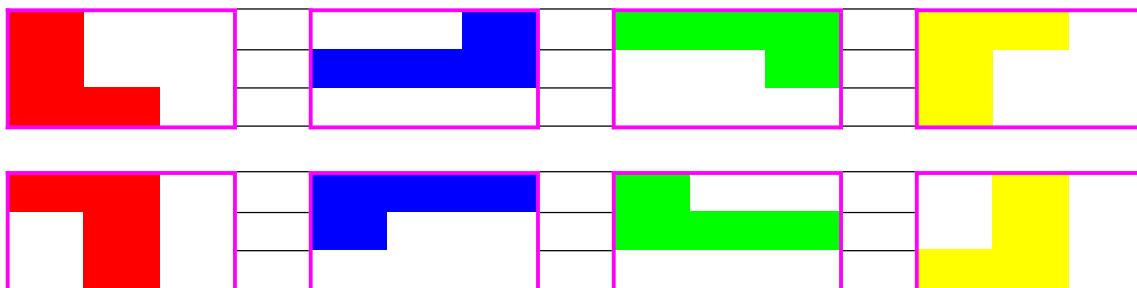
后，我们将通过一个并不难的搜索题，体会一下数学在优化算法方面的应用。

(读者可以自己比较一下原始算法和优化后的算法在时间效率上的差距)

### 例题四 骨牌覆盖问题

对于任意一个  $m \times n$  的矩阵，求 L 形骨牌覆盖后所剩方格数最少的一个方案。

L 形骨牌的 8 种形态



**[输入]**

输入包括一行，有由空格隔开的正整数  $m, n$ ，表示矩阵的大小。

**[输出]**

输出是一个  $m \times n$  的矩阵，矩阵中元素的值表示该格所在的骨牌编号（若该格不被骨牌覆盖，则值为 0），骨牌编号为从 1 开始的连续正整数。

### 算法分析

显然，这道题只有“搜索”，既然是搜索，我们就必然要把注意力集中在优化算法上。对于本题，显然在回溯搜索时，不能让已经“浪费”了的方格数太多，这样再继续搜索下去也是做无用功，于是我们就需要一个“浪费”方格数的上

界，也就是最优方案中，最多能覆盖的骨牌数目的下界；而事实上，我们完全可以求出它的确切值。

引理：对于  $m \times n$  的矩阵 ( $m, n > 3$ )，最多能覆盖的骨牌数目

$$number = \begin{cases} \left\lfloor \frac{m \times n}{4} \right\rfloor - 1 & \text{当 } m \times n \equiv 4 \pmod{8} \text{ 时} \\ \left\lfloor \frac{m \times n}{4} \right\rfloor & \text{当 } m \times n \not\equiv 4 \pmod{8} \text{ 时} \end{cases}$$

$$\text{证明：我们只证明： } number \leq \begin{cases} \left\lfloor \frac{m \times n}{4} \right\rfloor - 1 & m \times n \equiv 4 \pmod{8} \\ \left\lfloor \frac{m \times n}{4} \right\rfloor & m \times n \not\equiv 4 \pmod{8} \end{cases} \quad (1)$$

至于最大值为何能够取到，用数学归纳法是不难证明的，这里从略。

事实上，我们只需证明 (1) 命题对  $m \times n \equiv 4 \pmod{8}$  时成立，而其它情况是显然的。为此，我们只需证明：

如果一个  $m \times n$  的矩阵能用  $L$  形骨牌完全覆盖，则所用的  $L$  形骨牌数必为偶数。 (2)

为此我们引入  $L$  形骨牌的“头部”和“尾部”的概念：即字母 “ $L$ ” 底部的两个方格称为“尾部”，另外两个方格称为“头部”。“头部”和“尾部”都类似多米诺骨牌。不妨设  $n$  为偶数，则考虑  $m \times n$  矩阵的每一条水平线，这  $m$  条水平线共穿过“ $L$ ”形骨牌的“头部”或“尾部”的数目总和记为  $c$ ，一方面，对于每一条水平线，若它穿过奇数个“头部”或“尾部”，则在该水平线上部的其余奇数个方格内要实现多米诺骨牌的完全覆盖是不可能的。因此，每一条水平线必然穿过偶数个“头部”或“尾部”，故  $c$  为偶数。另一方面，每个“ $L$ ”形骨牌恰好被一条水平线切割到其“头部”或“尾部”，从而“ $L$ ”形骨牌的数目为偶数，(2) 得证。

由于我们计算出了最多能覆盖的骨牌数，搜索过程就大为优化。

算法的大致流程（回溯法）：

自左而右、自上而下搜索每一个方格的覆盖情况，如果可以以该方格为左上角放置骨牌，则试着放置该骨牌，设置相应的方格被覆盖标志；然后设置该



方格不被覆盖标志，搜索下一方格。搜索过程中，一旦已经产生的空格数超出空格数的上界，则回溯。

对比优化前和优化后算法的时效，我们发现，由于加入了阈值，使得优化后的算法减少了很多不必要的搜索，算法的效率自然提高了不少。

本题给了我们这样的启示：数学思想不仅仅能直接提供解题思路和方法，在更多的场合，它只是以一种辅助工具的形式出现的。这就要求我们具备将多种算法，多种思想结合使用的能力。

## [结束语]

从上面的几个例子中，我们可窥一斑——数学思想作为沟通问题与编程实现的一座桥梁，有着极其广泛的应用。它不仅可以直接为解题提供思路，如果与其他算法或思想相结合，也能够起到很好的辅助作用。最后，送给大家一句话，希望有所帮助：

在你觉得“山穷水复疑无路”时，不妨用数学的角度重新审视问题，也许就会“柳暗花明又一村”。

约定：凡是用(\*和\*)括住的都不是发言内容！！

No.1

(\*动画结束\*)大家好！我是南京市金陵中学的刘一鸣。今天我要和大家讨论的就是：一类搜索问题的优化思想——数据的有序化。

No.2

数据有序化的思想，就是将杂乱的数据，通过简单的分类和排序，变成有序的数据，从而加快搜索的速度。

(\*点击超级链接“为什么要进行数据有序化”， goto No.3\*)

No.3

为什么要数据有序化呢？

我们平时遇到的搜索问题，其数据往往有一大特点，那就是杂乱无章，毫无次序可言。

(\*点击鼠标，杂乱数据图出现\*)就是杂乱的数据。

(\*点击鼠标，有序数据图出现\*)而这则是有序的数据。

但是，在同一个题目中，应用的算法相同，而数据的有序程度不同，程序的效率往往会有较大的差异。

下面，我就给大家举一个例子，“装箱问题”。

No.4

请大家仔细看题目，注意，这里和一般的“装箱问题”不同的地方就在于，题目是要求“放满”集装箱，也就是说，货物体积总和必须恰巧等于集装箱总体积。

#### No.5

这里提供了两种算法的时间比较，我们不难发现，第 2 种算法在多数情况下运行得很好，而第 1 种算法则不很理想。

#### No.6

两个程序效率不同的原因在哪里？

(\*点击鼠标，图表出现\*)主要的原因是，我们在使用最优性剪枝的时候，最希望的就是能较早地得到一个逼近最优解的较优解。

(\*点击鼠标，"最优解"出现\*)这里就是最优解。

(\*点击鼠标，"不理想解"出现\*)这里是不太理想的初始解，不排序而直接搜索，很可能会产生这种初始解。

(\*点击鼠标，"最理想解"出现\*)这里是最理想的初始解，不排序而直接搜索和先排序再搜索都可能会产生这种初始解，不过后者的概率似乎略大。

(\*点击鼠标，"较理想解"出现\*)这里是较为理想的初始解，先排序再搜索，很可能会产生这种初始解，这就是它的优势所在。

#### No.7

当然，数据有序化的优点并不仅仅是这些。首先，对于大多数数据，它都有良好的优化效果，不过也不乏专门针对它的数据；其次，它实现起来很简单，对于刚才那个问题，在搜索前加上一个冒泡排序，只用加上几行就可以了；再其次，使用这种方法不会与其它优化方法形成冲突，甚至会为它们创造便利。所以，不难看出，数据有序化在实际应用中是大有裨益的。

## No.8

数据有序化大致可以分为两种。

第 1 种就是“预处理阶段的数据有序化”(\*点击对应的 HyperLink, goto No.9\*)

## No.9

(\*点击鼠标\*)

一般来说，我们解决一个问题，都是读入数据以后直接进行数据的加工。(\*点击鼠标，直至"加工"出现\*)

预处理阶段的数据有序化，就是在加工之前多一个数据的处理过程，把它们由杂乱的排成有序的。(\*点击鼠标，直至第 2 个箭头出现\*)

下面，我以 IOI2000 的"积木搭建"为例具体讲解。

## No.10

这道题目的题意大家应该比较熟悉了，我就不再多讲了。

传统的做法，是基于 3 维空间的算法，大家可以看看 Wang Renshen 同学的解题报告，这种方法虽然容易想到，但是效率并不高，有些官方测试数据甚至会超时。

现在，我们尝试在预处理阶段对数据进行有序化处理，来优化搜索。

#### No.11

先对构型的数据进行有序化处理。

(\*点击鼠标\*)将构型的所有小方块按照它们在空间中的顺序排序并编号。

(\*点击 2 次鼠标\*)用一个集合 $[1,v]$ 表示构型。

(\*点击 2 次鼠标\*)这样，就将原来的 3 维几何体转化成了一个 1 维的集合。

#### No.12

然后，我们对积木的数据进行有序化处理。

(\*点击鼠标\*)枚举所有能够插入构型的积木。

(\*点击 2 次鼠标\*)用积木所包含的方块的编号组成的集合分别表示每块积木。

(\*点击 3 次鼠标\*)这样，一个积木可以放进构型里，就可以用一个集合是否包含于另一个集合来表示。

(\*点击 2 次鼠标\*)

#### No.13

下面，我们看看怎样从一个构型里挖去一块积木。

这是一个构型，

(\*点击鼠标\*)我们用 $[1,10]$ 表示。现在，我们挖去一块积木，

(\*点击鼠标，并将鼠标指针指向浅绿色区域\*)大家请看，浅绿色区域代表挖掉的一块积木 $\{3,6,7,9\}$ ，那么，剩下的构型就是集合 $\{1,2,4,5,8,10\}$ 。

(\*点击 2 次鼠标\*)这个操作，数据有序化处理以后，我们可以用集合的减运算来表示。(\*点击鼠标\*)

#### No.14

我们现在对于剩下的那个构型

(\*点击 3 次鼠标\*)还想继续放一块积木

(\*点击 3 次鼠标\*)就是这块 $\{4,5,7,8\}$ 。我们发现， $\{4,5,7,8\}$ 并不包含于 $\{1,2,4,5,8,10\}$ ，

(\*点击鼠标\*)所以我们判定，

(\*点击 2 次鼠标\*)积木不能放入构型。

#### No.15

最后看看积木的冲突的判定

(\*点击 2 次鼠标\*)不难看出，左右两个积木单独放，都能放进构型

(\*点击 2 次鼠标\*)但是，这两个积木同时放，是存在冲突的，就是第 7 号方块  
(\*鼠标指向 7 号方块\*)

(\*点击鼠标\*)转化为集合表示，我们会发现，冲突的积木的交集不为空集。

#### No.16

至此，预处理阶段的数据有序化处理全部完毕，大家可以看一看有序化前后的比较。(\*点击鼠标\*)

我们现在已经成功地将 3 维几何体转化成了 1 维的集合，剩下的事情，就是对集合简单地进行搜索，这个算法很基本，也很简单，只要 DFS 就可以了。

事实上，这道题目的成功解决，主要就是在预处理方面下了功夫，数据的有序化使问题的数学模型得到了精简。(\*点击鼠标\*)

(\*点击鼠标，出现"Return"按钮，点击返回 No.8\*)

## No.17

接下来，我们再来看看"实时处理阶段的数据有序化"

传统的方法一般在计算出一些数据以后，(\*点击鼠标\*)，直接加工或保存。(\*点击 4 次鼠标\*)

而实时处理阶段的数据有序化，就是在计算出数据以后，先将其转化为有序的数据。(\*点击 2 次鼠标\*)，然后分别加工或保存，(\*点击 2 次鼠标\*)，在这时可能会发现一些无用的数据，就可以直接舍弃(\*点击 2 次鼠标\*)，这是它的额外的一种好处。

在这里，我们经常需要用到数据的最小表示法。

最小表示法是一种基于数据有序化思想的方法。其基本思想就是，对于同构的一类数据，在保存的时候只将最小的一个存储。

## No.18

传统的表示方法，就是得到一个合法状态以后，先得到所有的同构状态，(\*点击 2 次鼠标\*)，如果这些状态中有一个已经保存，(\*点击 3 次鼠标\*)，则  $S$  可以舍弃，(\*点击鼠标\*)，否则保存  $S$ (\*点击 2 次鼠标\*)。

#### No.19

而最小表示法，在得到一个合法状态以后，先将其转化为最小表示(\*点击 2 次鼠标\*)，如果最小表示已经保存，(\*点击 2 次鼠标\*)，则舍弃(\*点击鼠标\*)；否则，将最小表示保存(\*点击 2 次鼠标\*)。

最小表示有一个重要性质：唯一性。每一种状态都有且只有一个最小表示。这个性质对于搜索的优化是很有帮助的。下面，我以一道例题来详细讲解。

#### No.20

这是一个"N 皇后问题"的改进版，请大家仔细看题目。

#### No.21

首先要解决的是状态的表示，这里使用的是一种大家都很熟悉的方法，就是把 2 维的棋盘转化为 1 个  $n$  元组表示。(\*点击 3 次鼠标\*)

#### No.22

再来看看翻转、旋转的具体过程。以  $n=5$  为例，首先是以铅垂线为轴的翻转，(\*点击鼠标\*)，左边这个棋盘是一个合法的棋盘，经过翻转，得到右边这个



棋盘, (\*点击鼠标\*)。它们的  $n$  元组表示不难求出, (\*点击 3 次鼠标\*), 所以, 我们很容易地求出了以铅垂线为轴的翻转方法。(\*点击鼠标\*)

#### No.23

依照以上的方法, 我们还可以求出以水平线为轴的翻转(\*点击鼠标\*)

以对角线为轴的翻转(\*点击鼠标\*), 这里的  $b_i$  表示第  $i$  行的皇后所在的列。

还有 3 种旋转过程(\*点击鼠标\*)。

#### No.24

现在, 我们使用最小表示法。

一种方法就是按部就班地生成状态, 转化成最小表示, 再判断。(\*点击 7 次鼠标\*)

其实, 最后保存的只有最小表示, 所以, 不是最小表示的解可以在发现以后就立即回溯。

在枚举的过程中(\*点击鼠标\*), 如果发现由当前状态不可能生成最小表示, 则回溯(\*点击 3 次鼠标\*), 否则继续枚举(\*点击鼠标\*)。这就提供了一个新的剪枝(\*点击鼠标\*)。

#### No.25

由翻转、旋转的具体过程可知, 当前搜索到的状态如果满足最小表示, 必须符合这些条件(\*点击 2 次鼠标\*)。这是一个比较强的剪枝条件, 在实际应用中能剪掉一大半的无用枝杈。

## No.26

再看看这幅图，我们发现，利用新的剪枝条件，能够找到的解一定是最小表示的解。(\*点击 2 次鼠标\*)

由最小表示的唯一性，能够搜索到的、并满足最小表示的解一定是不同构的。所以，判断同构的过程可以省略，已经搜索到的状态也因此可以不保存。(\*点击鼠标\*)空间复杂度因此大幅下降。

## No.27

这是数据有序化前后的时空复杂度对比， $S$  是合法解的集合，(\*用鼠标指向  $|S|$ \*)。最小表示的优势是显而易见的。

## No.28

下面我们来比较一下两种实现方法。(\*点击鼠标\*)

预处理阶段的数据有序化，时间上的耗费小一些，但对空间的要求较高。

而实时处理阶段的数据有序化，更灵活一些，数据可以即时处理，所以空间要求小一些。但是，如果搜索树的结点较多，它不一定会很理想，因为它可能会重复处理某些相同的结点。

但是，这两种方法实际上是优势互补的。所以，在应用的时候，我们可以将两种方法并用，扬长避短，达到更好的效果。

## No.29

最后，我们对今天的研究做一个总结。

我们今天所讨论的，就是将混乱无序的数据，通过简单的方法，转化成为符合科学美的有序数据。科学本身就是一种美，我们努力创造出的符合科学美的数据，有时会令我们事半功倍。

也许大家会说，今天所讨论的题目，我所讲的方法都不是最好的方法，许多方法能使程序运行得更快。但是，我在这里要说的是一个性价比的问题，今天我所讲的优化方法，都只要在原来的程序作一些修改就可以了，不必将原来的程序推翻从头写，很经济，很划算。我们都知道，不论是在竞赛中，还是在生活中，解决一个问题，重要的是尽快设计出解决方案，进而获得答案，而不是让我们的程序运行的时间最短，只要能在限定时间内解决问题，就是成功。所以，只有合理的选择，才能获得最大的性价比。

No.30

我的发言到此结束，谢谢大家！

# 寻找最大重复子串

江苏金陵中学 林希德

## 【关键字】

后缀树 KMP 推广算法 最优子串

## 【摘要】

关于字符串处理，无论是国内的个大竞赛还是中文的经典宝书，都相对较少涉及。曾见各位高手在信息学论坛上就某前辈提出的此类问题讨论得热火朝天，于是我决心翻阅多方资料仔细研究，仅望对大家发表高见起抛砖引玉的小作用。

本文第一章给出了问题的详细描述和我对该问题的拙见，第二、三章简述了字符串处理的两个常用算法——后缀树和 KMP，第四章阐明了寻找最大重复子串的主算法。

## [目录]

### 一 问题提出和初步认识

#### 1.1 问题描述

#### 1.2 初步分析

### 二 后缀树

#### 2.1 后缀树的定义

#### 2.2 后缀树的构建

#### 2.3 后缀链接

#### 2.4 性能分析

### 三 模式匹配

#### 3.1 朴素模式匹配

#### 3.2 KMP模式匹配

#### 3.3 前缀函数

#### 3.4 性能分析

#### 3.5 KMP 推广算法

### 四 主算法

#### 4.1 问题转化

#### 4.2 字符串分解

#### 4.3 范围限定

## 4.4 找到循环节

## 4.5 辅助函数

## 4.6 性能分析

## 五 论文小结

## [正文]

## 一、问题提出和初步分析

## 1.1 章节

## 定义一：

对于字符串S，如果存在正整数p使得

$$\forall 1 \leq x \leq |S| - p : S_x = S_{x+p}.$$

那么称p是S的**循环周期**(Period)， $e = |S| / p$ 是S的**指数**(Power)，任何长度为p的S的子串都是S的一个**循环节**(Repetend)，特别的，当 $e \geq 2$ 时，S是个大小为p的**重复字符串**(Repetition or P-power word)。

## 问题描述：

给出一个由大写字母组成的字符串W，W长度为 $1 \leq n \leq 10^5$ ，请在W的所有子串中找出循环周期最长的那个重复字符串（Finding Maximal Repetition），即**最大重复子串**。

## 1.2 章节

## 定义二：

为方便表达，我们用符号 **$W(u,v)$** 表示开始于位置u结束于位置v的W的子串。

## 初步分析：

一个 $O(n^2)$ 的算法是乍一看最直观的收获。我们可以首先从n到1枚举循环周期p，然后针对p从位置1到n搜索重复子串，一旦找到立即输出并退出循环，如下：

```

For p = n → 1 do
  For i = 1 → n do
    If i > p 并且  $W_i = W_{i-p}$  then
       $G_i = \text{Max}(G_{i-1} + 1, 1)$ 
    Else
       $G_i = 0$ 
    End if
    If  $G_i = p$  then
      输出  $W(i-2p+1,i)$ ; 退出循环
    End for
  End for
End for

```

这个  $O(n^2)$  的算法简单易编，但速度较慢。那么，怎样让复杂度从  $O(n^2)$  降到  $O(n)$  呢？为说清楚这个线性算法，我们不得不先介绍一种关于字符串处理的数据结构——后缀树，还有就是模式匹配的 KMP。

## 二、后缀树

### 2.1 章节

**后缀树定义：**

→ 令  $W = W + \$$ ，这样  $W$  的最后一个字符从未在前面的任何一个位置上出现过。添置 '\$' 的目的，将在下一小节中予以说明。

后缀树其实就是一棵**检索树**(Trie)，和普通检索树一样我们不断往树中插入字符串和添置顶点，只不过，后缀树还有一些特殊的性质：

i) 我们从大到小依次往树中插入  $W$  的后缀，这其实是后缀树的名称由来，也是它的**构建过程**。

ii) 树上每条边  $E(u,v)$  有两个参数  $r$  和  $\lambda$ ，记为  $E(u,v) = (r, \lambda)$ ，代表子串  $W(r, \lambda)$ 。

iii) 树上每个节点均有 26 个儿子，代表 26 个大写字母。如果父亲节点  $u$  的第 1 个儿子是节点  $a$ ，那么  $E(u,a)$  上的子串一定以字母 A 开头；如果第 2 个儿子是节点  $b$ ，那么  $E(u,b)$  上的子串一定以字母 B 开头……以此类推。

**定义三：**

如果从根 Root 到节点  $x$  途径若干条边，将这些边上的子串顺次连接得到字符串  $S_x$ ，那么称  $S_x$  是  $x$  的**对应子串**， $x$  是  $S_x$  的**对应节点**。易知，“节点  $\rightarrow$  对应子串”和“字符串  $\rightarrow$  对应节点”都是**一一映射**的关系。

iv) 后缀树有  $n$  个叶子节点  $Leaf_1, Leaf_2, \dots, Leaf_n$ ， $Leaf_i$  的对应子串实际就是  $W$  的后缀  $W(i,n)$ 。这一点通过性质 i) 就可以得到，这里无非明确一下。

### 2.2 章节

## 定义四：

$Head_i$  是  $W(i,n)$  和  $W(j,n)$  的最长公共前缀，其中  $j$  是小于  $i$  的任意正整数， $Tail_i$  使得  $Head_i + Tail_i = W(i,n)$ 。

## 定义五：

只要子串  $S = W(u,v)$  满足  $u \leq x$ ，我们就说  $S$  在范围  $x$  内出现过。 $Head_i$  其实就是在范围  $i-1$  内出现过的  $W(i,n)$  的最长前缀。

## 后缀树的构建：

初始化后缀树为只有根结点的树

For  $i = 1 \rightarrow n$  do

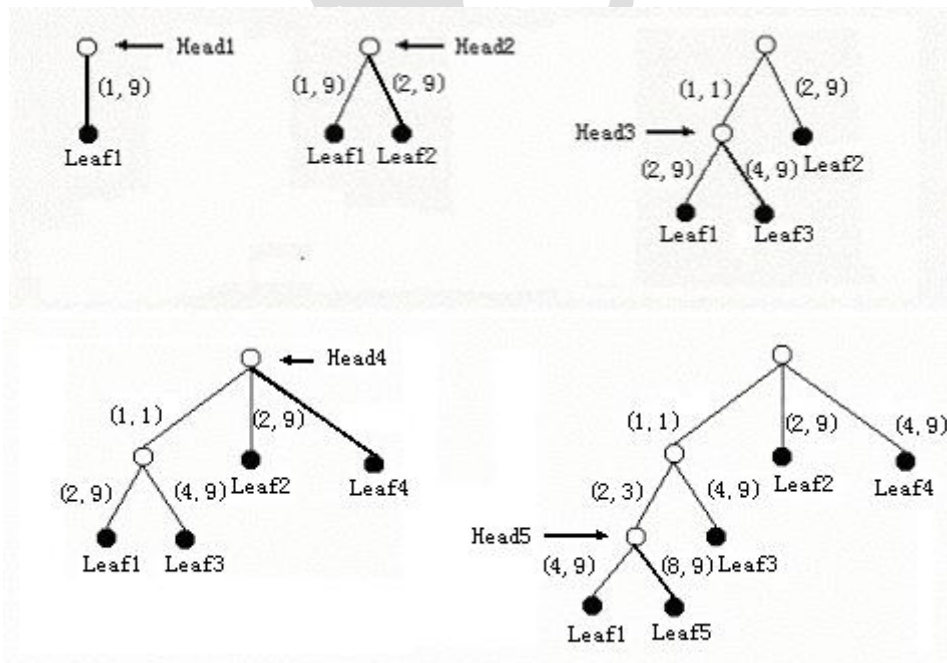
    找到  $Head_i$  的对应节点  $h_i$ ;

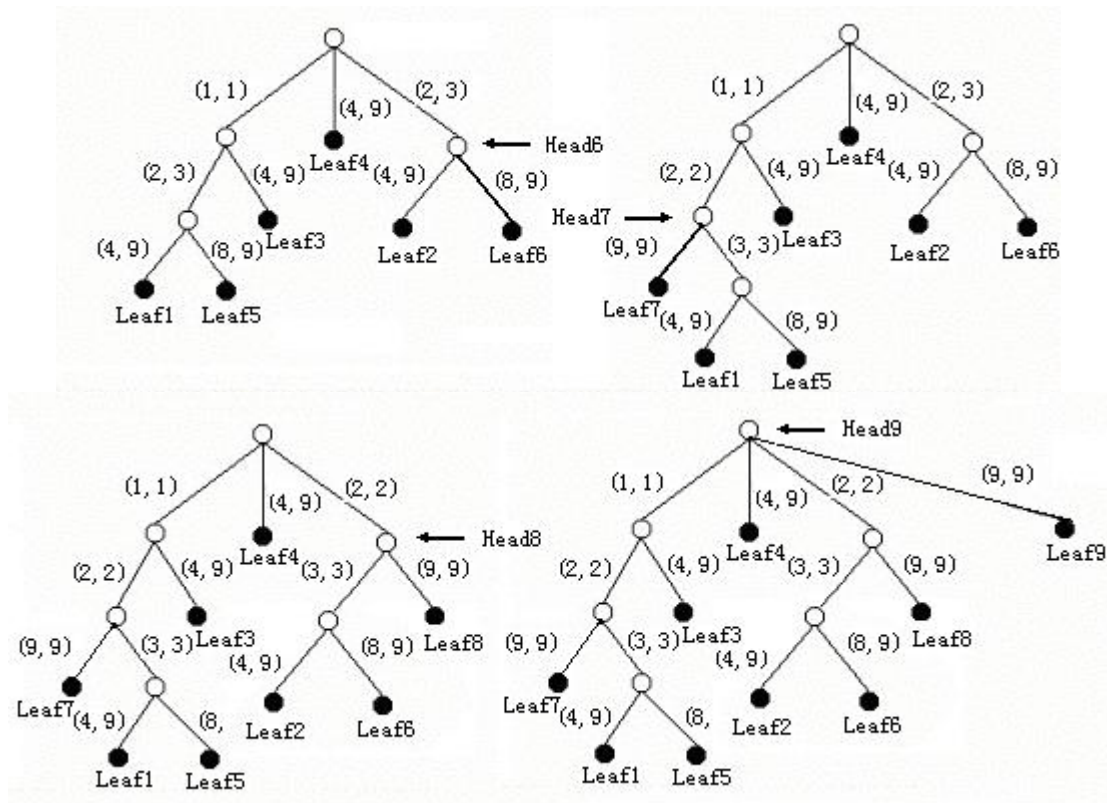
    增加叶节点  $Leaf_i$ ，使得  $E(h_i, Leaf_i) = (|Head_i| + 1, n)$ ;

End for

## 举例

下面让我们来看看字符串  $W = \text{"AGATAGAG\$"}$  的后缀树被逐步构建的具体例子，其中黑色圆圈是叶节点，圆括号内的是边参数  $(r, \lambda)$ ：






增加Leaf<sub>i</sub>只是O(1)的工作，关键问题是如何找到Head<sub>i</sub>的对应节点h<sub>i</sub>。最直观也是最野蛮的方法是从根节点开始对W(i,n)实行**逐个字符地扫描**：

```


hi = Find(Root, W(i,n)) ;
Function Find(实参：开始节点 p; 字符串 S): 指针类型;
While true do
    e = p 的第 ord(S1) - 64 个儿子节点;
    If e 不存在 then
        返回指针 p, Break;
    End if
    逐个字符比较找出 S 与 E(p,e)的最长公共前缀 U
    L = |U|
    IF L < |S| then
        p = e
        S = S(L+1, |S|)
    Else
        增加新节点 q, 令 V = E(p,e)
        令 q 成为 p 的儿子, E(p,q) = V(1,L)
        令 e 成为 q 的儿子, E(q,e) = V(L+1, |V|)
        返回指针 q, Break;
    End if

```



请看  处，或许你会担心  $V(L+1, |V|)$  可能是个无意义的空串，其实， $L = |V|$  的情况一定不会出现。

**反证法一：**


如果  $L = |V|$ ，那么就说明  $W(i, n)$  是某个  $W(j, n)$  ( $1 \leq j < i$ ) 的前缀，也就是说  $W$  的末尾字符 '\$' 至少出现过两次。这与刚才  处的规定违背。

故，任何增添进后缀树中的 **边都不是空串**。

这种野蛮的扫描算法使得时间复杂度高达  $O(n^2)$ ，还外加一个不小的常数因子。其实，这个 Find 函数并不是完全的没有用处，只是，为加快寻找  $h_i$  的速度我们需要使用辅助结构——**后缀链接**。

### 2.3 章节

**后缀链接的定义 (McCreight Arithmetic)：**

令  $\text{Head}_{i-1} = az$ ，其中  $a$  是字符串  $W$  的第  $i-1$  位字符。由于  $z$  在范围  $i$  内出现过至少两次，所以  一定有  $|\text{Head}_i| \geq |z|$ ， $z$  是  $\text{Head}_i$  的前缀。所谓  $h_{i-1}$  的后缀链接 (Suffix Link) 实际是由  $h_{i-1}$  指向  $z$  对应节点  $d$  的指针  $\text{Link } h_{i-1}$ 。当然， $z$  有可能是空串，此时  $\text{Link } h_{i-1}$  由  $h_{i-1}$  指向根节点  $\text{Root}$ 。

**创建方法：**

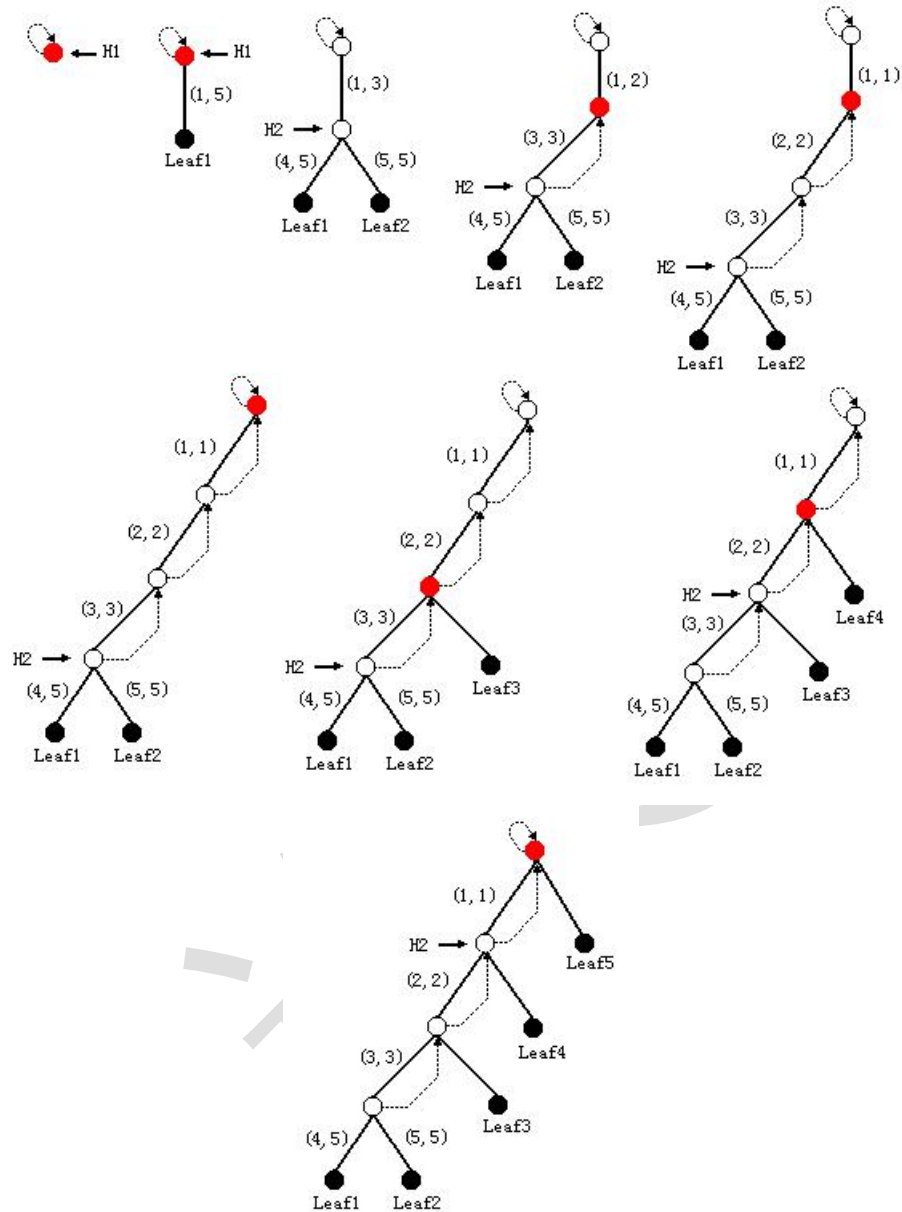
通过后缀链接，我们在  $O(1)$  时间内找到节点  $d$ ，然后再使用函数  $\text{Find}(d, W(i + |z|, n))$  找到  $h_i$ ，速度自然要快很多倍。现在的问题是，怎样在每次建立节点  $h_i$  后创建  $\text{Link } h_i$  呢？规定：

- 1) 根节点  $\text{Root}$  的后缀链接指向它自身
- 2) 任何一个非叶节点的后缀链接都在该节点出现以后被立即创建

根据这两条规定， $h_i$  的父亲节点  $c$  的后缀链接一定存在。我们从  $\text{Link } c$  出发，**向下搜索**，即可找到  $z$  的对应节点  $d$ 。

**举例二：**

下面让我们来看看字符串  $W = \text{"AAAA\$"}$  是如何利用后缀链接加快寻找  $h_i$  的速度的。图中，虚线箭头表示相应节点的后缀链接，黑色圆圈是叶子节点，红色圆圈是当前后缀链接指向的节点。



创建后缀链接的伪代码如下：

```

p = hi
While Link p 不存在 do
    c = p 的父亲节点
    V = E(c, p)
    If c = Root then
        Link p = Down(Link c, V(2, |V|))
    Else
        Link p = Down(Link c, V)
    End if
o = Link o
    
```

函数Down的伪代码如下：

```
Function Down(实参：开始节点 p；字符串 S)：指针类型；
  While true do
    e = p 的第 ord(S1) - 64 个儿子节点；
    If e 不存在 or S 是空串 then
      返回指针 p, Break;
    End if
    L = Min(|S|, |E(p,e)|)
    IF L ≤ |S| then
      p = e
      S = S(L+1, |S|)
    Else
      增加新节点 q, 令 V = E(p,e)
      令 q 成为 p 的儿子, E(p,q) = V(1,L)
      令 e 成为 q 的儿子, E(q,e) = V(L+1, |V|)
      返回指针 q, Break;
    End if
```

观察一下就会发现，函数 Down 和 Find 几乎一模一样，只不过<sup>①</sup>处，L 不是通过逐个的字符比较而是直接的长度比较得出。为什么呢？

反正法二：

令  $L = |S \text{ 和 } E(p,e) \text{ 的最长公共前缀}|$ 。

如果  $L < \min(|S|, |E(p,e)|)$ ，那么就证明 z 在范围 i 内从未出现过。这于<sup>②</sup>处的结论矛盾。

故，逐个字符比较的结果一定是  $L = \min(|S|, |E(p,e)|)$ 。

## 2.4 章节

算法主框架回顾：

回顾和整理一下算法主框架，看看我们究竟做了些什么？

```
For i = 1 → n do
  步骤 1、函数 Find 从 Link hi-1 开始向下搜索找到节点 hi
  步骤 2、增添叶子节点 Leafi
  步骤 3、函数 Down 创建 hi 的后缀链接 Link hi
```

### 后缀树性能分析：

接着刚才文本框内的伪代码来谈论。对于给定的 $i$ ，步骤2的复杂度为 $O(1)$ ，但由于无法确定Link  $h_{i-1}$ 到 $h_i$ 之间的节点个数，所以不能保证步骤1总是线性的。

局部估算失败，不妨从整体入手。有一点是肯定的，那就是 $i + |\text{Head}_i|$ 总随着 $i$ 的递增而递增。因此， $W$ 中的每个字符只会被Find函数遍历1次，**总体复杂度是 $O(n)$ 的**。

## 三、模式匹配的 KMP 算法

如何判断字符串 $T$ （称 $T$ 为模式）是否是字符串 $S$ （称 $S$ 为主串）的子串呢？

至今为止，解决这个问题最优秀的算法是1xxx年由D.E.Knuth、J.H.Morris和V.R.Pratt共同提出的模式匹配KMP算法。大多数选手早已熟练掌握KMP，故，我在此仅为论文的完整性对它进行简要叙述。

### 3.1章节

为方便说明，我们令 $n = |S|$ ， $m = |T|$ 。

#### 朴素的模式匹配算法


朴素模式匹配的基本思想是：将主串 $S$ 的第一个字符和模式 $T$ 的第一个字符进行比较，若相等则进一步比较二者的后续字符；否则，从 $S$ 的第二个字符开始重新和 $T$ 的第一个字符进行比较……以此类推，直至模式 $T$ 和主串 $S$ 中的某一个子串相等，则称匹配成功，否则称匹配失败。该算法复杂度是 $O(nm)$ ，显然令我们很不满意。

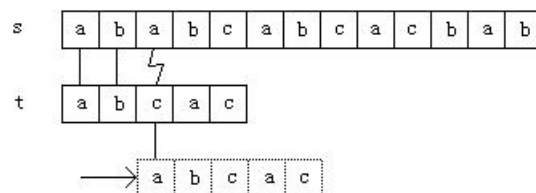
### 3.2章节

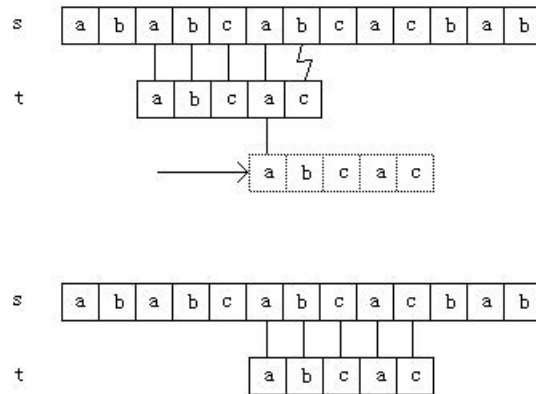
#### KMP模式匹配

如果说索引指针指向了当前有待比较的 $S$ 和 $T$ 中的两字符，那么朴素模式匹配算法效率不高的主要原因就是没有充分利用匹配过程中已经得到的部分匹配信息，而任由索引指针回溯。KMP正是针对这点缺陷对朴素算法做了实质性的改进，它主要的思想是：**当出现字符比较不相等时，让模式在主串上尽可能的向右滑动，然后接着进行比较。**

#### 举例三：

例如下图，其中， $S = \text{"a b a b c a b c a c b a b"}$ ， $T = \text{"a b c a c"}$ ，表示索引指针。





## 3.3 章节

## 前缀函数

一切问题集中在如何让模式T尽可能向右滑动上。

假设当前索引指针正向右移动，在它指向字符  $T_j$  和  $S_i$  时发现  $T_j \neq S_i$ ，那么对于一切正整数  $k$  ( $i - j + 1 < k \leq n - m + 1$ )，满足  $S(k, k + m) = T$  的必要条件 (记为※号) 是  $k \geq i$  或者  $T(1, i - k)$  是  $T(1, j - 1)$  的后缀。为防止重复或者遗漏，模式T向右滑动的距离应当等于  $\text{Min}\{x : i - j + 1 + x \text{ 满足必要条件※}\}$ 。因为空串  $T(1, i - i)$  一定是  $T(1, j - 1)$  的后缀，所以必要条件中的  $k \geq i$  完全可以省略。

KMP算法的核心就是：令  $\pi[j-1] = \text{Max}\{x : T(1, x) \text{ 是 } T(1, j-1) \text{ 的后缀}\}$ ，然后将模式T向右滑动，使得索引指针指向  $S_i$  和  $T$  的第  $\pi[j-1] + 1$  个字符。 $\pi[i]$  的大小与主串  $S$  并无关联，所以我们应该事先通过预处理求出并保存所有  $\pi[i]$  ( $1 \leq i \leq m$ ) 值，即所谓的前缀函数。过程如下：

```

 $\pi[0] = -1$ 
For  $i = 1 \rightarrow m$ 
   $K = \pi[i-1]$ 
  While ( $K \geq 0$ ) 并且 ( $t$  的第  $K+1$  个字符  $\neq t$  的第  $i$  个字符)
     $\rightarrow K = \pi[K]$ 
  End while
   $\pi[i] = K + 1;$ 

```

有了前缀函数，KMP 主算法就如下方框所述般简单明了：

```

Q = 0
For i = 1 → m
    While (Q > 0) and (t 的第 Q+1 个字符 ≠ s 的第 i 个字符)
        Q = π [Q]
    End while
    If t 的第 Q+1 个字符 = s 的第 i 个字符 then
        Q = Q + 1
        If Q = m then
            找到了模式匹配，退出循环。
        End if
    End if

```

### 3.4 章节

#### KMP复杂度计算

由于不清楚处到底循环了多少次，所以前缀函数的复杂度似乎不好分析。不过，因为K总是 $\geq 0$ 的，所以“操作 $K = \pi[K]$ 造成的K值减少量”一定不会比“操作 $K = \pi[i-1]$ 和 $\pi[i] = K + 1$ 造成的K值增加量”多，因而处循环总次数 $\leq m$ 。

主算法复杂度的证明同上，亦为线性，所以KMP算法的时间复杂度同空间复杂度一样，均为 $O(n+m)$ 。

### 3.5 章节

#### 定义六：

函数 $B_i = |T \text{与} S(i,n) \text{的最长公共前缀}|$ ，这里 $1 \leq i \leq n$ 。

#### KMP推广算法

普通KMP算法只是在判断是否存在整数满足 $B_i = m$ ，但如果希望求出所有 $B_1 \rightarrow B_n$ 的函数值并保持复杂度不变，又该怎么办呢？很自然的，我们继承经典KMP的核心思想但尝试对其进行改造以得到推广算法。

### 定义七：

函数 $A_i = |T \text{与} T(i, m) \text{的最长公共前缀}|$ ，这里 $2 \leq i \leq m$ 。

推广算法将借助函数A求出函数B。

### 函数A的求解

函数A的求解类似于数学归纳法。

- 1、首先通过逐个的比较字符求出 $A_2$ ，同时令 $k = 2$ 。
- 2、假设已经求出函数 $A_2, A_3 \dots A_{i-1}$ ，并且知道 $k$ 满足 $1 < k < i$ ， $k + A_k$ 最大。现在我们希望借助 $A_2 \rightarrow A_{i-1}$ 求出函数 $A_i$ 。
- 3、令 $Len = k + A_k - 1$ ， $L = A_{i-k+1}$ ，然后分类讨论：
  - a) 如果 $Len \geq i$ 并且 $L < Len - i + 1$ ，那么 $A_i = L$ 。  
理由：因为子串 $T(i, k + A_k - 1) = \text{子串} T(i - k + 1, A_k)$ ，所以字符 $T_{i+L} = T_{i-k+1+L} \neq T_L$ ， $A_i$ 延伸到长度 $L$ 时恰好不能匹配。
  - b) 如果 $Len \geq i$ 并且 $L \geq Len - i + 1$ ，那么 $A_i \geq L$ 。  
理由同上。  
此时，我们先令 $A_i = L$ ，然后通过逐个的比较字符将 $A_i$ 延伸到它应有的函数值大小，最后赋值 $k = i$ 。
  - c) 如果 $Len < i$ ，那么我们无法通过 $A_2 \rightarrow A_{i-1}$ 得到任何关于 $A_i$ 的信息。  
此时，我们先令 $A_i = 0$ ，然后通过逐个的比较字符将 $A_i$ 延伸到它应有的函数值大小，最后赋值 $k = i$ 。

### 复杂度分析

无论哪种情况， $k + A_k$ 的值都只增不减——这就是KMP算法的核心思想——所以逐个比较字符的总次数是 $O(m)$ 的，求函数A的复杂度为 $O(m)$ ，过程如下：

```

j = 0
While 字符  $T_{1+j}$  = 字符  $T_{2+j}$  do
    j = j + 1
End While
 $A_2 = j$ ,  $k = 2$ 
For i = 3  $\rightarrow$  m do
     $Len = k + A_k - 1$ ,  $L = A_{i-k+1}$ 
    If  $L < Len - i + 1$  then
         $A_i = L$ 
    Else

```

### 函数B的求解

函数B的求解和函数A的求解几乎一模一样，因而我就不再赘言而只给出求解过程的伪代码以供参考：

```
j = 0
While 字符  $T_{1+j}$  = 字符  $S_{1+j}$  do
    j = j + 1
End While
 $B_1 = j, k = 1$ 
For i = 2  $\rightarrow$  n do
    Len = k +  $B_k - 1$ , L =  $A_{i-k+1}$ 
    If L < Len - i + 1 then
         $B_i = L$ 
    Else
        j = Max(0, Len - i + 1)
        While 字符  $T_{1+j}$  = 字符  $S_{i+j}$  do
            j = j + 1
        End While
         $B_i = i, k = i$ 
```

函数B有着和函数A一样的复杂度分析，所以求解函数B的复杂度是 $O(n)$ 的。

综上，我们彻底阐述完了KMP推广算法的整个求解过程，推广算法的复杂度和普通KMP一样均是 $O(n+m)$ 的。



## 四、主算法

### 4.1 章节

#### 最优子串的定义：

对于子串  $S = W(u,v)$  和正整数  $L$ ，如果满足：

- 1)  $S$  是循环周期为  $L$  的重复子串
- 2)  $S$  不能向左扩展，即  $u = 1$  或者  $W(u-1,v)$  不满足条件1)
- 3)  $S$  不能向右扩展，即  $v = n$  或者  $W(u,v+1)$  不满足条件1)

那么称  $S$  是一个周期为  $L$  的**最优子串**。

#### 举例三：

例如当  $W = \text{"ABAABABAABAABA"}$  时，“ABABA”是周期为2的最优子串，而“ABAB”或者“BABA”则不是。“ABAABABAABA”是周期为5的最优子串，而“ABAABABAAB”或者“BAABABAABA”则不是。需要说明的是：即便周期相同，甚至部分重叠，两个不同子串也可能同时是最优子串。例如前缀“ABAABA”和后缀“ABAABAABA”就都是周期为3的最优子串，尽管它们有重叠部分  $W(4,3)$  是。

#### 问题的转化：

如果我们能够求出**所有最优子串连同它们的周期**，那么，从中找出周期最大的那个最优子串，最大重复子串的问题便迎刃而解。

怎样才能找出最优子串  $S$  呢？算法的主要思想是：

- 1、找到一个长度为  $L$  的循环节  $D$
- 2、分别将  $D$  向左和向右扩展到不能扩展为止
- 3、判断扩展以后的  $D$  是否长度  $\geq 2L$ 。

如果是，便找到了一个周期为  $L$  的最优子串  $S$ 。

以下，4.2、4.3 章节将说明如何寻找循环节  $D$ ，4.4 章节将说明如何将  $D$  快速地向两边扩展，4.5 章节将进行总结。

### 4.2 章节

#### 字符串分解：

将  $W$  分解成  $W = U_1 + U_2 + U_3 + \cdots + U_m$  的形式，其中  $U_i$  定义如下：

$$P = U_1 + U_2 + \cdots + U_{i-1}$$

Q 是 W 除去 P 的剩余部分，即  $W = P + Q$

如果字母  $Q_1$  从未在范围  $|P|$  中出现过，

那么  $U_i = Q_1$  单个字母组成的字符串

否则  $U_i =$  范围  $i-1$  中出现过的 Q 的最长前缀

#### 举例四：

字符串  $W = \text{"ABAABABAABAAB"}$  应该被怎样划分呢？

第一步：显然  $U_1 = \text{"A"}$ ；

第二步： $P = \text{"A"}$ ， $Q = \text{"BAABABAABAAB"}$ ， $W_2$  未在范围  $|P|$  中出现过，所以  $U_2 = \text{"B"}$ ；

第三步： $P = \text{"AB"}$ ， $Q = \text{"AABABAABAAB"}$ ， $W_3$  在范围  $|P|$  中出现过，所以  $U_3 = PQ$  的最长公共前缀  $\text{"A"}$ ；

第四步： $P = \text{"ABA"}$ ， $Q = \text{"ABABAABAAB"}$ ， $W_4$  在范围  $|P|$  中出现过，所以  $U_4 = PQ$  的最长公共前缀  $\text{"AB"}$ ；

第五步： $P = \text{"ABAAB"}$ ， $Q = \text{"ABAABAAB"}$ ， $W_6$  在范围  $|P|$  中出现过，所以  $U_5 = PQ$  的最长公共前缀  $\text{"ABAAB"}$ ；

第六步： $P = \text{"ABAABABAAB"}$ ， $Q = \text{"AAB"}$ ， $W_{11}$  在范围  $|P|$  中出现过，所以  $U_6 = PQ$  的最长公共前缀  $\text{"AAB"}$ ；

最终，W 被划分成为  $|A|B|A|AB|ABAAB|AAB|$ 。

你大概已经知道，字符串分解是利用后缀树算法实现的，因为  $U_i$  要么是单个的字符，要么就等于  $\text{Head}_{|P|+1}$ 。至此，我们终于可以诠释后缀树的作用。但是问题又来了，你不禁想问字符串分解的意义何在？

#### 4.3 章节

任何一个最优子串 S 的结束位置  $\text{End}(S)$  一定在某个片段  $U_i$  之内。如果我们暂且假设  $i$  是个已知常量，那么  $|S|$  和周期  $L$  是否都有一定的范围限制？为回答这些问题，我们需要对 S 的存在形式进行前后 4 层的分类讨论。

#### 第1层：

$\text{End}(S)$  在  $U_i$  内， $\text{Ini}(S)$  又在何处？

情况 1)  $\text{Ini}(S) < \text{Ini}(U_i)$

情况 2)  $\text{Ini}(S) \geq \text{Ini}(U_i)$

由于  $U_i$  是范围 P 内出现过的 Q 的最长前缀，所以被  $U_i$  包含的子串 S 一定在范围 P 内出现过。情况 2) 实际可以被情况 1) 包含。为避免重复劳动，情况 2) 将被忽略。

**定义七：**

我们把  $S(|S|-L+1, |S|)$  称为  $S$  的**最末循环节**，用符号  $R$  表示。

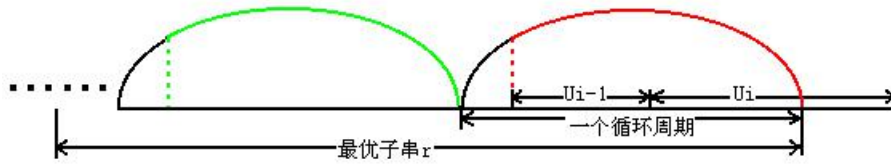
**第2层：**

既然  $\text{Ini}(S) < \text{Ini}(U_i)$ ，那么  $\text{Ini}(R)$  又应当在何处呢？

情况甲)  $\text{Ini}(R) > \text{Ini}(U_{i-1})$

情况乙)  $\text{Ini}(R) \leq \text{Ini}(U_{i-1})$

请看下图，图中一段弧线表示一个循环节：



显然，红色和绿色弧线标示了相同的子串，所以根据字符串分解的定义  $U_{i-1}$  应当等于红色弧线或者长度更长的子串。但事实上， $|U_{i-1}| < \text{红色弧线的水平长度}$ 。矛盾！**情况甲不会出现。**

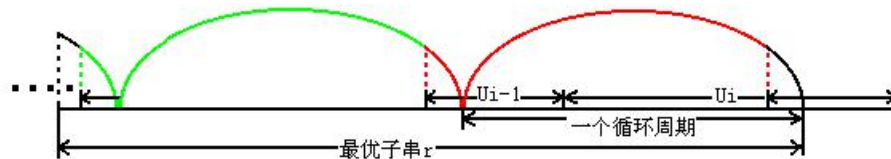
**第3层：**

虽然已经证明  $\text{Ini}(S) < \text{Ini}(U_i)$ ，不过我们还想进一步知道有关  $\text{Ini}(S)$  的信息。

情况 A)  $\text{Ini}(S) - \text{Ini}(U_{i-1}) < |U_{i-1} + U_i|$

情况 B)  $\text{Ini}(S) - \text{Ini}(U_{i-1}) \geq |U_{i-1} + U_i|$

再请看下图，图中所示为情况 B)：



因为周期  $L < |U_{i-1} + U_i|$ ，所以  $\text{Ini}(S) - \text{Ini}(U_{i-1}) > L$ 。同样道理，红色和绿色弧线标示了相同的子串，根据字符串分解的定义  $U_{i-1}$  应当等于红色弧线或者长度更长的子串。矛盾！**情况 B) 不存在。**

讨论完毕，综上所述，我们得出一个**重要结论**：

如果令  $V = \text{长度为 } |U_i| + 2|U_{i-1}| \text{ 的 } P \text{ 后缀，}$

$U = U_i$

那么，最优子串  $S$  的开始位置在  $V$  内，结束位置在  $U$  内。

至此，我们终于明白：字符串分解的重大意义在于**严格限制了最优子串的存在形式**。

#### 4.4章节

讨论了这么多，我们还是没有把S的循环节给找出来。其实，只要再进一小步分类，循环节就出来了。

#### 第4层：

因为 $\text{Ini}(S) < \text{Ini}(U_i) \leq \text{End}(S)$ 并且 $|S| \geq 2L$ ，所以以下两种情况S必然至少居其一：

情况i) S在V中的长度 $\geq L$

情况ii) S在U中的长度 $\geq L$

情况i) 和情况ii) 类似，为简便表达，我们仅以情况ii) 为例进行说明。易知， **$U(1,L)$ 就是S的一个循环节**。

上述 4 层分类都基于**处**的假设，现在把假设去掉，并整理一下得：

```

Answer = 0
For i = 2 → m do
    P = U1 + U2 + ... + Ui-1
    V = 长度为 |Ui| + |Ui-1| * 2 的 P 的后缀
    U = Ui
    For L = 1 → |U| do
        确定 U(1,L)为循环节 R
        ➡ 将 R 向左向右扩展到不能扩展为止
        判断扩展以后的R是否 |R| ≥ 2L ,
            如果是，那么 Answer = Max(Answer,R(1,2L))
    End for

```

枚举 L 的复杂度是  $O(|U|)$ ，而 U 的总长度又是  $O(n)$  的，所以我们必须在  **$O(1)$ 时间内完成**  
**处**的工作，才能保证算法总复杂度是线性的。

#### 4.5章节

#### 定义辅助函数：

$Lp_L = U$  与  $U(L+1, |U|)$  的最长公共前缀

$LS_L = V$  与  $V + U(1,L)$  的最长公共后缀

所谓将循环节向两边扩展，无非就是求得函数  $L_p$  和  $L_s$ 。如果单独求解某个  $L_{pL}$  的值是不能保证复杂度是  $O(1)$  的，但局部策略失败，不妨从整体入手，我们通过一次 KMP 推广算法求出所有  $L_{p1} \rightarrow L_{p|U|}$  的函数值，从而使得平摊复杂度为  $O(1)$ 。

#### 4.6 章节

### 主算法性能分析

其实主算法的复杂度已经在刚才几小节中陆陆续续的提到并证明了，现在我们将做的工作只是把这些证明汇总重申一遍。

1、首先是字符串划分。这步工作需要借助辅助结构后缀树，由于后缀树时间空间复杂度均是线性的，所以该步工作也是线性的。

2、然后是辅助函数。 $L_p$  和  $L_s$  的求解借助 KMP 推广算法，对于特定的  $V$  和  $U$ ，该步复杂度为  $O(|V+U|)$ ，因而总体复杂度为  $O(L)$ 。

3、最后是找出最优重复子串  $S$ 。 $S$  分为完整周期在  $v$  和完整周期在  $u$  两种情况，对于每种情况， $S$  可以根据周期  $j$  唯一地确定，由于周期  $j$  的范围是  $O(|V+U|)$  的，所以  $S$  的求解也是线性  $O(|V+U|)$  的。

### [小结]

后缀树和 KMP 尽管巧妙但卑之无甚高论，寻找最大重复子串算法本身也并没有多少可以广泛借鉴的价值。但是，如何将所学算法融会贯通、综合使用，却值得所有人认真思考。

### 参考文献

- [1]算法与数据结构（第二版） 傅清祥 王晓东 编著
- [2]The Art of Computer Programming Donald E.Knuth 著
- [3]Handbook of Computer Science and Engineering
- [4]Finding Maximal Repetition Roman Kolpakov 著
- [5]Discrete Applied Mathematics 1989
- [6]jsOI 内部资料

# 结果提交类问题

重庆外国语学校 雷环中

## 【关键词】

结果提交 数据分析 随机化 策略

## 【摘要】

结果提交类问题的历史非常短，但其崭新的模式、效率观、时空观，以及独特的解题技巧、解题策略，和对各种方法的鼓励，对选手更加深入、更加全面的考察而很快受到人们的青睐，在短期内快速的传播并发展，以至成为当今信息学奥林匹克中的一种重要题型。本文就结果提交类问题的两种重要技巧——数据分析和随机化展开讨论，提出此类问题所需要的不同策略。用一句话概括本文的核心，即：在最短的时间内得到正确结果，而不是过程。

## 【目录】

### 一、引言

### 二、数据分析

### 三、随机化

### 四、结果提交类问题的策略

1. 不同的效率观和策略观
2. 不同的时空复杂观
3. 手工运算与程序运算的协调策略

## 【正文】

### 一、引言

结果提交类问题可算是当今信息学奥林匹克竞赛中最年轻的一类问题，然而 IOI 连续两年出现此类问题的事实，使其成为主流竞赛中不争的必考题。其中最主要的原因，还是由于此类问题灵活新颖，以及命题人对殊途同归的倡导和对新奇方法的鼓励。

纵观在近两年来主流竞赛中出现的 Crack the code，Double Crypt，Birthday party，Tetris，Xor 等五道题目，可以说道道精彩，其中不仅充满了巧妙的构思，解题的方法也是百花齐放。但如何才能正确、迅速地完成任务，还需要我们深入地思考。本文以这五道题目为例，分析结果提交类问题的一些特点，介绍一些针对结果提交类问题的特殊方法和技巧。

伴随着结果提交问题的出现，很多方面都发生了深刻的变革：

#### 1. 考察内容的变革

在信息学奥林匹克竞赛发展已较为成熟的今天，仅仅就选手算法和数据结构方面能力作为主要考核标准是远远不够的。结果提交类问题的出现，使得考察的核心还涉及了一些更深层次的东西，如选手的思维能力、创造能力，全局策略、时间策略等等。这些方面的考察在以往题目中同样存在，但似乎都没有结构提交类问题体现得这样充分，考察得如此精妙。

#### 2. 区分度的变革

一道题目对选手的区分度发生了深刻的变革。我们不能仅仅依靠对算法时空复杂度的分析来衡量算法的优劣，还应该综合考虑，衡量我们完成输出文件所花的时间。原因很简单，在以往，1 秒钟出解的程序和 10 秒钟出解的程序，

测试的结果一般是前者得分要高不少；而结果提交类问题如果如果结果均正确，一名选手花 20 分钟，另一名花时 2 小时，前者则明显胜出，不论二人使用何种方法。

### 3. 题目的变革

我们后面可以看到，如 Birthday Party 这样的题目，是否结果提交，完全决定了我们的思考方式，决定了算法。既然命题者将一道题目以结果提交的形式出现，就说明并不希望选手将其视为一道普通题目，而是希望选手大胆使用各种方法，以在最快时间内得到正确结果。倘我们仍以老的思路来看待此类问题，不仅违背了命题者的意愿，浪费了所提供的输入文件，更重要的，我们将浪费更多的时间，甚至得不到理想的方法。结果提交，不是让我们放弃对完美算法的追求，而是鼓励我们站在综合效益的角度去得到结果。

### 4. 评卷方式的变革

大家都清楚，信息学奥林匹克的评卷是黑箱的，然而以前的黑箱似乎显得相对较狭义。自从结果提交类问题出现以来，黑箱的涵义得到了进一步拓展，更突出了对结果的唯一追求。过程，或者说是程序，更加显得无足轻重了。

我们用一句话来概括解决结果提交类问题的原则就是：永远提醒自己，你所需要的只是在最短的时间内得到正确结果，而不是过程。下面的讨论，也都是围绕此原则展开的。

## 二、数据分析



数据分析的方法始于结果提交类问题，由于这种方法对选手的观察、思考、猜想及动手实践能力的要求较其他的方法更高，而且灵活多变，所以备受重视。下面以 Crack the code 和 Birthday party 为例进行说明。

[例一]Crack the code(Baltic OI 2001)

题目大意：

有五个文件  $cra^*.out$ ，与其对应的，有  $cra^*.txt$ ，已知每一对文件出自同一篇文章。对于每个  $.out$  文件，有十个未知的 Byte 型数  $a_1 \dots a_{10}$ ，我们对文件的第  $i$  个字符  $c_i$  进行下述变换：

1. 如果  $c_i$  不是英文字母，则不做变化；
2. 如果  $c_i$  为小写字母，则变为大写，转 3；
3. 如果  $c_i$  为大写字母，则对其连续进行  $a_{(i-1) \bmod 10 + 1}$  次取后继操作。我们视 'A' 的后继为 'B'，'B' 的后继为 'C'..... 'Z' 的后继为 'A'。

然后将  $c_i$  依次写入对应的  $cra.in$  文件。你的任务是根据提供的  $cra^*.in$  和  $cra^*.txt$ (都在 11KB 以内)，得出  $cra^*.out$ 。

分析：

这道题是没有对于任何数据都非常有效的算法的，因为它毕竟是一种比较成形的加密算法。这也是命题者将此题作为结果提交问题的根本原因，只需你要破解给出的数据即可。

本题的解决方法很多，后面我们将要提到。还是先让我们试着用数据分析的方法来考虑此题，来看看数据分析方法的威力。

首先我们注意到本题数据只有 5 组，这和近年来 10 组甚至更多的测试数据不大一致，或许是在暗示这道题可以手算，换句话说，就为数据分析的方法提供了可能性。

注意到题目中的五篇文章都是英文的。我们清楚，英语中的许多高频词汇，如 a, I, the, he, are, of, in, after 等都是很短的，同时，单词长度越短，可能的情况就越少，如果长度为 1，我们就基本可以肯定是 a 或 I 了。

问题的关键是求出  $a_1 \dots a_{10}$ ，之后，cra.out 自然也就很容易得到了。而  $a_1 \dots a_{10}$  反映在微观上，就是原文和加密后的文章中每个单词（当然更是某些单词）的对应字母之间的差异。也就是说，我们只要确定了少数原单词和加密后单词的一一对应关系， $a_1 \dots a_{10}$  也就浮出水面了。比如  $A_1 A_2 \dots A_k$  是原单词， $B_1 B_2 \dots B_k$  是加密后的单词， $A_i$  是文章的第  $m$  个字母，则我们可以得到

$A_j \xrightarrow{\text{进行 } a_{(m+j-2) \bmod 10 + 1} \text{ 次取后继操作}} B_j$ ，其中  $j=1, 2, \dots, k$ 。反之，自然有  $B_j \xrightarrow{\text{进行 } a_{(m+j-2) \bmod 10 + 1} \text{ 次取前趋操作}} A_j$ 。

基于上述思路，我们完全可以仅用猜想+试探少数单词间对应关系的方法处理本题，当然还需要编两三个十余行的辅助小程序来提高效率。这种方法全靠观察能力和猜想能力，不涉及算法的复杂度。

下面仅以数据一和数据五为例进行简单说明。

数据一：

cra1.txt：

ALFRED TARSKI WAS ONE OF THE GREATEST MATHEMATICIANS, LOGICIANS  
AND PHILOSOPHERS OF THE PASSING CENTURY, AND ONE OF THE GREATEST LOGICIANS  
OF ALL TIME. 后略。

cra1.in

JK XHLMGX C PP ZEU OWL BM 1939 VP ZEU CRGBSHVOLD IJ UHGU XUK DYYXTM HPJ  
IYPIO MGLTK BLYV DBMJG, GCVCPPTSLF LFHMX LM SOG NXHPECW TUKBBHMMER ZUF ZEUE,  
EQMDY 1943, CZ QXY YYBULTYFJS SQ VZSKLLHHML TS IGXHUFIJ. 后略。

对于输入文件，稍加观察我们便可以发现，位于第三行的 EQMDY 1943 是突破口。英语常识告诉我们，EQMDY 只能是 AFTER！同时，EQMDY 的'E' 是文章的第 116 个字母，同时有  $AFTER \xrightarrow{(4,11,19,25,7)} EQMDY$ ，这样，10 个数就被破了 5 个，具体地说，就是  $a_6 = 4$ ， $a_7 = 11$ ， $a_8 = 19$ ， $a_9 = 25$ ，

$a_{10} = 7$ 。我们用得到的这 5 个数还原文本之后，就可以得到：

JK XHLIVED IP ZEU OSA IN 1939 OP ZEU CNVITAVOLD IF JOHN XUK DYUMAN APJ  
IYPED THETK BLYR SINCG, GCVCLIAEF LFHMT AT THG NXHPARD UNKBBHMITY ANF ZEUE,  
AFTER 1943, CZ QXY UNIVETYFJS OF CALKLLHHIA AT BGXHUFY. 后略。

很多单词都已露出了“马脚”，选择一两个较长的单词，即可轻松迅速地破解全文。如第二行的 THG NXHPARD UNKBBHMITY，我们不用想就知道应该是 THE HARVARD UNIVERSITY；还有第三行的 UNIVETYFJS OF CALKLLHHIA，我们也很容易看出是 UNIVERSITY OF CALIFORNIA，因为 cra1.txt 告诉我们这段文字是在描述一个人的生平。之后再做一次转换，得到  $a_1, a_2, a_3, a_4, a_5$ ，于是就可破解全文了。

数据五：

cra5.txt：

THE PLURAL IS USED HERE BECAUSE THE TERM "HAHN-BANACH THEOREM"  
IS CUSTOMARILY APPLIED TO SEVERAL CLOSELY RELATED RESULTS.

cra5.in

YSOEGOCYTV: OVN ZNGU IUIXS XB T ZXFTTJURYTR EUHBJF BLTIN N

NA OVN RXICEW AKOLA Q\* CQEXM ZZNIXTCI FZZ HQA VUWJNVPCDO  
EOWUFZ AIWYMOXDFTN CW T.

GUCU YPVH JZWOCYTV VBM OVGUQW UPZCEIRRSFBDWC WKK MUKQISM EG D\* KO (...)  
NB DG LHXGA JMIO HQALK XFJZVHRKGY ME NVYSNZ FGTU C\* QIHX W OKLJTZ NDJYX.

对于官方的参考程序而言，数据五是最难的，但对于我们数据分析的方法而言，却是最简单的，我们甚至可在一分钟内将它破解。

根据 cra5.txt 文件，我们可以大致判断这是一篇理科方面的文章，从英语语法的角度我们知道，cra5.in 开始的 OVN 可能是 the。“YSOEGOCYTV:”似乎是引出什么东西的定义。YSOEGOCYTV 共有 10 个字母，我们可以立即想到或许是 DEFINITION 这个单词。当然这不一定是对的，但试验之后，我们发现，文章到此已被完全破解：

DEFINITION: THE DUAL SPACE OF A TOPOLOGICAL VECTOR SPACE  $X$   
IS THE VECTOR SPACE  $X^*$  WHOSE ELEMENTS ARE THE CONTINUOUS  
LINEAR FUNCTIONALS ON  $X$ .  
  
NOTE THAT ADDITION AND SCALAR MULTIPLICATION ARE DEFINED IN  $X^*$  BY (...)  
IT IS CLEAR THAT THESE OPERATIONS DO INDEED MAKE  $X^*$  INTO A VECTOR SPACE.

正如我们所看到的，人脑在此处起了决定性的作用。如果说以前的题目是透过程序来对选手进行考察的话，那这种题目就是直接与选手面对面的对话。

下面我们来考虑如何通过程序来完成该问题。

由于加密文本是采用英语这种仅由 26 个字母组成的语言，而且理所当然，每种字母在一定文段中出现的频率是不相同的。如‘S’出现的频率一般都比‘V’高得多(我们统一使用大写字母)。基于此思想，我们可以设计一种频率比较的方法。

用  $(P_A, P_B, P_C, \dots, P_Z)$  来表示给出的未加密文本 `cra.txt` 中每个字母出现的相对频率，也就是  $P_{ch} = \frac{\text{文本中字母 } ch \text{ 出现的次数}}{\text{文本中字母的总数}} (ch = 'A', 'B', \dots, 'Z')$

我们分别计算  $a_1 \dots a_{10}$ 。对于  $a_i (i = 1, 2, \dots, 10)$ ，先将位于加密文件中第  $i, i+10, i+20, \dots$  位的字母做成集合  $S$ ，然后对于  $S$  做类似的统计：我们用

$(Q_A, Q_B, Q_C, \dots, Q_Z)$  来表示  $S$  中每个字母出现的相对频率，即

$$Q_{ch} = \frac{S \text{ 中字母 } ch \text{ 的数目}}{S \text{ 中字母的总数}} (ch = 'A', 'B', \dots, 'Z')。$$

之后从 0 到 25 枚举  $a_i$ ，同时作变换

$$Q_{ch} \xrightarrow{\text{字母 } ch \text{ 取 } a_i \text{ 次前趋得到字母 } ch'} H_{ch'}, \text{ 最后比较 } P \text{ 和 } H \text{ 的相似程度, 取最相似时的值作为真正的 } a_i。$$

令  $f(P, H)$  表示  $P$  和  $H$  的相似程度，于是问题的关键就转化为如何设计这个估价函数。一方面为了更好地反映各种函数之间的差异，另一方面由于  $a_1 \dots a_{10}$  间没有必然的联系，所以我们后面按照密码的个数来说明，一个测试数据有 10 个密码。

$$\text{方法一：最自然的一种，令 } f(P, H) = \sum_{ch='A'}^{Z'} |P_{ch} - H_{ch}|；$$

方法二：上面一种似乎不能很好地反映其相似程度，特别是在相差很微妙

$$\text{的时候。于是我们可以稍加改进：令 } f(P, H) = \sum_{ch='A'}^{Z'} (P_{ch} - H_{ch})^2；$$

方法三：第二种方法却又显得将权重夸大了，我们可以进一步改进：令

$$f(P, H) = \sum_{ch='A'}^{Z'} \frac{(P_{ch} - H_{ch})^2}{P_{ch}}。 \text{ 注意到这里的 } P_{ch} \text{ 不能太小(比如不能小于}$$

$$0.001), \text{ 于是我们将函数修正为 } f(P, H) = \sum_{ch='A'}^{Z'} \frac{(P_{ch} - H_{ch})^2}{P_{ch} + \text{Delta}}, \text{ 其中 Delta 等于}$$

最小的一个非零  $P_{ch}$ 。

下面是三种方法的比较，表中的数字为每个数据算对的密码个数：

	数据一	数据二	数据三	数据四	数据五
方法一	10	10	10	10	8
方法二	10	10	10	10	8
方法三	10	10	10	10	10

我们发现，随着输入文件的长度的缩短，样本容量越来越小，这是统计方法的大忌，前面两种方法便在数据五这种很小的样本容量面前失败了。

为了更好的区分程序的性能，我们另外设计四个数据，这四个数据都是相当短的，难度比前面五个大得多。

cra6.txt

It passes all vertical lines in increasing order of their x -values and keeps a list of active y -intervals. At each "stop", i.e. for every vertical line, it does the following:

cra6.in

NWOXNVSG CRF Z -GMXZHVLO UQ WLJ VYMESPWV WYUW

ERDI VKI XOGM XP UJH MSZLZEKM VKEY OZ BQO VPLSS UM IUV BEWMAK F -QWDFTYEQY,

cra6.out

MULTIPLY THE X -DISTANCE TO THE PREVIOUS STOP

WITH THE SIZE OF THE INTERVAL THAT IS THE UNION OF ALL ACTIVE Y -INTERVALS,

cra7.txt

Scan-line solution, (modification of previous solution with added y-axis compression).

cra7.in

TEDR-QOUM BYMWWMTT DQCR MCCC NSWTNWFPWEYOVV. CRF UFES RPNV ST CUVFE

VDNB SCQKJ 0... 30000 .

cra7.out

SCAN-LINE SOLUTION WITH LAZY IMPLEMENTATION. THE SCAN LINE IS ARRAY OVER RANGE 0... 30000 .

cra8.txt

My name is Mary.

cra8.in

J CP E HRLDNB HKUP. N GT VXD CCG.

cra8.out

I AM A CLEVER GIRL. I AM NOT BAD.

cra9.txt

All tests were generated automatically.

cra9.in

NQVX FXL ZJXEQP.

cra9.out

MOST ARE RANDOM.

下面是比较的结果。这次我们将官方程序及数据分析法一起加入比较。

	一	二	三	四	五	六	七	八	九	总计(上限 90)
方法一	10	10	10	10	8	8	5	1	1	63
方法二	10	10	10	10	8	8	3	1	1	61
方法三	10	10	10	10	10	7	5	2	1	65
官方程序	10	10	10	10	10	6	4	0	1	61

我们可以看到，由于问题本身的特殊性，没有一种方法可以给出非常理想的解答。其中四种程序实现的方法差异不大，随着样本容量的减小，正确率显著降低，直到数据八九的时候，可以说已经完全无能为力了。而数据分析方法一直表现良好，除非如数据九，文件内容简单得只有“NQVX FXL ZJXEQP.”，已经无从逻辑分析了，只有枚举。

另外值得一提的是，如方法一在数据五六的时候，得出的结果已经“比较”正确，这时候再结合数据分析的方法，就轻而易举了。这也说明，数据分析进行检查也是前面四种程序方法结束后的必经之路。

类似本题这样手工处理数据的方法在结果提交类题目中是非常普遍的，甚至可以说，用这种方法还能在相对更短的时间内得到正确答案(如本题)。既然如此，在比赛中又何乐而不为之呢？在不忽视“正统”方法的同时，我们应该针对题目的特点，想出更好的算法。

处理本问题的方法难以尽数，各种方法的优劣也难以仔细分辨，当然也包括数据分析方法。我想这也是命题人将本题作为结果提交类问题出现的原因之一。

	结果正确性	完成题目所需时间	易于实现性	对于加大测试数据数目的适应性
数据分析	好	短	很易	不好
程序方法	较差	较长	一般	好

下面我们将这种数据分析的方法和程序方法进行比较：

结果是一目了然的。仅对于本题结果提交的特点而言，数据分析方法相对而言是最好的，但是如果测试数据是二十个，那数据分析就显得太慢了。字母频率统计方法对于本题显得有些杀鸡用牛刀，而且结果的正确性不能得到很好的保证。但不论何种方法，都是需要人工+程序的，只是程度不同。

综上所述，不同方法的适应性是不同的，但在结果提交问题中，快速准确出解，才是最重要的。

## [例二] Birthday party(CEOI 2002)

题目大意：

有  $N$  个人，另给出  $M$  条“要求”。

要求的定义如下：

1.  $name$  是一个要求。 $name$  是一个长度不超过20的字符串，表示一个人的名字。这个要求当且仅当  $name$  被选中时满足。
2.  $-name$  是一个要求。这个要求当且仅当  $name$  不被选中时满足。



3. 如果  $R_1, \dots, R_k$  都是要求, 那么  $(R_1 \& \dots \& R_k)$  也是一个要求。这个要求满足当且仅当  $R_1, \dots, R_k$  都被满足。

4. 如果  $R_1, \dots, R_k$  都是要求, 那么  $(R_1 | \dots | R_k)$  也是一个要求。这个要求满足当且仅当  $R_1, \dots, R_k$  中至少有一个被满足。

5. 如果  $R_1, R_2$  是要求, 那么  $(R_1 \Rightarrow R_2)$  也是一个要求。这个要求不满足当且仅当  $R_1$  满足而  $R_2$  不满足。

提供给你 10 个输入文件 party\*.in, 每个输入文件包含  $N$  个人的名字和  $M$  个要求。对于每个输入文件, 分别找出应选中哪些人, 使得  $M$  要求全都被满足。(至少有一个解, 你也只需要输出一个解)

从输入文件我们可以看到,  $N \leq 25000, M \leq 129998$ 。

分析：

看完题目, 我们仅从数据规模就可以感觉到这道题应该是**线性规模或接近线性规模**的, 但此题的模型 SAT 却早被证明是 NPC 的, 这种怀疑使我们有理由相信, 此题的关键不是题目, 而是数据。命题者对数据精妙的设计, 也使本题成为结果提交类题目中最经典的一道。

每一个要求都是一个逻辑表达式, 其中, 我们视“-”为 NOT, “|”为 OR, “&”为 AND, “ $\Rightarrow$ ”为 THEN, 将每个人名(即逻辑变量)赋为真或假, 那题目也就是要找出一种赋值方案, 使得所有的逻辑表达式值都为真。

下面我们来对数据进行一一分析：

```
Party1.in
5
adam
bill
```

```
cindy
don
eve
7
(adam | bill | cindy | don)
(-adam | -bill | -cindy | -don)
(-don | -eve)
(-bill | -bill | -bill)
(bill | adam)
(-adam | don)
(eve | cindy | bill)
```

这是最简单的数据，我们既可手算，又可通过更强大的程序完成。究竟选择哪一种，则是一个策略问题。如果后面有些更大的数据有着相同的性质，那我们应该采用程序完成。否则手算是最划算的，因为花一分钟动笔比设计一个上百行的算法要容易得多。

```
Party2.in
15
b
c
d
e
f
g
h
i
j
ba
bb
bc
bd
be
bf
100
(-g | -bb | -c)
(e | -be | -b)
(-i | -bd | be)
```

后略，形式同上面三行。

毫无疑问，枚举或者是任何更好的方法都可以对付这个数据，枚举的时间复杂度为 $O(2^N \times M)$ 。

通过观察，我们还可以发现，如果将  $a, b, c, \dots, j$  分别和  $0, 1, 2 \dots 9$  一一对应，那么这些名字就是从 1 开始的  $N$  个自然数！后面的数据也满足这个特征，这为我们的数据处理提供了很大方便。这一点虽然很容易看出，但以前在题目中是不容易设计这种对的观察能力的考察的。也即，结果提交类问题，不仅给选手更广阔的空间，也将同样广阔的空间给了命题者。

Party3.in

35

b

c

d

其他名字略，规律如上。

700

(ba | -cb | -j)

(-cb | ch | bb)

(-ba | j | e)

后略，形式同上面三行。

形式和数据二是一样的，不过规模要大不少，如按  $2^{35} \times 700$  估算，枚举算法已不能在较短时间内出解(事实上已远远超过 5 个小时)。不过一个很一般的基于“要求”的搜索算法就可以对付这个数据。进一步，我们在搜索中应该去掉那些到目前为止已经成立的要求，并且将那些除了某个变量为真该要求就不能成立的首先考察。

Party4.in

和数据二差不多，枚举或者是任何更好的方法都可以对付这个数据。不过

值得注意的是，有下面六条要求的形式有所不同：

```
((bg & i) => (-be | -b | ba | j)
((i | g | -bf | g | -b | -g) & (-bh | -b | -bd | -bb | bc | -ba))
(b => (-b => ((c | (e & f)) => ((g & -h) | (e | h))))))
((-bd & h & -be & bh & g) => h)
((-j & -bb & -bf) => (-d | -bb | -e))
(bd => (-bc => (g => (bb => (c => -be))))))
```

我们可以先用一般算法得出解，然后人工判断其是否符合这 6 个要求。要知道，仅为这 6 个要求而去编写大量代码是很不划算的。当然，这也是结果提交类问题所带来的好处。

Party5.in

$N=500$  ,  $M=5000$

要求的形式：

全部为( $N_1 \Rightarrow N_2$ ) (其中  $N_1$  ,  $N_2$  为 *name* 或 *-name* , 下同)

Party6.in

$N=1000$  ,  $M=10094$

要求的形式：

全部为( $N_1 \Rightarrow N_2$ )

Party7.in

$N=5000$  ,  $M=50000$

要求的形式：

全部为( $N_1 \Rightarrow N_2$ )

Party8.in

$N=10000$  ,  $M=55000$

要求的形式：

全部为 $(N_1 | N_2)$

Party9.in

$N=25000$  ,  $M = 129998$

要求的形式：

基本上为 $(N_1 | N_2)$ ，另有六行 $(N_1 \Rightarrow N_2)$

此外还有两行非常怪异的

$(eebi \Rightarrow (eebf \& -eead \& -eeaj))$   
 $(bfjfg | fchf | fchf)$

从数据四和数据九我们可以看出，如果不仔细观察或者是编写一个小程序来扫描，有些信息我们是很容易忽略的，这也是结果提交类问题的特点，以前完全程序处理，程序是不会疏忽的，但人就不同了。结果提交，带给我们的既是蛋糕，又是陷阱。

Party10.in

$N=1000$  ,  $M = 87543$

这可以说是最可怕的一个数据，形式没有任何规律！但仔细看一看  $M$  个要求的

尾部，我们发现：

$(e \Rightarrow -e)$   
 $(-f \Rightarrow f)$   
 $(-g \Rightarrow g)$   
.....  
 $(jji \Rightarrow -jji)$   
 $(jjj \Rightarrow -jjj)$   
 $(baaa \Rightarrow -baaa)$

似乎 1000 个名字都齐了，但唯独缺少  $b, c, d$ ！可我们再到回去看看最前面的

三行要求：

$(d \Rightarrow (-b \ \& \ -c))$   
 $(b \Rightarrow d)$   
 $((-b \Rightarrow -c) \ \& \ (-c \Rightarrow d))$

这些情况已经足以出解了，因为由 $(name \Rightarrow -name)$ 我们可以得到不选 $name$ 。再结合余下的三行，我们可以得到唯一的解。根据题目所说的至少有一个解，并且只需要输出一个解，那么剩下的八万余行就可以完全不管了。

综合考虑：

数据一：手算。因为后面数据的组织方式和这个数据完全不同。

数据二三四：统一用一个搜索算法完成。事实上，从比赛策略的角度而言，我们甚至可以舍弃数据三，编一个枚举算法，这样可省些时间。

数据五六七八九：形式全部为 $(N_1 \Rightarrow N_2)$ 或 $(N_1 | N_2)$ ，需要一个线性规模或接近线性规模的算法。

数据十：还是手算，不过这里的手算和数据一有点不同，实质上是手工+简单程序计算。

另外，对于数据四和数据九的几行特例，最好的方法是人工验证，简单高效。正所谓“具体问题具体分析”，我们根据数据各自的情况，将一个 NPC 的问题完全分解了。

至此，本题的核心已非常明显，即是数据五六七八九的 **2-SAT**。该命题和 POI 0106 Peaceful Commission，Baltic OI 2001 的备用题目 Excursion 相似。让我们先来看看这两道题目的模型：

## Peaceful Commission(POI VIII Stage 2 Problem 2)

给出  $2N$  个点( $1 \leq N \leq 8000$ )，将点  $i$  和  $-i$  ( $1 \leq i \leq N$ ) 作为一组，另外给出  $M$  ( $0 \leq M \leq 20000$ ) 对整数  $i, j$ ，表示点  $i$  与点  $j$  不相容 ( $-N \leq i, j \leq N, i \neq 0, j \neq 0, i \neq j$ )。求出一含  $N$  个点的集合  $S$ ，包括且仅包括每一组中的一个点，且  $E$  中的点均相容。

## Excursion(Baltic OI 2001 备用题)

给出  $2N$  个点( $1 \leq N \leq 8000$ )，将点  $i$  和  $-i$  ( $1 \leq i \leq N$ ) 作为一组，另外给出  $M$  ( $0 \leq M \leq 20000$ ) 个整数对  $(i, j)$  ( $-N \leq i, j \leq N, i \neq 0, j \neq 0$ )。求出一含  $N$  个点的集合  $S$ ，包括且仅包括每一组中的一个点，且对任意一个前面给出的整数对  $(i, j)$ ，点  $i$  和点  $j$  中至少有一个属于集合  $S$ 。

我们暂且不具体讨论这两道题，但我们发现，两道题的数据规模竟然都是一样的。事实上，后面我们可以证明，这两道题与本题都是本质相同的。

Party 的模型如下：

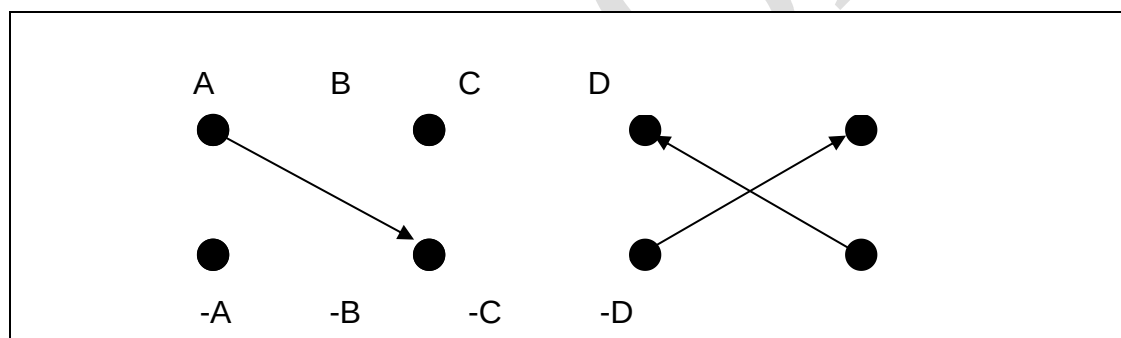
给出  $2N$  个点( $1 \leq N \leq 25000$ )，将点  $i$  和  $-i$  ( $1 \leq i \leq N$ ) 作为一组，另外给出  $M$  ( $0 \leq M \leq 129998$ ) 个要求，要求分两种，每个要求包括一个有序整数对  $(i, j)$  ( $-N \leq i, j \leq N, i \neq 0, j \neq 0$ )。求出一含  $N$  个点的集合  $S$ ，包括且仅包括每一组中的一个点，且满足所有要求。包括整数对  $(i, j)$  的要求被满足当且仅当：

1. 如果该要求是第一类要求，则如果点  $i \in S$ ，必有点  $j \in S$ 。
2. 如果该要求是第二类要求，则点  $i$  和点  $j$  中至少有一个属于集合  $S$ 。

我们先分析 Party：

首先，根据定义有 $\neg(\neg name)=name$ 。对于每个名字  $name$ ，我们建立顶点  $V_{name}$  和  $V_{\neg name}$  (由于名字和整数是一一对应的，后面我们将不再区分  $V_{name}$  和  $V_i$ )， $V_{name}$  表示要选， $V_{\neg name}$  表示不选。 $V_{name}$  和  $V_{\neg name}$  也分别代表一个布尔值，显然不能存在这样的  $V_{name}$  和  $V_{\neg name}$ ，他们的值都为真。同时我们定义符号“ $\rightarrow$ ”， $V_1 \rightarrow V_2$  表示如果  $V_1$  为真，则  $V_2$  为真，并且称  $V_2$  为  $V_1$  的后继。

于是我们很自然得到， $(N_1 \Rightarrow N_2)$  成立的条件是  $(V_{N_1} \rightarrow V_{N_2})$ ， $(N_1 | N_2)$  成立的条件是  $(V_{\neg N_1} \rightarrow V_{N_2})$  且  $(V_{\neg N_2} \rightarrow V_{N_1})$ ，这样，我们处理的任务中的所有要求(不包括几个特例)，都可以转化  $(V_1 \rightarrow V_2)$  这种形式，也就是在图中连一条从  $V_1$  到  $V_2$  的有向边。下图所示的为  $(A \Rightarrow \neg B)$  及  $(C | D)$  的情况：



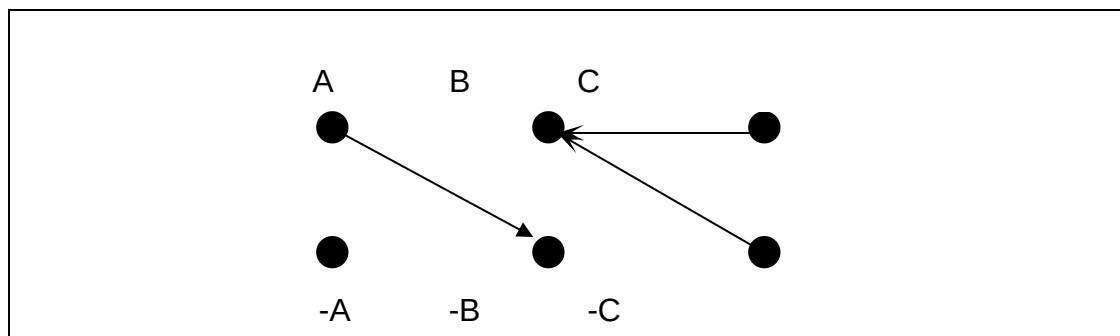
从定义  $V_1 \rightarrow V_2$  可以看到，一旦  $V_1$  为真，那么  $V_1$  的所有直接后继和间接后继都必须为真。这样，我们的目标就是找到一个顶点集合  $V$ ，满足：

1. 如果  $V_0 \in V$ ，则  $V_0$  的所有后继组成的集合  $V_{son} \subset V$ ；
2. 不存在这样的  $V_{name}$  和  $V_{\neg name}$  同属于  $V$ 。

对于每一个  $name$ ，我们可以先试探  $V_{name}$  为真，如果其间接后继中没有  $V_{\neg name}$ ，则说明  $V_{name}$  为真是可以满足题意的；否则我们再试探  $V_{\neg name}$  为真，如果其间接后继中没有  $V_{name}$ ，则说明  $V_{\neg name}$  为真是可以满足题意的；如果试探均失败则无解。事实上，题目已经明确指出至少是存在一个解的。



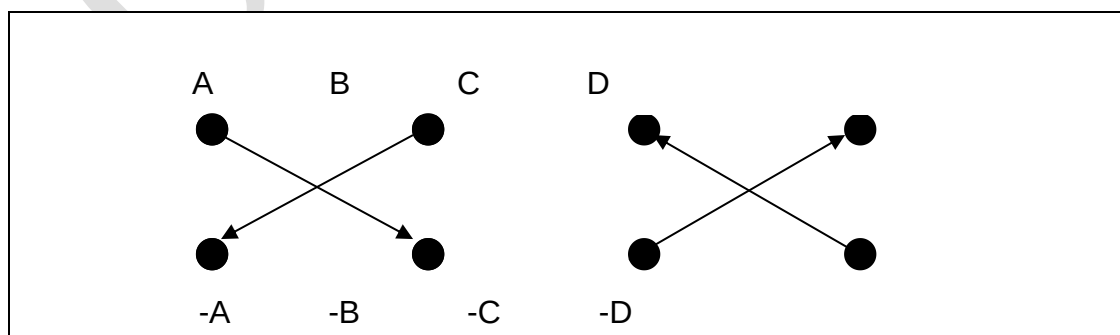
但我们可以清楚地看到，上述算法是**有后效性**的。我们完全可能在选中(注意这里不是试探性地选中，而是完成了试探的真正选中) $V_i$ 之后，当前又必须经由某个 $V_j$ 选中 $V_{-i}$ 。下图即是一个反例：



我们开始“顺利”地选中 A 与-B 以后，却在选 C 的时候面临无论如何都要选 B 的局面。这样，就造成了既选 B，又选-B 的错误局面。事实上，我们令  $S = \{-A, B, C\}$  或  $S = \{-A, B, -C\}$  都是满足题意的。这说明开始时我们很自然的构图方法是**不正确的**。

为了解决后效性问题，我们重新构图。

$(N_1 | N_2)$  成立的条件依然是  $(V_{-N_1} \rightarrow V_{N_2})$  且  $(V_{-N_2} \rightarrow V_{N_1})$ ，而  $(N_1 \Rightarrow N_2)$  成立的条件我们从  $(V_{N_1} \rightarrow V_{N_2})$  扩充为  $(V_{N_1} \rightarrow V_{N_2})$  且  $(V_{-N_2} \rightarrow V_{-N_1})$ 。于是前面的  $(A \Rightarrow -B)$  及  $(C | D)$  的情况变为：



后面的内容与前文一致。下面我们就来证明这种构图是**没有后效性**的：

根据前文所述,  $(N_1 \Rightarrow N_2)$  成立的条件是  $(V_{N_1} \rightarrow V_{N_2})$  且  $(V_{-N_2} \rightarrow V_{-N_1})$ ,  $(N_1 | N_2)$  成立的条件是  $(V_{-N_1} \rightarrow V_{N_2})$  且  $(V_{-N_2} \rightarrow V_{N_1})$ 。两者在本质上是相同的, 即  $(-N_1 \Rightarrow N_2)$  等同于  $(N_1 | N_2)$ 。也就是说, 我们可以将它们都视为  $(N_1 | N_2)$ 。

反设我们在选中(真正选中) $V_i$ 之后, 当前试探中必须经由 $V_j$ 选中 $V_{-i}$ , 这就表明存在 $(V_j \rightarrow V_{-i})$ 这条弧。根据构造的对偶性, 必然还存在 $(V_i \rightarrow V_{-j})$ 这条弧, 由于 $V_i$ 是已经肯定选中了的, 那么其后继 $V_{-j}$ 也必然选中, 于是当前既选中了 $V_j$ , 又选中了 $V_{-j}$ , 这是不可能的, 矛盾。所以问题不存在后效性。

于是 Party 就算解决了, 我们再回过头来看 Peaceful Commission 和 Excursion。

Peaceful Commission :

点  $i$  与点  $j$  不相容也就意味着必须满足  $(V_i \rightarrow V_{-j})$  且  $(V_j \rightarrow V_{-i})$ , 反推得到  $(V_{-i} | V_{-j})$ , 就是本题的  $(N_{-i} | N_{-j})$ 。于是 Peaceful Commission 与本题实质上完全一样。

Excursion :

前文已经说明其模型是要满足“对任意一个前面给出的整数对  $(i, j)$ , 点  $i$  和点  $j$  中至少有一个属于集合  $S$ ”, 这就已经是我们的  $(V_i | V_j)$ , 也就是本题的  $(N_i | N_j)$ 。于是 Excursion 与本题实质上完全一样。

下面我们来分析算法的复杂度。

空间复杂度: 线性, 可以忽略。

时间复杂度：枚举  $N$  个点，每次枚举最坏的情况遍历但不重复遍历  $M$  条弧，所以时间复杂度为  $O(M \times N)$ 。但如果确定了点  $i$  属于集合  $S$ ，那么  $V_i$  的所有后继就都必然属于集合  $S$ ，所以实际复杂度远小于  $O(M \times N)$ 。

需要指出的是，这三个问题的 2-SAT 模型是存在  $O(M)$  算法的，不过比较复杂，限于篇幅，此处从略。

Party 小结：

本题的核心内容 2-SAT 出得比较好，但本题的价值远不止此。它启示我们，当今的题目已不能单单从算法的角度来衡量了，对于本题，这样得到的结果无疑是 NPC 问题。但事实上这道题我们得到了解，而且是在相当短的时间内。这道题最完美的算法其实上是最笨的算法，最聪明的算法就是要求我们具体问题具体分析，在最短的时间内得到正确的结果。

本题目的精妙之处主要在于 10 个数据各具特点，“具体问题具体分析”，不要再以常规题目所有数据统一对付的思想来解决结果提交类问题，而是应该采用灵活多变的手法。对于一道题，我们可以使用各种各样的手段，但目标只有一个：快速得到正确结果！

此外，我们分析得到 Party 与 Peaceful Commission，Excursion 在本质上是完全一样的，在不到两年的时间里，同样的 2-SAT 模型屡次并且甚至变为结果提交的形式出现，这也需要引起我们的重视。

### 三、随机化

值得说明的是，这里的随机化和经典问题中的随机化有所不同。经典问题中的随机化需要证在规定时间内出解，且评测时只会运行一次。但结果提交类问题的随机化没有任何限制，目标只有一个：得到正确结果！

### [例三]Tetris(NOI 2002)

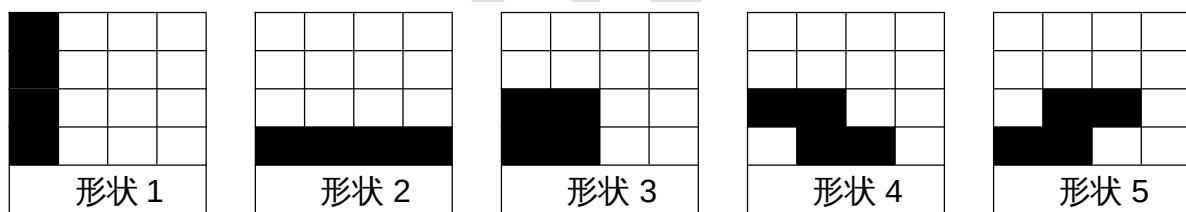
题目大意：

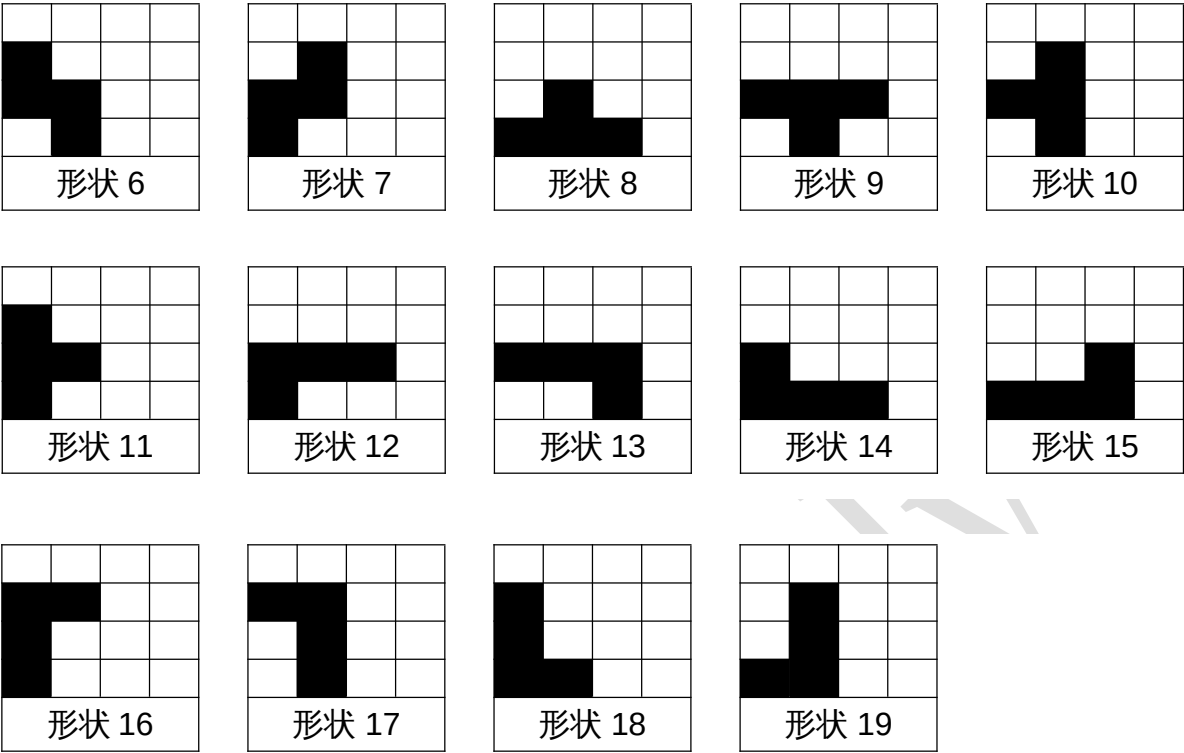
本题是关于俄罗斯方块的问题，只是本题所涉及的所有状态中，不能有悬空。

给出一个初始的棋盘状态，我们每次可以从标准的 19 种形状中任选一种，放到任意的位置上，只要不悬空，不超出边界即可。

你的任务是在 100000 步以内将棋盘消空。输入数据保证有解。

题目的测试数据中最大的棋盘状态有 1202 列。





分析：

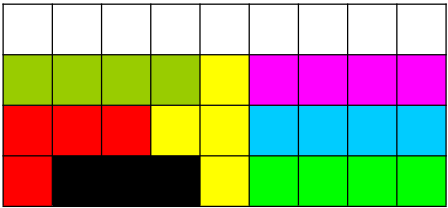
我们依然是先看测试数据。

Tetris1.in

9

0 1 1 1 0 0 0 0 0

如下即可解决，黑色为初始的格子。

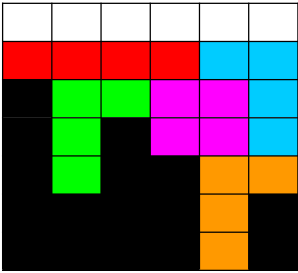


Tetris2.in

6

5 2 4 3 0 2

依然非常简单，我们一眼即可看出：



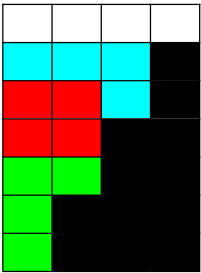
Tetris3.in

200

0 2 4 6 6 8 10 12 12 14 16 18 18 20 22 24 24 26 28 30 30 32 34 36 后略。

我们可以看到，数据以四列为单位有规律的出现。 $(0, 2, 4, 6)$ ， $(6, 8, 10, 12)$ ， $(12, 14, 16, 18)$ .....

对于每一个循环单位 $(a_i, b_i, c_i, d_i)(i=2, 3, \dots, 50)$ ，都是 $(a_{i-1} + 6, b_{i-1} + 6, c_{i-1} + 6, d_{i-1} + 6)$ ，而最基本的 $(0, 2, 4, 6)$ 是很容易解决的：



所以本数据只需要用一个小程序即可解决。

Tetris4.in

16

8 1 7 0 5 6 1 0 1 5 2 0 3 7 6 0

Tetris5.in~Tetris10.in

略

综合分析：

数据一、二：规模很小的数据，手工能在短时间内出解。

数据三： 规模很大，不过规律很明显，编一个很短的程序即可出解。

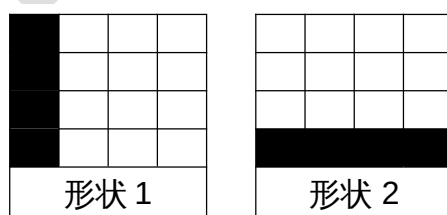
数据四、五：规模不是很小，也可以手工算，但需要一定时间，如手算的方法不好则很有可能算不出结果。

数据六...十：规模较大，规律很不明显或没有规律，手工已经完全无法胜任。

到这里，便有一个策略问题，人们总是习惯于在看到数据一、二、三等到手的分之后便匆忙动手，虽然得了 30 分，但浪费了很多时间。如果最后编出了理想的程序，那这里贪分就很不划算了。这也是命题者的意图所在，无形中考察了选手的比赛策略。

值得一提的是，对于数据 1, 2, 4, 5，我们甚至可以编一个模拟程序，这样可以极大提高手算速度。一方面，因为辅以简单程序，数据分析效率会大大提高；另一方面，依然是我们处理结果提交类问题的核心：不择手段，只求结果。

下面先谈一种可行的算法。玩儿过俄罗斯方块的人都知道，下面两种形状



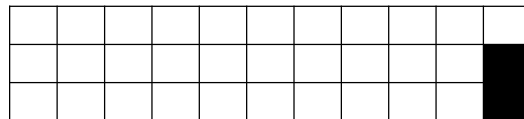
是最受欢迎的：

我们很容易发现，对于任何一个初始状态，可以仅用这两种形状，使棋盘成为高度不超过 3 的很矮的棋盘。接下来，我们只需用随机化算法将棋盘弄

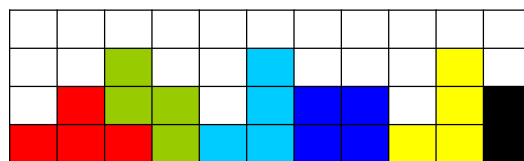
平。随机化的核心思想和实现都很简单，就是从左到右找“空”，然后随机用一个合法的形状填上此空，直到将棋盘完全弄平。

当然这个算法是很粗劣的，数据大一点就无能为力了，为此，我们还可以进行许多优化，主要是在随机的基础上加入一些人的思想。由于本问题的数据是给出了的，程序的优化就显得相对更加容易。

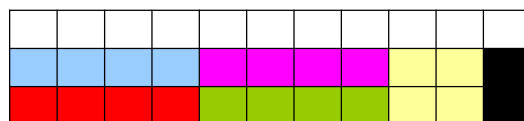
优化一：上述算法在棋盘“相当平”的时候，却在做大量在人看来非常愚蠢的操作。对于下面一个残局，



我们的算法却在这样随机生成形状：



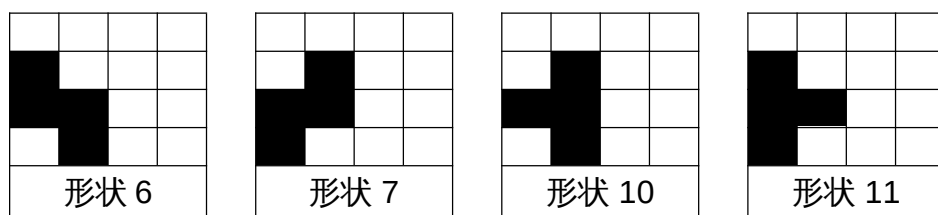
这样虽然合法，但是难以对付较大的数据，因为步数太多了。其实只需要：



所以在可能的情况下，我们应该尽量使用形状 2 和形状 3，这样会使棋盘尽量显得平整。

优化二：同时，我们也应该尽量避免下面 6，7，10，11 这四种形状，这样就不会过多引入那些突兀。





至此，虽然算法还比较粗糙，但对付本题已绰绰有余了。类似的人工因素还有很多，比如还可优先考虑形状 9 等等，这里就不再继续讨论了。此算法的优势在于其思考及实现的花时非常少，程序简短，劣势在于出解步数较大，最大的数据需要一万多步才能出解。虽然对付本题的数据没有问题，但数据再大就显得有些无能为力。算法的时间复杂度为  $O(M)$ ， $M$  为游戏的总步数。

顺便值得一说的是，该随机化算法在一般情况下是可以瞬间出解的，但偶尔会由于程序自身缺陷而陷入随机陷阱，造成“死机”，而我们只需要终止程序即可。这其实也是结果提交类问题带来的好处，那就是只管结果，不管过程，无论中途发生什么，只要输出正确结果。同时，这种多次运行的随机化思想还可以更有助于我们逼近最优解，这在后面将要提到的 Xor 中是非常有意义的。

除了随机化算法，我们还有两种较好的算法：

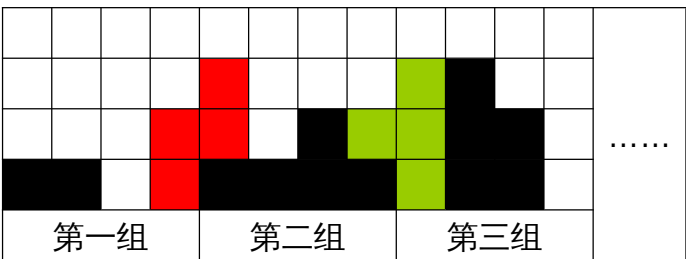
算法一：我们每一步都尽量使棋盘变得更平，直到最后达到完全平整。具体来说，对于任意个棋盘  $C$ ，我们定义  $F(C) = \sum_{i=1}^{n-1} |H_i - H_{i+1}|$ ，其中  $H_i$  表示棋盘第  $i$  列的高。我们每一步的决策就是从当前棋盘  $C$  的子棋盘集合  $S$  中找出一个  $C' \in S$ ，使得不存在  $C'' \in S$ ，满足  $F(C'') < F(C')$ 。如果有多个  $C'$  存在，我们可以随机<sup>[4]</sup>选取一个。容易看出，本算法的时间复杂度为  $O(M \times N)$ ，其中  $M$  为游戏总步数。这个算法是很容易实现的，不过时间复杂度高了些。

算法二：构造。

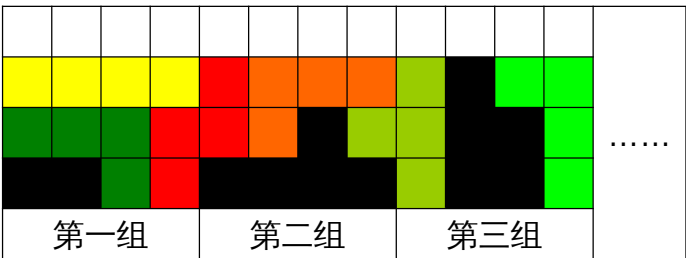
粗略描述如下：我们从左到右以 4 列为单位对棋盘进行分组，最后剩下的如果不足 4 列，则单独为一组。我们设法先将每组填平，然后再用形状 2 将整个棋盘填平。

但有些组我们是无论如何都无法填平的，那就需要我们在初识化时在相邻两组交界的地方放上某些特定的形状，使得该组的方块数为 4 的倍数。由于输入数据是保证有解的，所以这里的“该组”不包括未满 4 列的最后一组。

如下图所示，对于第一、二、三组，通过组内调节是无论如何都无法填平它们的，因为他们原有的块数分别为 2、5、5，都不是 4 的倍数。我们可以通过下面的组间调节使他们都变成 4 的倍数，调节之后三个组的块数分别变成 4、8、8。



之后每组的填平方法可以使用算法一。如下图所示：



此算法的时间复杂度大约为  $O(M)$ ，不过实现起来要麻烦些。

下面我们来将这三种算法做一下比较。由于三种算法的空间复杂度都是

	时间复杂度	运行速度	算法实现的复杂度	解的质量
随机化算法	$O(M)$	<1s	低	较差
算法一	$O(M \times N)$	<10s	低	较好
...	...	...	...	...

$O(N)$ ，在比较的时候，忽略此项指标。

从上表我们可以看出，对于结果提交类问题，随机化算法和算法一是最合适的，如果非结果提交，则算法二最好。不同的要求下，我们采取的算法可能是不同的，因为还要综合考虑算法实现的复杂度。

#### [例四]Xor(IOI 2002)

题目大意：

有黑白二染色的  $N \times N$  格子，我们每次操作先选择一个矩形，然后将此矩形包含的所有格子颜色取反。现要求用尽可能少的操作次数使这些格子全部变为白色。

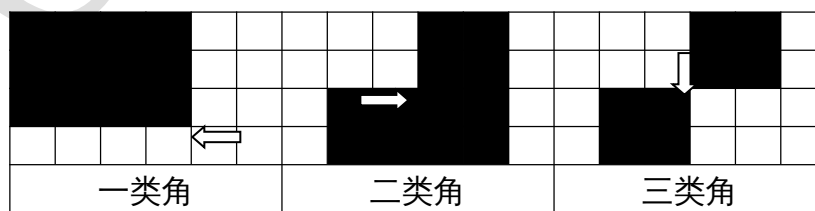
给出十个输入文件，其中  $N \leq 2000$ 。

每个数据得分的计算方法：

$$1 + 9 \times \frac{\text{该数据所有选手中最少的操作次数}}{\text{该数据你的操作次数}}。$$

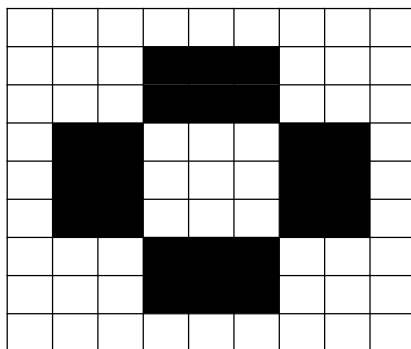
分析：

这类题目是比较普遍的，经验告诉我们应该首先考察长方形的“角”。角可以分为三大类：



显然，一、二类角必是某次操作中的长方形的角，而三类角则未必是。

如：



此位型只需要两次操作即可完成，其中的四个三类角都不是某次操作中的长方形的角。

于是我们的算法也就基本成型了：

1. 将所有的一、二类角做成集合  $A$ 。
2. 从  $A$  中找出所有能组成矩形四个顶点的角组  $(a_i, b_i, c_i, d_i)$ ，分别视为矩形操作并一一删去。
3. 从  $A$  中找出一个能组成某个矩形三个顶点的角组  $(e_i, f_i, g_i)$ ，设缺的顶点为  $h_i$ ，将  $(e_i, f_i, g_i, h_i)$  视为一次矩形操作，在  $A$  中删去  $e_i, f_i, g_i$ ，并添上  $h_i$ 。
4. 如果  $A$  为空，则中止，否则转 2。

算法的正确性容易证明，此处从略。

该算法其实上是贪心的，由于题目只要求我们求较优解，所以算法是可行的。每次四点组的删除会使  $|A|$  减少 4，每次三点组的删除会使  $|A|$  减少 2，所以  $|A|$  在算法中总是**递减**。最优解在最理想的情况下每次操作都能删去一个四点组，而最坏的情况我们也能每次使  $|A|$  减少 2，因此该算法在最坏的情况下每个数据至少可以得到  $1+9*2/4=5.5$  的分。当然在实践中，得分远不止此。

在算法的实现上，我们可以引入一个 $(N+1) \times (N+1)$ 的布尔数组 *Couple*，*Couple*[*i*,*j*]为真当且仅当在当前的图形中存在(*a*,*i*)，(*a*,*j*)两个角，而它们都属于一二类角集合。

引入随机算法优化：

我们可以看到，2 的处理是无所谓先后顺序的，而 3 的选择则直接关系到最终解的优劣。然而问题的规模 2000\*2000 非常大，即使是很简单的选择策略在时间上都难以承受。因此我们完全可以随机选择我们需要的三点组，每次选择三点组+删除+转回 2 反查四点组的总时间复杂度为  $O(N)$ ，这样程序主体的时间复杂度就降至完全可以忍受的水平了，如果最后得到的操作数为  $M$ ，则程序主体的时间复杂度就为  $O(M \times N)$ ，从实践中我们看出， $M$  最大也不超过 1000，时间主要花在预处理上面。除了速度，我们在后面还可以看到，解的质量也是可以接受的。

然而我们并不能保证次算法每次都能够得到相当优秀的解，虽然相差很小。但是在本题的评分方法下，这种差异对最后得分的影响还是不容低估的，于是我们选择多次运行取最佳解。我想这也应该是命题者将此题作为结果提交类问题的原因之一，考察程序的最佳表现，而不是随机表现。

下面是该算法的表现和目前已知的最优解(当然不包括此程序)比较的情况：

	Xor.pas 的解	已知最优解	得分
Xor1.in	3	3	10.0
Xor2.in	35	33	9.5
Xor3.in	28	27	9.7
Xor4.in	110	113	10.0
Xor5.in	74	74	10.0
Xor6.in	487	469	9.7

Xor7.in	515	507	9.9
Xor8.in	200	200	10.0
Xor9.in	500	500	10.0
Xor10.in	845	841	10.0
总分			99

注：以上结果是 10 次运行得到的。

结论：

本题用随机化的方法是相当有效的，特别是针对其结果提交的特点实行多次运行，使得有 5 个数据达到甚至超过了已知最优解，其它数据的结果也都非常接近已知最优解。当然，这种算法的实现要比其他的要容易得多——也就是，我们在尽可能短的时间内得到了较为满意的结果。

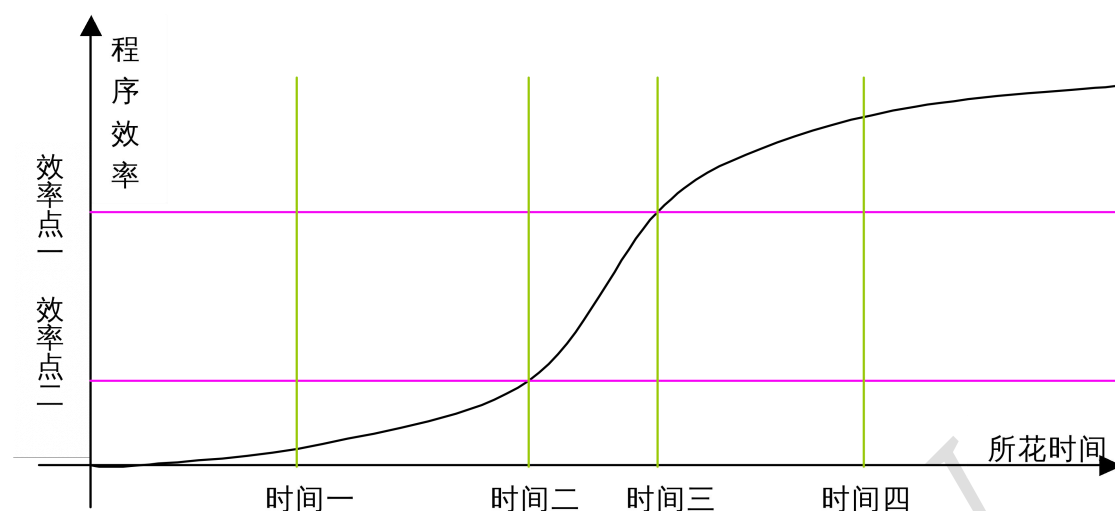
当然本题还有很多算法，比如匹配等，都是比较稳定而且质量较高的算法。这也再次说明结果提交类问题由于其开放型，一般都有多种方法，同时也需要我们能够优中选优。

#### 四、结果提交类问题的策略

正如前文所提到，结果提交类问题对选手的考察已不再局限于算法和数据结构，而是涉及一些更广泛、更深层次的东西，如比赛策略等。下面我们就来讨论一下该类问题所带来的不同的比赛策略。

##### 一、不同的效率观和策略观

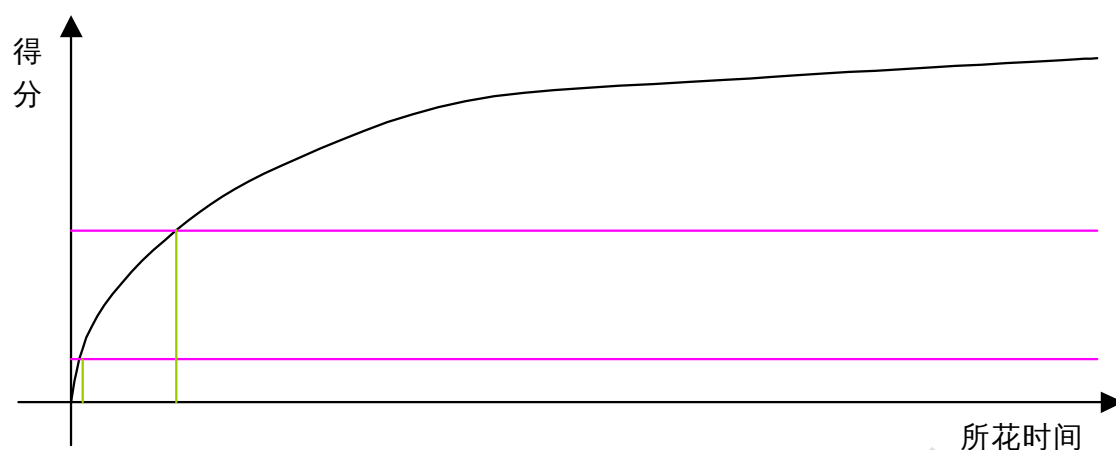
前文已经提到，结果提交类问题效率观的改变首先体现在对算法的衡量标准不再仅是时空复杂度，还要考虑算法的实现，考虑如何在最短的时间内得到正确结果。让我们先来看看经典的完成题目所花时间与程序效率关系图：



这种模式用“厚积薄发”四个字来形容最恰当不过了。而且可以发现：

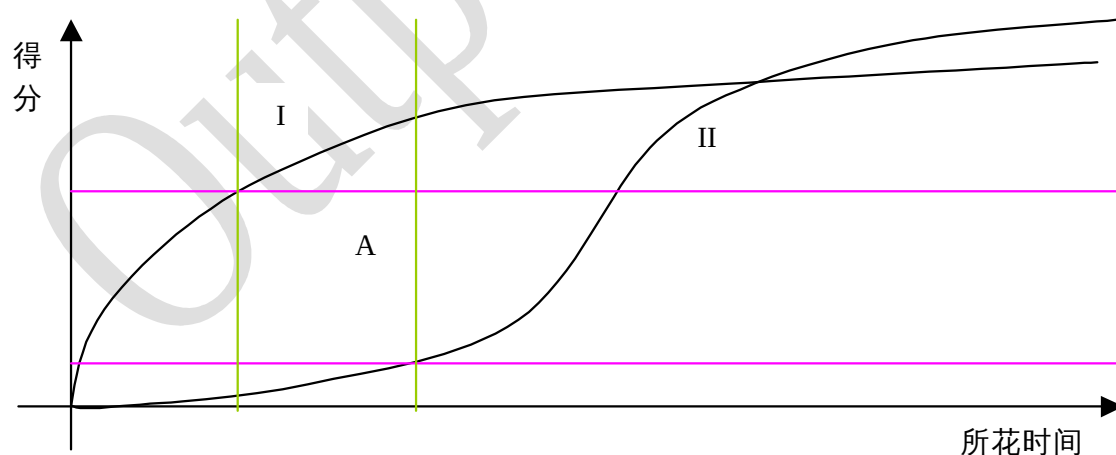
1. 我们不知道什么时候该“适可而止”。我们时常在时间四停止，但事实上，可能在时间三或更早时已经达到题目要求，从算法角度讲，我们得到了进一步的提高，但从策略角度讲，是不明智的。
2. 时间二到时间三这较短的时间内，我们的程序效率有了很大的提升，这对于最后的结果，往往是起着决定性的。注意这里的时间一和时间二离原点都比较远，在开始的相当长一段时间里，我们程序效率的提升很慢，主要是由于在设计算法和数据结构，根本还不涉及程序的实现。

然后我们再来看看结果提交类问题的所花时间与得分关系图，由于结果提交类问题不同的数据处理方法不一定相同，我们以难以用程序效率来衡量，所以后面我们统一使用得分来说明。



这种关系图是比较符合结果提交类问题的特点的。开始时一般都可贪分，所以图像斜率较大，后面斜率逐渐减小，以至最后得分的进展很不明显。当然，既然我们有测试数据，我们可以在心中对能够得的分数有一个比较准确地估计，所以决不会像经典问题那样多花无谓的时间。然而和前面经典问题不同的是，较短时间内使我们的程序效率有了很大提升的时间段(两条绿线之间)前移了不少。这就使得我们如果中途放弃，代价会相对较低。

下面我们来综合分析一下两种题目：



图中区域 A 是我们应当把握的关键，突出表现为曲线一速度快，效率(得分)高。对于同样的得分(水平红线)，我们可以看到，两者在完成时间上有着可观的差异。正如前文所说，结果提交类问题完成质量的标准是完成时间！所以曲



线 I 的优势是无容置疑的。此外，在比赛中，由于心态、策略或其他原因导致我们中途放弃某道题的时候(竖直绿线)，曲线 I 的优势则更为明显。

结果提交类问题对选手策略的考察，可以体现在题目的时间策略上及该试题的全局策略上。我们可以花较短的时间就对能得的分数有相当准确地估计（如 Birthday party），这是前所未有的。因此，如何在理想的时间内得到**得分/时间**的最佳值，如何舍弃部分数据以换取时间完成其他题目，如何有效快速的贪分，如何解决数据间方法的重叠，等等，都需要我们思考。

由此可见，结果提交类问题所带来的效率观是崭新的，倘若我们还停留在仅仅以算法的效率来衡量程序的层面上，那我们势必吃大亏。

## 二、不同的时空复杂观

可以说，结果提交类问题完全打破了经典的时空复杂观和程序优化模式：

1. 大大扩展了程序可用的**空间**，没有严格限制；
2. **时限**变得更加自由，甚至可以多达几个小时；
3. 由于近年来 Linux 在竞赛中的引入，**多线程**的方法也开始崭露头角，我们甚至可以因此将题目的时限扩至 5 个小时。

以前，我们只有少量的空间+很短的时间+单一的线程，而现在，我们有了测试数据+充足的空间+相当长的时间+多个线程。我们已没有理由再用原来的衡量标准和原来的思路来看待现在的问题，我们需要的只有一条：得到正确的结果，而不用管过程。当然问题应该一分为二地来看待，追求结果并不等于放弃对过程、对完美算法地追求，而是应该将实力与巧力有机结合起来。

让我们再来看一个例子。

[例五]Double Crypt(IOI 2001)

题目大意：

任给一个信息块  $p$  (纯文本 plaintext) 和一个密钥字块  $k$ ，通过 AES 的加密函数  $E$ ，便会得出一个加密后的块  $c$  (加密文本-ciphertext)： $c=E(p, k)$ 。

AES 加密函数  $E$  的反函数是解密函数  $D$ ，它满足

$$D(E(p, k), k)=p, \quad E(D(c, k), k)=c。$$

在 DoubleAES 双重加密法中，先后使用两个独立的密钥字块  $k_1$  和  $k_2$ ，先使用  $k_1$ ，然后用  $k_2$ ： $c_2 = E(E(p, k_1), k_2)$ 。

题目给出另一个整数  $s$ ，使得  $k_1$  和  $k_2$  都属于集合  $\{1, 2, 3, \dots, 2^{4 \times s}\}$

提供给你  $p$  和  $c_2$ ，求出  $k_1$  和  $k_2$ 。

输入文件有三行，分别是  $s$ ， $p$  和  $c_2$ 。

	$p=$	$c_2=$
1	00112233445566778899AABBCCDDEEFF	6323B4A5BC16C479ED6D94F5B58F
1	3FCFAA530B83FAC26C3AF18C958F0665	68F4005C13000809D9F5F090C36FD
2	20012001200120012001200120012001	9B4AF5F399003A4AFF6D1FC0743E3
2	CBF1F1840B32AD1FF76A86FAE9034526	2001200120012001200120012001200
4	7F5471276E955708D0BF998EB40E99AA	9E68E681898349BC4ECAB33DC530
4	7F5471276E955708D0BF998EB40E99AA	FB8E04CEE94601A32F30E35FC18C
4	7F5471276E955708D0BF998EB40E99AA	CB4AFEC5C0F4A1C381F014E740A2

输入文件：

简要分析：

很显然，本题用正反两个方向查找，然后用 Hash 表来比较的方法是比较理想的。如果不考虑冲突，则时间复杂度为  $O(2^{4 \times s})$ ，但实现是需要花一定时间的。这种算法比较简单，此处就不赘述。

如果我们退一步，采用时间复杂度为  $O((2^{4 \times s})^2)$  的枚举的方法，并注意到数据三和数据九输出应该完全一样( $p$  和  $c_2$  完全相同，只是  $s$  不同，但不影响结果)，数据五六七可以一起枚举等这些命题者有意用来考察选手观察能力的的数据特点，我们完全可以在非常短的编程时间内得到 80% 以上的分数。当然，程序运行时间很长，不过我们可以通过另一个线程来完成。虽然并不完美，但符合该类问题的特点，充分利用了时间和线程优势，为其他题目节省了大量的时间，这也是符合结果提交类问题的策略的。一言蔽之，就

$1h+1h+1s < 1s+0.5h+0.5h$ <sup>[2]</sup> !

### 三、手工运算与程序运算的协调策略

许多结果提交类问题的精妙之处，就是使手工运算与程序运算巧妙地结合了起来，如 Crack the code，Birthday Party 等都是这样。这样不仅将我们的脑力更加深层地更加高效地植入了程序之中，而且使任务的实现变得简洁迅速。

很显然，我们既不能在 Tetris 中大量依赖手算，也不能在 Party 中完全依靠程序运行，倘若不能很好地将二者有机地结合起来，往往不仅浪费了时间，而且还有可能导致无法出解。

正所谓智者善用物也，我们只有充分利用所有能够利用的主客观条件——大脑与计算机，才能在最短的时间内得到正确的结果。

## 【总结】

结果提交类问题的出现，使我们不能再以传统的观点和方法来处理此类问题，而是应该具体问题具体分析，从全局的角度设计最好的方法来得到需要提交的文件。我们只有善于观察、灵活分析、巧妙设计，同时，将手工与程序完美结合起来，才能快速准确地解决任务。

一道结果提交类问题，我们既可采用经典方法，又可以采取新的方式来完成，正如 Crack the code 等，同时也可能经典方法完全无法对付，如 Birthday party 等。同样的题目，是否以结果提交的形式出现，可能决定我们采用什么算法更好，同时各种方法各有优劣，各有适用范围。于是常出现以往我们觉得很漂亮很精妙的算法在结果提交类问题中就显得相形见绌了，因为我们的衡量标准已经改变。

总而言之，我们可以用一句话概括本文，那就是：永远提醒自己，你所需要的只是在最短的时间内得到正确结果，而不是过程。

奥林匹克的精神是什么？更高、更快、更强。

结果提交，让我们站得更高，做得更快，变得更强。

## 【附录】

[1]这里虽然也是随机，但这里的随机只是一个小模块，显得远没有前面的随机那样纯粹。

[2]在大多数情况下，这时间显得太多了，绝大部分程序都还是非常快的。

### 【参考文献】

1. The 7<sup>th</sup> Baltic Olympiad in Informatics BOI 2001 Tasks and Solutions  
By Marcin Kubica
2. Summary of the 9<sup>th</sup> Central European Olympiad in Informatics
3. 第十九届全国信息学奥林匹克竞赛 NOI 2002 试题
4. IOI 2002 试题
5. IOI'01 Competition (Second Edition)  
By Jyrki Nummenmaa , Erkki Makinen and Isto Aho

# 信息论在信息学竞赛中的简单应用

侯启明

## 引言

信息论是计算机科学中很重要的一个分支。虽然有关信息论的内容很少在以往各种关于信息学竞赛的材料里出现，但实际上，信息论作为证明某些问题解法最优性的理论基础，如果能在竞赛中适当地运用，往往可以取得事半功倍的效果。那么，什么是信息论呢？

## 信息论简介

信息论是关于信息的本质和传输规律的科学的理论。但是在竞赛中用到的主要是关于不确定性和信息的关系的知识。通过它可以很方便地得到某些交互式问题的一个较好的步数下界（这就是某些文献中所说的“信息论下界”）。不过，具体应该怎么做呢？让我们先来看一些信息论的核心理论：

**定义：**如果一个随机变量  $x$  共有  $n$  种取值，概率分别为  $p_1, p_2, \dots, p_n$ ，则其熵为  $H(x) = f(p_1, p_2, \dots, p_n) = -\sum C p_i \log p_i$  ( $C$  为正常数，一般取 1)

**定理 1：**在得到关于随机变量  $x$  的一个熵为  $h$  的信息后， $x$  的熵将会减少  $h$ 。

**定理 2：**当一个随机变量的各种取值概率相等时，它的熵最大。

这些理论看上去和某些题目关系密切，不是吗？那么，具体应该如何运用呢？让我们来看一些例子：

## 实际应用

例 1：验证一定理 1。

我们宿舍二楼到三楼之间楼梯的窗户外面是相邻的一个平房的房顶。在那一带栖息着三只浑身雪白，一只蓝眼睛，一只绿眼睛的——猫！三只猫分别是一只胖猫，一只瘦而尾巴健全的猫和一只瘦而尾巴不健全的猫。在天冷的时候，它们喜欢趴在楼内的暖气上。于是，每只猫就有了两种状态——在屋内和在屋外。显然，三只猫的状态共有 8 种可能情况，假设它们是等概率的。现在，我在一楼的小卖部。由于种种原因，我希望知道猫当时的状况，因此，我往上看了一眼，结果发现在这个位置只能知道屋内猫的只数……

问题 1：把所有猫的情况作为一个随机变量  $x$ ，则当我在小卖部的时候， $x$  的熵是多少？

解答 1：由于 8 种情况的概率相等，所以

$$H(x)=f(1/8,1/8,1/8,1/8,1/8,1/8,1/8,1/8)=\log 8$$

问题 2：我看一眼所得到的信息  $y$  的熵是多少？

解答 2：由于猫的只数共有 0,1,2,3 四种情况，概率分别为  $(1/8,3/8,3/8,1/8)$ ，所以：

$$H(y)=f(1/8,3/8,3/8,1/8)=-\\left(1/8*\log(1/8)+3/8*\log(3/8)+3/8*\log(3/8)+1/8*\log(1/8)\right)=\log 8-6\log(3/8)$$

问题 3：我看完之后， $x$  的熵  $H'(x)$  是多少？

解答 3：此时猫的只数为 0,1,2,3 的四种情况的概率依次是  $(1/8,3/8,3/8,1/8)$ ，而每种情况的熵分别为  $(0,\log 3,\log 3,0)$ ，所以此时  $H'(x)$  的数学期望为：

$$H'(x)=1/8*0+3/8*\log 3+3/8*\log 3+1/8*0=6\log 3$$

可以发现  $H(x)=H(y)+H'(x)$ 。

例 2：Rods(IOI2002)

一个 Rod 是一个由至少 2 个单位正方形连成的水平或竖直的长条。在一个  $N*N$  的方阵中，放了水平和竖直两个 Rod。在图 1 中，Rod 用 X 表示。两个 Rod 可

能有公共方格，比如在图1中，方格（4，4）无法确定是仅属于1个Rod还是同时属于两个Rod。因此，我们在这种情况下假定它同时属于两个Rod。这样，图中竖直Rod的上端点是（4，4）而不是（5，4）。

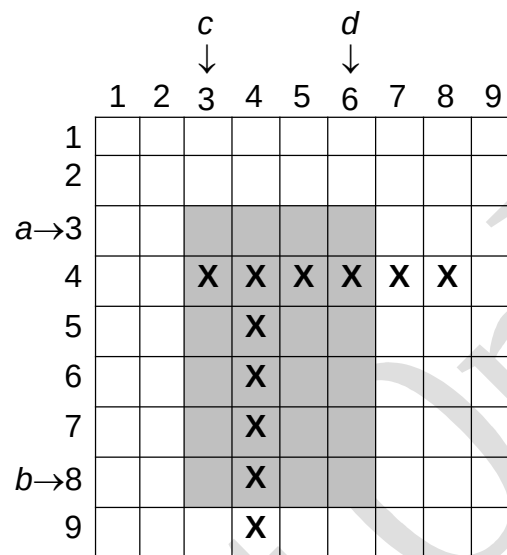


图1

最初我们并不知道两个 Rod 的位置，所以你的任务是编程序把它们的位置找出来。你只能通过库函数  $\text{rect}(a,b,c,d)$  来定位两个 Rod。该函数检验矩形  $[a,b] \times [c,d]$  (如图 1 中阴影区域)。注意参数的顺序，该函数要求  $a \leq b, c \leq d$ 。如果至少一个属于某个 Rod 的方格落在矩形  $[a,b] \times [c,d]$  内的话， $\text{rect}$  返回 1，否则返回 0。

考试当时我很快想到了一个最大步数为  $6\log_2 n + C$  (某个常数) 的方法，但是因为这个数差不多刚好达到步数限制，所以我在这时就开始试图优化步数中  $\log_2 n$  的系数，结果徒劳无功，反而耽误了时间，最后才发现丢分不是因为步数超限而是因为少考虑了一些特殊情况。因此，看过答案以后，我试着从信息论的角度分析了一下这个问题：



由于题目中没有涉及到概率，因此假设所有情况都是等概率的。所以，设 Rod 的摆放方法为随机变量  $x$ ， $x$  所有可能的取值数为  $f(n)$ ，那么  $x$  的熵  $H(x)$  就等于  $\log(f(n))$ 。而由于库函数只有两种可能的返回值，其熵最大为  $H_{\max}(y)=\log 2$ 。因此，询问次数的信息论下界就是  $L=H(x)/H_{\max}(y)=\log(f(n))/\log 2=\log_2 f(n)$ 。

下面讨论  $f(n)$  的值：在  $n \times n$  的方阵中放 1 个 Rod（无论横竖）共有  $n \times C(n+1, 2)$  种方案，放两个相交的 Rod 共有  $C^2(n+2, 3)$  种方案，所以：

$$f(n) = (n^2(n+1)/2)^2 - ((n+2)(n+1)n/6)^2 = (2n^6 + 3n^5 - n^4 - 3n^3 - n^2)/9$$

$$\text{当 } n \text{ 充分大时：} L = \log(f(n))/\log(2) > \log_2(2n^6/9) \approx 6\log_2 n - 2.2$$

由于各种原因，不一定总是使两种返回值概率相等，所以最坏情况下的调用次数往往达不到信息论下界，两者大约相差一个常数，因此，可以认为  $6\log_2 n + C$  是 rect 函数最大调用次数的下界。这样，在得到一个这样的算法之后，就没有什么必要再去徒劳地优化步数，完全可以把时间花在检查对特殊情况的处理上或干脆开始做别的题了。

例 3：Coins(选手推荐题 0024，推荐者饶向荣)

以前下面的难题曾被提议为地方数学奥林匹克竞赛：

有一堆 15 个硬币，其中有 14 个好的，一个坏的。所有好的硬币的质量是相同的，但坏的硬币的质量却不一样，现在告诉你某一枚是好的，要你用一架天平称出哪个是坏的硬币。看你是不是可以在 3 次比较之内，找出坏的硬币。

后来此题引申为：

有一堆  $n$  个硬币，其中有  $n-1$  个好的，一个坏的。所有好的硬币的质量是相同的，但坏的硬币的质量却不一样，现在告诉你某一枚是好的，要你用一架天平称出哪个是坏的硬币。看你是不是可以在  $k$  次比较之内，找出坏的硬币。

输入  $n$  和  $k$ ，如果能在  $k$  此比较中找到  $n$  枚硬币中的哪枚为坏的，就输出 'POSSIBLE'，否则输出 'IMPOSSIBLE'。

两年前我的一位远房亲戚曾给我出过一个类似的题目 ( $n=12$ ,  $k=3$ ，但开始时不知道哪一枚硬币是好的)，当时我苦苦思索了一晚上，终于想出来一个可行的解法。于是，那位亲戚加大了数据规模 ( $n=1100$ ,  $k=7$ ，IMPOSSIBLE)，我想了大概一周，觉得好像无解，但苦于无法证明我的解法的最优性，始终不能理直气壮地回答“IMPOSSIBLE”，只好回答说我最多做到  $n=1093$ ,  $k=7$ 。后来她给了我一个“说明”，但我始终觉得不太严密；拿来问我们班的 IMO 金牌，他的回答是“显然”，我觉得也不严密。于是，这件事就成了我这两年的一个遗憾。现在，有了信息论的武器，这个遗憾终于得到了解决！

首先，对硬币用 1 到  $n$  进行编号，设坏硬币的编号为  $x$ 。同样假设  $x$  的所有取值情况概率相等：

$$\therefore H(x) = \log n.$$

$\therefore$  用天平称一次的结果  $y$  只有 3 种可能情况（左边较重，右边较重，平衡）

$$\therefore H_{\max}(y) = \log 3.$$

$\therefore$  从  $n$  个硬币中通过天平找出一个坏硬币至少需要  $H(x)/H_{\max}(y) = \log_3 n$  步（结论 1）

虽然在本题的条件下，这个信息论下界是达不到的，但是如果没有这个结论，真正最优的解法的最优性就无从证明。得出这个结论后，证明中的困难就迎刃而解了。

进一步的分析：

设在有足够多的已知的好硬币的情况下， $k$  次比较最多从  $g(k)$  个硬币中找出一个重量未知的坏硬币。

当  $k=1$  时，通过枚举可以发现， $g(1)=2$ 。

当  $k>1$  时：

考虑第一次比较结果是平衡的情况，设  $t$  为这次没上天平的尚未确定好坏的硬币的个数，由于可以通过剩下的  $k-1$  次比较把坏硬币从这  $t$  个硬币中找出来，所以  $t \leq g(k-1)$ 。

现在考虑第一次比较结果不是平衡的情况，此时可以确定坏硬币在上了天平的  $g(k)-t$  个硬币中，同样由于可以通过剩下的  $k-1$  次比较把坏硬币从这  $g(k)-t$  个硬币中找出来，根据结论 1，得到： $g(k)-t \leq 3^{k-1}$

综合起来，得到  $g(k) \leq g(k-1) + 3^{k-1}$

现在通过构造一种称法来证明  $g(k) = g(k-1) + 3^{k-1}$ ：

第一次比较第 1 到  $3^{k-1}$  号硬币和  $3^{k-1}$  个好硬币，分以下情况讨论：

平衡：说明坏硬币在剩下的  $g(k-1)$  个硬币中，由  $g$  的定义，可以在  $k-1$  步内找出。

好球较轻：说明坏硬币在这  $3^{k-1}$  个硬币中，且坏硬币较重。此后每次把所有硬币分成三等份，比较其中两份，如果平衡，说明坏硬币在第三份中，否则

坏硬币就在重的一份中，这样就把坏硬币的范围缩小成了原来的三分之一。这个步骤重复  $k-1$  次之后，刚好找出坏硬币。

好球较重：与上一种情况类似，不再赘述。

构造完毕

这样，根据  $g(k)=g(k-1)+3^{k-1}$ ，计算得出： $g(k)=(3^k+1)/2$

于是，得到结论 2：

**结论 2** 在有足够多的已知的好硬币的情况下， $k$  次比较最多从  $g(k)=(3^k+1)/2$  个硬币中找出一个重量未知的坏硬币。

现在，开始解决原问题：

设在有一个已知的好硬币的情况下， $k$  次比较最多从  $f(k)$  个硬币中找出一个重量未知的坏硬币。

显然， $f(k) \leq g(k)$ 。

现在通过构造一种称法，来证明  $f(k)=g(k)$ ：

设硬币 1 为已知好硬币：

第一次比较 1 号到  $(3^{k-1}+1)/2$  号硬币和  $(3^{k-1}+1)/2+1$  号到  $3^{k-1}+1$  号硬币：

平衡：比较  $3^{k-1}+2$  号到  $2 \cdot 3^{k-1}+1$  号硬币和 1 到  $3^{k-1}$  号硬币，后面步骤参考有足量的好硬币的情形

不平衡：比较  $(3^{k-1}+1)/2+1$  号到  $(3^{k-1}+3^{k-2})/2$  号硬币、 $(3^{k-2}+1)/2+1$  号到  $(3^{k-1}+1)/2$  号硬币和 1 号到  $(3^{k-2}+1)/2$  号硬币加上足量的好硬币

平衡：转化成  $k-1$  的问题

不平衡：确定坏硬币在某  $3^{k-1}$  个硬币中，且知道是它比好硬币轻还是重

构造完毕。原问题解决， $k$  次最多在  $n=f(k)+1=(3^k+3)/2$  个硬币中称出一个坏硬币。

### 总结

细心的读者应该会注意到，本文中的例题不用信息论的知识都可以解决。那么，信息论在这里的意义是什么呢？实际上，作为一种纯粹的理论，信息论并非是解决具体题目的手段，而是用来对一类问题进行分析的通用工具。它可以为我们的解法提供强有力的理论依据，更可以通过估计上下界来指导解法的构造。综上所述，信息论在信息学竞赛中是大有用武之地的。

# 一类称球问题的解法

长沙雅礼中学 何林

## 关键字

判定树 三分 均匀

## 摘要

本文对一类天平称球问题提出了完整、严谨的解法，在此基础上总结了研究过程中的一些心得和方法。

## 引言

有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品，你有一架天平。现在要称出哪个次品来。

问题 1 已知次品的重量比其他的要重一些。

问题 2 不知道次品的重量。

问题 3 不知道次品的重量。不仅要求出次品，还要求次品的轻重。

问题 4 不知道次品的重量，要求次品和次品的轻重。另外你手里还得到了一个标准球。

面对这一系列类似的问题，我们该从何入手？

## 简单问题的分析

先来考虑最简单的问题 1。为了方便叙述，把  $n$  个球按  $1, 2, \dots, n$  顺次编号。

若  $n=3$ ，把一号球放在天平左边、二号球放在天平右边。如果天平：

- 1、左偏，一号重，是次品。
- 2、右偏，二号重，是次品。
- 3、保持平衡，那么一、二都是正常的球，因此就只有可能三号球是次品了。

因此  $n=3$ ，至多一次就能称出哪个是次品。记作  $f(3)=1$ 。

下面考虑  $n=9$ 。把所有的球分成三组： $A\{1,2,3\}, B\{4,5,6\}, C\{7,8,9\}$ 。A 组的球放在左边、B 组放在右边。如果天平：

- 1、左偏，则次品在 A 组里面。
- 2、右偏，则次品在 B 组里面。
- 3、保持平衡，次品在 C 组里面。

无论在哪个组里面，我们已经把次品的范围从“9”缩小到了“3”，也就是减少到原来的  $1/3$ 。之前我们已经研究过，3 个球 1 次就能称出来，故而  $f(9)=2$ 。

不难推广到一般的情况：

**定理 1.1**  $f(3^n)=n$ 。

证明： $n=1, 2$  时，已证。设  $n=k$  成立，则  $f(3^k)=k$ ；下面考虑  $n=k+1$  的情况。

将  $3^{k+1}$  个球分成三堆 A, B, C，使得  $|A|=|B|=|C|=3^k$ 。把 A 放在天平左边、B 放在右边，天平：

- 1、左偏，次品在 A

2、右偏，次品在 B

3、平衡，次品在 C

无论哪种结果，我们都把次品的范围缩小到了  $3^k$  个球里面。而  $f(3^k)=k$ ，故而  $f(3^{k+1})=k+1$ 。

综上，定理 1.1 成立。

稍经分析不难得到：

**定理 1.2**  $f(n)=\lceil \log_3 n \rceil$

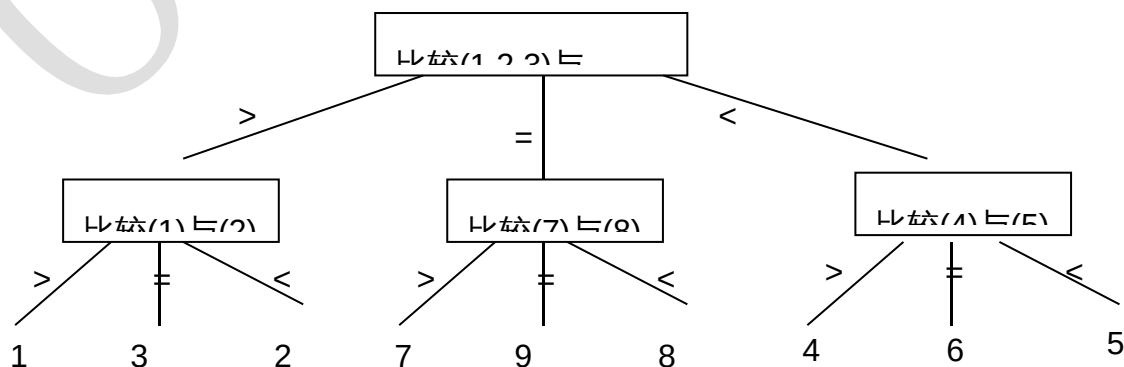
这个的证明和定理 1.1 完全类似，分  $n \bmod 3 = 0, 1, 2$  适当讨论即可。

我们必须注意到  $\lceil \log_3 n \rceil$  是可行的，因为我们能够构造出这样一个方案。问题是它是否最优？

我们采取的方案是每次将球尽量均匀的三分，这样做的根据就是天平只有三种结果：左偏、右偏、平衡。于是就能保证无论次品在哪一份都能将结果的范围缩小到原来的  $1/3$ 。从感性上认识， $\lceil \log_3 n \rceil$  应该就是最优解了。

为了更加严格的证明  $\lceil \log_3 n \rceil$  的最优性，我们引进判定树的概念。

下图就是  $n=9$  时的一种判定树：



此题的判定树是这样一棵树：

1、叶子节点代表一种可能的结果。



- 2、非叶子节点代表一次称量。
- 3、非叶子节点至多有三个儿子，分别代表天平的左偏、右偏、平衡三种情况。

任意一种称量方案都能唯一的表示成一棵判定树；反过来一棵判定树也唯一对应一种称量方案。

容易看出判定树的深度就是称量次数。这就是我们之所以引进它的原因。

做出判断之前，谁也无法预知哪个球是次品，每个都有可能是我们的目标；因此一个有意义的判定树应该具有至少  $n$  个叶子节点。

$n$  个叶子节点的树的深度  $h \geq \lceil \log_3 n \rceil$ ，故而可以证明， $f(n) = \lceil \log_3 n \rceil$  是最优的。

至此完整的解决了问题 1。

**我们的结论是：有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品，次品的重量比其他的要重一些。给一架天平，至少称  $\lceil \log_3 n \rceil$  次，就能找出那个次品。**

具体的方案是将球每次都尽量均匀的三分。（详见上文）

让我们总结一下。

“三分”是整个算法的核心。我们选择三分，而不是二分或者四分是有原因的，它的本质是由判定树的特殊结构——三叉树——所决定的。

同时还必须注意一点，我们在三分的时候有两个字很讲究：“均匀”。实际上  $h \geq \lceil \log_3 n \rceil$  中的“ $=$ ”当且仅当球被均匀的分配时才能达到。

这里说的“均匀”是指“在最坏情况下获得最好的效果”。因为一棵树的深度是由它根节点儿子中深度最大的儿子决定的，为了使得整个树深度最小，我们就要务必使得深度最大的儿子深度最小，这就是“均匀”分配的理论根据。

或许你觉得这些总结是显然的、多余的废话。面对一个简单的问题，你当然能够游刃有余，也不需要提炼出什么经验、方法；可是当问题被复杂化、隐蔽化，你还能如此得心应手吗？

## 称球问题的拓展

### 一、问题的提出与初步分析

问题 4 有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品，已知：

- 1、次品的重量与其他的球不同，**不知道轻重**。
- 2、你有一架天平和一个**标准球**

问至少要称多少次，才能找出那个次品，并且知道次品是轻还是重？

我们很自然的联想到问题 1，试着应用三分。

把  $n$  个球分成三堆，第一堆放在天平左边、第二堆放在天平右边。

天平平衡，毫无疑问，次品在第三堆中；但问题是如果天平发生了偏移，既可能第一堆中混入了一个重球、也可能第二堆中混入了一个轻球。我们根本无法准确地判断次品具体在哪堆中。

问题 4 较之问题 1 最大的困难就在于不知道次品的轻重，因此一次称量之后根本无法马上将次品缩小到理想的范围。

因此经过一次称量，如果天平不平衡，问题就被转化为一个新的结构，我们将其归纳、强化成如下引理：

**引理** 有两堆球，第一堆有  $n$  个、第二堆有  $m$  个，其中有一个是次品。并且次品如果在第一堆中只可能是重球、在第二堆中只可能是轻球。只要称  $\lceil \log_3(n+m) \rceil$  次就可以找到次品。

(这是一条重要结论，之后会反复引用，请注意)

为了解决问题 4，我们先探讨这个引理的证明。

## 二、引理的证明

总共  $n+m$  个球，每个球都可能是次品，所以判定树有  $n+m$  个叶子节点。于是判定树的深度  $h \geq \lceil \log_3(n+m) \rceil$ ，由此可知  $\lceil \log_3(n+m) \rceil$  是一个最优的界。

下面我们通过归纳构造，证明这个界是可以达到的。

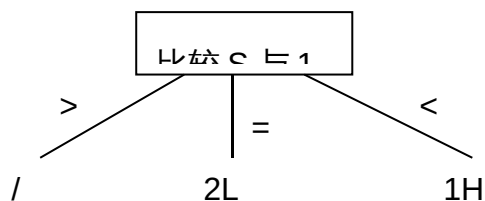
达到最优界必需且仅需满足一条原则：均匀分配叶子节点。

为了方便叙述，约定：

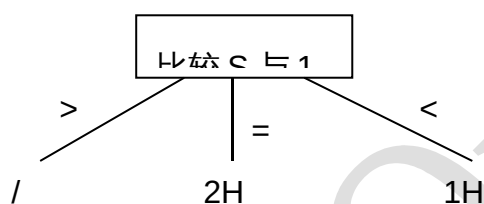
- 1、 $S$  表示标准球。
- 2、 $nH$  表示第  $n$  个球是重球
- 3、 $nL$  表示第  $n$  个球是轻球
- 4、 $\langle A, B \rangle$  表示将集合  $A$  中的球放在天平左边，集合  $B$  中的球放在天平右边的称量结果。 $\langle A, B \rangle = \text{Left}$ 、 $\text{Right}$  或者  $\text{Middle}$ ，分别表示左偏、右偏和平衡。

用归纳法证明，归纳  $n+m$ 。

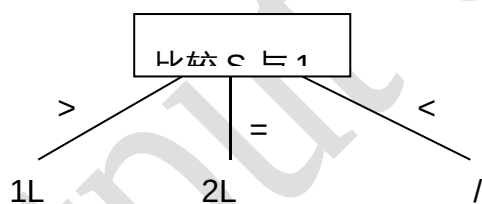
$n+m=2$ 。如果  $n=m=1$ ，根据题设只有可能是 1H, 2L。判定树如下：



如果  $n=2, m=0$ ，只有可能是 1H、2H，判定树如下：



若  $n=0, m=2$ ，只有可能是 1L、2L，判定树如下



无论  $n, m$  如何取值，均只需要 1 次称量即可。而  $\lceil \log_3(n+m) \rceil = 1$ ，因此  $n+m=2$  时引理成立。

假设  $n+m < k$  ( $k \geq 3$ ) 时引理成立，下证  $n+m=k$  时也成立。

$k=n+m$ ，按照  $n, m$  除以 3 的余数来讨论。

情况一： $n=3p, m=3q$

将第一堆  $n$  个球均分成三堆：A, B, C，使得  $|A|=|B|=|C|=p$ 。（这里前  $a$  个球地位完全相等，因此只要知道每一堆的球数即可，至于具体每堆球的编号则不是我们所关心的）

第二堆  $b$  个球也均分成三堆：A', B', C'，使得  $|A'|=|B'|=|C'|=q$ 。

根据题设次品在 A, B, C 中只有可能是重球；在 A', B', C' 中只有可能是轻球。

称  $\langle A+A', B+B' \rangle$ ，若：

1、 $\langle A+A', B+B' \rangle = \text{Middle}$ 。次品在 C 或者 C' 中，问题归结到  $p+q$  时的引理。

2、 $\langle A+A', B+B' \rangle = \text{Left}$ 。由于次品在 A' 中只有可能是轻球、在 B 中只有可能是重球，所以次品必不在这两者中。可以把范围缩小到 A 和 B' 中，问题又归结到  $p+q$  时的引理。

3、 $\langle A+A', B+B' \rangle = \text{Right}$ 。类似于 2，可以把次品的范围缩小到 A' 和 B 中，问题也归结到  $p+q$  时的引理。

无论如何，一次称量后都可以把问题归结到  $p+q$  时的引理。根据归纳假设还要称  $\lceil \log_3(p+q) \rceil$  次。所以原问题总共称  $\lceil \log_3(p+q) \rceil + 1 = \lceil \log_3(n+m) \rceil$  次即可。

于是情况一引理成立。

情况二： $n=3p+1, m=3q+2$ 。

将第一堆  $n$  球分成三堆 A, B, C，使得  $|A|=p, |B|=p, |C|=p+1$ 。

第二堆  $m$  个球也分成三堆 A', B', C'，使得  $|A'|=q+1, |B'|=q+1, |C'|=q$ 。

称  $\langle A+A', B+B' \rangle$ ，根据天平偏移的情况做出适当的判断即可。此不赘述。

我们之所以把  $A+A', B+B', C+C'$  分别作为一组，就是因为：

1. 天平左偏，次品只有可能在 A, B' 中，有  $p+q+1$  种可能的结果。

2. 天平右偏，次品只有可能在 A', B 中，有  $p+q+1$  种可能的结果。

3. 天平平衡，次品只有可能在 C, C' 中，有  $p+q+1$  种可能的结果。

也就是说无论天平怎么偏，我们总可以把原来的  $n+m=3(p+q)+3$  种可能的结果，缩减到  $p+q+1$  种；换一个角度看，也就是把  $3(p+q)+3$  个叶子节点分摊到三棵子树上，每棵分摊到的节点个数都是  $p+q+1$ ！这是是准确意义上的均匀，保证了在最坏的情况下也能得到最好的结果。

总之无论你怎么分，只要保证了“均匀”，就是一种可行的方案。所谓的均匀就是“最坏的情况下的结果达到最优”。

另外还有四种情况（见下表）都可以按照以上的方法类似证明，结构完全相同，关键是抓着“均匀”的原则不放。下表是这几种情况的具体划分方法：

	a 的划分，三个数依次代表 $ A ,  B ,  C $	b 的划分，三个数依次代表 $ A' ,  B' ,  C' $
$n=3p, m=3q+1$	$p, p, p$	$q, q, q+1$
$n=3p, m=3q-1$	$p, p, p$	$q, q, q-1$
$n=3p+1, m=3q+1$	$p, p+1, p$	$q+1, q, q$
$n=3p+1, m=3q+2$	$p, p, p+1$	$q+1, q+1, q$

需要特别说明的是，这里的均分的是“可能的结果”（即判定树的叶子节点），而不是单纯的均分球。

综上，对于任意的自然数  $n, m(n+m \geq 2)$ ，引理成立！

### 三、问题 4 的解决

回顾一下问题 4。

**问题 4** 有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品，已知：

- 1、次品的重量与其他的球不同，**不知道轻重**。
- 2、你有一架天平和一个**标准球**

问至少要称多少次，才能找出那个次品，并且知道次品是轻还是重？

问题 1 中每个球都有可能是次品，所以有  $n$  个叶子节点；我们必须注意到问题 4 不仅每个球都有可能是次品，而且次品还有两种情况：轻或者重！所以本题实际上有  $2n$  种不同的可能结果，而不是  $n$  种。

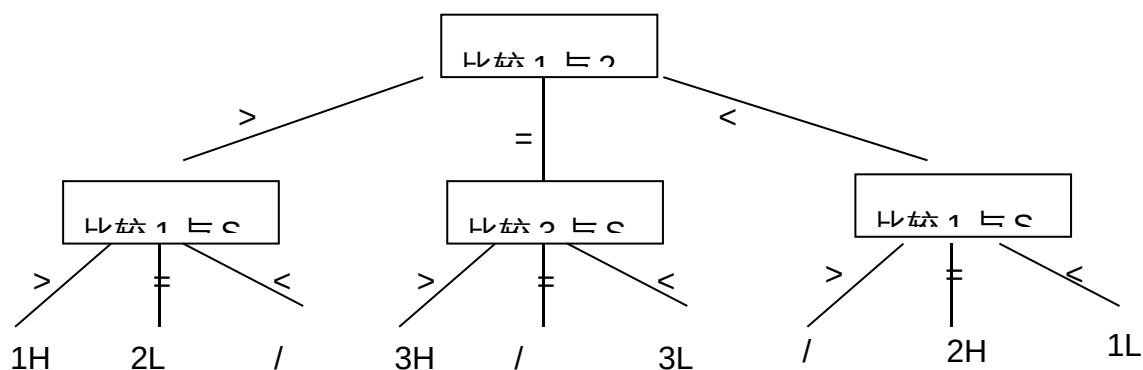
所以此问题的判定树至少要有  $2n$  个叶子节点，它的深度  $h \geq \lceil \log_3 2n \rceil$ 。

比如  $n=3$ ，有如下 6 种结果：

- 1、 1 号是次品，且比标准求重。记作 1H (H 是 heavy 的缩写)
- 2、 2H
- 3、 3H
- 4、 1 号是次品，且比标准球轻。记作 1L (L 是 light 的缩写)
- 5、 2L
- 6、 3L

拥有 6 个叶子的判定树的深度  $h \geq \lceil \log_3 6 \rceil = 2$ ，所以 3 个球至少要称 2 次才能找到次品并知道轻重，而不是我们想象的 1 次。

实际上  $n=3$  时 2 次也就足够了，判定树如下：



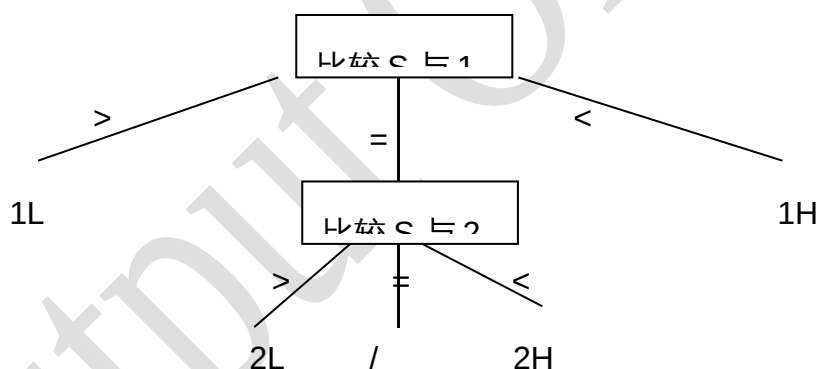
我们很自然的联想：是不是 $\lceil \log_3 2n \rceil$ 就是问题的解？ $\lceil \log_3 2n \rceil$ 的最优性是毋庸置疑的，可是可行性呢？我们是不是总能构造出这样的一棵判定树呢？

将这个想法归纳一下就是

**猜想** 给定一个标准球，一架天平。有  $n$  个球，其中一个是次品。每个球既有可能轻、也有可能重。只需要 $\lceil \log_3 2n \rceil$ 次就能称出次品，并且知道它是轻还是重。

仿照之前解决的问题，采用归纳构造。

$n=2$  时判定树如下，2 次即出解。 $\lceil \log_3 2n \rceil=2$ ，因此  $n=2$  时成立。



假设  $n < k$  ( $k \geq 3$ ) 时猜想成立，下证  $n=k$  时也成立。

情况一： $k \bmod 3 = 0$ 。可以设  $k=3p$ 。

将球分成三堆： $A, B, C$ ，使得 $|A|=|B|=|C|=p$ 。称 $\langle A, B \rangle$ 。若：

I.  $\langle A, B \rangle = \text{Left}$ 。结果可能是  $AH$  或者  $BL$ ，问题归结到  $p+p$  时的引理，还要称 $\lceil \log_3 2p \rceil$ 次。总共要称 $\lceil \log_3 2p \rceil + 1 = \lceil \log_3 6p \rceil = \lceil \log_3 2k \rceil$ 次。

II.  $\langle A, B \rangle = \text{Middle}$ 。次品在  $C$  中，问题归结到  $n=p$  时的猜想，根据归纳假设还要称 $\lceil \log_3 2p \rceil$ 次。所以总共称 $\lceil \log_3 2p \rceil + 1 = \lceil \log_3 6p \rceil = \lceil \log_3 2k \rceil$ 次即可。

III.  $\langle A, B \rangle = \text{Right}$ 。类似 I，总共称 $\lceil \log_3 2k \rceil$ 次。



无论如何称 $\lceil \log_3 2k \rceil$ 次即可。综上  $k \bmod 3 = 0$  时猜想成立。

情况二： $k \bmod 3 = 2$ 。可以设  $k=3p+2$ 。

分成三堆 A, B, C，使得  $|A|=p+1$ ,  $|B|=p+1$ ,  $|C|=p$ 。将  $6p+4$  个叶子节点分摊为  $2p+2, 2p+2, 2p$ ，是均匀的。

然后称  $\langle A, B \rangle$ ，和情况一完全类似，此不赘述。

情况三： $k \bmod 3 = 1$ 。可以设  $k=3p+1$ 。

先看一种错误的想法。

分成三堆 A, B, C，使  $|A|=p$ ,  $|B|=p$ ,  $|C|=p+1$ ，这样看起来是“均匀”的。接着称  $\langle A, B \rangle$ ：

- 1、 $\langle A, B \rangle = \text{Middle}$ 。次品在 C 中，可能是 CH 或者 CL 这  $2p+2$  种结果。
- 2、 $\langle A, B \rangle = \text{Left}$ 。可能是 AH 或者 BH 这  $2p$  种结果。
- 3、 $\langle A, B \rangle = \text{Right}$ 。可能是 AL 或者 BH 这  $2p$  种结果。

不难发现原问题的  $6p+2$  个叶子节点被分摊成  $2p+2, 2p, 2p$ 。这是均匀的吗？显然不是！

比如  $p=1$ ,  $n=4$ ，共有 8 种可能的结果，根据猜想只需要 2 次即可得出解答。可是如果把球划成 1,1,2 三堆，2 个球需要 2 次才能称出，整个问题就至少需要称 3 次！

我们往往容易被表面的假象所迷惑，把  $3p+1$  个球分成  $p$ 、 $p$ 、 $(p+1)$  这三堆看似是均匀的，然而我们必须时时谨记：均分的不是球，而是判定树的叶子节点——那才是保证结果最优的充要条件。

既然有  $6p+2$  个叶子节点，我们就应该按照  $2p+1, 2p+1, 2p$  的方式均分。

可以如此划分： $A\{S, 1, 2, \dots, p\}$ ,  $B\{p+1, p+2, \dots, 2p, 2p+1\}$ ,  $C\{2p+2, 2p+3, \dots, 3p+1\}$ 。即 $|A|=p+1$ ,  $|B|=p+1$ ,  $|C|=p$ 。特别注意的是在  $A$  中加入了一个标准球。

然后称 $\langle A, B \rangle$ ，若：

1、 $\langle A, B \rangle = \text{Middle}$ 。次品在  $C$  中，问题归结到  $n=p$  的猜想，根据归纳假设，还要称 $\lceil \log_3 2p \rceil$ 次。既总共称 $\lceil \log_3 2p \rceil + 1 =$

$\lceil \log_3 6p \rceil = \lceil \log_3 6p + 2 \rceil = \lceil \log_3 2k \rceil$ 次。

2、 $\langle A, B \rangle = \text{Left}$ 。有可能是  $1H, 2H, \dots, pH, (p+1)L, (p+1)L, \dots, (2p+1)L$  这  $2p+1$  种结果（由于  $S$  是标准球，所以不在我们的筛选范围之内。因此参加称量的虽然有  $2p+2$  个球，但是实际有可能是次品的却只有  $2p+1$  个。这也是此方案和之前错误想法的根本区别），可以归结到  $p+(p+1)$  时的引理，还要称 $\lceil \log_3 2p+1 \rceil$ 次。总共称 $\lceil \log_3 2p+1 \rceil + 1 = \lceil \log_3 6p+3 \rceil = \lceil \log_3 6p+2 \rceil = \lceil \log_3 2n \rceil$ 次即可。

3、 $\langle A, B \rangle = \text{Right}$ 。类似于 2，不赘述。

综上， $k \bmod 3 = 1$ ，猜想成立。

综上，对于任意的自然数  $n$  ( $n \geq 2$ )，猜想成立！

于是问题 4 解决了，我们的结论是：

给定一架天平，一个标准球。称出  $n$  个球中的次品并且知道轻重，需要且仅需要 $\lceil \log_3 2n \rceil$ 次。

## 四、小结

在已经研究过问题 1 的基础上，我们很自然的把解决问题 1 的一系列方法和主观经验搬到此题来；这对启迪思路起了很重要的作用。

然而由于问题的相异性，表面的规律和经验却在进一步的应用中失败了。

我们以“判定树”为桥，深入的研究了问题 1 中间方案的本质规律，得到了两条重要原则：

- 1、 均匀三分。
- 2、 均分的是叶子节点，而不是球。

正是这两条原则引导我们迅速的构造出完整的称量方案，得以正确的解决问题。

称球问题的其他变化形式，也完全遵循上述两个原则。所谓万变不离其宗，只要我们牢牢抓住上述两条原则，以判定树为研究的桥梁，无论问题形式怎么变，都能手到擒来。

## 称球问题的一些其他变化形式

**问题 3** 有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品。已知：

- 1、 次品的重量与其他的球不同，不知道轻重。
- 2、 你只有一架天平。

问至少要称多少次，才能找出那个次品、并且知道次品到底是轻还是重？

问题 3 和 问题 4 相比，唯一的不同就是没有标准球。

实际上一次称量之后，无论天平怎么偏，我们至少可以得到一个标准球。

具体地说，第一次称量：

- 1、 天平平衡。放在天平上的都是标准球。
- 2、 天平不平衡。没放在天平上的球都是标准球。

因此，从第二次称量开始，问题 3 就完全等价与问题 4。只要考虑第一次称量。

解决问题 4 时， $n \bmod 3 = 0, 2$  我们没用标准球即达到最优解；唯有  $n \bmod 3 = 1$  时需要借助标准球实现叶子节点的均摊。

稍加试验就会发现， $n \bmod 3 = 1$  无论如何也无法在没有标准球的前提下均分叶子节点，只能舍而求次，采用一种次优的划分方案。

设  $n=3k+1$ ，划分成 A, B, C，使得  $|A|=|B|=k$ ,  $|C|=k+1$ ，然后称  $\langle A, B \rangle$ 。具体的判断和计算并不复杂，留给读者完成。

我们的结论是：

给定一架天平，有  $n$  个球，其中一个是次品。称  $\lceil \log_3 2n + 2 \rceil$  次就可以找到次品，并且知道次品的轻重。

容易发现  $n$  个球只有  $2n$  种可能的结果，因此从理论上说，判定树的最优下界是  $\lceil \log_3 2n \rceil$ 。此处的结论为什么要  $\lceil \log_3 2n + 2 \rceil$  次呢？原因就在于缺少一个标准球，使得  $n \bmod 3 = 1$  时，第一次我们无论如何也不可能实现完美意义上的均分，只能采取了一种次优的方案，因此最优界也是不可能达到的。

以上的问题都要求次品的轻重，可有时候我们关心的是哪个球是次品。至于具体是轻还是重，倒是无关紧要的。

因此我们提出一个新问题：

**问题 5** 有  $n$  ( $n \geq 3$ ) 个球，其中一个是次品。已知：

- 1、次品的重量与其他的球不同，不知道轻重。
- 2、给你一架天平、一个标准球。

问至少要称多少次，才能找出那个次品？

这个问题又复杂一点。

首先是下界不好估计。客观上说每个球都有可能是次品，且能轻能重，似乎有  $2n$  个叶子节点；可是我们关心的是哪个是次品，具体的轻重无关紧要，因此  $n$  个不同的叶子节点足矣。

到底算  $n$  个还是  $2n$  个呢？之前的研究方法行不通了，要另辟蹊径。

设  $f(n)$  表示  $n$  个球时的最少称量次数。

先假设有无穷多个标准球，第一次取  $a$  个球在天平左边， $b$  个球在天平右边 ( $a \leq b$ )。由于天平两边的球数必须相等，所以在左边还补进  $b-a$  个标准球。

如果天平：

- 1、左偏。有可能是左边  $a$  个球中有重球、或者右边  $b$  个球里有轻球，根据上节引理，还要称  $\lceil \log_3(a+b) \rceil$  次，因此总共称  $\lceil \log_3(a+b) \rceil + 1$  次。
- 2、右偏。类似 1，也是称  $\lceil \log_3(a+b) \rceil + 1$  次。
- 3、平衡。次品必然在剩下的  $n-a-b$  个球中，要称  $f(n-a-b)$  次。

综上，此时称量次数是： $\max\{\lceil \log_3(a+b) \rceil, f(n-a-b)\} + 1$  次。

注意到最后的称量次数只和  $a+b$  有关，因此可以对  $a, b$  适量调整，使得  $|a-b| \leq 1$ ，于是补充的标准球顶多 1 个，完全满足题目要求。

设  $a+b=p$ ，则：

$$f(n) = \min\{\max\{\lceil \log_3 p \rceil, f(n-p)\} + 1\} \quad (1 \leq p \leq n)$$

$$f(0)=0, f(1)=0, f(2)=1$$

这就是此问题的一个递推式，若对时间复杂度要求不高，如此既可求解。

但是当  $n$  较大时，时空效率都是不能令人满意的，因此我们求出部分  $f(n)$  的值：

N	$f(n)$
1	0
2	1
3—5	2
6—14	3
15—41	4
42—122	5

很容易发现： $f(n) = \lceil \log_3(2n-1) \rceil$ 。

考虑从递推式入手来证明这个猜想。同样采用归纳法。

$n=1,2$  时候容易验证猜想成立。

设  $n < k$  时猜想成立，下证  $n=k$  也成立。

先证最优性。

$$f(k) = \min\{\max\{\lceil \log_3 p \rceil, f(k-p)\} + 1\} \quad (1 \leq p \leq k)$$

假设存在这样的  $p$  使得  $f(k) < \lceil \log_3(2k-1) \rceil$ ，那么：

$$\lceil \log_3 p \rceil + 1 = \lceil \log_3 3p \rceil < \lceil \log_3(2k-1) \rceil$$

$$f(k-p)+1 = \lceil \log_3 2(k-p)-1 \rceil + 1 = \lceil \log_3(6k-6p-3) \rceil < \lceil \log_3(2k-1) \rceil$$

也就是：

$$3p < 2k-1 \quad (1)$$

$$6k-6p-3 < 2k-1 \quad (2)$$

(1)\*2 + (2)得到：

$$6k-3 < 6k-3$$

矛盾。因此对于  $f(k)$  必然有  $f(k) \geq \lceil \log_3(2k-1) \rceil$ 。

再证可行性。

根据  $k \bmod 3$  的余数分类。

情况一： $k \bmod 3 = 0$ 。设  $k = 3p$ 。

分成三堆  $A, B, C$ ，使得  $|A|=|B|=|C|=p$ ，称  $\langle A, B \rangle$ ：

1.  $\langle A, B \rangle = \text{Middle}$ 。次品在  $C$  中，问题归结到  $n=p$  时的问题 5，还要称  $\lceil \log_3(2p-1) \rceil$  次。所以总共称  $\lceil \log_3(2p-1) \rceil + 1 = \lceil \log_3(6p-3) \rceil = \lceil \log_3(2k-3) \rceil$  次。

2.  $\langle A, B \rangle = \text{Left}$ 。可能是  $AH$  或者  $BL$ ，问题归结到  $p+p$  时的引理，还要称  $\lceil \log_3 2p \rceil$  次。所以总共要称  $\lceil \log_3 2p \rceil + 1 = \lceil \log_3 2k \rceil$  次。

3.  $\langle A, B \rangle = \text{Left}$ 。类似 2，总共称  $\lceil \log_3 2k \rceil$  次。

综上， $k=3p$  要称  $\lceil \log_3 2k \rceil = \lceil \log_3(2k-1) \rceil$  次。

情况二： $k \bmod 3 = 1$ 。设  $k=3p+1$ 。

分成三堆  $A\{S, 1, 2, \dots, p\}$ ,  $B\{p+1, p+3, \dots, 2p+1\}$ ,  $C\{2p+2, 2p+4, \dots, 3p+1\}$ ，此时  $|A|=p+1$ ,  $|B|=p+1$ ,  $|C|=p$ 。特别注意在  $A$  中加入了一个标准球。称  $\langle A, B \rangle$ ：

1.  $\langle A, B \rangle = \text{Middle}$ 。次品在  $C$  中，问题归结到  $n=p$  时的问题 5，还要称  $\lceil \log_3(2p-1) \rceil$  次。所以总共称  $\lceil \log_3(2p-1) \rceil + 1 = \lceil \log_3(6p-3) \rceil = \lceil \log_3(2k-3) \rceil$  次。

2.  $\langle A, B \rangle = \text{Left}$ 。可能是  $AH$  或者  $BL$ ，问题归结到  $p+(p+1)$  时的引理（因为标准球不要考虑，所以是  $2p+1$  而不是  $2p+2$ ），还要称  $\lceil \log_3(2p+1) \rceil$  次。所以总共要称  $\lceil \log_3(2p+1) \rceil + 1 = \lceil \log_3(2k+1) \rceil$  次。

3.  $\langle A, B \rangle = \text{Left}$ 。类似 2，总共称  $\lceil \log_3(2k+1) \rceil$  次。

综上， $k=3p+1$  要称  $\lceil \log_3(2k+1) \rceil = \lceil \log_3(2k-1) \rceil$  次。

情况三： $k \bmod 3 = 2$ 。设  $k=3p+2$ 。

分成三堆  $A\{\mathbf{S}, 1, 2, \dots, p\}$ ,  $B\{p+1, p+3, \dots, 2p+1\}$ ,  $C\{2p+2, 2p+4, \dots, 3p+2\}$ , 此时  $|A|=p+1$ ,  $|B|=p+1$ ,  $|C|=p+1$ 。特别注意在  $A$  中加入了一个标准球。称  $\langle A, B \rangle$  :

1.  $\langle A, B \rangle = \text{Middle}$ 。次品在  $C$  中，问题归结到  $n=p+1$  时的问题 5，根据归纳假设还要称  $\lceil \log_3[2(p+1)-1] \rceil$  次。所以总共称

$\lceil \log_3(2p+1) \rceil + 1 = \lceil \log_3(6p+3) \rceil = \lceil \log_3(2k-1) \rceil$  次。

2.  $\langle A, B \rangle = \text{Left}$ 。可能是  $AH$  或者  $BL$ ，问题归结到  $p+(p+1)$  时的猜想 II（因为标准球不要考虑，所以是  $2p+1$  而不是  $2p+2$ ），还要称  $\lceil \log_3(2p+1) \rceil$  次。所以总共要称  $\lceil \log_3(2p+1) \rceil + 1 = \lceil \log_3(2k-1) \rceil$  次。

3.  $\langle A, B \rangle = \text{Left}$ 。类似 2，总共称  $\lceil \log_3(2k-1) \rceil$  次。

综上， $k=3p+2$  要称  $\lceil \log_3(2k-1) \rceil$  次。

综合三种情况， $f(n) \leq \lceil \log_3(2k-1) \rceil$ 。

最优性和可行性均证，故而  $f(n) = \lceil \log_3(2k-1) \rceil$ 。

我们的结论是：

给定一架天平和一个标准球，从  $n$  个球中找出不知道轻重的次品至少要称  $\lceil \log_3(2n-1) \rceil$  次。

我们还可以把问题 5 中的标准球去掉（也就是变成本文一开始提出的问题

2），有兴趣的读者可以自己推导一下看看。



## 研究方法总结

本文的有关结论如下： $(n \geq 3)$

给定一架天平，有  $n$  个球，其中一个是次品。

结论 1 次品的重量比其他的重，称  $\lceil \log_3 n \rceil$  次就能找出那个次品。

结论 2 轻重不详。有一个标准球。称  $\lceil \log_3 2n \rceil$  次就可以找到次品，并且知道轻重。

结论 3 轻重不详。称  $\lceil \log_3 (2n+2) \rceil$  次就可以找到次品，并且次品的轻重。

结论 4 轻重不详。有一个标准球，称  $\lceil \log_3 (2n-1) \rceil$  次就可以找出次品。

当然，还可以变化出一些其他的形式，但是万变不离其宗，只要牢牢抓住以一个原则：“均匀”，任何此类问题都能迎刃而解。

本文全面而严格的提出了“天平称物”这一类问题的统一解法；在给出解法和有关结论的同时，特别重视对思路的产生过程作重点地阐述。

在研究此类问题的过程中我们得到了不少有益的经验：

1、从简单入手。这是本文的构篇基础。从简单的问题 1 步步深入，一直到较复杂的问题 5；从有标准球开始研究，进而推广到无标准球的情况；从要求次品的轻重情况，到只要求次品……这些无不包含着“从简单入手”这一重要研究手段。

2、总结经验，合理外推。通过将研究问题 1 而获得的部分经验进行合理的推广，我们在解决其后的难题时都能够很快的产生初步思路、迅速踏上正轨。

3、求同存异。世界上没有两片相同的叶子。哪怕是一个条件发生了细微的改动，题目的结构也许就发生了本质的变化。因此在推广的同时必须谨慎的考虑问题的相异性，做出积极的调整。

4、大胆猜想，严格证明。思路不甚明朗的时候，猜想就是一个很好的突破口。本文最重要的几个结论以及最后的问题 5，都是通过猜想取得进一步进展的。

## 附录

- 1、 本文作者针对问题 3，编写了一个简单的模拟程序。[Game.pas](#)
- 2、 判定树还可以用来证明：任何一个借助“比较”进行排序的算法，在最坏情况下所需要的比较次数至少为 $\lceil \log_2 n! \rceil$ ；根据斯特林公式，有 $\lceil \log_2 n! \rceil = O(n \log n)$ 。因此  $O(n \log n)$  是借助比较法排序的理论时间复杂度下界（快速排序、堆排序均是这个数量级的）。本文借助了这一思想，用来证明某些天平称物问题的最优性。具体的判定树内容可以在参考书籍的 P292 找到。

## 参考书籍

数据结构（第二版），清华大学出版社，严蔚敏 吴伟民 编著

## 染色法和构造法在棋盘上的应用

广东北江中学 方奇

### 1 基本概念

### 2 棋盘的覆盖

(1) 同行覆盖

(2) 异性覆盖

(3) 小结

### 3 马的遍历

(1) 马的哈密尔顿链

(2) 马的哈密尔顿圈

### 4 其它问题

(1) Warm world

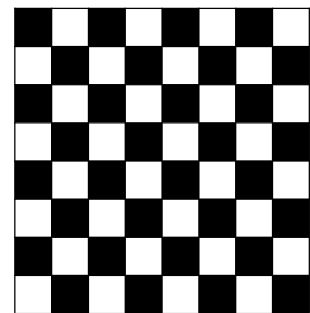
(2) 删除数字

### 5 结语

棋盘：

所谓  $m \times n$  棋盘，指由  $m$  行  $n$  列方格构成的  $m \times n$  矩形。每个方格成为棋盘的格，位于第  $i$  行  $j$  列的格记为  $a(i, j)$ 。当  $i+j$  为奇（偶）数时，称  $a_{ij}$  为奇（偶）格。

染色法：



用不同颜色将棋盘格子进行染色，起到分类的效果。特别地，类似国际象棋盘上的黑白二染色，我们称之为“自然染色”。

构造法：

直接列举出某种满足条件的数学对象或反例导致结论的肯定与否定，或间接构造某种对应关系，使问题根据需要进行转化的方法，称之为构造法。

棋盘的覆盖

指用若干图形去覆盖  $m \times n$  的棋盘。覆盖的每个图形也由若干格子组成，称为覆盖形。

约定任两个覆盖形互不重叠，任一覆盖形中任一格总与棋盘上某格重合。

按覆盖效果，可分为完全覆盖、饱和覆盖、无缝覆盖和互异覆盖。（只讨论）

完全覆盖：各个覆盖形的总格子数等于棋盘的总格子数

按覆盖形分，可分为同行覆盖和异型覆盖。

同形覆盖：只有一种覆盖形；

异型覆盖：有多种覆盖形

同形覆盖

例 1 给出  $m, n, k$ ，试用若干  $1 \times k$  的矩形覆盖  $m \times n$  的棋盘。

分析：

定理 1  $m \times n$  棋盘存在  $1 \times k$  矩形的完全覆盖的充分必要条件是  $k|m$  或  $k|n$ 。

证明：

充分性是显然的。用构造法。当  $k|n$  时，每一行用  $n/k$  个  $1 \times k$  的矩形恰好完全覆盖。 $k|m$  情况类似。

必要性：

1 2 3 ... K	1 2 3 ... k	.....	1 2 3 ... S
2 3 4 ... 1	2 3 4 ... 1	.....	2 3 4 ... S+1
3 4 ... .. 2	3 4 ... .. 2	.....	3 4 ... .. :
: : : :	: : : :	.....	: : : :
K 1 ... .. k-1	K 1 ... .. k-1	.....	k 1 ... .. S+k-1
1 2 3 ... K	1 2 3 ... K	.....	1 2 3 ... S
2 3 4 ... 1	2 3 4 ... 1	.....	2 3 4 ... S+1
3 4 ... .. 2	3 4 ... .. 2	.....	3 4 ... .. :
: : : :	: : : :	.....	: : : :
K 1 ... .. k-1	K 1 ... .. k-1	.....	k 1 ... .. S+k-1
: : : : :	: : : : :	.....	: : : : :
: : : : :	: : : : :	.....	: : : : :
1 2 3 ... K	1 2 3 ... K	.....	1 2 3 ... S
2 3 4 ... 1	2 3 4 ... 1	.....	2 3 4 ... S+1
3 4 ... .. 2	3 4 ... .. 2	.....	3 4 ... .. :
: : : :	: : : :	.....	: : : :
R r+1 ... .. R+k-1	r ... .. r+k-1	.....	R r+1 ... .. r+s-1
1	1	.....	1

设  $m=m_1 \times k+r, 0 < r < k$

设  $n=n_1 \times k+s, 0 < s < k$

约定  $r \geq s$

由上面的定理 1，可彻底解决  $m \times n$  棋盘的  $p \times q$  矩形完全覆盖问题

定理 2  $m \times n$  棋盘存在  $p \times q$  矩形的完全覆盖充分必要条件是  $m, n$  满足下列条件之

一：

(i)  $p|x$  且  $q|y$

(ii)  $p|x, q|y$ , 且存在自然数  $a, b$ , 使  $y=ap+bq$

其中  $\{x, y\} = \{m, n\}$

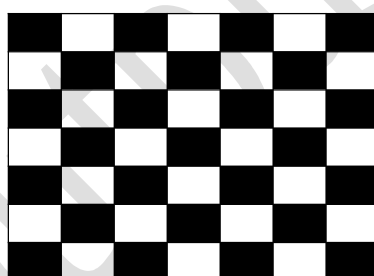
### 异型覆盖

例 2 设有  $m \times n$  的棋盘，当  $m \times n$  为奇数时，尝试删去一个格子，剩下部分用若干  $1 \times 2$  的矩形覆盖；当  $m \times n$  为偶数时，尝试删去两个格子，剩下部分用若干  $1 \times 2$  的矩形覆盖。

分析：

(1) 先来考虑  $m \times n$  为奇数的情况

一方面，将棋盘自然染色。无论怎么放，一个  $1 \times 2$  的矩形必盖住一个黑格和一个白格，而棋盘上的黑格比白格多 1，于是只能去掉一个黑格(即偶格)

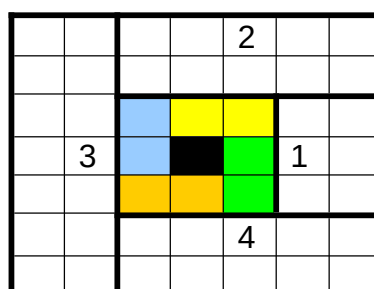


另一方面，设去掉偶格为  $a(i, j)$ ，用构造法必能得到可行解

1)  $i$  与  $j$  同为奇数



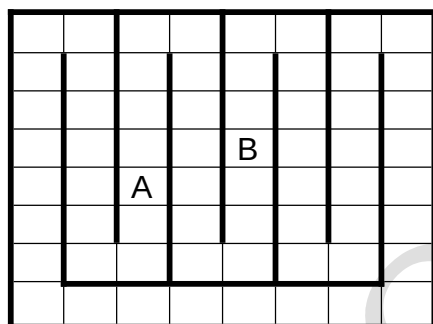
2)  $i$  与  $j$  同为偶数



## (2) 再考虑 $m \times n$ 为偶数的情况

类似地，由自然染色法得知，去掉的两格必定异色，即一个奇格，一个偶格（不然两种格子总数不等）

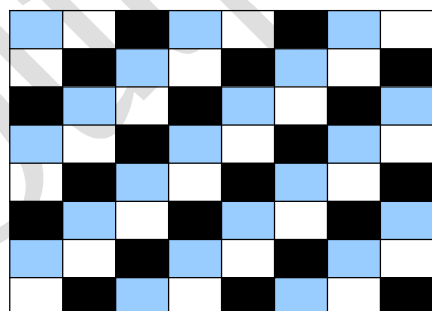
另一方面，用构造法，将用一些粗线将棋盘隔成宽为 1 的长条路线，使从任一格出发可以不重复地走遍棋盘并回到出发点。



针对染色法，上面的例子都是利用“各类颜色格子总数必须相等”这一条件推出矛盾，但有些时候，只考虑这个条件是不够充分的。

例 3  $8 \times 8$  棋盘剪去哪个方格才能用 21 个  $1 \times 3$  的矩形覆盖？

分析：



蓝色：21 个      白色：22 个      黑色：21 个

考虑到对称性，只有剪去  $a(3,3)$ 、 $a(3,6)$ 、 $a(6,3)$ 、 $a(6,7)$  中的某一个才能满足题意。

小结

覆盖类问题其实是一个难度较大的课题，这里只讨论了一些简单的情况，以说明染色法与构造法的应用

需要补充的是，染色法的种类形形色色、五花八门。考虑到可推广性和易操作性，本文只着重研究了“间隔染色法”（即自然染色法的推广）

### 马的遍历

马行走规则：从  $2 \times 3$  的矩形一个角按对角线跳到另一个角上

棋盘中马的遍历问题分两类

(1) 马的哈密尔顿链

(2) 马的哈密尔顿圈

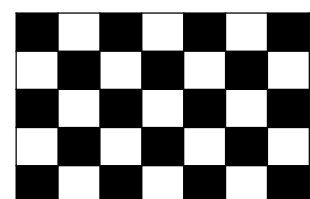
### 马的哈氏链

通常有四种方法

- 1 贪心法——每一步跳向度最小的点
- 2 分治法——将棋盘分成几个小棋盘，分别找哈氏链，再连接起来
- 3 镶边法——先在一个小棋盘中找到哈氏链，然后在棋盘四周镶边，已产生大棋盘的哈氏链。

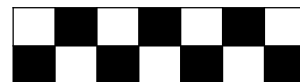
按上述方法不难得到下面结论

$n \times n$  棋盘存在哈氏链的充要条件是  $n > 3$ 。





## 马的哈氏圈



### 例 4 求 $n \times n$ 棋盘的哈氏圈

#### 分析

将棋盘自然染色，考察无解情况。

马无论怎么走，都必须按黑格 - 白格 - 黑格 - 白格 . . . . . 如此循环。由于要回到起点（起点与终点同色），途经两种颜色的格子数必相等，可知  $n$  为奇数时无解。

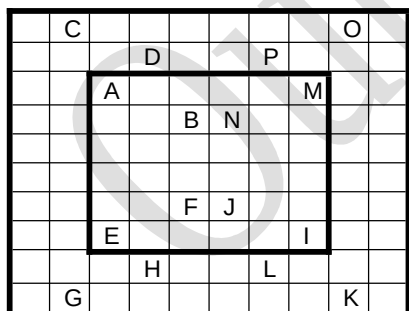
因为大小限制， $n < 6$  时也无解

当  $n \geq 6$  且为偶数时，用镶边法构造

假设  $(n-4) \times (n-4)$  的棋盘已找到哈氏圈

1)  $n$  除以 4 余 2 时，

在内矩形四个角（A、E、I、M）上分别开口。



1	1	1	2	7	4
	6	9	6		
2	2	2	5	1	2
0	5			8	7
1	2	1	8	3	6
5	6	7			
2	2	3	1	2	9
4	1	2	1	8	
3	1	2	3	3	1
5	4	3	0	3	2
2	3	3	1	1	2
2	1	4	3	0	9

1 将 C 与 D 所在的外回路与“内矩形”的回路在 A、B 上对接，变成 A-C- . . . -D-B。

2 将 G 与 H 所在的外回路与“内矩形”的回路在 E、F 上对接，变成 E-G- . . . -H-F。

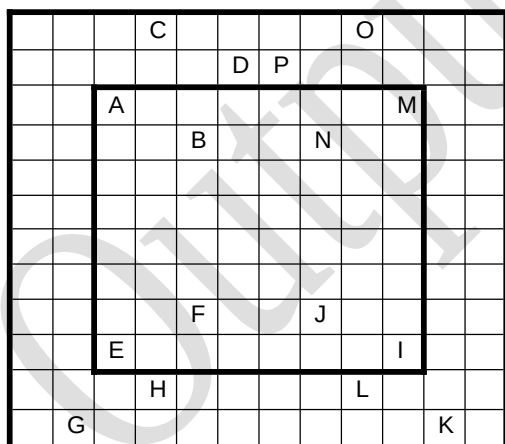
3 将 K 与 L 所在的外回路与“内矩形”的回路在 I、J 上对接，变成 I-K- . . . - L-J。

4 将 O 与 P 所在的外回路与“内矩形”的回路在 M、N 上对接，变成 M-O- . . . -P-N。

{ 在这里，要注意一个问题，就是作为基础矩形的“内矩形”的回路，首先要满足：A 的下一步到 B，E 的下一步到 F，I 的下一步到 J，M 的下一步到 N。只有这样，构造成的新矩形才能继续作为“内矩形”按上述规则向外扩展。现给出满足要求的基础矩形的一组解 (N=6) }

2) n 除以 4 余 0 时

在内矩形四个角 (A、E、I、M) 上分别开口。



1	54	47	38	49	52	31	26
46	39	2	53	32	27	22	51
55	64	37	48	3	50	25	30
40	45	56	33	28	23	4	21
63	36	61	44	57	20	29	24
60	41	34	15	12	5	8	19
35	62	43	58	17	10	13	6
42	59	16	11	14	7	18	9

1 将 C 与 D 所在的外回路与“内矩形”的回路在 A、B 上对接，变成 A-C- . . . - D-B。

2 将 G 与 H 所在的外回路与“内矩形”的回路在 E、F 上对接，变成 E-G- . . . - H-F。

3 将 K 与 L 所在的外回路与“内矩形”的回路在 I、J 上对接，变成 I-K- . . . -L-J。

4 将 O 与 P 所在的外回路与“内矩形”的回路在 M、N 上对接，变成 M-O- . . . -P-N。

一个猜想：

$m \times n$  ( $m \leq n$ ) 棋盘不存在哈氏圈的充要条件是：

$m, n$  满足下列条件之一

- (1)  $m, n$  都是奇数
- (2)  $m=1, 2$  或  $4$
- (3)  $m=3$  且  $n=4, 6, 8$

其它应用

例 5 蠕虫世界 (Uva)

蠕虫在一张  $N \times N$  的网上爬行。每个网格上有一个数字，蠕虫不能经过相同的数字两次。开始的时候，蠕虫任意选择一个格子作为起始点。它爬行只能沿水平或竖直方向，且不能超出网外。蠕虫如何移动才能到达尽可能多的网格呢？下面是一个样例。

6	8	18	15	24	20	2	20
6	2	15	2	17	15	3	7
0	11	18	16	20	15	1	11
6	2	6	13	4	17	20	16
5	12	7	2	3	5	18	23
7	13	3	2	2	11	4	23
16	23	10	2	4	12	5	20
17	12	10	1	13	12	6	20

分析：

采用“染色法”贪心出一个上界。

1 自然染色

2 设 Tfree, Tblack, Twhite 分别记录三类格子数量

对每一种数字 (1, 2, 3……) 分析

1) 只存在标有该数字的白色格子,  $Twhite \leftarrow Twhite + 1$

2) 只存在标有该数字的黑色格子,  $Tblack \leftarrow Tblack + 1$

3) 存在标有该数字的黑白两色格子,  $Tfree \leftarrow Tfree + 1$

3 估价上界

$$L_{\max} = \begin{cases} (Twhite + Tfree) * 2 + 1 & (Twhite + Tfree < Tblack) \\ Tblack + Twhite + Tfree & (Twhite + Tfree \geq Tblack) \end{cases}$$

(假设  $Twhite \leq Tblack$ , 否则交换即可)

结语

存在性问题——> 染色法

可行性问题——> 构造法

在以棋盘为模型的问题中，综合运用这两种方法，双管齐下，往往能收到事半功倍的效果！

谢谢

Output Only