In a playground, a group of kids is playing hide and seek. As the name suggests, the game is about kids hiding and seeking other kids. Each kid is either a *hiding* kid or a *seeking* kid. Hiding kids are kids that just try not to be found, while seeking kids are kids that try to find (hiding and seeking) kids.

As you may note, both hiding and seeking kids try not to be found, and for doing this they use some walls that there are in the playground. Each wall is represented by a line segment and each kid is represented by a point in the XY plane. Two kids see each other if and only if the line segment between them does not intersect any wall segment.

Your task is to calculate how many other kids each seeking kid can see. To simplify the problem, you may assume that walls do not intersect even at their endpoints. Moreover, no three points are collinear within the set formed by kids and endpoints of walls; this implies that kids are not inside walls, and that no two kids have the same location.

Input

The input file contains several test cases, each of them as described below.

The first line contains three integers S, K and W representing respectively the number of seeking kids, the total number of kids and the number of walls in the playground ($1 \leq S \leq 10$; $1 \leq K, W \leq 10^4$ and $S \leq K$). Each of the next K lines describes a kid with two integers X and Y ($-10^6 \leq X, Y \leq 10^6$), indicating that the location of the kid in the XY plane is the point (X, Y); the first S of these lines describe seeking kids. Each of the next W lines describes a wall with four integers X_1, Y_1, X_2 and Y_2 ($-10^6 \leq X_1, Y_1, X_2, Y_2 \leq 10^6$), indicating that the two endpoints of the wall in the XY plane are (X_1, Y_1) and (X_2, Y_2). You may assume that wall segments do not intersect and no three points given in the input are collinear.

Output

For each test case, output S lines, each of them containing an integer. In the i-th line write the number of other kids the i-th seeking kid can see.

Sample Input

```
2 3 2
0 0
100 0
0 100
50 -1 48 3
49 49 51 52
4 4 4
-100 0
0 100
0 -100
100 0
3 3 -2 -2
-101 50 101 50
-101 -101 101 -101
-49 -50 49 -50
5 6 4
40 40
60 10
70 30
60 80
30 81
20 40
0 10 40 50
10 61 30 61
-100 90 200 90
50 20 50 50
```

Sample Output

```
1
0
1
0
2
1
1
2
3
5
2
```