浅谈信息学中状态的合理设计与应用

福建省福州第三中学　　刘弈

目录

浅谈信息学中状态的合理设计与应用 1
摘要 2
关键字 2
正文 2
一、　引言 2
二、　状态设计与应用的相关应用及例题分析 3
1、合理分析状态 3
例一、 square root 3
问题描述: 4
问题分析: 4
初步优化: 4
深入分析: 5
小结: 5
2、合理改进状态 6
例二、 banal tickets 6
题目描述: 6
问题分析: 7
初步分析: 8
进一步分析： 10
小结： 10
3、合理设计状态 11
例三、 shoot your gun 11
问题描述: 11
题目分析: 12
进一步分析： 14
深入思考： 16
小结： 16
三、　总结 17
附录 18

浅谈信息学中状态的合理设计与应用

摘要

状态分析与设计是信息学中一个重要的部分，在动态规划、搜索、枚举等众多算法中均有很多应用。对状态本身的分析与研究也是十分重要的，而本文就针对这个方面进行了探讨研究。

本文通过三个例题阐述了对于状态的合理设计在解题中所发挥的重要作用。

关键字

状态、分析、优化、改进

正文

一、 引言

在日常生活中，工作时间与工作数量、单位效率的关系可以用下面的这个式子来表达：

工作时间=工作数量*单位效率……①

（上式中的单位效率是指完成一个工作所需花费的时间）

在信息学中，程序的运行时间是由两个因素决定的：程序中所处理的状态总数和处理每个状态所花费的时间，因此程序运行的总时间可以用下面的式子来表示：

程序运行时间=状态总数*单位效率……②

众所周知，①式中的工作数量是无法减少的，因此大多数人选择了减少对每一个工作所花费的时间，即提高单位效率，从而减少工作总时间，达到优化的目的。从表面上看，式①与式②并没有什么本质区别，然而在信息学中，式②中的状态总数不完全等同于式①中的工作数量，因为不同的状态表示可能产生不同的状态量。合理的状态设计有时能有效地减少冗余，从而通过减少状态总数而达到优化的目的。此外，好的的状态表示还能够帮助编程人员理清思路，为我们的

解题带来崭新的思维方式，降低了解题难度。而对单位效率进行优化有时则无法达到这样的效果。

二、 状态分析与设计的相关应用

本文将重点介绍三道例题：《square root》、《banal tickets》、《shoot your gun》，从三个方面描述了如何根据问题的核心要求分析出问题的本质，并合理改进状态的设计与表示方法，由此带来解题新思路。

其中，例题一《Square Root》，通过合理地分析，减少状态总数，用朴素算法解决问题。例题二《Banal Tickets》，改进状态表示方法，通过对状态进行优化，取得良好的效果。例题三《Shoot Your Gun》，在常规状态表示法无法奏效时，重新设计了状态表示，极大的减少了状态总数。并由此得出了更为简洁的算法，成功的降低了编程复杂度及时间复杂度。

1、合理分析状态

人们在解决问题时，往往要先分析算法的复杂度是否能够满足题目的空间及时间的限制，之后才进行编程。如果在分析复杂度时出现错误，过高或过低地估计了算法的复杂度，都会直接影响问题的解决。复杂度估计过低，往往使得编程人员盲目下手，就算最后恍然大悟，发现算法不可行，也已经浪费了不少编程时间。如果不能及时找到有效的改进，往往就只能选择放弃该算法而另起炉灶。这样的错误在严格限定时间编程的信息学赛场上导致的后果是很严重的，甚至会直接影响选手的心理及比赛的结果；反之，如果复杂度估计过高，可能会使选手放弃本来可行的解法而另寻出路。由于这样一点分析上的失误而与正确的算法失之交臂，难道不是很可惜吗？下面的例子就是讲述如何通过分析状态总数做相应优化，消除其中的冗余，并正确判断与运用算法，减少程序运行总时间，提高效率，使问题得以解决。

例一、square root[1]

[1] 题目来源 acm.timus.ru/problem.aspx?space=1&num=1132

问题描述:

若整数 x 满足 $x^2 \equiv a \pmod n$，则称 x 是以 n 为模时 a 的平方根，记 $root(a,n)$ 为满足以上条件的 x 的集合。

题目包含 k 个询问 $(1 \le k \le 100000)$，每次询问给出 a 和 n $(1 \le a,n \le 32767)$，其中 n 为质数，且 a 与 n 互质，要求出所有在(0,n-1)区间内的 $root(a,n)$。

问题分析:

这题是一道模平方根(Modular Square Roots)的问题，有专门的数学方法来解决此类问题，如有兴趣，可以参考相关资料。

但是，解决模平方根的问题需要学习和掌握较高深的数学知识，若想在竞赛时推导数学公式，需要花费不少的时间。假如在推公式时遇到了阻碍，我们就会考虑用其他替代方法来解决这样的问题。

首先考虑最普通的方法——枚举。枚举的基本思路是穷举 x，计算 $value(x) = x^2 \pmod n$，如果 $value(x)$ 等于 a，那么就称这个 $x \in root(a,n)$。对于一个询问，枚举的复杂度为 $O(n)$，则整体复杂度为 $O(kn)$，在极限数据时，总枚举量可高达 30 多亿，显然无法在要求的时限内出解。经过上述分析，枚举法似乎很难满足题目要求。但枚举法是否就真的不可行呢？我们需要对状态进行合理的分析。

初步优化:

枚举法虽然无法直接解决这个问题，但是我们注意到这样一个信息：每次对一个询问 (a,n) 使用枚举法处理，实际上可以得到所有的询问 (m,n) 的答案，$(0 \le m < n)$。对一个询问花费了 $O(n)$ 的时间去枚举，却只利用了这中间的一小部分，浪费严重。那么，应该如何合理利用这些 “冗余”操作呢？

再次回顾题目，可以看到 k（即询问数）最多有 100000 个，但是 n 的取值只有 32767 个。显然，根据简单的鸽笼原理可得知，这些询问中最多出现 32767

个不同的n。而对于相同的n，可以对第 1 个询问进行枚举，之后的询问就可以$O(1)$得出结果。而只需要一个排序，就可以对所有询问进行分类。这样就成功地对原算法进行了优化。

但是即使如此，最多还是要处理 32767 个不同的询问，时间复杂度为$O(n^2)$，仍然无法在时限内出解。因此，我们还不能停下优化的脚步。

深入分析:

仔细分析枚举的算法，我们发现在枚举时并没有对题目条件进行合理的利用。题目条件中给出了这样的信息：n为质数，且a与n互质。题目给出这样条件就肯定有利用的价值，那么该如何利用这个条件呢？

注意到n为质数这个重要的信息，根据平时总结的经验，n以内的质数个数总会比n小很多，能否把这个作为突破口呢？

在使用筛法把 32767 中的质数都列出来后，发现质数的个数仅为 3500 个左右，比 32767 小了近 10 倍！

如此分析后发现，原算法的状态数其实没有$O(n)$，而仅为$O(prime(n))$，其中$prime(n)$表示 n 以内的质数个数，这样就可以大胆地采用枚举算法来解决此题，时间复杂度约为$O(k\log k + prime(n)*n)$，已经可以很好的满足了题目的时间限制。如果采取桶排，可以将时间复杂度降为$O(k + prime(n)*n)$，但是由于排序不是这题时间的瓶颈，所以不影响总的时间复杂度。

小结:

要想顺利解决数学题，一般需要很深的数学造诣，而对于大部分选手，这是十分困难的。但是信息学竞赛不同于数学竞赛，信息学竞赛中的许多数学问题并不是只有通过使用数学方法才能解决的，总会有一些替代算法。这些算法一般不能满足题目的限制条件。但这些做法并不总是一无是处的，只要我们细心发掘，也许成功就在不远处。本题的做法虽然不一定能适用于所有数学问题，但是却代

表了状态设计的思想。

在这题中，面对巨大的数据量，正当不知所措的时候，通过合理地分析了数据中状态的量，肯定了算法的效率。如果没有及时对状态进行分析，恐怕会陷入“优化”的“深渊”而无法自拔。此题的解法虽然简单，但是却给予我们很大的启发。

2、合理改进状态表示

动态规划是信息学中的重要算法之一，以其灵活多变的思维方式及状态表示，成为许多初学者难以逾越的一道难关。甚至对于一些已经在信息学竞赛中取得优异成绩的人，动态规划有时仍然显得那么高不可攀。

众所周知，动态规划的效率是由两部分决定的——状态数目及状态转移的效率。在以往大多数动态规划的题目中，优化状态转移效率的方法层出不穷，种类繁多，此前，许多优秀的选手已经对此进行过了许多细致的研究，例如 2001 年毛子青的论文《动态规划算法的优化技巧》，2007 年杨哲的论文《凸完全单调性的一个加强与应用》等等。在不断发掘优化转移效率的新方法的同时，关于状态总数——作为决定动态规划效率的另一重要部分，这方面的优化却鲜有人问津。不可否认，状态上的优化不像转移优化那样具有普遍性，有不少题目甚至是无法对其状态的表示进行优化的。但在某些特定的题目中，对状态的表示作必要的改进可以收到良好的效果。以下实例就是描述当对状态转移进行优化而无法解决问题时，我们应该如何处理：

例二、banal tickets[2]

题目描述:

给定一个长度为 $2*n$ 的数字串 $(1 \le n \le 18)$，数字串中有的位置数字是已知的，以 0..9 表示；有的位置的数字是未知的，以？表示。下图给出了一个长度为 4 的数字串：

[2] 题目来源 acm.pku.cn/JudgeOnline/problem?id=1608

2	?	?	2

当且仅当一个数字串满足以下条件时，称这个数字串 *interesting*，否则为 *banal*：

$$\prod_{i=1}^{n} s[i] = \prod_{i=n+1}^{2*n} s[i]$$

要求求出所有长度为 n 的 *interesting* 串和 *banal* 串的个数。

问题分析：

首先，不难发现，求 *interesting* 串的个数和求 *banal* 串的个数这两个问题是等价的，两者为互补关系。这样，就可以通过求其中的一个命题，来直接得到另一命题的解。而求 *interesting* 串的个数明显比求 *banal* 串的个数简单，因此只考虑求 *interesting* 串的个数的命题。

用 $dp[i][j]$ 表示前 i 位，乘积为 j 的方案数。不难得出这样的一组状态转移方程：

边界条件：

$$dp[0,0] = 1$$

当 $s[i] \neq ?$ 时：

$$dp[i, j] = \begin{cases} dp[i-1, \frac{j}{s[i]}] & (j \bmod s[i] = 0) \\ 0 & (j \bmod s[i] \neq 0) \end{cases}$$

当 $s[i] = ?$ 时：

$$dp[i, j] = \sum \begin{cases} dp[i-1, \frac{j}{l}] & (j \bmod l = 0, l \in [0,9]) \\ 0 & (j \bmod l \neq 0, l \in [0,9]) \end{cases}$$

设 dpa 表示对前半部分进行动态规划所得出的结果，dpb 表示对后半部分进行动态规划所得出的结果，则 *interesting* 串的个数为：

$$\sum_{i=0}^{m} dpa[n,i] \times dpb[n,i]$$

其中， m 为最大的状态数。

似乎，此题已经被很好地解决了。可是仔细分析下状态总数，不难发现，当 s 每位都取 9 时，总乘积最大，竟然达到了 9^{18}=150094635296999121。这么大的状态数根本存不下，更不要提在规定时间内出解！需要进行优化！

初步分析:

首先需要明确的是，这题中状态在转移时最多进行 10 次计算，因而，对转移进行优化并不能有效提高时间效率，我们迫切需要解决的是如何合理地改进状态表示来减少状态总量。

明确了这点后，就要花心思对状态表示进行优化。稍加分析不难发现，一类数例如：

$$11,13,14,17,19,22,23,26,28,29,31,33,34,37,38,39.....$$

这些数均不可能出现在状态中，但是这些数有什么共同点呢？质数？不完全是。光是看这些数可能不能很明显的发现一些规律，我们不如把可能出现在状态中的数字列举出来：

$$0,1,2,3,4,5,6,7,8,9,10,12,15,16,18,20,21,24,25,27.....$$

以上这些数字有什么共同之处呢？除了 0 以外，其他数字都能写成 $2^a3^b5^c7^d$ 的形式。仔细回想题目，状态 j 只可能是 0..9 这 10 个数的乘积，而 0..9 这几个数只包含 2,3,5,7 这 4 个质因子!!!

因此可以将状态改为 $dp[i,a,b,c,d]$，表示前 i 位，乘积为 $2^a3^b5^c7^d$ 的方案数，转移方程只需稍加改动：

边界条件：

$$dp[0,-1,0,0,0]=1$$

当 $s[i]\neq ?$ 时：

$$dp[i,a,b,c,d]=\begin{cases} dp[i-1,a-r(s[i],1),b-r(s[i],2),c-r(s[i],3),d-r(s[i],4)] \\ (a\ge r(s[i],1),b\ge r(s[i],2),c\ge r(s[i],3),d\ge r(s[i],4)) \\ 0 \\ (a<r(s[i],1)或b<r(s[i],2)或c<r(s[i],3)或d<r(s[i],4)) \end{cases}$$

当 $s[i]=0$ 时，

$$dp[i,-1,0,0,0]=dp[i-1,-1,0,0,0]+\sum dp[i-1,a,b,c,d]\quad (1\le 2^a3^b5^c7^d)$$

当 $s[i]\neq 0$ 时，

$$dp[i,-1,0,0,0]=dp[i-1,-1,0,0,0]$$

当 $s[i]=?$ 时：

$$dp[i,a,b,c,d]=\sum\begin{cases} dp[i-1,a-r(l,1),b-r(l,2),c-r(l,3),d-r(l,4)] \\ (a\ge r(l,1),b\ge r(l,2),c\ge r(l,3),d\ge r(l,4),l\in[0,9]) \\ 0 \\ (a<r(l,1)或b<r(l,2)或c<r(l,3)或d<r(l,4),l\in[0,9]) \end{cases}$$

$$dp[i,-1,0,0,0]=dp[i-1,-1,0,0,0]+\sum dp[i-1,a,b,c,d]\quad (1\le 2^a3^b5^c7^d)$$

因为 0 无法用 $2^a3^b5^c7^d$ 表示，所以用 $(-1,0,0,0)$ 替代 0 。其中， $r(l)$ 是一个四元组，表示将 l 分解为 $2^{r(l,1)}3^{r(l,2)}5^{r(l,3)}7^{r(l,4)}$ 时对应的系数。

采用这样的方法，就明显地减少了冗余状态，优化了状态表示。接下来再分析状态总数：

a 最多为 $3n$ (考虑全部数字为 8 的情况)， b 最多为 $2n$ (全部为 9)， c 最多为 n (全部为 5)， d 最多为 n (全部为 7)。所以当 $n=18$ 时，状态数目达到最大，为 12×18^4 (使用滚动数组)，转移为 $O(10)$，总的最坏时间复杂度为 $12\times 18^4\times 10=1259712$，总的最大运算量为1259712，不管时间还是空间都十分理想。

但是这样做真的就能 AC 此题了吗？

n 最多为 18，总的字符串长度为 36，因此总的数字个数为 10^{36}，需要使用高精度！

假如选择压 4 位(10000 进制)，那么高精度最大位数为 9，则运算时间和所需

空间均为原来的 9 倍，这样时间勉强能承受，但是这题的空间限制为 32MB，稍加计算即可知道不能满足题目要求。而选择大于 5 位的压缩方法，则必须使用 int64 强制转换运算类型(因为最后计算结果时需要用到高精度乘法)。由于 int64 的空间为 longint 的 2 倍，只有在压缩 9 位的情况下才能使空间略微减少，仍然超过了空间限制。

进一步分析：

我们的优化已经初见成效了，仅仅因为超了一点空间而选择放弃实在太可惜。这样的状态真的没有一点冗余了吗？让我们尝试列举 50 以内的所有状态，以期发现一些新的思路：

1，2， 3， 4， 5， 6， 7， 8， 9，10，11，12，13，14，15，16，17，18

19，20，21，22，23，24，25，26，27，28，29，30，31，32，33，34，35，36

37，38，39，40，41，42，43，44，45，46，47，48，49，50

上面的数列中，涂红色的数字是有用的状态，共 33 个。但是按照之前的状态表示，2 最多可能出现 5 个，3 最多可能出现 3 个，5 最多可能出现 2 个，7 最多可能出现 2 个，因此需要开的空间为 $(5+1)*(3+1)*(2+1)*(2+1)=216$，从这里可以看出，所用的状态表示仍然包含大量冗余！例如，对于 $2^{3n}3^{2n}5^n7^n$ 这个状态就不可能出现，因为 2^{3n} 就已经决定了 n 位数字全为 8，所以其他质数的个数不可能大于 0。

因此可以使用 *hash*，只保留所有可能出现的状态，就可以很好地减少了总状态数。具体编程只需要使用一个 *BFS* 的过程，从 $(0,0,0,0)$ 这个状态开始扩展，把所有有用的状态筛选出来然后用 hash 建立一一对应的映射关系即可。

经实践发现，对于 $n=18$ 的极限状态，使用 *hash* 可以将原来的 69 万左右的状态数减少到 5 万以下。这样，就能很好地解决此题了。

小结：

对于本题，很多人在对这题稍加分析，都会想到使用动态规划来解决。但是

当我们采用熟悉的动态规划不能在题目要求的时间和空间限制内出解时，我们应该怎么做呢？转换模型？优化转移？这两者都不失为很好的解题方法，但它们并非万能灵丹，有时还是需要依靠优化状态总数来解决问题。

如在解决这道问题时，转换模型并不是那么容易，优化转移则事倍功半，最终还是要通过对状态进行分析，改进状态表示，发现并减少了冗余，才能满足题目中空间和时间的要求。有时候，状态中的冗余并不是很容易发觉，需要通过细心的分析，果断的尝试，以百折不挠的精神来解决问题。

3、合理设计状态

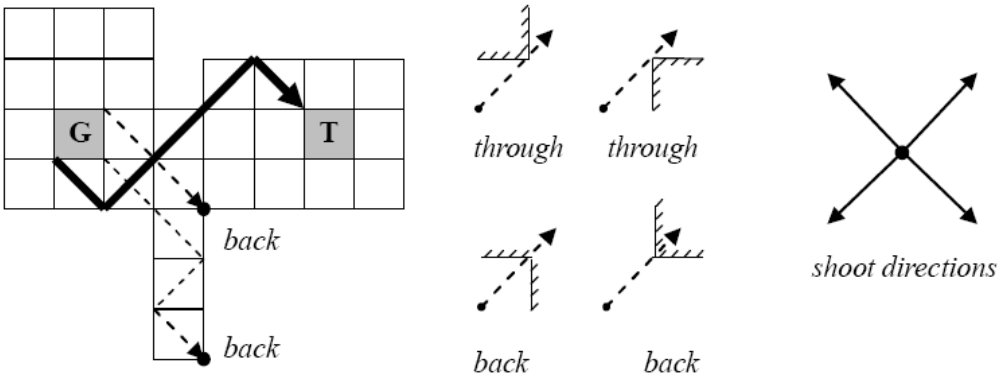
对于许多人来说，学习信息学是从学习经典算法开始的。了解及应用了经典算法之后，能帮助我们拓宽思维，举一反三。但需要注意的是，信息学中的题目是千变万化的，生搬硬套经典算法有时候并不能有效解决问题，这时就需要我们动脑筋思考，用创新的思维来改进已有的算法。学而不思则罔，思而不学则殆。我们不仅要学习某种算法本身，更要学习其中蕴涵着的思维方法。下面的例子就是讲述如何在经典算法遇到障碍时，合理地设计状态，巧妙地解决障碍，并最终以较低的编程难度实现较高的程序效率。

例三、shoot your gun[3]

问题描述:

定义 *rectangular polygon* 为一个仅包含 90 度或者 270 度内角的简单多边形，并且其边平行于坐标轴。已知，在一个大的 *rectangular polygon* *M* 中有两个小的 rectangular polygon *G* 和 *T* 。对于每一个格点，可以往其左上，左下，右上，右下四个方向发射出一条射线(如下图右)，射线在其轨迹路线上遇到障碍物，则遵循光路的镜面反射规律进行反射。具体的反射规则（如下图中）。图下图左描绘出了一个例子：

[3] 题目来源 The 2007 ACM Asia Programming Contest Beijing Site, Problem G

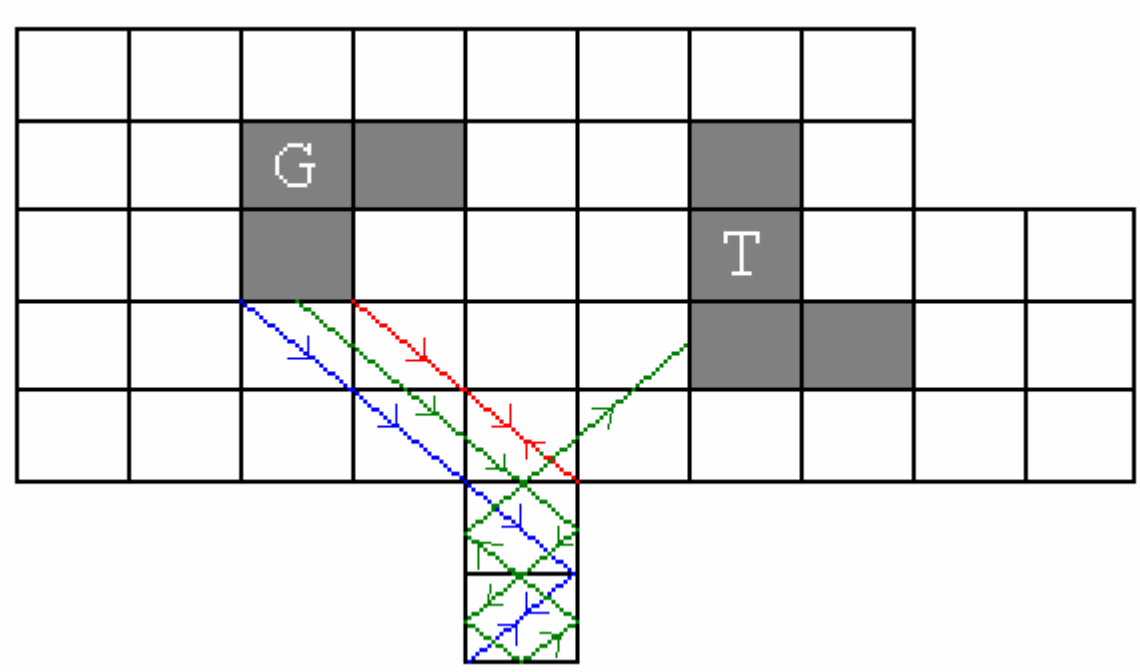


题目要求求出从 G 边上的任意一个位置到达 T 边上的任意一个位置的路径的最少反射次数，并且需要保证路径在到达 T 之前不能再次经过 G 。

一个 *rectangular polygon* 最多包含 50 条边，顶点坐标范围在[0..4000]。

题目分析:

需要注意到，题目中 G 边上的任意一点都可以发射出射线。如下图，分别用红线和蓝线代表一条边的两个顶点所发出的射线，用绿线代表边上除顶点外任意一点发出的射线。

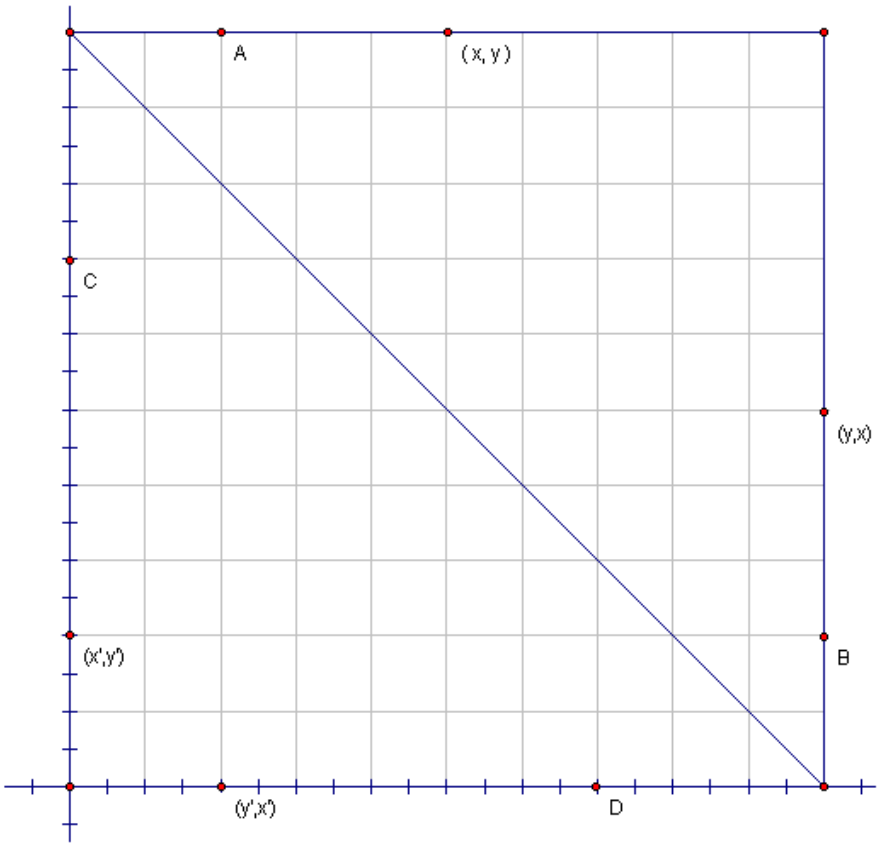


很明显，绿线所代表的非整点发射出的轨迹与红线与蓝线明显不同，因此是不能忽略的，否则可能导致最优解的缺失。由于一条边上有无穷多个点，枚举似乎是不可行。但是稍加分析后可以发现，图中的绿线，无论往上或往下平移，只要不碰到红线或蓝线，其路线轨迹上反射的次数都是相同的，也就是说这些非整点发出的射线对于问题的解是等价的。我们选择边的中点作为这些点的代表。这样，实际上只需要处理整点和中点，状态是有限多个的。

结论：对于同一条边上的非整点，其路线轨迹上的反射次数是相同的。

证明：

不妨考虑向右下发射的情况，其他方向同理可证。如图，以方格的左下角为原点建立平面直角坐标系：



图中从 A 点向右下方向发射的射线与方格的边交于 B，C 点向右下方向发射的射线与方格的边交于 D。更一般的非整点(x,y) 向右下方向发射的射线与方格的边交于点(y,x)。同样的，因此，对于所有在同一条边上的非整点(x,y)，会且只会落在该方格的另一条边上。即非整点只能到达非

整点，且落点在同一条边上。而落点所在的边上的点又可以同理证明其发射的射线在到达另外一个方格的边时，仍有如上性质。因此同一条边上的非整点发射出的射线轨迹上落点的个数是相同的，即反射次数相同。

命题得证。

由于这题中的点可以朝 4 个方向发射射线，简单地把每个点作为一个状态难以区分方向，因此可以采用经典的拆点法，把一个点拆成 4 个点，分别代表左上，左下，右上，右下四个方向，这样就能合理地表示状态了。

在选定了状态后，接下来是设计转移。设 $m[i][j][k]$ 表示 (i, j) 这个格子并且射线发射方向为 k 的状态 ($k=1..8$ 分别表示左上，左下，右上，右下角的点和上，下，左，右边的中点)。

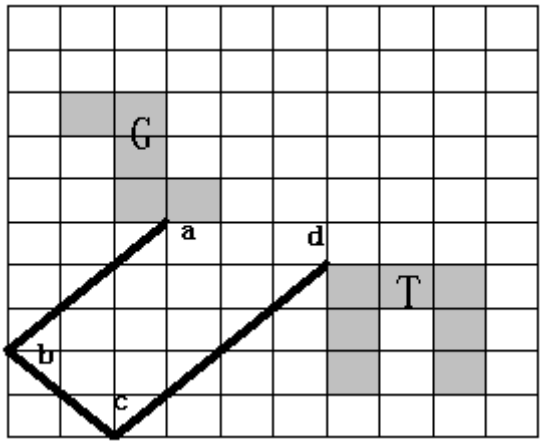
可以将所有状态构成一张图，相邻的格子上的点之间连边，问题就可以转化为求 G 集合到 T 集合的最短路。由于 $|V|$ 和 $|E|$ 同级别(V 为顶点集，E 为边集)，所以使用 *spfa* 或堆优化的 *dijkstra* 效果会比较好。但是考虑到顶点坐标的范围在 [0..4000]之间，所以状态总数最多可达 4000^2*8，就算使用 integer 类型的话， 也需要 $4000^2*8*2\approx 244\text{MB}$ 的空间，完全无法承受。

进一步分析：

之前的状态表示有很明显的冗余，例如 (i, j) 这个格子的右下角和 $(i+1, j+1)$ 的左上角是同一个点，但是我们却把这个点重复表示了好几次($(i, j+1)$ 和 $(i+1, j)$ 也会分别把这个点表示一次)。这样的表示方法造成了很大的浪费！

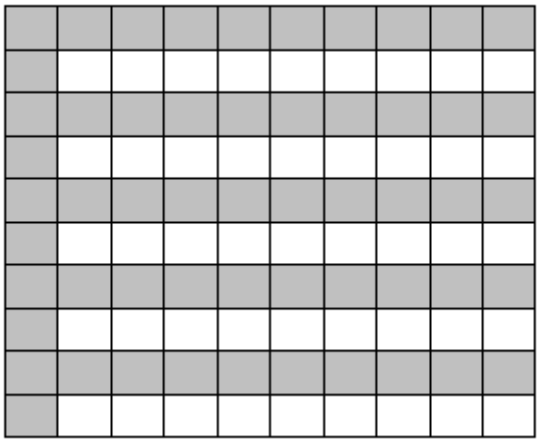
在我们排除了这些本质相同的状态后，最大状态数约为 4000^2*2，空间需要依然十分巨大(为原来的 $\frac{1}{4}$)，仍然不是很理想。要想彻底的优化，还必须在状态表示上多下些工夫。

此前的构图方法并没有用到题目中给的“线路轨迹遵循光的传播路线”这个条件，这个条件能给我们一些有用的启示吗？大家知道，光是沿直线传播的，只有在遇到障碍物时才会发生反射！例如下图：



图中加粗的边为射线的轨迹。

在这个图中，$a \to b$，$b \to c$，$c \to d$ 的路径中，中间的边上的转移是固定的，只有在路径遇到多边形的边界时才会发生不同的转移，而我们却仍多此一举，把这些点纳入状态表示范围，产生了大量的冗余状态。由于路径的变化是发生在多边形的边上，因此只需要处理多边形边上的点即可。这样状态数将大大减少。我们来估算一下改进后最坏情况下的时空复杂度。可以想象，当图形如下图所示时，边上的点数会达到最大，大约为 $\frac{50}{4}*4000*2$（每一个凸出来的部分有上下两个面，而构成一个凸出部分（图中灰色部分）需要 4 个点）。



这样，点数大概在 10 万左右，边数与点数同阶，如果使用 *spfa* 算法，所需空间仅为 $O(|E|)$，大约为 $2*100000$，大约为之前的状态总数的 $\frac{1}{160}$。而且，*spfa* 算法的期望时间复杂度约为 $O(|E|)$，这样的状态表示方法已经能够很好满足题目空间及时间的限制。

深入思考：

使用*spfa*算法虽然已经可以解决此题，但是考虑到*spfa*算法常数因子并不小，编程较为麻烦(构图部分)，有没有更简洁一些的方法呢？

我们对题目中“光路”的条件并未充分利用：光路是不会部分重叠的！要么完全不重叠，要么完全重叠。因此，根本不需要用 *spfa* 之类的最短路算法，只需要枚举起点，然后每次遇到多边形的边的时候模拟折射，直到到达*T* 集合。然后从中挑选一条折射次数最少的光路即可。这样做最多将每个点的 4 个方向枚举一次，因此总的最大计算量为 10 万*4。

伪代码如下：

1. 枚举起点(x,y)和方向 d，若全部起点和方向都已枚举过则退出该过程
2. 对于当前点(x,y)，对所有边 v 按从近到远的顺序进行枚举，
   找到第一条可以发生折射的边 k
3. 当前点(x,y)和方向 d 关于边 k 发生反射，并得出射线与边 k 的交点(x’,y’)和方向 d’
4. if 到达 *T* 集合
   更新反射的次数最优解并返回 1
   if 到达 *G* 集合
   返回 1
5. 替换当前点(x,y)为(x’,y’),替换当前方向 d 为 d’,并返回 2

这样做之后，程序实现起来十分简单，运行效率也很高。至此，我们很好地解决了此题。

小结：

这题是今年 ACM 预选赛北京赛区的题目，在比赛现场并没有人通过此题。但是，经过仔细分析后发现，程序实现的方法十分简单，并没有什么很复杂的证明及推理过程。但是为什么在当时没人通过呢？可能大部分人在看到这题都把题目想复杂了，而我们对题目进行了仔细的分析，合理地设计了状态，有效的去除

了冗余，并用简单的方法解决了此题。由此可以看到，对题目的进一步思考以及对状态进行合理设计，可发挥出重要作用。

三、 总结

状态优化的方法是基于对状态的表示和对题目条件的深入分析而设计的。优化状态并不是只需要单纯地使用 *hash* 就可以有效地减少状态表示的空间，减少冗余。事实上，这里还有不少的优化技巧。

在第一题中，数学方法应该是理论上的最优解决办法。但我们却另辟蹊径，通过对状态的分析成功地使用了朴素的枚举算法来解决。这也说明了合理地分析和设计状态可以让我们用一些简单的做法来解决更困难的问题。

在第二题中，我们很容易想到 *dp* 模型，但是规模十分巨大。我们通过合理分析题目条件，改进了状态表示，并使用 *hash* 进一步优化了冗余，这说明了状态表示的重要性以及选择好的状态对解题所发挥出的巨大作用。

在第三题中，常规的思路不能满足题目的时间及空间限制。但是分析了状态的实质后，我们把那些用处不大的状态给排除了，极大地减少了状态的数目，并很好地解决了问题。在最后的时候，又利用题目条件，巧妙地使用简单的方法来代替原来略显繁杂的方法，使原方法锦上添花。

对状态的合理设计与应用能帮助我们以简化繁、优化算法和理清思路。但是要做到合理地设计状态，合理地应用题目条件，就需要我们严谨地分析题目，对题目条件进行合理地组织与应用，还要有一点点创造性的思维以及不断累积的解题经验。当我们成功地解决题目后，要善于归纳总结解题思想，累积解题经验，才能更自信地走进信息学赛场。

感谢

感谢福州三中魏丽真老师的指导；

感谢集训队刘汝佳教练对论文修改提出的宝贵意见及帮助；

感谢清华大学杨沐同学及上海交通大学王航同学在论文修改中给予的帮助；

感谢孙林春前辈在论文修改中给予的帮助。

参考文献

《算法艺术与信息学竞赛》

附录

例一原题：

Square Root

Time Limit: 1.0 second
Memory Limit: 16 MB

The number x is called a square root of a modulo n (root(a,n)) if

$$x*x = a \pmod{n}$$

Write the program to find the square root of number a by given modulo n.

Input

One number K in the first line is an amount of tests ($K \leqslant 100000$). Each next line represents separate test, which contains two numbers a and n (a, n are natural, $1 \leqslant a$, $n \leqslant 32767$, n is prime, a and n are relatively prime).

Output

For each input test the program must evaluate all possible values root(a,n) in the range (0,n−1) and output them in increasing order in one separate line using spaces. If there is no square root for current test, the program must print in separate line: ‘No root’.

Sample

input	output
5 4 17	2 15 No root

3 7 2 7 14 31 10007 20011	3 4 13 18 5382 14629

Problem Author: Michael Medvedev

例二原题：

Banal Tickets

Time Limit: 5000MS **Memory Limit:** 32000K
Total Submissions: 693 **Accepted:** 59

Description

Peter is fond of number theory. That's why he is looking for interesting bus tickets. Ticket with the number of length 2N is called interesting if the product of the first N digits of its number is equal to the product of the last N digits. Other tickets are called banal.
Peter has found a used ticket in his pocket. Unfortunately the ticket was punched, so Peter cannot recognize some digits. He wonders whether this ticket was an interesting one. Moreover he wants to know how many different interesting and banal tickets could be punched to get this one.
Help Peter to find answers to his questions.

Input

The first line of the input file contains an integer N(1 <= N<=18). The next line contains a string representing the ticket number. If some digit is punched out it is denoted by "?" otherwise it is denoted by itself.

Output

On the first line of the output file print the number of interesting tickets. On the second line print the number of banal tickets.

Sample Input

2
2??3

Sample Output

4
96

Source

Northeastern Europe 2003, Northern Subregion

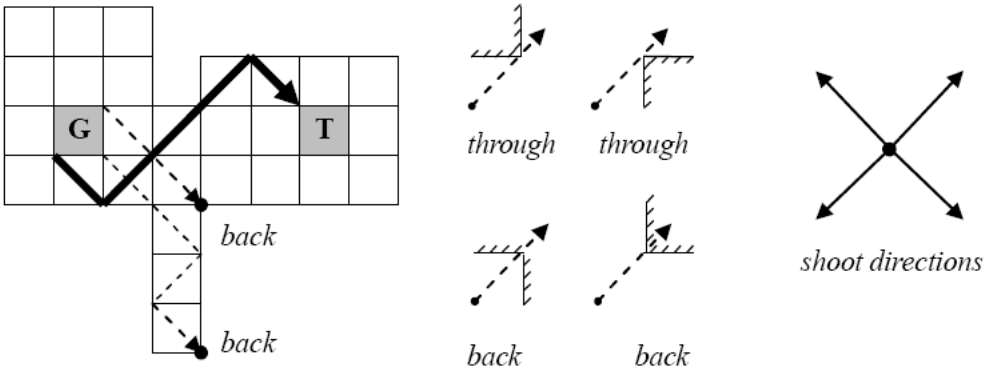
例三原题:

Problem G

Shoot Your Gun!

Input: gun.in

There are two rectangular polygons (simple polygons with interior angles of only 90 or 270 degrees) *G* and *T*, inside another rectangular polygon *M*. You can place a gun ***anywhere*** on the boundary of *G*, then shoot a bullet in one of four diagonal directions, and then touch the boundary of *T*. You may shoot across an edge of *T*, but touching only a corner is also allowed. Your bullet is not allowed to touch *G* again (even touching a corner of *G* is ***not*** allowed), before touching *T*.



The edges of *M* can reflect the bullet. When the bullet touches a vertex of *M*, it may simply go through it (and not regarded as a reflection), or go back. These special cases are shown in the figure above.
Write a program to find the minimal number of reflections needed from *G* to *T*.

Input

The input contains several test cases. The first line of each case contains three positive integers *nG*, *nT*, *nM* ($4 \le nG, nT, nM \le 50$). The next line contains *nG* pairs of integers, the coordinates (non-negative integers not greater than 4000) of the vertices of *G*, in counter-clockwise order. The next two lines describe polygon *T* and *M*, in the same format. It is guaranteed that *G* and *T* are outside each other (their boundaries will not touch), and are both inside *M* (they do not touch the boundary of *M*). The last test case

is followed by a single zero, which should not be processed.

Output

For each test case, print the case number and the minimal number of reflections to touch *T*. If it's impossible, output `-1`.

Sample Input

```
4 4 12
1 4 2 4 2 5 1 5
6 4 7 4 7 5 6 5
0 3 3 3 3 0 4 0 4 3 8 3 8 6 4 6 4 5 3 5 3 7 0 7
4 4 4
1 1 2 1 2 2 1 2
3 1 4 1 4 2 3 2
0 0 5 0 5 3 0 3
4 4 4
1 1 2 1 2 2 1 2
6 1 7 1 7 2 6 2
0 0 8 0 8 3 0 3
0
```

Output for the Sample Input

```
Case 1: 2
Case 2: 0
Case 3: 1
```