混淆与破解

【问题描述】

小强和阿米巴是好朋友。

阿米巴研发出了一套相当高端的图片识别系统,并把它写成了一个手机 app。这个识别系统具备特殊的识别能力,比如说,它能够识别一张图片里是否有萌萌的小狗。

这个 app 由两个模块组成:特征提取模块和分类模块。每当小强拍摄一张图片,特征提取模块就从中提取出一个长度为 n 的 01 串并存储起来。当小强希望进行识别的时候,分类模块就会根据提取出的 01 串进行分类(即,输出一个 0 或者 1 的答案)。

为了保护分类算法,阿米巴的这个 app 是经过加密处理的。经过对阿米巴的死缠烂打,小强弄明白了分类模块的工作原理。

分类模块会从输入的这个 n 位 01 串中恢复出 m 位的“有效信息”。每个“有效信息”都是某些输入变量的异或。之后,分类模块利用这些“有效信息”进行运算来得到真正的结果。为了进一步加密,阿米巴还会加入“噪声”。所谓“噪声”,是指这个分类模块会故意按一定的比例将结果反转。小强拿到的可能是经过了反转的结果。

举个例子,分类模块的算法步骤可能是这样的:

```
function f(x[]):  
    z[0] = x[0] xor x[4] xor x[7]  
    z[1] = x[12] xor x[2]  
    z[2] = x[0] xor x[1] xor x[2] xor x[3]  
    result = h(z[])  
    return result xor g(x[])
```

其中 $x[]$ 是一个 01 串, $x[i]$ 表示其中的第 i 个位,即一个 0 或 1 的数。

$g(x[])$ 是某个大多数情况下返回 0,偶尔返回 1 的函数。 h 是某个关于 $z[]$ 的函数,其返回值为 0 或 1。

$z[0], z[1], z[2]$ 就是“有效信息”。

为了让小强无法从 app 的源代码中看出算法,这个算法被进行了混淆。为了方便起见,我们把混淆之后的算法叫做“混淆版算法”。混淆版算法的代码共有 Q 行,它的每一行都是这个样子:

```
y[u] = (not (y[v] and y[s])) xor y[d] xor y[e]
```

其中 $y[]$ 是一个长度为 L 的 01 数组；xor 表示异或，and 表示与，not 表示非。 u, v, s, d, e 是这一行的参数。初始的时候， $y[0] \sim y[n-1]$ 里面放置了 $x[0] \sim x[n-1]$ 这 n 个输入位，其他地方都是 0。执行完这 Q 行代码之后， $y[0]$ 这个位就是输出。

对于阿米巴的这种以损失性能为代价进行加密的行径，小强感到很愤怒。于是，小强打算从混淆版算法中破解出阿米巴的分类算法。为了方便起见，我们把破解得到的算法叫做“破解版算法”。小强希望你能够帮他破解出：

1. 如何提取有效信息。这个可以表述为 m 个 $\{0, 1, \dots, n-1\}$ 的子集，每个子集对应了一个有效信息是从哪几个输入位异或得到的；
2. 把这 m 位有效信息映射到分类结果上的函数 h 。该函数用一个长度为 2^m ，每一位均为 0 或 1 的查找表表示；这 2^m 位分别对应了 m 位有效信息每一种可能的情况。

当然，这种破解算法是不唯一的，即，可能会有多种有效信息提取方法和查找表的组合。你只需要给出其中一种即可。

阿米巴保证，引入的噪声比例不超过 p 。即，你要求出的破解版算法，和混淆版算法至少在 $2^n(1-p)$ 个不同的输入上得到的结果是一样的；并且阿米巴保证这样的算法是存在的。

同时，阿米巴也保证，这 m 个有效信息都是必须的，即， h 无法化简为少于 m 个输入的函数。

【输入格式】

输入文件 *pdt.in* 表示一个混淆版算法。

输入文件的第一行包含 4 个整数 n, m, L, Q 。接下来 Q 行，每行包含 5 个整数 u, v, s, d, e ，表示每行的参数。

【输出格式】

输出文件 *pdt.out* 表示你给出的破解版算法。

先输出 m 行，每行包含 1 个 n 位 01 串，表示每个有效信息是由哪些输入位异或得到的。其中 1 表示包含该输入位，0 表示不包含。

接下来输出一行一个长度为 2^m 的 01 串，表示 h 函数的查找表。查找表中的项按字典序进行排列。即，先排第一个有效信息是 0 的，再排第一个有效信息是 1 的。排第一个有效信息是 0 的项的时候，先排第二个有效信息是 0 的，再排第二个有效信息是 1 的。以此类推。

【样例输入 1】

```
3 2 4 1
0 1 2 2 2
```

【样例输出 1】

```
001
010
1110
```

【样例说明 1】

样例输入等价于如下代码

```
y[] = 0000
input x[0..n-1]
y[0..n-1] = x[0..n-1]
y[0] = (not (y[1] and y[2])) xor y[2] xor y[2]
output y[0]
```

其中 $x[0..n-1]$ 表示 01 串 x 的第 0 位到第 $n-1$ 位。
在这段代码中，每一种输入对应的输出如下

input	000	001	010	011	100	101	110	111
output	1	1	1	0	1	1	1	0

样例输出是一种破解方案，等价于如下代码

```
input x[0..n-1]
z[0] = x[2]
z[1] = x[1]
output h(z[])
```

---

h 函数的输入与输出有如下对应关系

$z[]$	00	01	10	11
$h(z[])$	1	1	1	0

可以发现，对于每一种输入，破解版算法和混淆版算法的输出是相同的。

【样例输入输出 2】

见选手目录下的 *pdt/pdt.in* 与 *pdt/pdt.ans*。

【数据规模和约定】

对于 10% 的数据, $m = 1, p = 0$;

对于另外 30% 的数据, $m = 1$;

对于另外 20% 的数据, $m = 2$;

对于另外 20% 的数据, $m = 3$;

对于另外 20% 的数据, $m = 4$ 。

对于所有的数据, $1 \leq n \leq 64, 1 \leq L \leq 256, 1 \leq Q \leq 1024, 0 \leq p \leq 0.01,$
 $0 \leq u, v, s, d, e < L$ (注意, 输入文件中并没有把 p 的值给你)。

【提示】

使用位运算一次在多个输入上求出函数值可以极大的加速你的程序。