

# IOI2013 Day1试题讨论 及几题非传统趣题讨论

南京外国语学校 许昊然

给定一幅分辨率不超过500\*500的油画照片，要求判断这幅油画是题目给定的4种类型中的哪一种。题目免费附赠了36幅图片（每种类型9张）以帮助你测试自己的程序。

得分公式：

正确率小于25%：0分

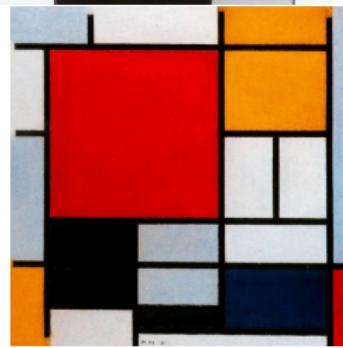
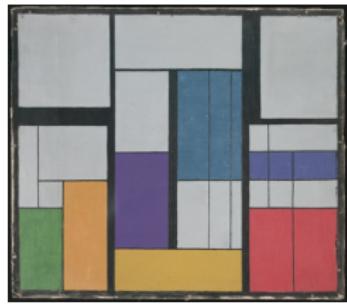
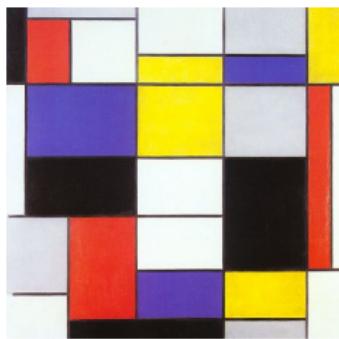
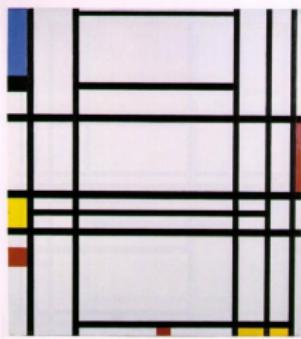
正确率25%~50%：0分~10分，得分与正确率线性关系

正确率50%~90%：10分~100分，得分与正确率线性关系

正确率90%以上：100分

下面是作品鉴赏时间.....

第一种可能的风格是：新造型主义现代画。



看起来像是铺地砖工人出身的画家创造的作品，因此简称为地砖画。

第二种可能的风格是：**印象派风景画**。



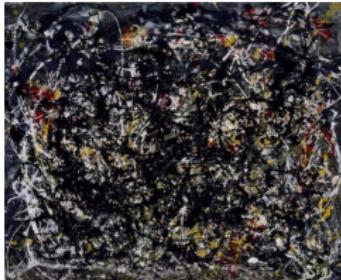
这个看上去很正常向啊，就叫风景画好了。

第三种可能的风格是：表现派细节组合画。



你们随意感受一下艺术家的世界.....

第三种可能的风格是：表现派细节组合画。



不知道在座各位有没有艺术细胞比较丰富的同学，能感受到这些作品中蕴含的美感……反正我是没感受到，于是就简称乱码画好了……

第四种可能的风格是：色块组合画。



如果说地砖画的作者是铺地砖工人出身的话，本类作品的作者大概是刷墙工人出身。于是就叫刷墙画吧。

讨论时间.....



朴素思路：随机返回1~4的一个值，多次提交取最优解。

时间、空间、编程复杂度：极小

期望得分：个位数。

更靠谱的做法？……  
观察四种风格的规律。

地砖画：主要由色块组成，色块内颜色十分均一，但色块间颜色可以差异很大；色块边界十分规整，色块间一般由深色线条分割。

风景画：有一个基础色调，各个像素点颜色大致差不多，但又不像地砖画或刷墙画那样整齐划一。

乱码画：杂乱无章，十分混乱。各种颜色的乱线交错在一起。

刷墙画：整幅画主要由三四块大块的颜色块构成，色块内部颜色比较均一。可能有一些细线条之类的点缀。

可以发现这四类画作中，乱码画显得十分突兀。其他画作要么有一个整体的“大致色调”，要么至少在局部有一个“大致色调”，其中的颜色都比较接近。而乱码画中充满了乱码一样的东西。

但是，怎样用计算机程序来进行判断呢？计算机程序没有像人类一样的视觉，显然是无法直接“看”出来的。

很容易想到，对于图片中每个点，计算它与它周围的一些点颜色的差距，然后利用平均数/方差等数据进行判定。

怎么计算两个点色差？

最简单的方法：

直接把RGB值视为三维坐标，算欧几里德距离或曼哈顿距离之类的。

当然也有一些更高端的方法：

有理论认为RGB色彩空间的表示方式与人眼感受色彩是有一定差距的，因此提出了一些新的表示方法。

例如HSV色彩空间、HSL色彩空间、LAB色彩空间等一听起来就很高端的东西。

据说转换成HSV色彩空间的表示法后再算欧几里德距离/曼哈顿距离效果更佳。

但对这题来说，直接用RGB表示法算色差就足够了。

我们不妨试验一下我们的策略对题目自带图片的效果。

效果如下：（考虑每个像素为中心 $7 \times 7$ 的点，欧几里德距离的平方作为差距估价，第一列为每个点与周围点RGB差值的平均值，第二列为标准差的平均值，四张图依次对应地砖画、风景画、乱码画和刷墙画）

style-1-0.in	2938	5176	style-2-0.in	2032	2618
style-1-10.in	7703	11318	style-2-1.in	1116	1388
style-1-11.in	3271	4784	style-2-2.in	3546	4122
style-1-3.in	1909	2564	style-2-3.in	1159	1468
style-1-4.in	2200	3102	style-2-4.in	3586	4365
style-1-5.in	1223	1745	style-2-6.in	1558	1880
style-1-6.in	2578	3718	style-2-7.in	5287	6457
style-1-8.in	2666	3182	style-2-8.in	3323	3960
style-1-9.in	7810	11461	style-2-9.in	3060	3553
style-3-10.in	13075	14954	style-4-15.in	204	286
style-3-11.in	8741	9756	style-4-16.in	21	26
style-3-1.in	10151	11366	style-4-1.in	392	465
style-3-2.in	10240	11774	style-4-2.in	284	356
style-3-4.in	14918	16196	style-4-3.in	440	560
style-3-5.in	9911	11732	style-4-4.in	237	262
style-3-6.in	13958	16535	style-4-7.in	61	70
style-3-7.in	20259	22459	style-4-8.in	94	100
style-3-8.in	16996	19135	style-4-9.in	101	128

可以发现，刷墙画的两个参数明显地小于其他三类画（均远小于1000），因此可以直接用这个标准来判断刷墙画。

但是其他三类画并没有明显拉开差距（虽然乱码画的参数值明显比较高，但乱码画中参数值较低的与地砖画、风景画中参数值较高的并没有明显拉开差距）。因此我们需要寻找更合适的判定标准。

进一步观察可以发现：

风景画中每个像素点与周围像素点RGB值普遍有差距但差距都不大

地砖画中每个像素点要么与周围像素点RGB差距很小（色块内部），要么与周围像素点RGB差距很大（边界）

乱码画中每个像素点与周围像素点RGB差距普遍比较大

新策略：统计每个点到周围点的RGB差距值中，大于某个值的差距所占的比例。

试验一下效果（各列分别表示RGB差距

在1000、2000、3000、4000、5000以上的像素点对占的比率）：

style-1-0.in	0.089	0.081	0.076	0.072	0.069
style-1-10.in	0.119	0.110	0.105	0.102	0.099
style-1-11.in	0.067	0.061	0.057	0.055	0.053
style-1-3.in	0.128	0.110	0.098	0.088	0.081
style-1-4.in	0.105	0.087	0.077	0.071	0.066
style-1-5.in	0.069	0.058	0.051	0.047	0.044
style-1-6.in	0.082	0.067	0.062	0.058	0.055
style-1-8.in	0.156	0.137	0.125	0.116	0.109
style-1-9.in	0.138	0.123	0.116	0.112	0.109
style-2-0.in	0.293	0.191	0.142	0.112	0.093
style-2-1.in	0.223	0.128	0.087	0.064	0.049
style-2-2.in	0.492	0.355	0.279	0.228	0.191
style-2-3.in	0.199	0.118	0.084	0.065	0.052
style-2-4.in	0.422	0.296	0.231	0.189	0.160
style-2-6.in	0.276	0.171	0.122	0.093	0.074
style-2-7.in	0.489	0.367	0.300	0.254	0.222
style-2-8.in	0.430	0.293	0.223	0.180	0.150
style-2-9.in	0.482	0.330	0.248	0.196	0.159
style-3-10.in	0.710	0.619	0.558	0.512	0.474
style-3-11.in	0.706	0.601	0.529	0.473	0.427
style-3-1.in	0.723	0.613	0.541	0.486	0.442
style-3-2.in	0.720	0.613	0.540	0.485	0.440
style-3-4.in	0.789	0.697	0.633	0.583	0.541
style-3-5.in	0.691	0.574	0.499	0.443	0.399
style-3-6.in	0.694	0.602	0.543	0.498	0.462
style-3-7.in	0.778	0.701	0.648	0.606	0.570
style-3-8.in	0.764	0.683	0.626	0.582	0.544

我们发现，这三类图片在这几个特征值的大小上并没有完全拉开差距……

但是我们可以发现，地砖画和乱码画的5个参数值都比较接近，而风景画中，5个参数值相对大小变化的很明显，第一个参数值都达到了最后一个参数值的好几倍。这个是因为风景画中颜色渐变相对平滑，所以颜色差距在 $1000\sim2000$ ,  $2000\sim3000$ ,  $3000\sim4000$ ,  $4000\sim5000$  的像素点对都有不少。

而地砖画和乱码画中，颜色要么几乎一样（地砖画色块内、乱码画线条内），要么差距极大（地砖画地砖边界处，乱码画各个线条之间），导致大部分差距较大的点对差距都在5000以上，所以从第1个参数到第5个参数变化比较小。

因此我们得到了风景画的判定：如果5个参数值在合理的范围内，且第一个参数达到了第5个参数的若干倍（我考场上写的是2倍），那么就判定为风景画。

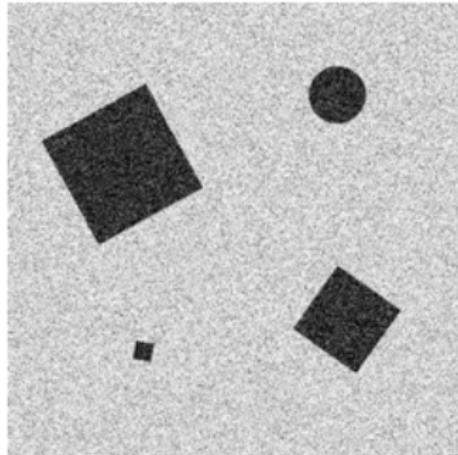
最后只剩下地砖画和乱码画的判定了。我们发现，在排除了风景画后，这5个参数值差距已经拉的非常明显了，直接设一个标准进行判定即可。

我考场上基本按这个思路写的判定，第一次提交就做到了98%的准确率（120幅画中只错了2幅）。

# Codeforces 178E The Beaver's Problem

给定一张图片（边长不超过2000像素），白色背景上有一些黑色的圆和正方形（可能有旋转），但加入了噪音（每个点有20%概率被翻转成相反的颜色）。

问图片中有多少个圆多少个正方形。保证正方形的边长/圆的直径不小于15像素，任意两个图形间隔不小于10像素，图形总数不超过50个，且图片用肉眼可识别。



假如没有噪点？

Floodfill每个连通块

每个连通块内，试图找到距离最远的两个点，它们构成了正方形的对角线或圆形的直径

进而估测出正方形/圆形的面积，与实际面积相比较，确定是正方形还是圆形

(距离最远的两个点不太好找?)

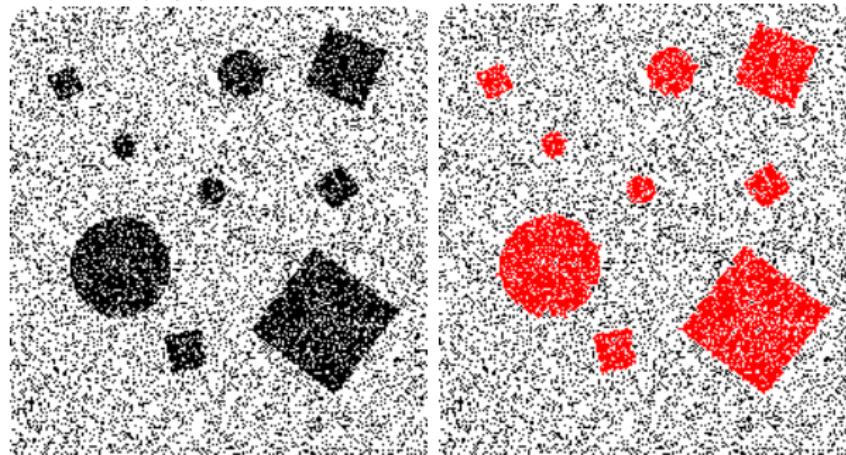
(实际只要找距离重心最远的那个点就行了.....)

# Codeforces 178E The Beaver's Problem

有噪点怎么办？

无视噪点直接Floodfill，最后面积除以0.8？

确实，仅仅20%的随机颜色翻转几乎不可能把一个图形分成两块，或把两个分离的图形连起来（注意直径/对角线至少15像素，且图形间隔至少10像素）



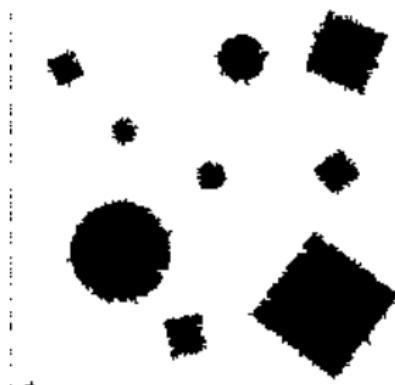
# Codeforces 178E The Beaver's Problem

但是这么做实际效果很糟糕，难以正确识别比较小的图形（事实上连样例都过不了）。

考虑如何降低噪声对结果的干扰。

最简单的策略：直接Floodfill，如果连通块大小小于某个阈值，直接视为噪声，反色。

实战效果：能够去除大部分的噪声，但有效图形和背景的边缘部分依然很糟糕。



更好的方法？使用一个听起来很高大上的技术：高斯模糊（Guassian Blur）

回忆一下，大家刚看到这幅充满噪声的图时，是否本能地眯着眼睛去数圆和方块的个数，以尽量减少烦人的噪音干扰？

人的眼睛相当于一个透镜，眯着眼睛实际相当于刻意避免完美对焦，让原图中每个点实际投影到视网膜的一片区域而不是一个点上。

高斯模糊实际就是模仿了这个过程。把每个点的色值变成其周围的点的色值按某种方式**加权平均**得到的结果。大家可以想象一下，这样一来，因为噪音是随机的，背景里的每个点周围噪点数量都差不多，从而都会变成颜色差不多的点；前景里的噪点同理。而真正的前景不会受到太大影响。

# Codeforces 178E The Beaver's Problem

具体方法是这样的：首先选择一个值 $\sigma$ , 表示对每个点考虑其周围行差距和列差距不超过 $\sigma$ 的像素点。

然后设定一个 $(2\sigma + 1) * (2\sigma + 1)$ 的权值矩阵 $G$ , 表示各个像素点对中心点贡献的权重, 矩阵里所有权重的和应当是1 (以防止图片颜色越来越深或越来越浅) 。

0.04	0.04	0.04	0.04	0.04
0.04				
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04

例如左图就是一个可行的矩阵 ( $\sigma = 2$ )

最后像上文所说的, 对每个点算一下带权平均数就可以了:

$$a'_{i,j} = \sum_{|x|, |y| \leq \sigma} a_{i+x, j+y} * G_{x,y}$$

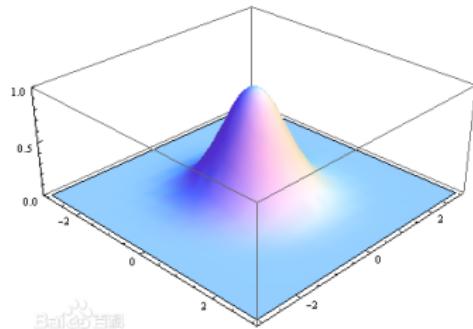
# Codeforces 178E The Beaver's Problem

权值矩阵里填的内容？

方法一：随手敲一个，看着舒服就行了，效果不好就再敲一个。

方法二：每个元素都取  $\frac{1}{(2\sigma+1)^2}$  (相当于不给权重，直接进行平均)

方法三：可以想象靠近中心点的点权重应该比距离较远的点权重更大一些。不妨用正态分布公式来建立矩阵  $G(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$



上图是运用正态分布公式建立出的权值矩阵直观的样子。

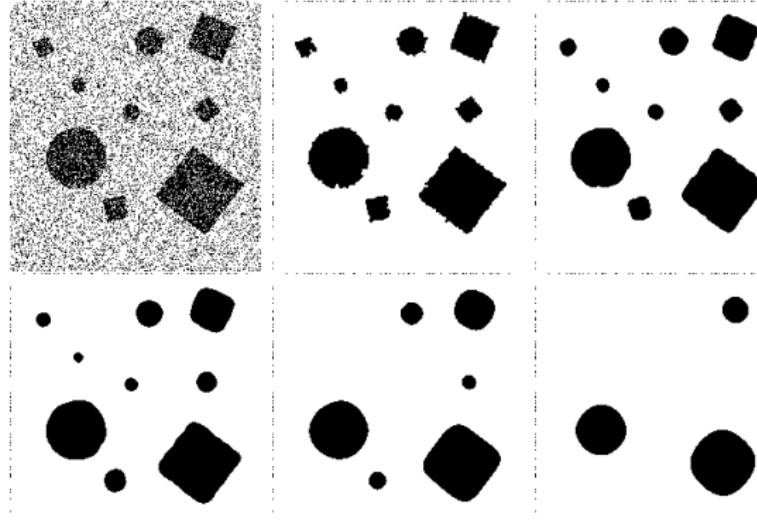
# Codeforces 178E The Beaver's Problem

权值矩阵的大小？

$\sigma$ 太小，去噪效果不佳，给判定算法带来麻烦。

$\sigma$ 太大，对图片边缘损坏严重，给判定算法带来麻烦。

通过试验和实际情况选择合适的 $\sigma$ 值。



上图分别是 $\sigma = 0, 1, 2, 3, 5, 10$ ，以正态分布公式建立矩阵进行高斯模糊的效果。可以发现取 $\sigma = 1$ 是效果最好的。

降低噪声后，再使用之前的算法（对每个连通块找到距离最远的两个点作为对角线/直径，估测面积，与实际面积比较）来判，就可以过掉Codeforces上的数据了。但这个算法仍然过不掉tsinsen版本的变态数据（即使加上一些随机化乱搞和调参数也只能得到95分左右）。

更优秀的判定方法？

考虑把每个图形的边界求出来（对每一列，记录下最高和最低的黑色像素），然后计算重心到边界各点的距离。

如果是圆，距离应该都差不多；如果是方块，距离应该有较大的差异。利用最大最小值的差/方差等进行判定

因为考虑了更多的信息，判定更准一些。

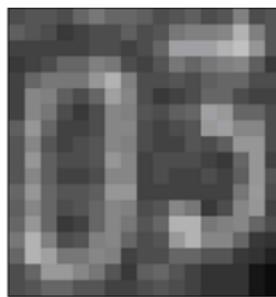
其他的一些想法？

对每个连通块求一个凸包，可以想象如果是正方形，凸包上的点应该很稀少；如果是圆形，凸包上的点应该很多。

通过实践才能得知各个算法的优劣。

# HNOI2007 所罗门的咒语

有一张图片，深色背景上面写了一行浅色字，字体字号一样，没有重叠没有扭曲，不超过3个字符，且字符只可能是阿拉伯数字或A,D,E,L,X。现在要求你识别图片中写了什么。  
文字可能有轻微旋转，也可能有少量噪音。



例如，上图分别为03、038、A、E。

与上一题一样，我们首先应该试图把文字从背景里提取出来，然后再考虑如何识别它是什么字母。

如何提取？

背景比较黑，前景比较亮……设一个阈值，灰度小于它的设为黑，大于它的设为白。

怎么比较科学地取这个阈值呢？直接猜一个显然是不靠谱的……

再引入一个听起来很高大上的东西：OTSU方法。

思想：背景和前景之间的颜色差距比较大，因此应该选一个阈值 $k$ 把灰度分成两类，使得这两类的类间方差尽可能大。因为如果有前景被错误的归类到背景中，或背景被错误的归类到前景中，都会导致类间方差变小。

数学化的表达：设 $\omega_1$ 为第一类（前景）的元素个数， $\mu_1$ 为第一类元素的平均灰度；设 $\omega_2$ 为第二类（背景）的元素个数， $\mu_2$ 为第二类元素的平均灰度； $\mu_T$ 为全体元素的平均灰度。那么我们应当最大化：

$$\omega_1 * (\mu_1 - \mu_T)^2 + \omega_2 * (\mu_2 - \mu_T)^2$$

实现应该很简单吧，预先排序后扫一遍就可以了。

OTSU二值化处理的效果:



然后我们试图把这些字符分离开来，以进行识别。

如何分离？

用一个土办法就行了：统计一下每列有多少个像素是白点，把连续的有白点（或白点比较多）的列视为一个字母。

因为字符没有重叠或粘连，这么做的效果已经足够了。

如何识别字符？

最简单的想法：对每个字符写一个判定。

比如说我们考虑如何识别字符“L”吧。首先我们大概可以在左下角的地方枚举一下，确定那个拐点的位置。

嗯嗯，然后我们往上面和右边扫，看看能不能找出“L”的横线和竖线。

嗯，从图片里识别出一条线，有点难度的样子。不过大概判定写详细一些还是可以的。

然后把识别出的部分删掉吧。看看删掉后白点是不是已经基本没了，如果答案是肯定的，那就应该是L了。

嗯嗯，然后我们来看下怎么识别字符“2”吧。我们大概得先想办法识别出顶上那条弧线。

这个好像有点难度的样子……试试看枚举圆心？嗯，然后……



什么？考试已经结束了？

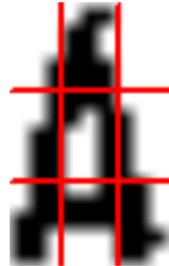
为每个字符都写一个判定方法显然是不可行的.....

不仅时间上不可行，即使真的写出来了，恐怕效果也会很糟糕。几个噪点、一点轻微的旋转都很可能会给这些判定算法造成巨大的麻烦。

回想一下我们怎么解决我们最初讲的Artclass这道题目的。我们显然没有教电脑去感受艺术，只是让电脑分析了作品的几个特征值，并以此判定。

注意到字符只有0123456789ADELX这几个。这些字符长得都不太像，如果能提取出几个合适的特征值就能相对靠谱地识别出来了。

我们不妨考虑切成9块，统计“上层、下层、左侧、右侧、中间的白点所占的比例”这5个值作为特征值。



预先处理出0123456789ADELX这些字符的特征值，选择最接近的一个作为答案。

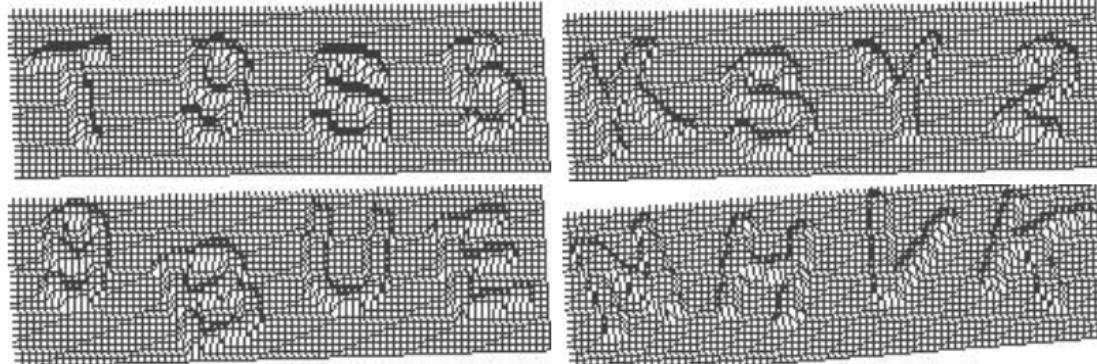
通过适当地对着数据调参数，可以过掉本题。

# 一个有趣的东西

在上一题中，我们已经实现了一个简陋的OCR（图片文字识别），虽然它只能不太靠谱地识别出几个简单的字符。

下面我们来挑战一下更困难也更有趣的任务：识别网站的验证码。

首先我尝试了识别中国国航网站的验证码：



基本信息：图片大小为80像素\*240像素，验证码由大写字母和阿拉伯数字组成，但没有I和1，O和0等过于容易混淆的字符。如图所见，验证码的形式是网格上“浮起”了4个字母，网格和字母有小幅旋转，但字母没有重叠，且字体、字号固定。

# 一个有趣的东西

首先，与之前一样，我们应该去除噪音，提取出真正的文字部分。那张网格肯定很碍眼，我们应该想办法把它去掉。

观察若干验证码的图案后，可以发现，所有网眼的大小都是4或者3（因为错位）。

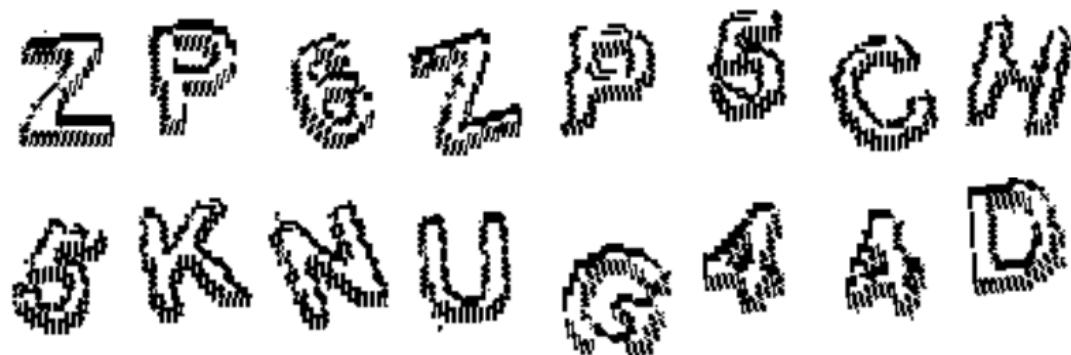
因此，如果一个白色联通块的大小既不是3也不是4，那么说明它很可能验证码文字的一部分，否则就很可能只是网眼。

还可以发现，大小小于3的黑色四联通块、大小为 $2*2$ 的黑色方块也很可能是文字的一部分，其余的很可能只是网格线。

# 一个有趣的东西

我们只保留这些可能是文字的部分，把其余部分全删光，然后删除图片中残余的过于稀疏的黑点（我选择的是 $7 \times 7$ 矩形中不超过4个黑点就删掉， $11 \times 11$ 矩形中不超过10个黑点就删掉，当然这也是根据实际试验结果得到的方案），进一步降低噪音。

这时候图片已经非常可看了：



# 一个有趣的东西

然后我们需要把字符分割开来。这道题中字符分割比上一题还要简单：只要等分成4块就行了。

接下来就是识别了。直接用上一题粗制滥造的画九宫格法进行识别肯定是不行的，可以想象这种方法在分辨5和S等比较接近的字符时会有多无力。

我用的识别方法是这样的：因为我们已经能分割出字母了，直接找100个字母人工进行识别，作为样本。

然后对每个要判定的字母，找到样本中最接近的字母作为答案。

# 一个有趣的东西

怎么判定最接近的字母？

字母有旋转，位置也不一定相同，我们需要找一种比较优的匹配方案。

我们观察到，字符大小都是相同的。因此，很容易想到，找到每个字符的重心，在匹配两个字符时，最优的匹配应该让重心重合。

然后，我们可以枚举一个旋转角度。这样，我们就只剩下确定两个字符的相异程度这个问题了。

很容易想到的估价是，对于两个字符中所有有色像素点，找到另外一个字符中与其最接近的像素点，以其距离作为估价。得到了一组距离后，用它们的和/平均数/标准差等得到最终估价。

经过尝试，可以发现用“它们的和”作为估价是一个比较好的选择。并且，可以发现如果有点根本不能在近距离匹配上，那么这个字母不太可能是答案。为了突出这一点，我设定以所有距离的 $2.5$ 次方后的和为估价。这样即使很小的不匹配也能造成比较大的答案差距。

# 一个有趣的东西

上述算法已经能识别出绝大多数字符了，但仍然偶尔会错，经过测试可以发现错误主要集中在对“5”和“6”，“8”和“9”等相近字符的识别上。



我当时尝试了很多方法，但都效果不佳。后来尝试了一个很好玩的方法：在原图中，因为是从网格上“凸起”了一些字符，这些字符是有“阴阳面”的！也就是说，字符凸起的侧面有面向我们的，也有背对我们的。我们可以尝试利用这个信息，在匹配时考虑只允许阴面匹配阴面，阳面匹配阳面。

# 一个有趣的东西

可以想象，字符背对我们的侧面从我们的视角看，应该是一团阴影。

进一步观察发现，大小 $2 \times 2$ 的黑色方块很可能是文字的“阴影”造成的，我们把这些方块标记为红色。



# 一个有趣的东西

加上阴阳面的设定后，在匹配寻找最近点时，应该更优先找同色点。也就是说，异色的点虽然也可以作为答案，但要承受一个额外的代价，我设定的是距离会额外乘2.5。



加上这个设定后，程序便可以无压力分辨出5和6、8和9之类的相似字符了。

还可以加一些优化，比如在快速找最近点时可以用k-d树加速等，或者可以考虑对准重心后，再按照比如“从重心出发包含最多黑点的射线”等方式对齐，以省掉枚举旋转角的时间等等。

最后可以做到98%以上的识别正确率。

# 更大的挑战？

前段时间我又尝试了识别12306的验证码，示例图片如下：



基本特点：图片大小只有26像素\*78像素，字符集包括大小写字母和阿拉伯数字。字母粘连严重，有旋转，有缩放。

# 更大的挑战？

这个问题我也没有想到特别好的解决方法。下面是我尝试过的一些思路：

1. 第一个字符左半部分绝大多数情况都是“干净”的，直接用上道题的方法识别第一个字符的左半部分就能读出第一个字符。
2. 第二个字符与第三个字符粘连一般不严重，使用一些启发式的方法（比如按某种方式设权值找一条从顶部到底部的最短路等）可以将它们大致分离开。如果分离成功可以基本正确读出第三个字符。

但因为第二个字符几乎总是与第一个字符严重粘连，即使能读出并移除第一个字符，也往往会影响第二个字符的很多特征。加上图片特别小，读第二个字符错误率很高。第四个字符情况也差不多。

最后只做到了大约10%的准确率，大部分都是识别出了粘连不严重的情况。与新闻中说的直接用现成OCR软件破解正确率差不多。如果同学们有更好的想法欢迎与我交流。

# 网站应该怎么做？

思考：为了增加验证码的强度，可以采取什么措施？

12306曾经采取过动图验证码的措施，但我个人认为这个措施反而降低了强度：在动态验证码中每个字符是不同颜色显示的，给程序分离字符串带来了巨大的方便。同时，动图对程序不但没有任何害处，反而相当于给程序一次提供了很多幅不同角度拍的图片进行分析，恐怕反而能增加程序的识别率。倒是真正老老实实用眼睛的用户面对眼花缭乱的动态验证码不知所措了。

还有一些网站使用汉字作验证码，比如用一到九的繁体字作为验证码。这样看似难识别了，但如果总共就几个汉字的话，只要识别出任意一小部分属于哪个字就可以确定这个字了（以前看过一篇破解天涯的繁体数字验证码就用的这个方法）。还有一些网站（比如百度贴吧）直接找四个汉字让用户输，但这样往往很影响用户体验，弄到最后还是妥协成提供几个选项让用户去选答案，但这样又为识别提供了方便。

# 网站应该怎么做？

我设想过几条策略来增强验证码的强度：

1. 字符使用相同的颜色，不让程序能从颜色上分离字符。
2. 在人眼可识别的前提下，尽可能重叠、粘连字符，增加分离字符的难度。
3. 字符要扭曲、缩放、旋转、用多种不同字体，最好用罕见字体、空心实心字体混用等，给样本搜集、模式匹配尽可能制造麻烦。
4. 如有必要还可以加入噪音、干扰线等，并尽可能缩小图片。

# 我们应该怎么做？

有很多更高端的方法可以更有效的识别验证码，如图形上下文、神经网络、深度学习等听起来就很神的算法。

同时，我们作为攻击方，可以充分发挥人类智慧，发现和利用设计者的疏忽。

图像识别领域也是计算机科学近年研究的热点，不断有新的识别方法和应对方法出现，这是一场没有止境的攻防战。

你需要编写一个交互式的程序，该程序需要猜测10000段来自维基百科上的摘录（每段摘录100字符）的语言类别。

每次系统会给你一段摘录，你的程序要猜测这段摘录的语言，然后系统会告诉你正确的答案 以方便你的程序从中学习。

总共有56种可能的语言，它们是IOI2010的所有参赛国家的母语。每个摘录所用的语言随机选自这56种语言之一，而摘录内容来自维基百科的随机文章的第一段。所有字符都会被编码为1至65535之间的整数，且不对应任何标准编码。

得分公式：

低于0.3的准确度没有分

达到0.3的准确度就可以获得30分

接下来你的得分会随准确度线性增加，达到0.91的准确度就可以获得100分

题目还给出了一种示例做法，这种被称为Rocchio法的算法可以达到大约0.4的准确率。

这种算法思路是这样：对每段摘录 $E$ ，选择与到目前为止遇到的语言中最相似的作为答案。摘录 $E$ 与语言 $L$ 的“相似度”被定义为 $E$ 中没有在任何已知语言为 $L$ 的摘录中出现的字符个数。

直接实现这个算法，发现对题目中的数据实际准确率达到53%，56分就这么到手了。

更好的做法？

官方做法是一个看起来有点不可思议的算法：改进一下Rocchio算法，找一个合适的 $k$ 值，把连续的 $k$ 个字符当作 $k$ 元组处理。

然后摘录 $E$ 与语言 $L$ 的相似度定义改为： $E$ 中没有在任何已知语言为 $L$ 的摘录中出现的 $k$ 元组的个数。

这个算法我最初也没有理解为什么会有很好的效果，直到后来偶然看到了一篇与自然语言处理相关的科普文后才知道这个算法是n-gram算法的一个变形。在这里介绍一些相关的比较有意思的东西。

首先我们需要了解一些概率论基础知识。

我们用 $P(A)$ 表示事件A发生的概率。

$P(A|B)$ 表示，在事件B发生的前提下，事件A发生的概率。

$P(AB)$ 或 $P(A, B)$ 或 $P(A \cap B)$ 都是等效的写法，均表示既发生事件A也发生事件B的概率。

$$P(AB) = P(A) * P(B|A)$$

$$P(A|B) = P(AB)/P(B)$$

一个重要的公式：贝叶斯（Bayes）公式。

$$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$$

举个很简单的例子以帮助理解这个公式：

我们知道女生中参加OI竞赛的比例是远低于男生的。假设某学校中60%是男生40%是女生，其中男生有40%是Oler，女生有20%是Oler。有一天你偶然看见学校机房里有一个Oler正在对着题目苦思冥想（这里假设每个人出现在机房是等概率的），那么这名Oler是女生的概率是多少呢？

我们实际要求的是 $P( Girl | Oler )$ ,

也就是 $\frac{P( Girl | Oler )}{P( Oler )}$ ,

也就是 $\frac{P( Girl ) * P( Oler | Girl )}{P( Oler )}$ ,

也就是有 $P( Girl | Oler ) = \frac{P( Oler | Girl ) * P( Girl )}{P( Oler )}$ ,

把 $Girl$ 和 $Oler$ 分别换成 $A$ 和 $B$ , 这就是贝叶斯公式。

利用贝叶斯公式我们可以简化很多问题。

比如，我们敲键盘时候有时会手抖敲错单词，这时候有些文本编辑器会提示你单词拼错了，并给出可能的正确拼法。

这是怎么做到的呢？

直接找词典里每个单词，看看哪个和当前单词最接近？

不够靠谱，比如用户手抽打了collwge这个词，用户更可能想输入的是college（大学）还是collage（把.....制作成拼贴画）呢？字母w和字母a、e在键盘上都靠得很近，但college这个词很明显比collage常用很多，用户打出collwge时十次里可能有九次心里想的是college只有一次是collage，显然最佳的更正建议是college而不是collage.

记"敲出来的单词是 $x$ "这个事件为 $A_x$ , "需要敲出单词 $y$ "这个事件为 $B_y$ , 概率 $P(B_y|A_x)$ 就是敲出的单词是 $x$ , 而想敲的单词是 $y$ 的概率.

那么文本编辑器要做的是, 在已知用户敲出了 $x$ , 找一个单词 $y$ , 使得用户想敲的是 $y$ 的概率最大. 翻译成概率语言, 即找一个 $y$ , 使得 $P(B_y|A_x)$ 最大。

运用一次贝叶斯公式, 我们能得到

$$P(B_y|A_x) = \frac{P(A_x|B_y)*P(B_y)}{P(A_x)}$$

$P(A_x)$ 项是个常数, 可以不考虑. 那么我们实际要优化的是 $P(A_x|B_y) * P(B_y)$ 的大小, 用大白话说, 就是我使用单词 $y$ 的概率乘上想要输入单词 $y$ 却把单词 $y$ 手抖打成单词 $x$ 的概率。这两部分显然都不难求。

和刚才那个直接找字典里最接近的单词的方法不同之处? 刚才的做法求的实际是 $P(A_x|B_y)$ , 而不是我们想求的 $P(B_y|A_x)$ 。

再看一个很常见的例子：垃圾邮件过滤。

大多数邮箱系统都提供了反垃圾邮件的功能。如果我们要实现一个简单的判断垃圾邮件的系统应该怎么做呢？

我们可以统计出一个邮件是否是垃圾邮件的先验概率，不妨设为 $P(\text{好})$ 和 $P(\text{坏})$ 。（也就是所有邮件中有多大比例是垃圾邮件）

不妨设邮件A由n个单词组成。我们把一封邮件包含A的第*i*个单词称为事件 $T_i$ 。运用贝叶斯公式，可以计算出此时的后验概率：一封包含了邮件A的每个单词的邮件是垃圾邮件的概率为

$$\begin{aligned} & P(\text{坏} | T_1, T_2, \dots, T_n) \\ = & \frac{P(\text{坏}) * P(T_1, T_2, \dots, T_n | \text{坏})}{P(T_1, T_2, \dots, T_n)} \\ = & \frac{P(\text{坏}) * P(T_1, T_2, \dots, T_n | \text{坏})}{P(\text{好}) * P(T_1, T_2, \dots, T_n | \text{好}) + P(\text{坏}) * P(T_1, T_2, \dots, T_n | \text{坏})} \\ = & \frac{P(\text{坏}) * P(T_1 | \text{坏}) * P(T_2 | \text{坏}, T_1) * P(T_3 | \text{坏}, T_1, T_2) * \dots + P(\text{坏}) * P(T_1 | \text{坏}) * P(T_2 | \text{好}, T_1) * P(T_3 | \text{好}, T_1, T_2) * \dots}{P(\text{好}) * P(T_1 | \text{好}) * P(T_2 | \text{好}, T_1) * P(T_3 | \text{好}, T_1, T_2) * \dots + P(\text{坏}) * P(T_1 | \text{坏}) * P(T_2 | \text{坏}, T_1) * P(T_3 | \text{坏}, T_1, T_2) * \dots} \end{aligned}$$

也就是  $P(\text{坏} | T_1, T_2, \dots, T_n) =$

$$\frac{P(\text{坏}) * P(T_1 | \text{坏}) * P(T_2 | \text{坏}, T_1) * P(T_3 | \text{坏}, T_1, T_2) * \dots}{P(\text{好}) * P(T_1 | \text{好}) * P(T_2 | \text{好}, T_1) * P(T_3 | \text{好}, T_1, T_2) * \dots + P(\text{坏}) * P(T_1 | \text{坏}) * P(T_2 | \text{坏}, T_1) * P(T_3 | \text{坏}, T_1, T_2) * \dots}$$

然后我们就会发现，这个式子一切都好，惟一的问题就是没法算。

问题：假设邮件只有20个单词，想靠谱地算出  $P(T_{20} | \text{好}, T_1, T_2, \dots, T_{19})$  的值，需要多大的语料库？

一封信的前19个单词有多少种不同的可能性？

想想都能吓死人啊，恐怕把全世界的硬盘都拿来也装不下吧.....更别提找这么多的邮件了。

那怎么办？

于是产生了一个被逼无奈的办法：一个单词大概只与前面若干个单词关系比较紧密吧，与更靠前的其实关系不算特别大。

于是n-gram算法就产生了：我们干脆假设每个单词只受其前面连续 $k$ 个单词影响，不再受更靠前的单词影响了。

通过取一个适当的 $k$ 值，就可以有效避免数据稀疏性的问题了。而且实践表明这个算法效果很好。

为什么武断地认为每个单词只与前 $k$ 个单词有关不会玩脱呢？要知道，很多时候一句话可能与老早以前的某句话都有逻辑关系。

一个解释是，很早以前的单词确实可能会对当前单词产生影响，但这个影响太小了，而且很可能正面影响和负面影响都差不多，很可能基本抵消掉，因此对最终答案没有什么影响。

而且从现实角度看，有很多词语是“专属”于广告的。正常人看到“大减价”“优惠券”“物美价廉”“快来抢购”等词语时候，不管前面说了啥，就已经能基本确定是广告了。至于到底是“吮指原味鸡优惠券”还是“黄金脆皮鸡优惠券”并不重要～ :) 所以以短词组为单位识别还是相对靠谱的。

怎么选取 $k$ 的值？

$k$ 值太小，区分度不够高，统计“快来抢购”这个词组显然比统计“快”“来”“抢”“购”这四个单字分别出现的次数更能确定广告。

$k$ 值太大，数据稀疏性问题又来了。

根据实际情况选择合适的 $k$ 值。

回到原题，我们要识别一段摘要是哪个语言。

这个和之前讲的识别一段话是否是广告本质是一个问题啊，相当于识别一段话有多大程度符合某种语言的特征。

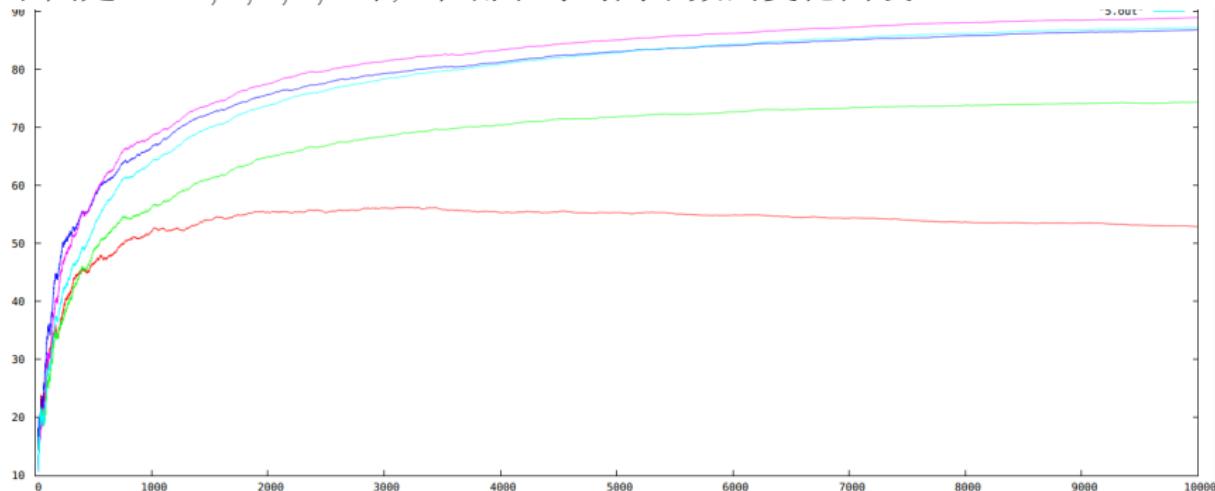
这时候大家应该很容易理解题解中的“把 $k$ 个连续元素组成 $k$ 元组”了吧。也可以发现，题目中给出的那个Rocchio方法就是 $k = 1$ 的n-gram，相当于假设所有字符都是独立事件。

适当地选取 $k$ 值进行试验，结果如下：

k值	准确率
1	53%
2	74%
3	87%
4	91%
5	87%
6	84%
7	79%
8	74%

可以发现 $k = 4$ 时效果最好，正确率达到了91%，可以得到100分。

下图是 $k = 1, 2, 3, 4, 5$ 时，准确率与询问个数的变化曲线。



我也尝试过随着询问个数的增加，逐步加大 $k$ 等优化，但发现优化效果极为有限。

能否做到更高的准确率？欢迎有兴趣的同学与我讨论。

有一个 $n$ 点 $m$ 边的边权均为1的无向图和一个值 $H$ 。

你被要求写两个程序，第一个程序可以获知这个图的信息和 $H$ 的值，并发送一串二进制串给第二个程序。

第二个程序只能获知这串二进制串，要求输出结点 $1 \sim H$ 与结点 $1 \sim n$ 两两之间的最短路。

$$n \leq 1000, m \leq \frac{n*(n-1)}{2}, H \leq 36$$

得分规则：

如果你发送的二进制串长度不超过16000000，你可以得到25分；

如果你发送的二进制串长度不超过360000，你可以得到50分；

如果你发送的二进制串长度不超过80000，你可以得到75分；

如果你发送的二进制串长度不超过70000，你可以得到100分；

朴素算法：

由于所有边权都是1，最短路长度很明显不会超过 $n$ 。直接暴力BFS求出最短路，总共最多有 $n * H = 36000$ 个数，每个数不超过1000，直接用10位二进制位表示，发送过去，正好360000位。  
这样就50分了。

特殊性质？边权都是1。如何利用？

注意到，如果存在一条边权为1的边 $(u, v)$ ，那么对于任意点*i*，必然有 $|dis(i, u) - dis(i, v)| \leq 1$ 。

建立从1开始的最短路径树，树上每个结点记录该结点到1~*H*的各个最短路径长度。

考虑父结点u与孩子结点v上的*H*个值的关系。由上述结论，对任意*i*， $dis(u, i)$ 与 $dis(v, i)$ 必定只相差-1，0或1。因此我们只要记录这个-1，0，1序列，就能从父结点u的答案推出孩子结点v的答案。

把这棵最短路径树传递给程序2就可以了。

需要传递的信息：每个结点的父结点以还原树结构，每条树边的的-1，0，1序列以还原答案。

每个树结点需要的长度： $\log_2 3^{36} + 10 = 68$ 位。

总长度：68000位，可以得到满分。

你需要写两个程序，第一个程序读入 $n$ 个0~255之间的整数，输出不超过 $L$ 个0~255之间的整数。

然后第一个程序的输出会被随机洗牌打乱次序后提供给第二个程序。

第二个程序必须还原出第一个程序所读入的 $n$ 个数（ $n$ 个数的次序也要一样）。

得分规则：

任务一：限制 $n \leq 16, L \leq 10n$

任务二：限制 $n \leq 32, L \leq 10n$

任务三：限制 $n \leq 64, L \leq 5n$

想法一：

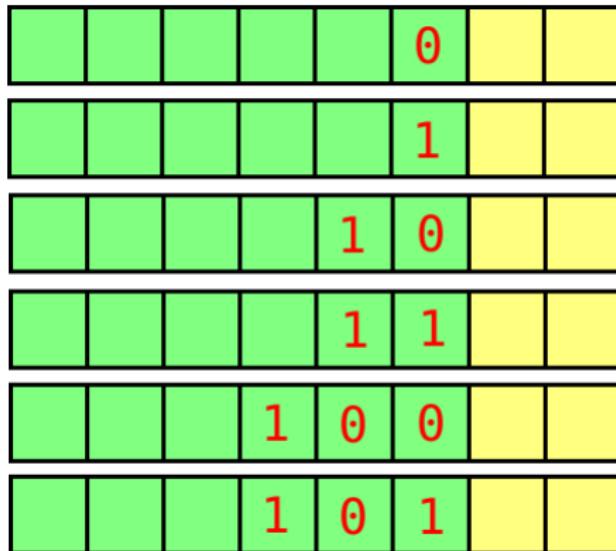
被打乱顺序了怎么办？

想办法保存额外信息，记录每个数是序列里的第几个。

设用 $x$ 位记录自己是序列里的第几个，那么剩下的 $8 - x$ 位可以保存有效信息。

那么最多一共能传送 $2^x * (8 - x)$ 位信息。

取 $x = 6$ 时能传送的位数最多。此时能传送128个二进制位信息，也就是16字节， $L$ 的长度是 $4n$ ，可以满足任务一的要求。



前6位作为位置码，依次存储0~63。后两位存储真正的有效信息。

解码时，先按前6位进行排序，然后把后两位依次拼起来，即可次序不变地还原出信息。最多能传递 $2^6 * 2 = 128$ 位二进制位。

怎么解决任务2？

考虑把要求的序列的二进制形式直接写下来（第二个数直接接在第一个数后面这样）。

总共长度最多 $32 * 8 = 256$ 位。

如果第*i*位是1，我们就把第*i*位输出出来。

## 任务3?

上一个做法的实质是，用一个长度为256的2进制整数与原序列（一个长度为32的256进制整数）一一对应起来。

于是.....为什么我们必须使用二进制整数？进制数并不是限制。

真正的限制是“代价等于所有数位的和”。因此我们应该按照“所有数位的和”从小到大为次序进行编码。

利用组合数学，很容易发现数位和为 $k$ 的编码方案数共有 $C_{255+k}^k$ 种。  
而64个0~255的整数共有 $256^{64}$ 种可能性。

至少要数位和达到多少，才能让编码数量达到 $256^{64}$ 呢？  
也就是找到最小的 $k$ 使得

$$\sum_{i=0}^k C_{255+i}^i \geq 256^{64}$$

写个程序算一下，发现答案是261。因此最多只要用不超过261个数，  
就能传递64个0~255以内的整数。

效率？ $\frac{L}{n} = \frac{261}{64} \approx 4.08$ ，足以满足任务3的要求。

但上述方法的代码太难写了，为了进行编码，不得不写高精度等麻烦东西.....

更简单的方法？

使用更浪费但更好写的编码方案。

16个数位一组，每组使用不超过20个数。总方案数约 $7.3 * 10^9$ 种，足以编码32个二进制位，也就是4字节。

效率正好是 $\frac{20}{4} = 5$ ，恰好满足任务3要求。

这样就不用高精度了，直接用动态规划求编码序号即可。

# 娱乐向：程序猜数

曾经，有一个30位（二进制）的整数摆在我的面前我没有珍惜，直到我失去时才后悔莫及，如果上天再给我一次机会，我一定牢牢的记住这个整数。

没办法，已经失去了，只好再问回来了。

你的程序可以调用一个系统提供的函数 $\text{isX}(x)$ ，这个函数会返回 $x$ 与这个数有多少二进制位不同。

你需要在不超过5次调用内确定这个整数。

特殊附加条件：本题有20组数据，可以多次提交，每次提交都可以看到各个测试点的评测结果。

# 娱乐向：程序猜数

正常的做法可能在5次操作内确定一个整数吗？

每次调用的返回值都在0~30之间，共31种可能。

因此，5次调用最多只有 $31^5$ 种不同结果，也就最多只能分辨出 $31^5$ 个不同整数，远远达不到题目从 $2^{30}$ 种可能性中找到正确的整数的要求。

因此正常做法理论上就不可能解决此题。

怎么办？

利用特殊条件，我们每次提交都可以看到各个测试点的评测结果。

我们发现，第一次询问 $\text{isX}(0)$ ，第二次询问 $\text{isX}(2^k)$ ，我们就能确定当前要猜的数的第 $k$ 位是0还是1。

然后？如果第 $k$ 位是0，就让程序直接退出，得到Wrong Answer；  
如果第 $k$ 位是1，就让程序进入死循环，得到Time Limit Exceeded；  
这样就把第 $k$ 位的结果通过提交结果传递到了我们的手上。

# 娱乐向：程序猜数

每次提交都能确定各个数据答案的某一位是0还是1；  
只要30次提交就能取得全部数据的答案；

得到全部数据的答案后，只要根据这些答案人工找一个分辨规则就行了。

# 娱乐向：一道防AK好题

请大家无视搞笑的题目名字.....这题其实是我出的某套NOIP模拟赛的第一题.....

有一个长度为 $n \leq 500000$ 的数列，第 $i$ 个数为 $x_i$ 。

有很多（不超过500000组）询问，给定的 $a, b, c \leq 10^{18}$ ，要找到一个 $i$ ，使得 $a * (i + 1) * x_i^2 + (b + 1) * i * x_i + (c + i) = 0$ 成立。

如果有多个 $i$ 满足，要求返回最小的那个 $i$ 。

输入数据中 $a = b = c = 0$ 标志着询问的结束。

为了加大难度，出题人对数据进行加密以防止离线算法的出现。假设你在输入文件中读到的三个数为 $a_0, b_0, c_0$ ，那么真正要询问

的 $a = a_0 + lastans, b = b_0 + lastans, c = c_0 + lastans$ 。

*lastans*的值是你对前一个询问的回答。如果这是第一个询问，那么 $lastans = 0$ 。所有的询问都将会按上述方式进行加密，**包括标志着询问的结束的那个询问也是这样**。（也就是最后一个询问是解密后才得到 $a = b = c = 0$ 的）

# 娱乐向：一道防AK好题

话说这道题的平均分似乎是整场比赛三道题中最低的（虽然做法是最简单的），在赛后这道题也得到了很多同学的好评。



总算见到比AHOI2012还坑爹的题了.....

[Mato完整版](#)



Mato完整版：回复 Mato完整版 :我有一个想法.....以后在TYVJ上办一个坑爹模拟赛，专出坑爹题，请@sillycross +AHOI2012出题者出题

[封](#) | [删除](#) | 2012-8-25 22:16 回复

同时这道题的思想似乎还被用到了山东省选的某一道题中又坑了一些人，作为始作俑者我表示十分抱（xin）歉（wèi）。

首先我们发现传统的数据结构维护的思路对题目中的式子基本都是无效的。

实际这题的突破口在于，最后那个标志询问结束的询问也是加密的。而它解密后一定是 $a = b = c = 0$ 。

也就是说， $a_0 + \text{lastans} = 0$ ，于是我们可以直接得出倒数第二个询问（也就是要回答的最后一个询问）的答案是 $-a_0$ .

然后有趣的事情就发生了。得到了倒数第二个询问的答案后，我们将答案和倒数第二个询问的 $a_0, b_0, c_0$ 带入原题的式子中，可以得出一个关于倒数第三个询问的答案的一元一次方程。于是可以 $O(1)$ 直接解出倒数第三个询问的答案。

如此推下去，就可以 $O(n)$ 推出全部询问的答案。

看上去是一个复杂的数据结构题，实际考察的却是选手打破思维定势的能力。

# 总结?

嗯，和非传统题有关的就讲到这里吧。

一定要总结一点什么的话？

除去考察一些常见的思路与算法外，主要的考察点的依旧是选手针对题目特点，充分发挥人类智慧，探索、测试、改进解决方案的能力。

接下来讨论一下IOI2013 Day1的剩下两道题。

给定一个包含  $n$  个结点的森林 ( $n \leq 10^5$ )，边上有权值。

给定一个值  $L$ ，你需要增加若干边权均为  $L$  的边，使得：

得到的新图是一棵  $n$  个结点的树

这棵树的直径尽可能小

问最小可能直径是多少。

$n \leq 10^5$

首先我们试图发现一些性质。

1. 如果某个结点是我们新加的某条边的端点，那么我们称这个点为“接点”。可以证明存在一个最优解使得在每个原图中的树里，都有且仅有一个“接点”。

证明：如果有一个最优解使得在某个树中有超过一个“接点”，我们不妨取其中任意两个，设为 $A$ 与 $B$ 。我们不妨定义 $dis(T)$ 表示结点 $T$ 所在树中，距离结点 $T$ 最远的点的距离。不妨设 $dis(A) \leq dis(B)$ ，那么我们将接到结点 $B$ 的边全部改为接到结点 $A$ 上，可以发现，直径不会增加。如此往复，直至这棵树中只剩一个接点，此时解没有变差。

2. 可以证明存在一个最优解，使得对于原图中每个树，这个树的接点 $T$ 必定是 $dis()$ 值最小的点。

证明：如果有一个满足上个性质的最优解中，某个接点 $T$ 的值不是所在树中 $dis()$ 最小的结点，不妨设 $dis()$ 最小的结点为 $T'$ ，我们把所有接到结点 $T$ 的边都改为接到节点 $T'$ 上，可以发现，直径不会增加。

3. 有了前两条性质，我们现在需要考虑的就仅仅是这些“接点”的连接方式了。那么最优的连接方式一定是：选择某一个接点，其他所有接点都向这个接点直接连边。

我也不可能严格写这一步的证明。不过感性理解一下的话，正确性是显然的。

对严格证明这一步有想法的同学欢迎与我讨论。

有了上文的分析，算法已经非常显然了。首先我们需要求出原图中每个树的直径，并求出原图的每个树中 $dis()$ 的值最小的结点。

这一步只需对每个结点用树形dp求出其向下最长延伸、次长延伸的长度，以及向上最长延伸的长度即可 $O(n)$ 完成。方法十分经典，在此不再赘述。

我们不妨用 $D[i]$ 来表示我们求出的树*i*的 $dis()$ 的最小值。接下来，我们考虑枚举那个“其他接点都向它连边”的接点，那么当前方案的答案可以利用 $D$ 数组的最大的三个值 $O(1)$ 计算出来。

最后还有一个小细节，上文所算的答案仅仅考虑了包含我们新加的边的路径，因此最后还应该和原图中各个树的直径取 $\max$ ，才能得到最终答案。最终复杂度 $O(n)$ 。

有一个 $R * C$ 的网格，其中所有的竖向边只能从上向下单向通行，而横向边可以双向通行。每条边都有一个初始权值。

要求支持：

修改一条边的权值

查询从网格最上面一行的某一点到最下面一行的某一点的最短路的长度

$R \leq 5000, C \leq 200$ , 修改操作不超过500次, 查询操作不超过 $2 * 10^5$ 次, 时间限制20秒。

最最朴素的做法？

对每个询问暴力用Dijkstra求最短路。

$O(RC \log R)$ 每次询问。

稍稍改进一些？

因为竖向边只能从上向下单向通行，这道题具有很明显的阶段性。

直接进行dp，同一行内部的转移可以用BFS通过一些技巧做到 $O(C)$ 。

可以做到 $O(RC)$ 每次询问。

再改进一些？

因为竖向边都是单向的，这里的最短路是满足区间合并性质的。如果我们有两个 $X * C$ 的网格中，最上面的行和最下面的行两两之间最短路的长度，那么我们显然可以在 $O(C^3)$ 内合并它们，得到这两个网格拼起来后的情况。

我们不妨用线段树维护 $R$ 这一维，那么复杂度就是 $O(C^3 * \log R)$ 每次修改， $O(1)$ 每次查询（因为查询只查询整个网格的情况）。

做到这一步可以拿55分或76分（根据实现不同）。我考场上只想到了这一步，于是只有55分，成为了中国队中唯一没有AC这道题的队员。

怎么继续优化？

我们固定上半部分网格的起点，然后考虑拼接处每个点能作为哪些端点的最优答案。可以发现，以其为最优答案的终点必定是一个区间。

归纳证明：

首先，如果只有一行，结论显然成立。

如果结论对 $k$ 行的网格成立，考虑拼上第 $k + 1$ 行，发现结论依然成立。

也就是说，转移是满足单调性的，一个老的决策一旦劣于一个新决策，那么这个老决策就可以永远地被丢弃了。

因此，只要套用经典的1D/1D动态规划优化方法，用一个单调栈+二分来维护最优决策即可把合并复杂度优化到 $O(C^2 * \log C)$ 。

最终复杂度是，预处理 $O(C^2 * R * \log C)$ ，修改 $O(C^2 * \log C * \log R)$ ，查询 $O(1)$ ，足以通过本题数据。

# 谢谢大家

感谢胡渊鸣、罗雨屏、王康宁、乔明达、贾志鹏、罗翔宇等同学提供的大量修改建议和帮助

感谢CCF提供了这个交流的平台

感谢父母、学校对我参与OI竞赛的支持

感谢所有帮助、关心我的老师们和同学们

祝大家新年快乐～

如果看到这道题目，说明我前面准备的题目太少了……T\_T  
定义：

$$g(x) = x \text{ xor } (x/2)$$

$$h_1(x) = x/m_1 * m_1 + (x + s_1) \% m_1$$

$$h_2(x) = x/m_2 * m_2 + (x + s_2) \% m_2$$

$$f(x) = g(h_2(g(h_1(g(x)))))$$

其中`xor`是按位异或，`/`是整除，`%`是取模。

你只知道 $m_1, m_2, s_1, s_2$ 都是不超过345678的正整数，

且 $0.3m_1 < s_1 < m_1, 0.3m_2 < s_2 < m_2$ ，

但你不知道这四个常量具体的值。

现在样例给了大约5000组 $x$ 的值及对应的 $f(x)$ 的值，你只需要用任意方法解出 $m_1, m_2, s_1, s_2$ 这四个常量的值就行了。

给出的数据中，大约100组数据有 $x \leq 1000$ ，其余的 $x$ 大致等概率分布在 $[0, 2^{32})$ 间。

首先发现函数 $g$ 是有反函数的。

因此可以把 $y = g(h_2(g(h_1(g(x)))))$ 最外层和最里层的 $g$ 去掉，变成 $y' = h_2(g(h_1(x')))$ 。

接下来，最自然的想法是枚举 $h_1()$ 的参数 $m_1$ 和 $s_1$ ，但这时会发现 $m_2$ 和 $s_2$ 依然有很多可行解，再乘上对5000组数据一一验证的复杂度，完全无法接受。

换一个想法？考虑一旦确定了 $m_1$ ，随着 $s_1$ 取不同的值， $h_1(x')$ 能取到的值会形成两个（或一个）区间。

但很不幸，在碰到 $g$ 函数后，这些形成的区间又被打散了，没法利用.....

突破口在哪里呢？

提供的数据中有100组数据满足 $x \leq 1000$ 。

大胆猜想：此时如果 $m_1$ 不特别小的话，那

么 $h_1(x) = x/m_1 * m_1 + (x + s_1)\%m_1$ 中，前半部分 $x/m_1 * m_1$ 基本就是0了，而且如果 $s_1$ 和 $m_1$ 不特别接近，后半部分也可以无视取模！

然后 $g$ 函数并不会显著改变传进来参数的大小（因为是位运算），因此 $g(h_1(x))$ 大概只有 $s_1$ 这么大。

于是……如果 $m_2$ 也比较大的

话， $h_2(x) = x/m_2 * m_2 + (x + s_2)\%m_2$ 中，前半部分 $x/m_2 * m_2$ 说不定也是0……

于是我们带着这两个猜测试验一下。这时候，我们发现，枚举了 $s_1$ 的值后， $s_2$ 的值也就确定了。

而且发现，符合条件的 $(s_1, s_2)$ 组并不多，大概只有几十组。

我们再对每一组符合条件的 $(s_1, s_2)$ 枚举 $m_2$ ，此时 $m_1$ 也就可以求出了，然后逐一验证即可。这样可以在大约3分钟内找到解。

如果我们运气不好，刚才的猜测是错误的呢？留给同学们思考～：）