Problem A
Grandpa’s Walk

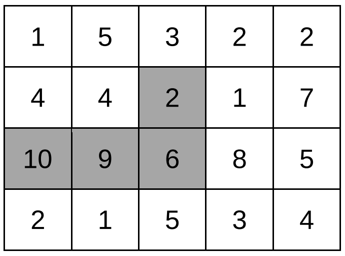
In spite of his old age, Grandpa Pierre likes to take a walk. He believes that by taking more exercises, he will be healthy and live a longer life. The area which Grandpa used to explore can be represented as a N x M map, where each cell contains the height information of that area. Grandpa can start from any cell and walk from a cell to its adjacent cells (north, south, west, east) if and only if the destination cell’s height is strictly lower than his current cell’s height.

For example, consider a 4 x 5 map below.

1	5	3	2	2
4	4	2	1	7
10	9	6	8	5
2	1	5	3	4

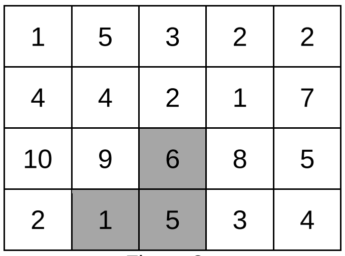
Figure 1

There are many possible routes, for example:



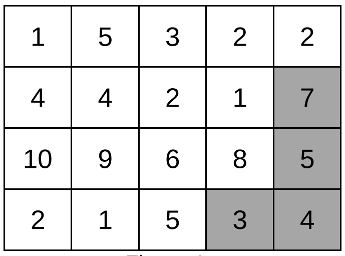
1	5	3	2	2
4	4	**2**	1	7
10	**9**	**6**	8	5
2	1	5	3	4

Figure 2



1	5	3	2	2
4	4	2	1	7
10	9	**6**	8	5
2	**1**	**5**	3	4

Figure 3



1	5	3	2	2
4	4	2	1	**7**
10	9	6	8	**5**
2	1	5	**3**	**4**

Figure 4

- Figure 2 shows a route of length 4 which starts from a cell with height 10 and ends at a cell with height 2 (10 - 9 - 6 - 2).
- Figure 3 shows a route of length 3 which starts from a cell with height 6 and ends at a cell with height 1 (6 - 5 - 1).
- Figure 4 shows a route of length 4 which starts from a cell with height 7 and ends at a cell with height 3 (7 - 5 - 4 - 3)

There are many other routes we can get from Figure 1 which are not shown.

A route is considered maximal if and only if one cannot extend the route at the beginning or at the end to make it a longer route.

- Figure 2 is not a maximal route. We can extend the end of the route from cell with height 2 to its east, a cell with height 1, such that the route is 10 - 9 - 6 - 2 - 1 with length of 5.
- Figure 3 is not a maximal route. We can extend the beginning of the route from cell with height 6 to its east, a cell with height 8, such that the route is 8 - 6 - 5 - 1 with length of 4.
- Figure 4 corresponds to a maximal route because we can extend neither the beginning nor the end of the route.

Your task is to help Grandpa Pierre to count how many maximal routes there are in the given map.

Input

The first line of input contains an integer T (T ≤ 100) denoting the number of cases. Each case begins with two integers N and M (1 ≤ N, M ≤ 50) denoting the number of rows and columns in the map respectively. The next N lines each contains M integers A_{ij} (1 ≤ A_{ij} ≤ 100) denoting the height of a cell in i^th row and j^th column.

Output

For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is the number of maximal route we can find in the given map such that the route visits cells in strictly decreasing height order.

Sample Input	Output for Sample Input

4
4 5
1 5 3 2 2
4 4 2 1 7
10 9 6 8 5
2 1 5 3 4
3 4
8 8 8 8
8 8 8 8
8 8 8 8
2 5
2 1 6 7 8
6 7 1 4 5
1 10
4 3 2 1 2 3 4 5 6 7
	Case #1: 20
	Case #2: 12
	Case #3: 7
	Case #4: 2

Explanation for 2nd sample input.
All cells have the same height, so there are only routes with length 1, and there are 12 of them.

Explanation for 3rd sample input.
List of visited cells' height of all maximal routes are:

1. 7 - 1
2. 7 - 1
3. 7 - 6 - 2 - 1
4. 8 - 5 - 4 - 1
5. 8 - 7 - 4 - 1
6. 8 - 7 - 6 - 1
7. 8 - 7 - 6 – 1

Notice there are two (7 - 1) above and both are different routes. The first one corresponds to a route from cell (1, 1) to cell (0, 1), while the second one is from cell (1, 1) to cell (1, 2). A same explanation applies for (8 - 7 - 6 - 1).

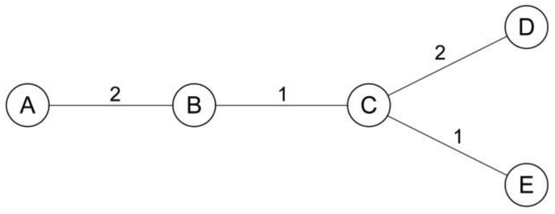
Problem B
Let's Go Green

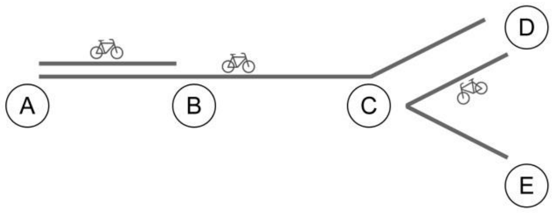
There are a lot of cities in Tree Land kingdom and each city is connected to other cities with roads such that there is exists exactly one path to go from one city to any other city. Each road in Tree Land connects exactly two different cities.

The king of this kingdom wants to encourage his citizen to have a good sense of natural environmental responsibility. One of his agenda is to promote bicycle, a wheeled human-powered transportation, as the main vehicle for his citizen to travel between places, rather than motor-cycle or auto-mobile which might produce a bad pollution to the environment.

In order to determine the progress of his campaign, the king asked the chancellor to monitor the number of bicycles which pass each road in the kingdom. The chancellor executed the king's order immediately and reported the result to his king, i.e., for each road in the kingdom he reported the number of bicycles passing those roads in one full day. The king was happy, but it didn't take a long time for him to realize that this report could be misleading. The report only says the number of bicycles passing each road and doesn't say anything about the total number of bicycles being used in that day. It also doesn't contain any information on the direction of reported bicycles.

For example, consider a city structure shown in the figure (left) below. There are 2 bicycles passing road connecting city A and B, 1 on road connecting B and C, 2 on C - D and 1 on C - E. The total number of bicycles recorded is 2 + 1 + 2 + 1 = 6, but it doesn't mean the actual number of bicycles used is 6. There could be cases where a bicycle is used to travel between city X and Z passing some other cities, such that this one bicycle is recorded several times, once on each road between X and Z. The figure on the right describes one possibility where there are only 3 bicycles, i.e., one is used to travel between A and B, one is used to travel between A and D passing B and C, and one is used to travel between D and E passing C.





Assume that no bicycles are used to pass any road more than once. If a road detects P passing bicycles, then those are P different bicycles.

Now your task is to help the king to determine the minimum number of bicycles used on the report.

Input

The first line of input contains an integer T (T ≤ 10) denoting the number of cases. Each case begins with an integer N (2 ≤ N ≤ 100,000) denoting the number of cities in the kingdom numbered from 1..N. The next N-1 lines each contains three integers A, B and C (1 ≤ A, B ≤ N; A ≠ B; 0 ≤ C ≤ 100) denoting that there are C bicycles recorded passing road connecting A and B. Note that this N-1 lines describe the complete city structures in the kingdom.

Output

For each case, output “Case #X: Y”, where X is case number starts from 1 and Y is the minimum number of bicycles used in the respective case.

Sample Input	Output for Sample Input
`2`	`Case #1: 3`
`5`	`Case #2: 5`
`1 2 2`	
`2 3 1`	
`3 4 2`	
`3 5 1`	
`7`	
`1 2 1`	
`2 3 2`	
`2 4 1`	
`2 5 3`	
`5 6 2`	
`5 7 3`	

Problem C
Stop Growing!

You are given five integers: A_0, B_0, C_0, D_0 and E_0. Each number in the sequence, A_k, B_k, C_k, D_k and E_k are calculated by the following formula:

$$\begin{aligned} A_k &= A_{k-1} + B_{k-1} \\ B_k &= B_{k-1} + C_{k-1} \\ C_k &= C_{k-1} + D_{k-1} \\ D_k &= D_{k-1} + E_{k-1} \\ E_k &= E_{k-1} + A_{k-1} \end{aligned}$$

For example, let's start with $A_0 = 2$, $B_0 = 1$, $C_0 = 0$, $D_0 = -1$, $E_0 = 4$.

k	A	B	C	D	E
0	2	1	0	-1	4
1	3	1	-1	3	6
2	4	0	2	9	9
3	4	2	11	18	13
4	6	13	29	31	17
…	…	…	…	…	…

The table above shows the iteration up to k = 4. These numbers might (or might not) grow to infinite.

Your task is to determine the minimum value $r \geq 0$ where $A_r + B_r + C_r + D_r + E_r \geq M$ for some integer M. In the example above, if M = 50, then r = 4, because:

$$A_4 + B_4 + C_4 + D_4 + E_4 = 6 + 13 + 29 + 31 + 17 = 96.$$

You can check for k = 0..3, there will be no k such that the sum of $A_k..E_k$ is no less than 50.

There might be some cases where it is not possible for the integers to reach M, output -1 if such case happened.

Input
The first line of input contains an integer T (T ≤ 1,000) denoting the number of cases. Each case contains six integers in a line A_0, B_0, C_0, D_0, E_0 and M ($-10^8 \leq A_0, B_0, C_0, D_0, E_0, M \leq 10^8$) as stated in the problem statement.

Output
For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is the minimum value r such that the sum of $A_r..E_r$ is no less than M. output Y = -1 if there's no such r.

Sample Input	Output for Sample Input

`4`
`2 1 0 -1 4 50`
`500 10 70 -100 -200 100`
`0 0 0 0 0 10`
`1 1 1 1 1 500`
	`Case #1: 4`
	`Case #2: 0`
	`Case #3: -1`
	`Case #4: 7`

Problem D

Retrenchment

ACM Telco is one of the well-known mobile telecommunication services providers in Byte Land. Lately ACM Telco has a hard time maintaining their revenue because of the high operational cost in maintaining their cell towers across the land.

To save their company, the board of directors have a plan to build a new tower and demolishing two existing towers such that they don't sacrifice the signal coverage by much, i.e. by choosing two nearest existing towers from the newly build one to be demolished, but there is one restriction: these two towers to be demolished should be chosen from towers inside a certain region (which resembles a closed simple polygon). Tower which located at the boundary of the region is considered inside the region.

The distance function used in this problem is Euclidean distance. Let there be two points (x_1, y_1) and (x_2, y_2), the distance between them is:

$$D = ((x_1 - x_2)^2 + (y_1 - y_2)^2)^{0.5}$$

You are given the location of all N existing towers (x_i, y_i) in a Cartesian plane and R number of regions. For each region there is M positions in which for each position you should find the two nearest towers in the region.

Input

The first line of input contains an integer T (T ≤ 10) denoting the number of cases.

The first line of each case contains a positive integer N (5 ≤ N ≤ 20,000) denoting the number of towers in the land. The next N lines each represents the location of a tower given in format of "x_i y_i" (0 ≤ x_i, y_i ≤ 1,000,000). Each tower has an id from 1 to N given in such order.

The following line contains an integer R (1 ≤ R ≤ 10) denoting the number of regions. The next R blocks each represents the region boundaries and queries on that region.

Each block begins with an integer B_k (3 ≤ B_k ≤ 20) denoting the number of corner points of the region boundaries. The points will be given in clockwise order in the next B_k lines with the same format as the tower position. You may assume that the given region will form a closed simple polygon. The next line contains an integer M_k (1 ≤ M_k ≤ 5,000) denoting the number of position queries for that region. The following M_k lines, each contains the query position (the new tower) in the same format as the tower position.

Output

For each case, output "`Case #X:`" in a line, where `X` is case number starts from 1. For each region in the case, output "`Region K`" in a line where `K` is the region number starts from 1. The next M_k lines for each region contains two integers "`A B`" denoting the id of two nearest towers in the region from the query position. A is the id of the nearest tower while B is id for the second nearest tower. In case of tie, prioritize the tower with smaller id. Assume that there are at least two towers in the region.

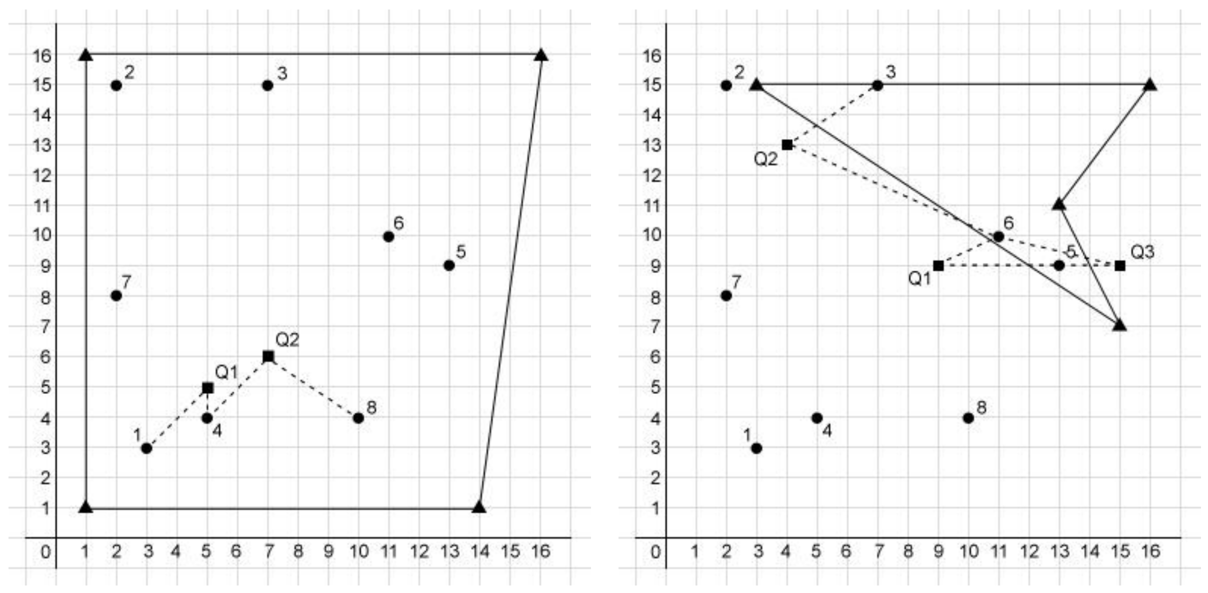
Sample Input

```
1
8
3 3
2 15
7 15
5 4
13 9
11 10
2 8
10 4
2
4
1 1
1 16
16 16
14 1
2
5 5
7 6
4
3 15
16 15
13 11
15 7
3
9 9
4 13
15 9
```

Output for Sample Input

```
Case #1:
Region 1
4 1
4 8
Region 2
6 5
3 6
5 6
```

Explanation for 1st sample input.



The left figure corresponds to query on the 1st region, while the right figure corresponds to query on the 2nd region.

Problem E
Bee Tower

There are N towers in Bee Land known as Bee Towers. These towers are located in a straight line and the height may vary between one tower and the other.

Jolly is a unique bee, even though he is very smart and strong, he can't fly. Jolly can jump from one point to any other point if the destination height is not H higher than his current height and no farther than W width away. He can jump to any point that is lower than his initial point given the horizontal W still can be reached. Jolly can only jump from one tower to its adjacent tower(s). He cannot jump and skip a tower even though the destination is within his reach. Note that he can only jump to/from ground or the top of each tower.

In Bee Land, the highest towers (might be more than one) are considered sacred. Jolly loves high place and he wants to reach the top of (any) sacred tower. Initially he's at the ground, and in order to reach the top of highest tower he might need to jump to several lower and reachable towers before he could reach the highest one. He's able to jump to any tower in the land from the ground if the tower's height is no more than H.

For example, let there be 5 towers located at 5, 7, 8, 12 and 13 with height 3, 5, 8, 5 and 3 respectively as shown in Figure 1. As you can see, there is only one highest tower which is the third tower which has height 8. If H = 3 and W = 2, then Jolly can jump from the ground to first tower, and then to second tower, and finally reach the third which is the highest tower. He can't reach third tower by jumping from fifth tower as from the fourth to the third tower, the horizontal distance is 4 which is more than W = 2.

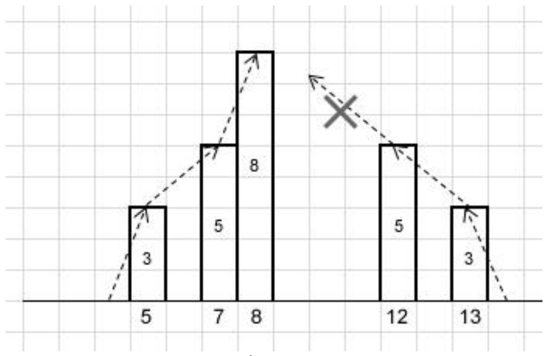


Figure 1

Did we mention that Jolly is strong? Jolly could move the towers such that the horizontal distance between towers are not more than W. There are some restrictions though when moving towers:

- He cannot move a tower pass through another tower (the tower order must remain).
- The sacred towers cannot be moved.

The cost of moving a tower is equal to the move distance multiplied by the tower's height, e.g. moving a tower of height 10 from position 3 to position 7 will cost (7 - 3) * 10 = 40.

For example, let there be 5 towers located at 2, 5, 8, 13 and 14 with height 4, 3, 6, 5 and 9 respectively as shown in Figure 2.

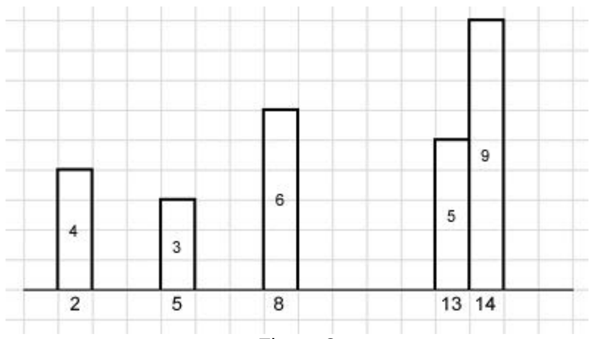


Figure 2

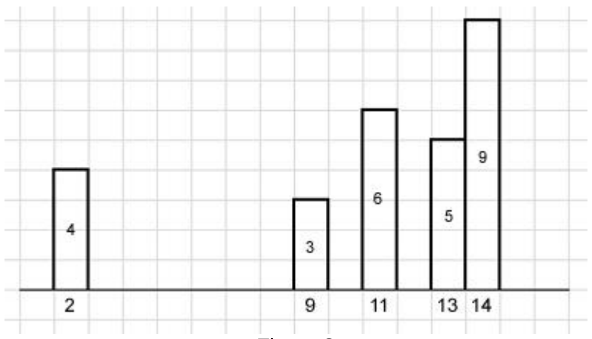


Figure 3

The highest tower is the 5th tower with height 9. Let H = 4 and W = 2. With such H and W, Jolly cannot reach the highest tower without moving any tower. One possible solution is to move 3rd tower from 8 to 11 and 2nd tower from 5 to 9, so he can reach the highest tower from the 2nd tower (Figure 3). The cost of that solution is = moving 3rd tower + moving 2nd tower = (11 - 8) * 6 + (9 - 5) * 3 = 18 + 12 = 30.

There is another solution by moving the 4th tower from 13 to 12, the 3rd tower from 8 to 10 and the 2nd tower from 5 to 8 as shown in Figure 4. The cost of this solution is = moving 4th tower + moving 3rd tower + moving 2nd tower = (13 - 12) * 5 + (10 - 8) * 6 + (8 - 5) * 3 = 5 + 12 + 9 = 26; which has a smaller cost than the previous solution.

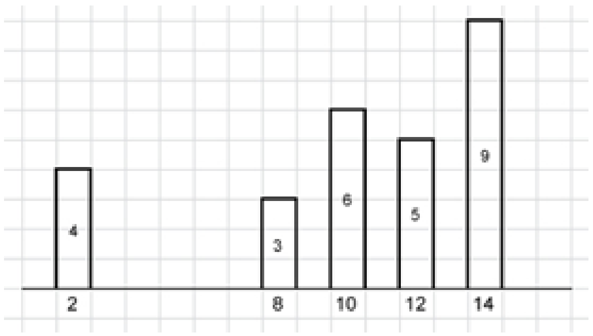


Figure 4

Given the location and height of all towers in Bee Land, your task is to determine the minimal cost needed for Jolly to reach the top of (any) sacred tower. If it's not possible for Jolly to reach any sacred tower, output -1.

Input

The first line of input contains an integer T (T ≤ 50) denoting the number of cases. The first line of each case contains three integers N (1 ≤ N ≤ 50), H (1 ≤ H ≤ 500) and W (1 ≤ W ≤ 100) denoting the number of towers, H and W as defined in the problem statement respectively. The next N lines each contains two integer p_i and h_i ($1 \le p_i, h_i \le 500$) denoting the position and the height of ith tower.

Output

For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is the minimum cost required for Jolly to reach the top of (any) sacred tower, or -1 if it's not possible to do so.

Sample Input	Output for Sample Input

4
5 4 2
2 4
5 3
8 6
13 5
14 9
2 4 10
5 3
6 100
6 3 1
1 3
2 6
3 9
4 4
5 7
6 10
4 6 2
1 8
4 5
10 5
14 8
	Case #1: 26
	Case #2: -1
	Case #3: 0
	Case #4: 5

Explanation for 4th sample input.
There are two sacred towers: 1st and 4th tower. Reaching the 1st tower costs less as he only needs to move the 2nd tower with height 5 one position to the left in order to reach the 1st tower.

Problem F

Knots

Indonesian children like to play with rubber bands, partly because they are very widely available, and partly because a lot of villagers could not afford modern, more expensive toys for their kids.

The original shape of the rubber band is a simple circle like in Figure 1. Figure 2 and 3 are some origami kids can make with rubber bands.

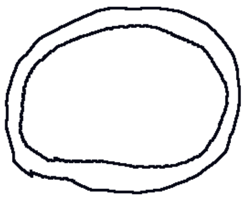


Figure 1

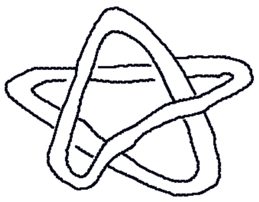


Figure 2

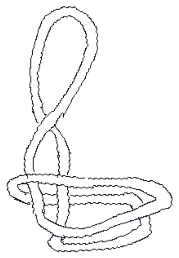
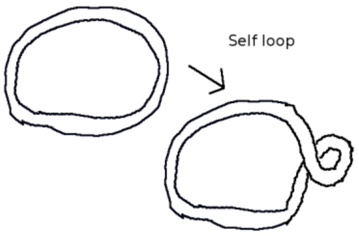
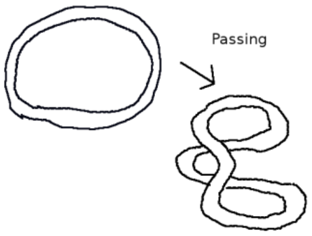


Figure 3

There are only two kinds of basic transformations of a rubber band:

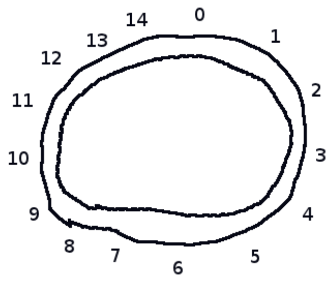
1. Self-loop
2. Passing





The two transformations, when repeated multiple times on different parts of the rubber band, can yield any of the valid origami.

Now your task is to detect whether a given origami is possible or impossible to be formed from the original rubber band shape.



The description of the origami is as follows. Suppose the rubber band has length L, and points in the rubber band is marked 0, 1, 2, ..., L-1, as illustrated on the right. Then the origami can be described with exactly P pairs (A_i, B_i) of distinct numbers, such that when the origami is placed in a flat plane and looked at from the top of the plane, position A_i in the rubber band overlaps and occludes (is at the front of) position B_i.

Given the description of the overlappings of the flattened origami as seen from the top, determine whether the origami position is reachable from the original shape of the rubber band.

Input

The first line of input contains an integer T (T ≤ 100) denoting the number of cases. The first line of each case contains two integers, L (1 ≤ L ≤ 1,000,000) and P (1 ≤ P ≤ 5,000) as defined by the problem statement. The next P lines each contains A_i and B_i ($0 \le A_i, B_i < L$). A blank line follows the last line in each case. The input is guaranteed to satisfy these conditions:

- A position in an origami is mentioned in at most one pair with some other position, describing a unique overlap.
- The origami represented by the input is planar, that is, the overlappings result from the origami being flattened on a plane.

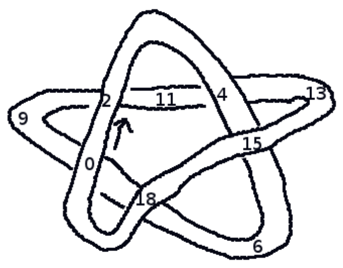
Output

For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is either a "YES" or a "NO" (without quotes). A "YES" indicates that the origami is possible to be formed from the original shape of the rubber band. A "NO" indicates otherwise.

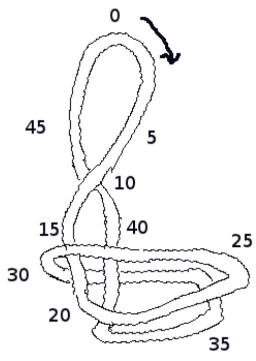
Sample Input	Output for Sample Input

3
20 5
0 8
2 10
4 12
15 5
18 7
 	
50 7
10 42
28 15
27 39
18 31
38 32
21 37
24 34
 	
20 5
0 8
10 2
4 12
15 5
7 18
	Case #1: YES
	Case #2: YES
	Case #3: NO

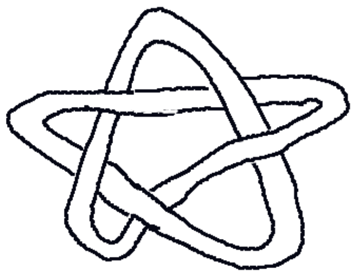
The following 3 figures correspond to sample input.



Sample #1



Sample #2


Sample #3

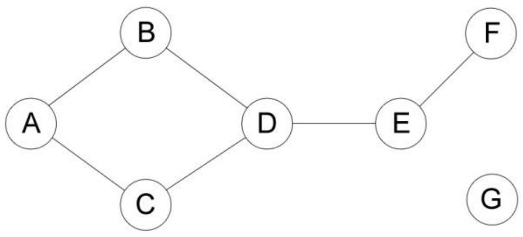
Problem G
Unique Path

Given an undirected graph, your task is to determine the number of pair of nodes (A, B) which has exactly one path connecting them. Path connecting (A, B) is defined as sequence of nodes a_1, a_2, … a_K where:

- $a_1 = A$
- $a_K = B$
- $A \neq B$
- Any two adjacent nodes in the sequence have an edge connecting them
- There are no edges used more than once in the sequence.

As you might notice, the reverse path from A to B is a path from B to A, so in this problem we consider pair (A, B) is the same as pair (B, A).

For example, consider the following graph.



In the example above, only pair (E, F) has exactly one possible path while the other pairs have more than one or zero possible path. Here are some pairs that have more than one possible paths, note that this list is not exhaustive.

- (A, B) : A-B, A-C-D-B.
- (A, C) : A-C, A-B-D-C.
- (A, D) : A-B-D, A-C-D.
- (A, E) : A-B-D-E, A-C-D-E.
- (B, C) : B-A-C, B-D-C.
- (B, E) : B-D-E, B-A-C-D-E.
- (D, E) : D-E, D-B-A-C-D-E, D-C-A-B-D-E.
- …

Note that some paths visit one or more nodes more than once, e.g. pair (D, E): path D-B-A-C-D-E visits D twice, but the path does not use any edges more than once, so it is a valid path. Also notice that there is no possible path from/to G.

Input

The first line of input contains an integer T ($T \leq 30$) denoting the number of cases. The first line of each case contains two integers N ($2 \leq N \leq 10{,}000$) and M ($1 \leq M \leq 100{,}000$) denoting the number of nodes and edges respectively. The nodes are numbered from 1 to N. The next M lines each contains two integers a_i and b_i ($1 \leq a_i, b_i \leq N$; $a_i \neq b_i$) denoting that there is an edge connecting node a_i and b_i. Assume that there is at most one edge connecting a_i and b_i.

Output

For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is the number of pair of nodes (A, B) which has exactly one path connecting them.

Sample Input	Output for Sample Input

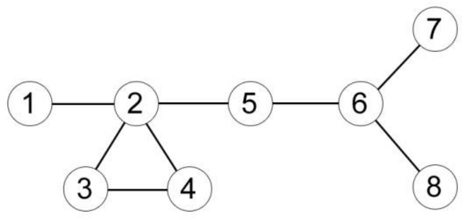
4
7 6
1 2
1 3
2 4
3 4
4 5
5 6
5 4
1 2
2 3
2 4
4 5
4 4
1 2
2 3
3 4
4 1
8 8
1 2
2 3
2 4
2 5
3 4
5 6
6 7
6 8
	Case #1: 1
	Case #2: 10
	Case #3: 0
	Case #4: 6

Explanation for the 1st sample input.
This is the example from the problem statement.

Explanation for the 2nd sample input.
The graph is a tree, so any pair of nodes has a unique path.

Explanation for the 3rd sample input.
All the nodes lie in the cycle, so no pair of nodes has unique path.

Explanation for the 4th sample input.
The graph input corresponds to the figure below.



The pairs which have unique path are: (5, 6), (5, 7), (5, 8), (6, 7), (6, 8) and (7, 8).

Problem H

Alien Abduction

In the past few days, there is a strange phenomenon where some people suddenly disappear and re-appear somewhere else randomly. Scientists from all over the world have worked so hard to figure out what has happened to these people, and the result is not pleasant at all, those people were abducted by an alien ship!

One scientist managed to figure out how the alien abduction works. For simplicity, let's assume our world is in a Cartesian plane (W x H), and each person is at position (X_i, Y_i). At any time, the cloaked alien ship flies over to a coordinate (X_k, Y_k), and then, with its highly advanced transporter technology, it abducts people whose position is within the (Manhattan) distance from the alien ship. To be precise, a person at (X_i, Y_i) is abducted by the alien ship if $|X_i – X_k| + |Y_i – Y_k| \le E_k$ where E_k is the transporter energy level.

Apparently, the alien wants to keep this abduction a secret operation. The abducted people are immediately returned back to Earth after the alien is "done" with them. The alien is supposed to return the abducted people exactly to the positions where they were abducted, but for some reasons, the people were returned to positions based on a certain pattern.

One of our brilliant scientists develop a device to capture various energy signals emitted from the alien ship when it is in operation and from those signals, the scientist is able to decipher the pattern and determines the exact location where the abducted people will be returned! According to the scientist, an abducted person will be returned to the following position on Earth:

$$X_i' = ((X_i * a) + (Y_i * b) + (i * c)) \bmod W$$
$$Y_i' = ((X_i * d) + (Y_i * e) + (i * f)) \bmod H$$

where:

- i is the person identification number
- (X_i, Y_i) is the position of person with identification number i where he/she is abducted,
- (X_i', Y_i') is the position where person with identification number i will be returned to, and
- a, b, c, d, e and f are the alien ship's energy signal captured by the scientist's device at the time of abduction.

By measuring other signals, the scientist also knows the total energy the alien ship has and is able to prove that the alien ship can only abduct (and return) at most 50,000 people in total across all abduction operations. Note that the same person may be abducted many times.

Knowing this, the Earth government wants to be able to locate all the people after the alien has finished all of its abduction operations. The brilliant scientist is asked to produce this report as soon as the alien leaves Earth for good. Since there are too many people on Earth, even the brilliant scientist is overwhelmed to manually produce this report and thus needs to create a program for this. Surprisingly, the brilliant scientist cannot program. Knowing that you are one the best programmers on Earth, the scientist asks for your help!

Input

The first line of input contains an integer T ($T \le 10$) denoting the number of cases. The first line of each case contains four integers N, Q, W and H denoting the number of people on Earth, the number of abduction operations recorded by the scientist's device, and the size of the world as stated in the problem statement respectively. The next N lines each contains two integers Xi and Yi denoting the initial position of the ith person with identifier i (the identifier starts from 1 to N). The next Q lines each contains 9 integers X_k, Y_k, E_k, a, b, c, d, e and f representing an alien abduction operation, which means the alien ship is currently at (X_k, Y_k) and abducts people using its transporter and emitting various energy signals (E_k, a, b, c, d, e, f) which are captured by the brilliant scientist's device. The abducted people in this operation will be returned before the next abduction operation is executed.

The constraints for all inputs are:

- $1 \le N, Q \le 50{,}000$
- $5 \le W, H < 2^{15}$
- $0 \le X_i, X_k \le W$ and $0 \le Y_i, Y_k \le H$
- $0 \le E_k$, a, b, c, d, e, f $< 2^{31}$
- It is guaranteed that the total number of abduction across all operations is no more than 50,000.

This problem has a huge judge input, therefore using C/C++ cin or Java Scanner will possibly lead to Time Limit Exceed. Instead use C/C++ scanf or Java BufferedReader.

Output

For each case, output "Case #X:" in a line, where X is case number starts from 1, followed by N lines which report the final N positions of each person (X_i, Y_i) after the alien ship leaves Earth for good. X_i and Y_i are separated by a single space.

Sample Input

```
2
6 3 11 8
5 4
8 2
4 2
3 1
8 2
11 6
1 1 4 3 7 3 1 7 3
4 5 9 3 3 2 7 1 3
3 1 8 5 8 8 7 3 5
5 8 12 7
0 2
0 3
5 5
12 2
3 1
6 0 9 3 3 7 3 8 7
1 1 7 4 6 9 1 3 8
4 3 7 4 7 1 3 4 8
1 2 5 4 7 9 3 1 4
5 0 2 4 8 2 4 2 9
1 0 7 1 9 3 1 4 2
5 0 9 5 8 9 7 3 1
3 1 3 2 9 3 8 1 3
```

Output for Sample Input

```
Case #1:
4 4
10 1
2 3
9 0
6 5
8 5
Case #2:
9 2
9 3
9 5
9 2
7 1
```

Explanation for 1st sample input.

Abduction #1

#3 (4,2) - (2,3)	#4 (3,1) - (6, 6)

Read:
person 3 abducted at (4, 2) returned to position (2, 3)
person 4 abducted at (3, 1) returned to position (6, 6)

Abduction #2

#1 (5,4) - (7,2)	#2 (8,2) - (1,0)	#3 (2,3) - (10,2)
#4 (6,6) - (0,4)	#5 (8,2) - (7,1)	#6 (11,6) - (8,5)

Abduction #3

#1 (7,2) - (4,4)	#2 (1,0) - (10,1)	#3 (10,2) - (2,3)
#4 (0,4) - (9,0)	#5 (7,1) - (6,5)	

Problem I
Tiling

Analyzing floor tiles is a newly trending hobby for Jakartan school children who are accustomed to decorated floor grids in various buildings, especially in Jakarta shopping malls. A curious child wonders: If she could reduce an infinitely-repeating patterns in a grid into the smallest possible unit, such that, when the unit is repeated without any overlap it could cover all the patterned cells without exception, what is the smallest possible unit size?

For simplicity, we assume that each cell in the grid can be either blank or dotted. For example, given a tile pattern as shown in Figure 1. She can deduce that it indeed can be reproduced with a minimal unit of size 5 (Figure 2), such that the unit, if infinitely tiled without overlapping, can cover the whole pattern (Figure 3).

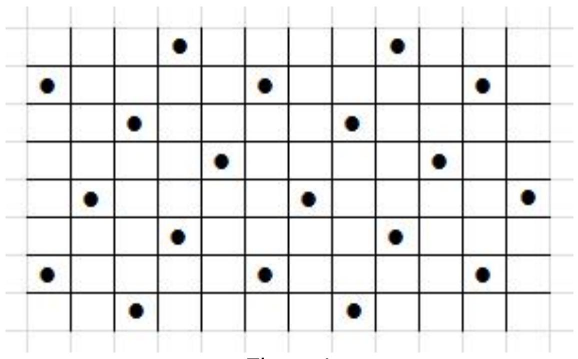


Figure 1

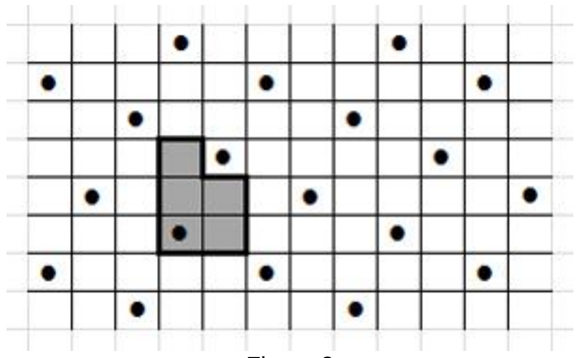


Figure 2

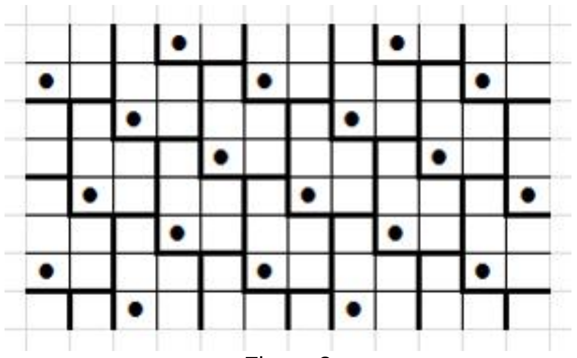


Figure 3

Now we're attempting to solve a more general problem. Suppose you are given the positions of every single dot in the infinite grid, described by:

(DX1, DY1), (DX2, DY2), (DX3, DY3)

which means: Only the cells at position (X, Y) that satisfies

X = i • DX1 + j • DX2 + k • DX3 and
Y = i • DY1 + j • DY2 + k • DY3
for some integers i, j, and k

are dotted and the rest is blank.

For example, given
(DX1, DY1) = (5, 1)
(DX2, DY2) = (-1, -5)
(DX3, DY3) = (2, 2)

The pattern will look like Figure 4, and a tile with minimal size of 8 can reproduce the pattern as shown in Figure 5.

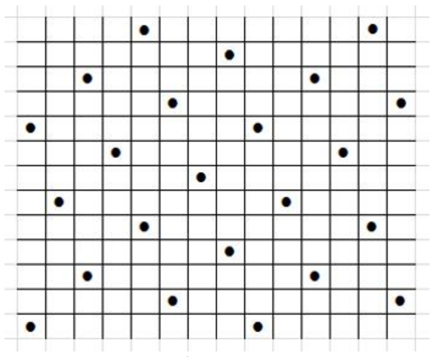


Figure 4

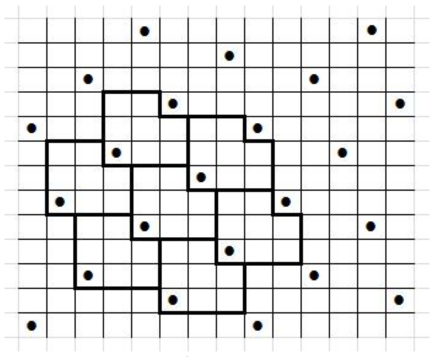


Figure 5

Given the positions of the dotted cells in an infinite grid described by the manner above, what is the smallest unit size, that, if repeated as is (no mirroring, no rotation) without overlaps, can cover the pattern in the grid without leaving some cells uncovered?

Input

The first line contains the number of cases, T (1 ≤ T ≤ 1000). Each case consists of a line containing six integers: DX1, DY1, DX2, DY2, DX3, DY3 (each integer is in the range -10000 to 10000, inclusive). It is guaranteed that the three vectors (DX1, DY1), (DX2, DY2), and (DX3, DY3) all point to different directions (no vector is simply a scalar multiple of another vector).

Output

For each case, output "`Case #X: Y`", where `X` is case number starts from 1 and `Y` is the smallest number of cells in a unit that if infinitely repeated without overlap can cover the pattern in the grid.

Sample Input	Output for Sample Input

4
1 2 2 -1 -5 0
2 4 7 4 -3 -1
5 1 -1 -5 2 2
1 2 2 1 2 2
	Case #1: 5
	Case #2: 5
	Case #3: 8
	Case #4: 1

Explanation for 1st and 2nd sample input.
These samples demonstrate two different ways to describe the first example in the problem statement.

Explanation for 3rd sample input.
This sample corresponds to the second example in the problem statement.

Explanation for 4th sample input.
This sample input corresponds to all cells dotted in the pattern, hence the minimum tile size is 1.

Problem J

Perfect Matching

A palindrome is a word that can be read the same way in either direction (front to back, or back to front). E.g., “RACECAR”, “DEED”, “LEVEL” are palindromes, while “ADAM”, “CAR”, “WATER” are not palindromes. Of course there are trivial palindromes which are words that consist of only one character such as “A”, “B”, “C”, etc.

In this problem, you have N strings and a function named perfect which takes two arguments string A and string B which will return whether concatenation of A and B (A·B) is a palindrome or not.

For example, let there are 4 strings {“race”, “car”, “ac”, ecar”}. We can take any permutation of 2 strings out of those 4 strings:

- perfect(“race”, “car”) yes, “racecar” is a palindrome.
- perfect(“race”, “ac”) no, “raceac” is not a palindrome.
- perfect(“race”, “ecar”) yes, “raceecar” is a palindrome.
- perfect(“car”, “race”) no, “carrace” is not a palindrome.
- perfect(“car”, “ac”) yes, “carac” is a palindrome.
- perfect(“car”, “ecar”) no, “carecar” is not a palindrome.
- perfect(“ac”, “race”) no, “acrace” is not a palindrome.
- perfect(“ac”, “car”) no, “accar” is not a palindrome.
- perfect(“ac”, “ecar”) no, “acecar” is not a palindrome.
- perfect(“ecar”, “race”) yes, “ecarrace” is a palindrome.
- perfect(“ecar”, “car”) no, “ecarcar” is not a palindrome.
- perfect(“ecar”, “ac”) no, “ecarac” is not a palindrome.

Out of those 12 possible 2-permutation of 4 strings, there are 4 which have “yes” as the result: (“race”,”car”), (“race”,”ecar”), (“car”,”ac”), (“ecar”,”race”); and we call those as perfect matching. Note that A·B is considered as different from B·A, you can find in the example above both (“race”,”ecar”) and (“ecar”,”race”) are counted as a different matching although they are composed of a same set of string.

Given N strings, your task is to count how many perfect matching there are in those strings.

Input

The first line of input contains an integer T (T ≤ 50) denoting the number of cases. Each case begins with an integer N (1 ≤ N ≤ 1,000) denoting the number of strings. The next N lines, each contains a string S_i. S_i contains only lowercase alphabets (a-z) and its length is between 1 and 500, inclusive. You may assume that all strings in each case are unique.

Output

For each case, output “`Case #X: Y`”, where `X` is case number starts from 1 and `Y` is the number of perfect matching in the corresponding case.

Sample Input	Output for Sample Input

`4`
`4`
`race`
`car`
`ac`
`ecar`
`3`
`xyz`
`abac`
`aba`
`5`
`abc`
`ba`
`xab`
`x`
`bax`
`5`
`abc`
`ab`
`cba`
`ba`
`c`
	`Case #1: 4`
	`Case #2: 1`
	`Case #3: 4`
	`Case #4: 6`

Explanation for the 1st sample input
This is the example from the problem statement.

Explanation for the 2nd sample input
(“abac”, “aba”) is the only perfect matching.

Explanation for the 3rd sample input
(“abc”, “ba”), (“ba”, “xab”), (“xab”, “bax”), (“bax”, “xab”) are perfect matching.

Explanation for the 4th sample input
(“abc”, “cba”), (“abc”, “ba”), (“ab”, “cba”), (“ab”, “ba”), (“cba”, “abc”), (“ba”, “ab”) are perfect matching.