3655 Onion Layers

Dr. Kabal, a well recognized biologist, has recently discovered a liquid that is capable of curing the most advanced diseases. The liquid is extracted from a very rare onion that can be found in a country called Onionland. But not all onions of Onionland are worth to take to the lab for processing. Only those onions with an odd number of layers contain the miraculous liquid. Quite an odd discovery!

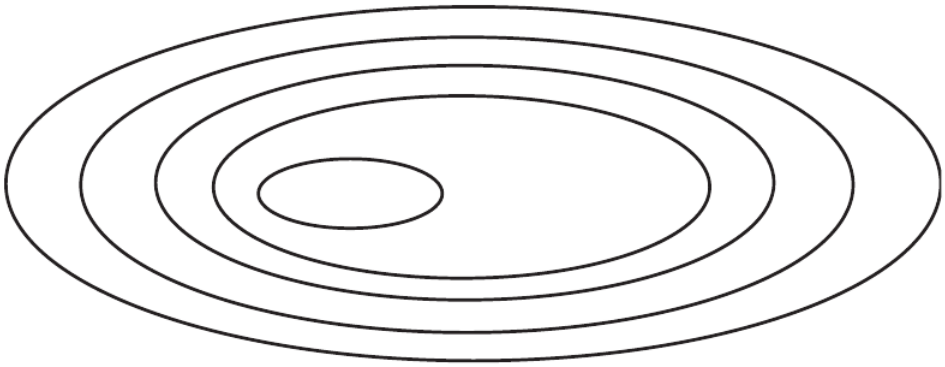


Figure 1: Onion from Onionland

Dr. Kabal has hired a lot of research assistants to collect and analyse onions for him. Since he does not want to share his discovery with the world yet, he didn't tell the assistants to look for onions with an odd number of layers. Instead, each assistant was given the task of collecting onions, and selecting points from each of the layer's outer borders, so that an approximation of the layer structure of the onion can be reconstructed later. Dr. Kabal told the assistants that the next step will be a "complicated analysis" of these points. In fact, all he will do is simply to use the points to count the number of layers in each of the onions, and select the ones with an odd number of layers.

Figure 2: Points collected by an assistant

It is clear that the approximation obtained by Dr. Kabal, from the points collected, might have a different *shape* than the original onion. For instance, only some of the points of the onion shown in Figure 1 would be extracted in the process, giving rise to a set of points as shown in Figure 2. With these points Dr. Kabal will try to approximate the original layers of the onion, obtaining something like what is shown in Figure 3. The approximation procedure followed by Dr. Kabal (whose result is shown in Figure 3) is simply to recursively find nested convex polygons such that at the end every point belongs to precisely one of the polygons. The assistants have been told to select points in such a way that the *number of layers in the approximation, if done in this recursive manner, will be the same as*

in the original onion, so that is fine with Dr. Kabal. The assistants are also aware that they need at least three points to approximate a layer, even the innermost one.

Figure 3: Dr. Kabal's approximation

Your task is to write a program that, given a set of points collected by an assistant (as shown in Figure 2), determines if the respective onion should be taken to the laboratory.

Input

The input contains several test cases. Each test case consists of an integer $3 \leq N \leq 2000$ in a single line, indicating the number of points collected by the assistants. Following, there are N lines, each containing two integers $-2000 \leq X, Y \leq 2000$ corresponding to the coordinates of each point.

The input is finished by a problem with $N = 0$ points, which should not be processed.

Output

There should be one line of output for each test case in the input. For each test case print the string

`Take this onion to the lab!`

if the onion should be taken to the laboratory or

`Do not take this onion to the lab!`

if the onion should not be taken to the laboratory.

Sample Input

```
7
0 0
0 8
1 6
3 1
6 6
8 0
8 8
11
2 6
3 2
6 6
0 0
0 11
1 1
```

```
1 9
7 1
7 9
8 10
8 0
0
```

Sample Output

```
Do not take this onion to the lab!
Take this onion to the lab!
```

6532 Hide and seek

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
```

6528 Disjoint water supply

Nlogonia is a queendom that consists of several cities located on a big mountain. The capital city is Logville, located on the mountain peak. Logville has a huge lake with a perfectly round shape, appropriately named "The Big O". This is the only lake with drinkable water in the entire queendom, so it is used to supply all cities. Some cities in Nlogonia are connected with water pipes that allow the distribution of the water. As there are no pumps, each pipe carries water from a city to another city at a lower altitude, using gravity.

Nlogonia's water system has been a source of worries for the Queen, because since cities depend on other cities for their water supply, hot discussions occur about how much water a city is allowed to use. A water supply path is a sequence of cities in decreasing order of altitude, starting in Logville and such that there is a pipe connecting each pair of consecutive cities in the sequence. Two cities have disjoint water supply if and only if there exist two water supply paths, one supply path ending in each of the cities, such that Logville is the only city that is present in both paths. Notice that Logville itself has disjoint water supply with every other city.

The Queen considers disjoint water supply a nice property because it reduces dependency problems and also avoids water outages to spread as quickly through Nlogonia. She therefore ordered a survey to assess the current state of water supply disjointness in the whole queendom. Being the cleverest advisors in the Queen's court, you have been summoned to help calculate the number of pairs of distinct cities that have disjoint water supply.

Input

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

The first line contains two integers C ($2 \leq C \leq 1000$) and P ($1 \leq P \leq 10^5$), representing respectively the number of cities and the number of water pipes in Nlogonia. Cities are identified with different integers from 1 to C, in strictly decreasing order of altitude (no two cities have the same altitude); Logville is city 1. Each of the next P lines describes a pipe with two integers U and V ($1 \leq U < V \leq C$) indicating that the pipe connects city U with city V. You may assume that no two pipes connect the same pair of cities, and that for each city in Nlogonia there is at least one water supply path that ends in it.

Output

For each test case, output a line with an integer representing the number of pairs of distinct cities that have disjoint water supply.

Sample Input

```
6 6
1 2
1 3
1 4
2 5
2 6
3 6
8 11
1 2
```

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

Sample Output

```
14
26
```

6026 Asteroid Rangers

The year is 2112 and humankind has conquered the solar system. The Space Ranger Corps have set up bases on any hunk of rock that is even remotely inhabitable. Your job as a member of the Asteroid Communications Ministry is to make sure that all of the Space Ranger asteroid bases can communicate with one another as cheaply as possible. You could set up direct communication links from each base to every other base, but that would be prohibitively expensive. Instead, you want to set up the minimum number of links so that everyone can send messages to everyone else, potentially relayed by one or more bases. The cost of any link is directly proportional to the distance between the two bases it connects, so this doesn't seem that hard of a problem.

There is one small difficulty, however. Asteroids have a tendency to move about, so two bases that are currently very close may not be so in the future. Therefore as time goes on, you must be willing to switch your communication links so that you always have the cheapest relay system in place. Switching these links takes time and money, so you are interested in knowing how many times you will have to perform such a switch.

A few assumptions make your task easier. Each asteroid is considered a single point. Asteroids always move linearly with a fixed velocity. No asteroids ever collide with other asteroids. Also, any relay system that becomes optimal at a time $t \geq 0$ will be uniquely optimal for any time s satisfying $t < s < t + 10^{-6}$. The initial optimal relay system will be unique.

Input

Each test case starts with a line containing an integer n ($2 \leq n \leq 50$) indicating the number of asteroid bases. Following this are n lines, each containing six integers x, y, z, v_x, v_y, v_z. The first three specify the initial location of an asteroid ($-150 \leq x, y, z \leq 150$), and the last three specify the x, y, and z components of that asteroid's velocity in space units per time unit ($-100 \leq v_x, v_y, v_z \leq 100$).

Output

For each test case, display a single line containing the case number and the number of times that the relay system needs to be set up or modified.

Sample Input

```
3
0 0 0 0 0 0
5 0 0 0 0 0
10 1 0 -1 0 0
4
0 0 0 1 0 0
0 1 0 0 -1 0
1 1 1 3 1 1
-1 -1 2 1 -1 -1
```

Sample Output

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

7208 Fence the vegetables fail

At the early age of 40, Alice and Bob decided to retire. After more than two decades working as examples for networking protocols, game theoretical books and several other texts, they were tired. To remain active, they decided to get into gardening.

Alice and Bob planted several vegetable plants in a huge field. After finishing, they realized that their plants needed protection from wild animals, so they decided to build a fence around them. The field is represented as the XY plane, and each vegetable plant as a different point in it. A fence is represented as a polygon in the plane. However, not every polygon is a valid fence. The fence needs to be a single simple polygon with each of its sides parallel to one of the axes. Of course, the polygon should contain all the points representing vegetable plants. A fence too close to the plants or to itself could make it difficult to walk around, so each side of the polygon needs to be away from all plants and all non-adjacent sides.

Unfortunately, Alice and Bob subcontracted the construction of the fence to a nasty multinational. The company had a lot of lawyers on payroll, but no good fence designers, so they failed to comply to all requirements. They built a fence which is a simple polygon with sides parallel to the axes and whose sides are away from plants and itself. However, they forgot to make the fence contain all the plants!

Alice and Bob want to assess the extent of the problem. Since not all plants are equally valuable to them, they want to know the total value of the plants that were left outside the fence.

Input

The input contains several test cases; each test case is formatted as follows. The first line contains two integers P and V, representing respectively the number of plants and the number of vertices of the polygonal fence ($1 \leq P, V \leq 10^5$). Each of the next P lines describes a different plant with two integers X_p and Y_p, indicating the coordinates of the plant ($-10^9 \leq X_p, Y_p \leq 10^9$). The value of the p-th plant in the input is p, for $p = 1, 2, \ldots, P$. Each of the next V lines describes a vertex of the fence with two integers X_v and Y_v, indicating the coordinates of the vertex ($-10^9 \leq X_v, Y_v \leq 10^9$). Vertices are given in counter clockwise order. Each of these points is an actual vertex of the polygon, that is, it is not collinear with its two adjacent vertices. The represented polygon is a simple polygon with each side parallel to an axis. No two plants are in the same position, and no plant lies on a fence's side.

Output

For each test case in the input, output a line with an integer representing the sum of the values of all the plants that lie outside the fence.

Sample Input

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

```
4 3
2 3
2 5
0 5
6 12
6 5
1 9
3 6
3 4
2 0
4 4
5 8
5 3
2 3
2 5
4 5
4 7
0 7
0 1
7 1
7 10
0 10
0 8
1 4
1 1
2 0
2 2
0 2
0 0
```

Sample Output

```
6
15
0
```

CVXPOLY - Convex Polygons

no tags

English Vietnamese

You are given n points in the 2-D cartesian coordinate system. You are to determine the number of convex polygons with 3 or more vertices which can be formed by choosing a subset of the given points. To make matters simple, the input obeys the following conditions:

(1) No 3 points in the input are collinear.

(2) No 2 points will have the same coordinates.

Since the result can be quite large, you are required to output (result % 1234567) instead.

Input

First line contains an integer T, the number of test cases. In each test case, first line contains n, the number of points in the corresponding test case, next n lines contain 2 space separated integers denoting the coordinate of ith point. Absolute value of the coordinates do not exceed 10000.

Output

T lines each corresponding to the answer of corresponding test case.

Example

```
Input:
2
4
0 0
2 0
2 2
0 2
6
0 0
2 0
2 2
0 2
1 -1
1 3

Output:
5
42
```

Constraints

```
Input Set 1 : numberOfTestCases <= 100, 3 <= n <= 10 timeLimit: 5 seconds
Input Set 2 : numberOfTestCases <= 50, 3 <= n <= 100 timeLimit: 5 seconds
```

C. Edo and Magnets

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Edo has got a collection of n refrigerator magnets!

He decided to buy a refrigerator and hang the magnets on the door. The shop can make the refrigerator with any size of the door that meets the following restrictions: the refrigerator door must be rectangle, and both the length and the width of the door must be **positive integers**.

Edo figured out how he wants to place the magnets on the refrigerator. He introduced a system of coordinates on the plane, where each magnet is represented as a rectangle with sides parallel to the coordinate axes.

Now he wants to remove no more than k magnets (he may choose to keep all of them) and attach all remaining magnets to the refrigerator door, and the area of the door should be as small as possible. A magnet is considered to be attached to the refrigerator door if **its center** lies on the door or on its boundary. The relative positions of all the remaining magnets must correspond to the plan.

Let us explain the last two sentences. Let's suppose we want to hang two magnets on the refrigerator. If the magnet in the plan has coordinates of the lower left corner (x_1, y_1) and the upper right corner (x_2, y_2), then its center is located at $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$ (may not be integers). By saying the relative position should correspond to the plan we mean that the only available operation is translation, i.e. the vector connecting the centers of two magnets in the original plan, must be equal to the vector connecting the centers of these two magnets on the refrigerator.

The sides of the refrigerator door must also be parallel to coordinate axes.

Input

The first line contains two integers n and k ($1 \le n \le 100\,000$, $0 \le k \le min(10, n - 1)$) — the number of magnets that Edo has and the maximum number of magnets Edo may not place on the refrigerator.

Next n lines describe the initial plan of placing magnets. Each line contains four integers x_1, y_1, x_2, y_2 ($1 \le x_1 < x_2 \le 10^9$, $1 \le y_1 < y_2 \le 10^9$) — the coordinates of the lower left and upper right corners of the current magnet. The magnets can partially overlap or even fully coincide.

Output

Print a single integer — the minimum area of the door of refrigerator, which can be used to place at least $n - k$ magnets, preserving the relative positions.

Examples

input Copy

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

output Copy

```
1
```

input Copy

```
4 1
1 1 2 2
1 9 2 10
9 9 10 10
9 1 10 2
```

output Copy

```
64
```

input Copy

```
3 0
1 1 2 2
1 1 1000000000 1000000000
1 3 8 12
```

output Copy

```
249999999000000001
```

Note

In the first test sample it is optimal to remove either the first or the third magnet. If we remove the first magnet, the centers of two others will lie at points (2.5, 2.5) and (3.5, 3.5). Thus, it is enough to buy a fridge with door width 1 and door height 1, the area of the door also equals one, correspondingly.

In the second test sample it doesn't matter which magnet to remove, the answer will not change — we need a fridge with door width 8 and door height 8.

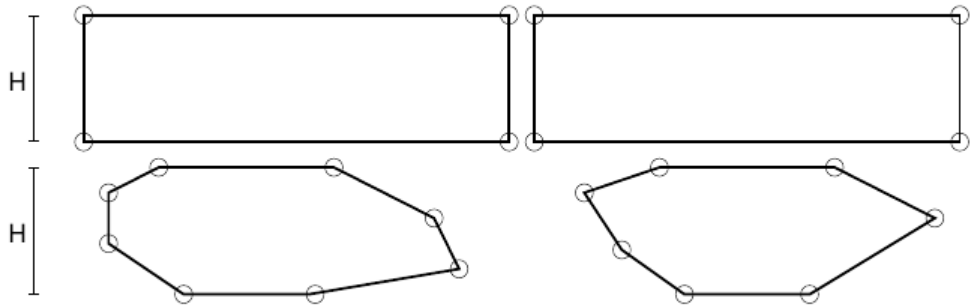
In the third sample you cannot remove anything as $k = 0$.

8188 Arranging tiles

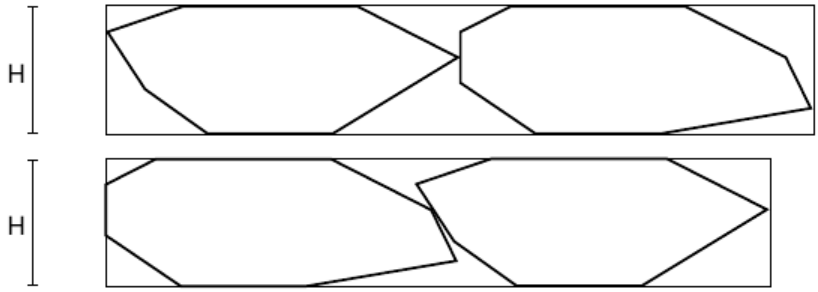
A set of rectangular stone tiles, all of them having the same height H, had their original four corners cut in different ways so that two properties were kept:

1. Each tile is still a simple convex polygon.
2. Each tile has two parallel sides that are part of the bottom and top sides of the original rectangular tile, which implies that the height H was preserved.

The figure below illustrates two tiles before and after the cuts. The corners are highlighted with small circles.



We need to place all tiles, side by side and without overlap, along a frame of height H, for transportation. The tiles can be translated from their original positions, but they may not be rotated or reflected. Since their convex shapes may be very different, the order in which we place the tiles along the frame matters, because we want to minimize the width of the frame. The next figure shows the two possible orders for the tiles from the previous figure, the second order being clearly the one that minimizes the width of the frame.



Given the description of the set of tiles, your program must compute the minimum width for a frame of the same height of the tiles that contains all of them, side by side and without overlap.

Input

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

The first line contains an integer N ($1 \leq N \leq 14$) representing the number of tiles. Following, there are N groups of lines, each group describing a tile, all of them having the same height. Within each group describing a tile, the first line contains an integer K ($4 \leq K \leq 10^4$) representing the number

of corners of the tile. Each of the next K lines describes a corner of the tile with two integers X ($-10^8 \leq X \leq 10^8$) and Y ($0 \leq Y \leq 10^8$), indicating the coordinates of the corner in the XY plane.

The corners are given in counterclockwise order. The first corner is (0, 0) and the second corner is of the form $(X, 0)$ for $X > 0$, this side being the bottom side of the tile. The tile has the shape of a simple convex polygon with a top side parallel to its bottom side.

Output

For each test case, output a single line with a rational number indicating the minimum width for a frame of the same height of the tiles that contains all of them, side by side and without overlap.

The result must be output as a rational number with exactly three digits after the decimal point, rounded if necessary.

Sample Input

```
3
4
0 0
1 0
0 5
-1 5
4
0 0
1 0
2 5
1 5
4
0 0
3 0
2 5
1 5
3
4
0 0
204 0
412 1031
-253 1031
6
0 0
110 0
290 436
100 1031
0 1031
-400 750
5
0 0
120 0
100 1031
0 1031
-281 93
```

Sample Output

```
5.000
1420.754
```

Problem B

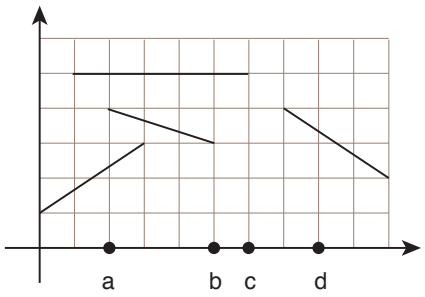
Balloon

File: `balloon.[c|cpp|java]`

One of the main difficulties of organizing a Programming Contest is collecting the balloons that escape and are trapped on the roof of the competition hall: often the contract with the hall owner requires that the hall must be cleaned soon after the event, otherwise a fine is applied.

This year the organization of our competition have been more prudent: it got the project design of the ceiling of the hall, and wants your help to determine what will happen to a loose balloon depending on the position on the ground where it is released (that is, whether it is blocked by the ceiling or escapes to the outside of the hall).

The ceiling of the hall consists of several plans that, viewed from the side, can be described by line segments, as shown in the figure below:



The balloon may be considered to be a point. When a balloon touches a line segment that is horizontal, it gets stuck. When a balloon touches a segment that is tilted, the balloon glides to the highest point of the segment and escapes. It may then escape from the hall or it may touch more segments. There are no points in common between the line segments that form the ceiling.

For example, if a balloon is released at the positions marked as a or b, it will be stuck in the position with coordinates $(2, 5)$; if a balloon is released at the position marked c, it will be stuck in the position of coordinates $(6, 5)$; and if the balloon is released at the position marked as d, it will not be blocked and will escape out of the hall in the position of coordinate $x = 7$.

Write a program that, given the description of the ceiling of the hall as line segments, answers a series of queries about the final positions of balloons released at the hall floor.

Input

The first line of input contains two integers N and C indicating, respectively, the number of segments describing the ceiling, and the number of queries. Each of the next N lines contains four integers X_1, Y_1, X_2, Y_2, describing a line segment from the ceiling, with end points at coordinates (X_1, Y_1) and (X_2, Y_2).

Each of the next C describe a query and contains an integer X, indicating that the query wants to determine what happens to a balloon releaset at the point of coordinates $(X, 0)$.

Output

For each query in the input your program must output a single line. If the balloon escapes the hall, the line must contain a single integer X, indicating the x coordinate where eht balloon escapes the hall. Otherwise, the line must contain two integers X and Y indicating the position (x, y) where the balloon gets stuck in the ceiling.

Restrictions

- $1 \leq N \leq 10^5$
- $1 \leq C \leq 10^5$
- $0 \leq X_1, X_2 \leq 10^6$, $0 < Y_1, Y_2 \leq 10^6$, $X_1 \neq X_2$
- no two x coordinate values are equal, considering all segments.
- $0 \leq X \leq 10^6$

Examples

Input	**Output**
4 4 0 1 3 3 1 5 6 5 5 3 2 4 7 4 10 2 2 5 8 6	2 5 2 5 7 6 5

Input	**Output**
4 3 1 3 4 2 10 3 7 4 2 3 8 3 3 5 5 4 4 9 8	1 7 8 3