Day2

	编码干扰	括号序列计数	里面还是外面
程序名称	code.h/pas decode.h/pas	brackets.cpp/c/pas	**insider.cpp/c/pas**
输入文件	code.in	brackets.in	**insider.in**
输出文件	code.out	brackets.out	**insider.out**
时间限制	5s	见题目	2s
内存限制	512M	512M	**512M**
题目类型	交互题	传统	传统

编码干扰

（code）

题目描述：

Alice 和 Bob 此刻分别身处于两个相距遥远的国度。他们因为不能相见，所以只能通过互联网传输信息。简单来说，Alice 每次会向 Bob 发送一条长度为 k 位的 q 元码。所谓 q 元码，即编码每一位都是小于 q 的非负整数。

然而，信息在传输的过程中，往往会受到一些干扰。Bob 已经知道这个干扰的规模为非负整数 d，如果取 e = [(d-1) / 2] （即 d-1 除以 2 向下取整），则干扰会最多造成 e 位的错误。这里 e <= k，如果 e = 0 则不会产生任何错误，若 e = k，则对于一条长度为 k 的编码，有可能因为干扰而变成任意一条长度为 k 的编码。同时 Bob 还知道，这个干扰的规模不会因为编码长度的增加而变化，只是由地理因素决定的。也就是说，如果 Alice 传送一条长度为 k+1 的 q 元码给 Bob，编码在传输中收到的干扰规模仍然是 d。但是编码长度的增加却会带来更多的费用问题。

Bob 希望可以设计一个纠错系统，通过发送长度 n >= k 位的 q 元码，来避免因为干扰而造成收到错误的信息。这个纠错系统由两部分组成：CODE 系统与 DECODE 系统。Bob 希望设计好后，将 CODE 系统寄给 Alice，将 DECODE 系统留在自己手中。

之后 Bob 与 Alice 约定：每一次 Alice 在发出长度为 k 的编码 I 前，先通过 CODE 系统得到长度为 n 的编码 II（这里编码 I 与编码 II 都是 q 元码），然后将编码 II 发送给 Bob，编码 II 在发送途中会受到规模为 d 的干扰，所以 Bob 真实收到的编码 III 可能与 II 不同。最后 Bob 利用 DECODE 系统，通过编码 III 计算出编码 I。这就完成了纠错工作。

注意到：Alice 每一次发送的编码总是长度为 k 的 q 元码，且收到的干扰规模总是 d。传送编码的长度直接决定了传送费用（这里，通过互联网传送信息并不被认为是免费的，而需要与传送编码长度成正比的费用）。所以 Bob 希望这个系统在能实现正确纠错的基础上，每一次发出的编码 II 的长度 n 尽可能小。

程序要求：

本题要求选手编写两个独立的程序，分别对应 CODE 系统与 DECODE 系统。

对于 c/cpp 选手，需要提交 code.h 与 decode.h 程序。code.h 中的过程：

```
int Alice_solve(int k , int d , int q , int B[]);
```

为 CODE 系统，对于给定系数 k，d，q 和编码 I（用 B[1]到 B[k]保存），需要返回编码 II 的长度 n，并将编码 II 保存在 B[]中。

而 decode.h 中的过程：

```
void Bob_solve(int k , int d , int q , int B[]);
```

为 DECODE 系统，对于给定系数 k，d，q 与编码 III，假设编码 III 的长度为 n，则 B[1]

到 B[n]存储了编码 III，需要通过 B[]找出最早的编码 I（Alice 发出的原始编码），并保存在 B 中（B[1]到 B[k]）。

对于 code.h 与 decode.h，我们要求选手的两份程序之间不存在互相的调用关系，且我们禁止在程序中出现“#define”、“#if”等字符串。比赛中所提供的 forbidden_c.txt 列出了对于 c/cpp 选手所提交的 h 文件中禁止出现的字符串。

我们提供有评测程序 code_administrator.cpp，当选手的代码（code.h 与 decode.h）提交上来之后，我们会将 code_administrator.cpp 与 code.h、decode.h 一起编译（code_administrator 中有调用到 code.h 与 decode.h）。注意到 code_administrator.cpp 从输入文件 code.in 读入信息并对选手提供的纠错系统进行检测，检测结果将反馈到输出文件 code.out 中。需要注意的是，因为 code.h 与 decode.h 作为了一个程序被编译，所以重复的变量名定义（包括与 code_administrator.cpp 中变量出现重复的情况）有可能会造成程序编译无法通过。

我们为选手提供了一份参考代码，code.h 与 decode.h，以便选手可以更好地理解所需要完成的工作。

对于 pascal 选手，需要提交 code.pas 与 decode.pas。code.pas 中，前四行需要保证是：

```
unit code;
interface
function Alice_solve(k,d,q:longint;var B:array of longint):longint;
implementation
```

且最后两行需要保证是：

```
begin
end.
```

对于 decode.pas，后两行有着相同的要求，前四行要保证是：

```
unit decode;
interface
procedure Bob_solve(k,d,q:longint;var B:array of longint);
implementation
```

详细可以参见比赛中提供的文件 code.pas 与 decode.pas。

选手只需要编写 Alice_solve 和 Bob_solve 过程内的内容，变量可在 implementation 和过程申明之间申明。选手的 code.pas 与 decode.pas 不得存在相互调用的情况。为此，比赛中提供 forbidden_pas.txt 列出了所有对于 pascal 选手提交的程序应该被禁止出现的所有字符串。

我们提供了评测程序 code_administrator.pas。注意到 code_administrator.pas 从输入文件 code.in 读入信息并对选手提供的纠错系统进行检测，检测结果将反馈到输出文件 code.out 中。

我们为选手提供了一份参考代码，code.pas 与 decode.pas，以便选手可以更好地理解所需要完成的工作。

输入文件：

对于输入文件 code.in，第一行给出了系数 k，d，q。之后一行有一个整数 T，表示 Alice

要给 Bob 传送 T 条长度为 k 的 q 元码。之后 T 行，每行 k 个非负整数，描述了一条 q 元码。比赛中，我们提供了一份合法的 code.in 以供选手测试使用。

输出文件

输出文件一共 T 行，由 code_administrator 生成，每一行先是字符串“YES”或“NO”，说明了纠错系统是否正确，若为“YES”，之后一个整数为 n，即编码 II 的长度。比赛中，选手可以通过提供的程序和输入文件 code.in 得到一份合法的输出文件 code.out。

得分判定：

对于输出文件 code.out，如果出现“NO”，则纠错系统失败，不得分，考虑编码 II 的最大长度。根据评测系数 p(1)到 p(10)我们给出这一个数据点的得分情况。如果编码 II 的最大长度不超过 p(u)则得到至少 u 分，且每一个测试点的得分不超过 10 分。

数据规模：

数据编号	k	d	q
1	<=10	7	<=4
2	<=20	5	2
3	116	3	3
4	502	3	2
5	<=100	5	1000<=q<=10000
6	<=50	7	1000<=q<=10000
7	<=40	9	1000<=q<=10000
8	120	4	2
9	247	4	2
10	42	8	2

对于所有数据，T<=100。

括号序列计数

（brackets）

题目描述：

Alice 和 Bob 知道，一个由空格、左括号、右括号组成的序列被称为括号序列。有一类特殊的括号序列被称为“合法括号序列”。已经知道：

（1）“（）”是合法的括号序列，空串是合法的括号序列。

（2）如果 S1 是合法的括号序列，S2 是合法的括号序列，则 S1 与 S2 拼接起来的序列 S1＋S2 也是合法的括号序列。

（3）如果 S 是合法的括号序列，在其左右分别插入一个左括号和一个右括号所得到的字符串“（”＋S1＋“）”也是合法的括号序列。

（4）如果 S 是合法的括号序列，在 S 的任何位置（包括头尾位置）插入一个空格，得到的序列也是合法的括号序列。

现在，Alice 希望知道：对于某个已知的有限状态自动机中的状态 s 与 t，存在多少以 s 为起点，t 为终点的长度为 k 的合法括号序列。

所谓有限状态自动机，又可以被认为是一个有向图 G，由 n 个结点组成，每一个结点表示一个状态，且存在三类以此为起点出去的有向边，对于每一个状态（或结点）来说其出去的同一类有向边将指向同样的状态（或结点）。三类有向边分别代表三种符号：左括号“（”，右括号“）”和空格。

这里，我们将状态（或结点）从 0 开始编号。对于第 i 个状态，用 dfa[i][0]，dfa[i][1]，dfa[i][2]分别表示从 i 出发，代表了左括号、右括号和空格的那一类边指向的状态（或结点），再用 dfa2[i][0]，dfa2[i][1]，dfa2[i][2]表示每一类边的个数。

对于一条从 s 出发到 t 结束的路径，满足长度为 k 且路径经过的边对应的符号组成了一组合法的括号匹配，则称作“满足[G,s,t,k]的合法括号序列”。

现在，Alice 为 Bob 提供了自动机 G，并提出 Q 组询问。对于每一组询问，Alice 会给出 s、t 和 k，她希望 Bob 可以告诉她满足[G,s,t,k]的合法括号序列有多少组。她只需要知道答案除以 47 后的余数。

输入文件：

第一行一个整数 n，表示状态数（或结点数）。

之后 n 行，对于其中的第 i 行，有 6 个 32 位整数，依次为 dfa[i-1][0]，dfa2[i-1][0]，dfa[i-1][1]，dfa2[i-1][1]，dfa[i-1][2]，dfa2[i-1][2]。

之后一行有一个整数 Q，表示 Alice 的询问次数。

之后 Q 行，每一行有 3 个 32 位的整数，依次为 s，t，k。

输出文件：

输出文件有 Q 行。

其中第 i 行对应了第 i 个询问，只有一个整数，表示满足[G,s,t,k]的合法括号序列的个数，答案只需要除以 47 后的余数。

样例输入：

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

样例输出：

```
45
9
10
2
19
25
```

样例说明：

对于第一组询问，长度为 3 的合法括号序列依次有：

（1）三个空格“　　”，则在自动机中的合法方案数为：3×3×3＝27。

（2）对于“（ ）”、“（） ”和“ （）”，则在自动机中的合法方案数为：1×3×2＝6。

所以总的可行方案数为：27＋3×6＝27＋18＝45。

数据规模：

存在 10%的数据：k <= 1000。时间限制 1 秒。

另存在 10％的数据：

n=1，dfa[0][0] = dfa[0][1] = dfa[0][2] = dfa2[0][2]=0，dfa2[0][0] = dfa2[0][1] = 1。

时间限制 5 秒。

另存在 20％的数据：

n=1，dfa[0][0] = dfa[0][1] = dfa[0][2] = 0。

时间限制 5 秒。

另存在 30％的数据：k <= 30000。时间限制 15 秒。

另存在 30％的数据：k <= 100000。时间限制 20 秒。

对于 100％的数据，n <= 2，k <= 100000，Q <= 10。

里面还是外面

（insider)

题目描述：

Alice 给出了平面上的一个简单 N-多边形。所谓简单 N-多边形，包括 N 个给定的端点，和连接相邻点的直线段，特别的，我们认为 1 号点与 N 号是相邻的。对于边界上不同的直线段，保证它们只会在公共端点处相交。

有的时候 Alice 会指着平面上一个点，然后问 Bob：“这个点是在多边形的里面呢，还是外面呢，还是在边界上呢？”

这个时候，如果她所指的点是多边形的一个顶点或者在多边形某条边的边界上，都将被认为是在多边形的边界上。

还有的时候，Alice 为了加大难度，会删除连接 a 和 b 的边，并插入新的点 c（新插入的点保证不与任何已有的端点重合，也不在任何边界上），然后新增 a 到 c 的边与 b 到 c 的边，从而得到一个新的简单多边形。

Alice 保证这样的操作得到的新图形总是简单多边形。

Bob 要做的，就是准确回答出 Alice 的提问。而实际上，Alice 的每一次提问都将由 Bob 上一次的回答决定，虽然这个回答是唯一的，但却意味着如果 Bob 不能回答出前一个问题，就不能拿到 Alice 的下一个问题。不过，Alice 对多边形的修改确实事先准备好的。

详细来说：Alice 的每一次修改命令可以看作是一个六元组：

〈x_a，y_a，x_b，y_b，x_c，y_c〉

表示删除了坐标位置（x_a，y_a），与坐标位置（x_b，y_b）的点之间的连边，并插入新的点（x_c，y_c）。这里我们保证坐标为（x_a，y_a）的点与坐标为（x_b，y_b）的点总是存在的。

因为 Alice 保证了所有出现的点（这包括了询问点）的坐标都是非负整数，且都小于 1000000000，且多边形中（这不包括询问点）任意两个点的 x 坐标不同，y 坐标也不同。

所以每一次询问 Alice 将给出 7 个非负整数：

r，x_{in}，y_{in}，x_{out}，y_{out}，x_{bd}，y_{bd}

而 Alice 这一次询问真正要询问的点（X，Y）的坐标将由上一次询问的点（x_0，y_0）与上一次询问的回答而决定。例如，若上一次询问的点在多边形外，则：

$$X = (r * x\_0 + x\_\{out\}) \bmod 1000000000$$
$$Y = (r * y\_0 + y\_\{out\}) \bmod 1000000000$$

对于第一次讯问，我们假设 x_0 = y_0 = 0。

输入：

输入文件的第一行有一个整数 N，表示初始时多边形的点数。

之后 N 行，每行一对非负整数 x 和 y（0<=x,y<1000000000）。按照某一顺序依次描述了多边形的所有顶点的坐标，并编号为 1 到 N。

这里我们只认为，对于平面上的一点(10^100,10^100)一定是处在多边形以外的。

之后一行有一个整数 Q，表示总的操作次数。

之后 Q 行，每行第一个数字 p，如果 p=0，则表示询问，如果 p=1，则表示修改。

对于询问，之后给出了 7 个非负整数，它们是：

r，x_{in}，y_{in}，x_{out}，y_{out}，x_{bd}，y_{bd}

对于修改，之后给出了 6 个整数，它们是：

x_a，y_a，x_b，y_b，x_c，y_c

输出：

对于每一次询问操作，单独输出一行且只包含一个字符串，它或者是 in、或者是 out、或者是 bd（均为小写字符），分别表示询问点在多边形的内、外或边界。

样例输入：

```
6
249999999 499999998
583333331 83333333
83333333 333333332
333333332 999999996
833333330 749999997
499999998 833333330
12
0 1 872826049 679758020 472526437 270998755 15447952 502239247
1 833333330 749999997 499999998 833333330 916666663 666666664
1 833333330 749999997 916666663 666666664 416666665 916666663
0 1 371653715 747730364 409617871 21996163 118531999 759280767
1 249999999 499999998 583333331 83333333 666666664 166666666
0 1 195920917 488293591 322952040 262793733 678458193 506876149
0 1 203963007 782710007 391614158 831643205 340800821 896322422
0 1 498571077 461554269 765704840 973009111 152064733 114249255
1 499999998 833333330 249999999 499999998 999999996 583333331
```

0 1 159294077 702544938 787871788 619972292 941209243 950700951
0 1 791254252 411705638 382076333 263993056 306662346 47793905
0 1 13359599 513224793 415037020 28305143 48117026 34994422

样例输出：

out
out
in
in
out
out
out
in

数据规模:

对于 10%的数据：N<=1000，Q<=5000。
存在 10%的数据：N<=1000，Q<=50000，没有修改操作。
存在 20%的数据：N<=50000，Q<=50000，没有修改操作。
存在 10%的数据：N<=50000，Q<=50000，每次询问操作的系数 r 都恒为 0。
存在 20%的数据：N<=50000，Q<=50000，每一次修改操作中，x 或者与 a 的横坐标相差不超过 1，或者与 b 的横坐标相差不超过 1。
对于 100%的数据：N<=50000，Q<=50000，所有坐标非负且均小于 1000000000，而 r 或者为 1 或者为 0。