Bay Area Programming Contest 2024

BAPC 2024

March 3, 2024

CITADEL | CITADEL Securities

Problems

Do not open before the contest has started.

Problem A. An X-Camp Transformer Game

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

X-Camp Academy is a coding academy with a full spectrum of levels of class offerings. In its summer camp, students play a game during breaks and winners get a transformer badge.

The students are first given non-negative integer sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ of length n.

In one operation, they can choose an integer $1 \le i \le n$, and do the following:

- For all $1 \le j < i$, replace a_j with $a_j | a_i$.
- For all $i < j \le n$, replace a_j with $a_j \& a_i$.

Here, | and & denote the bitwise `OR` and `AND` operations respectively.

The goal of the game is to perform the operation **exactly n times, choosing a distinct integer in each operation**, so that in the end, $a = b$.

Please help construct the solution to win the X-Camp transformer badge, or determine that there is no solution!

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^5$).

The first line of each test contains n ($1 \le n \le 3 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$).

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($0 \le b_i \le 10^9$).

It is guaranteed the sum of n over all tests does not exceed $3 \cdot 10^5$.

Output

For each test, if the goal is unachievable, output `-1`.

Otherwise, output n integers $p_1, p_2, \ldots, p_n$, meaning that at the i-th step, you perform the operation with index p_i. ($1 \le p_i \le n$).

All p_i must be distinct.

If there are multiple answers, you can output any.

Example

standard input	standard output
4 3 1 2 3 3 2 2 1 1000000000 1000000000 2 1 5 69 420 5 0 1 2 3 4 0 1 2 3 4	2 1 3 1 -1 -1

Note

In the first test,

- After the operation with $i = 2$, $a = [1|2, 2, 3\&2] = [3, 2, 2]$.
- After the operation with $i = 1$, $a = [3, 2\&3, 2\&3] = [3, 2, 2]$.
- After the operation with $i = 3$, $a = [3|2, 2|2, 2] = [3, 2, 2]$.

Note that other sequences of operations may be possible and would be accepted, such as $[2, 3, 1]$.

In the second test, note that the goal may already be achieved at the start. Regardless, we are still forced to perform an operation.

In the third and fourth tests, we can show there are no solutions.

Problem B. Big Data

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

> Big data, machine learning, blockchain, artificial intelligence. Digital manufacturing, big data analysis, quantum communication, and internet of things.
>
> —*Narendra Modi, 2019*

You are given a sequence $a_1, a_2, \ldots, a_n$ of length n, consisting of ones and zeros.

Define the *blockchain* of a binary sequence as the list of lengths of blocks of equal elements in a. For example, blockchain$([1, 0, 0, 1, 1, 1, 0]) = [1, 2, 3, 1]$, since the sequence consists of a single 1, followed by two 0s, followed by three 1s, followed by a single 0.

You are also given an integer sequence $b_1, b_2, \ldots, b_m$ of length m. Your goal is to make blockchain(a) into a permutation of b, using the following operation:

- Choose some $1 \le i \le n$, and flip the i-th element of a (so if $a_i = 0$ it becomes 1, and vice versa).

What's the minimum number of operations you need?

Input

The first line contains n and m ($1 \le m \le n \le 100$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 1$).

The third line contains m integers $b_1, b_2, \ldots, b_m$ ($1 \le b_i \le n$).

It is guaranteed that $b_1 + b_2 + \ldots + b_m = n$.

Output

Print the answer. We can show the goal is always achievable under the constraints of this problem.

Examples

standard input	standard output
7 4 1 0 0 1 0 1 0 1 3 1 2	1
4 1 0 1 0 1 4	2

Note

In the first test, initially, blockchain$(a) = [1, 2, 1, 1, 1, 1]$. But if we flip the value at index 5, we get $a = [1, 0, 0, 1, 1, 1, 0]$, and so blockchain$(a) = [1, 2, 3, 1]$. Since $[1, 2, 3, 1]$ is a permutation of $[1, 3, 1, 2]$, we have achieved our goal.

In the second test, initially blockchain$([0, 1, 0, 1]) = [1, 1, 1, 1]$. We have two options, both of which take two operations — we can either turn a into $[0, 0, 0, 0]$ or into $[1, 1, 1, 1]$. Note that these two sequences are equivalent to the blockchain function.

This page is intentionally left blank.

Problem C. Crazy Dance

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

There are n dancers on the number line. Each dancer is at an integer coordinate. It is possible for multiple dancers to be at the same location.

During a *dance*, every dancer moves either 1 unit to the left or to the right. They make this decision uniformly at random and independently of the other dancers.

Let's say a dance is *crazy*, if for all integers x, the number of dancers at position x stays the same. For example, if the dancers are at positions $[1, 2, 3, 2]$, and after the dance the positions become $[2, 1, 2, 3]$, then the dance is crazy. But if the positions instead become $[0, 3, 4, 3]$, then the dance is not crazy.

What is the maximum probability that the dance is crazy, if you place the dancers optimally?

Input

The only line contains n ($1 \le n \le 40\,000$).

Output

The output format is a bit unusual. In particular, let ans be the desired probability. Then you will need to output $\log_2(ans)$. If $ans = 0$, then you should output 0.

Your answer will then be considered correct if its absolute or relative error does not exceed 10^{-9}. Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-9}$.

Examples

standard input	standard output
4	-3.0000000000000000000
1	0

Note

In the first test, one optimal placement is $[1, 2, 3, 2]$. We can show this yields a probability $ans = \frac{1}{8}$.

In the second test, there is only one dancer. It is impossible for the dance to be crazy, so the answer is 0 and we output 0.

This page is intentionally left blank.

Problem D. Deviously Disorganized Documents

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Om and Mo are rivals in the college application process, both applying to the prestigious Red Panda University.

Recently, after getting access to Om's document of n essays (ordered 1 through n), Mo decided to sabotage Om by duplicating and shuffling his essays. Since Om blindly copy-pastes his essays into Common App, the RPU admissions officers are sure to reject him!

Mo is in the middle of this process, when he finds out that there has been a complication. Om's parents just donated a building to RPU! Now, Om only needs one correctly ordered essay to get in. In other words, Om will be admitted if and only if $a_i = i$ for some $1 \le i \le n$.

In one operation, Mo can swap any two essays a_i and a_j. How many operations does he need to perform to ensure Om does not get into RPU?

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^4$).

The first line of each test contains n, the number of essays in Om's document ($1 \le n \le 10^5$).

The second line of each test contains n integers $a_1, a_2, \ldots, a_n$, the order of Om's essays ($1 \le a_i \le n$).

It is guaranteed that the sum of n over all tests does not exceed 10^5.

Output

For each test, if Mo cannot prevent Om from getting admitted, output `-1`.

Otherwise, output the minimum number of operations he needs to perform to prevent Om from getting admitted.

Example

standard input	standard output
4 5 1 5 3 2 4 3 1 1 1 3 1 2 3 2 2 1	1 -1 2 0

Note

In the first test, $a_1 = 1$ and $a_3 = 3$ initially. We can swap a_1 and a_3 to get $a = [3, 5, 1, 2, 4]$, which satisfies $a_i \neq i$.

This takes 1 operation, and we need at least one operation, so the answer is 1.

In the second test, no matter what swaps we do, $a_1 = 1$.

In the third test, we can swap a_1 and a_2 to get $a = [2, 1, 3]$, then swap a_2 and a_3 to get $a = [2, 3, 1]$.

In the fourth test, note that the answer might already be achieved.

This page is intentionally left blank.

Problem E. Ezra and Experiments

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Recently, Ezra has been interested in Conway's Game of Life (you don't need to know it to solve this problem though!) — specifically, trying to generalize it to trees.

His favorite tree is undirected, with vertices numbered from 1 through n. It is rooted at vertex 1.

In the Game of Life, it is optimal for cells to have three living neighbors. Ezra has decided that for his experiment, the optimal number of neighbors is described by a constant l.

More specifically, he defines the *aliveness* of a vertex v recursively:

- Let S be the sum of aliveness across the direct children of v, plus one (in particular, if v is a leaf, then $S = 1$).
- Then, the aliveness of v will be $\max(0, l - |l - S|)$.

Ezra can easily calculate the aliveness of vertices in a given tree using his programming skills, but the issue comes when he tries to modify the tree. Help him answer the following question, for all $1 \le i \le n$ independently:

- Suppose you create a new vertex and attach it with an undirected edge to vertex i. What will the aliveness of vertex 1 in this new tree be?

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^5$).

The first line of each test contains n and l ($1 \le n \le 2 \cdot 10^5$, $1 \le l \le 10^9$).

The next $n - 1$ lines contain two integers u_i and v_i, the edges of the graph ($1 \le u_i, v_i \le n$).

It is guaranteed that in each test, the given graph is a tree, the sum of n over all tests does not exceed $2 \cdot 10^5$.

Output

For each test, output n integers, the answers for each $1 \le i \le n$.

Example

standard input	standard output
3 4 3 1 2 1 3 1 4 1 1000000000 8 2 2 1 3 1 4 2 5 2 6 2 7 3 8 4	1 1 1 1 2 0 1 2 1 1 1 2 1

Note

In the first test, if we attach a new vertex to the root of the tree, the aliveness of each leaf will be $\max(0, l - |l - 1|) = \max(0, 3 - 2) = 1$. There will be 4 leaves, so the aliveness of the root is $\max(0, l - |l - (4 + 1)|) = \max(0, 3 - 2) = 1$.

Now, suppose we attach a new vertex to vertex 2. The aliveness of that leaf will be 1, so the aliveness of vertex 2 is $\max(0, l - |l - 2|) = 2$. Then, the sum of aliveness for the direct children of the root, will be $2 + 1 + 1 = 4$. This is the same as in the case where we attach a vertex directly to the root, so the answer is again 1.

In the second test, the aliveness of the root would become $\max(0, l - |l - 2|)$. Substituting in values, we get $\max(0, 1\,000\,000\,000 - 999\,999\,998) = 2$.

Problem F. Funky Finding

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Everyone knows the standard ordering of the positive integers:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots$$

Using the usual ordering, it's easy to figure out how far apart any given integers x and y are. We can just do subtractions like $6 - 4 = 2$ (6 appears two positions ahead of 4) or $8 - 12 = -4$ (8 appears four positions behind 12).

In this problem, we consider a different ordering, the *Sharkovskii ordering*:

$$3, 5, 7, 9, \ldots, 6, 10, 14, 18, \ldots, 12, 20, 28, 36, \ldots, 24, 40, 56, 72, \ldots, \ldots, 32, 16, 8, 4, 2, 1$$

First, we list out the odd numbers greater than 1 in increasing order. Then we list out 2 times these same odds in increasing order. Then we list out 4 times these odds in increasing order. And so on for all the powers of 2. Finally, at the very end, we list all the powers of 2 in decreasing order. Notice that every positive integer appears exactly once in this ordering.

It just got a bit harder to figure out how far apart two numbers are. 2 now appears three positions in front of 16, 20 appears one position behind 28, and for some pairs of integers, the answer might be infinite — for example, 6 appears infinitely many positions in front of 3!

But surely a little bit of infinity never scared you. You're given integers x and y, and you need to find how far ahead of x y appears in the Sharkovskii ordering.

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^4$).

The only line of each test contains two integers x and y ($1 \le x, y \le 10^9$).

Output

For each test, output the difference between the positions of y and x in the Sharkovskii ordering. If x appears before y, this number should be positive, and it should be negative otherwise. Note that if $x = y$, you should output `0`.

If there are an infinite number of integers between x and y, output either `inf` or `-inf`. See the sample input for more details.

Example

standard input	standard output
5 7 7 3 5 1 3 4 128 93 92	0 1 -inf -5 inf

Note

In the first test, we output `0` since $7 = 7$.

In the second test, we can see that 3 appears one spot before 5 in the above ordering.

In the third test, note that 1 comes at the very end of the order, and 3 appears at the very beginning. So 1 appears after 3, and there are infinitely many elements between them.

In the fourth test, note that the powers of 2 are ordered backwards by magnitude, so 4 appears after 128 (specifically, 5 elements after 128).

In the fifth test, we can show that 93 appears before 92, and there are infinitely many elements between them.

This following is for those curious about the context behind the Sharkovskii ordering. **You don't need to read it to solve the problem.**

- Let $f : \mathbb{I} \to \mathbb{I}$ be a continuous function over some interval of real numbers.
- A *cycle* of length n over f, is a sequence $b_1, b_2, \ldots, b_n$ of distinct real numbers such that $b_{i+1} = f(b_i)$ for all $1 \leq i < n$, and $b_1 = f(b_n)$.
- A theorem by Oleksandr Sharkovskii (1964) proves that if f has a cycle of length n, it must also have a cycle of length m for **any** m which appears after n in the Sharkovskii ordering. So, for example, if f has a cycle of length 3, it must also have cycles of any positive integer length!

Problem G. GCD Spanning Tree

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Consider a weighted undirected graph on n vertices, numbered 1 through n. For all $1 \le i < j \le n$, there is an edge between vertices i and j with a weight $\gcd(i, j)$.

You are given an integer k. Construct a spanning tree on the graph with total weight exactly equal to k, or report that it doesn't exist.

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 5 \cdot 10^5$).

The only line of each test contains n and k ($2 \le n \le 10^6$, $1 \le k \le 10^{12}$).

It is guaranteed the sum of n across all tests does not exceed 10^6.

Output

For each test, if the answer does not exist, output `-1`.

Otherwise output $n - 1$ lines, containing two integers each, describing the edges in your spanning tree.

If there are multiple answers, you can output any.

Example

standard input	standard output
5 5 5 2 1 2 2 10 1000000000000 6 7	1 2 2 4 2 3 4 5 1 2 -1 -1 1 2 2 4 2 6 1 5 5 3

Note

In the first test, all edges in our outputted spanning tree have weight 1 (since $\gcd(1, 2) = \gcd(2, 3) = \gcd(4, 5) = 1$), except for the edge $(2, 4)$, with a weight $\gcd(2, 4) = 2$. So the total weight of our spanning tree is $1 + 2 + 1 + 1 = 5$, as desired.

In the second test, there is only a single possible spanning tree, with a weight $\gcd(1, 2) = 1$. $k = 1$, so we output it.

In the third and fourth tests, we can show that there are no possible solutions.

This page is intentionally left blank.

Problem H. Haphazard Reconstruction

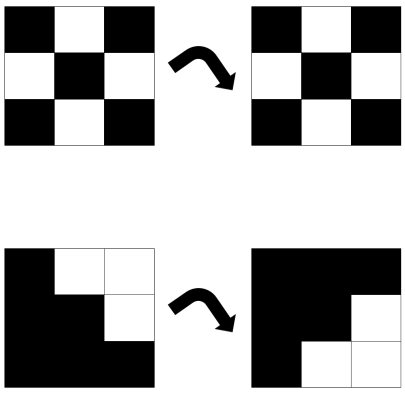
Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Ench Lolz was playing chess, when he accidentally dropped his board on the ground, and everything shattered! All he remembers is that the chessboard was a square, with some number of white and black cells. He goes to the store and buys an n-by-n grid and k black tiles. Unfortunately for him, he forgot what a chessboard looks like and is now trying to construct it with the following instructions.

You are given a n-by-n square, divided into 1-by-1 cells. Initially every cell is white. You want to color exactly k cells black in a way that satisfies the following condition:

- Suppose you rotate the square by 90 degrees clockwise. Then the coloring of the cells stays the same.

Determine if such a coloring is possible. For example, the first square shown here is valid, while the second one is not:



Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 1\,000$).

The only line of each test contains n and k ($1 \le n, k \le 30\,000$).

Output

For each test, print `YES` if a coloring is possible, and `NO` otherwise.

Example

standard input	standard output
4 3 5 3 6 1 30000 30000 30000	YES NO NO YES

Note

The first test corresponds to the square shown in the problem statement.

The second test corresponds to the invalid square shown. We can show that no valid placements are possible.

This page is intentionally left blank.

Problem I. Interesting Constructive

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

You are given a n-by-m grid. Initially every cell is white, and your goal is to make all cells black.

On your first move, you can pick any cell and color it black. After that, you can only perform the following operation:

- Choose some white cell that has exactly 1 or 3 black neighbors, and color it black. Here, a cell's neighbors are those cells that share a side with it. See the examples for clarification.

You need to construct a sequence of operations that colors in the grid, or report it is impossible.

Input

Input consists of multiple tests. The first line contains t, the number of tests ($1 \le t \le 10^3$).

The only line of each test contains n and m ($1 \le n, m \le 50$).

Output

If the goal is impossible, output `-1`.

Otherwise, output $n \cdot m$ lines, each containing two integers r and c ($1 \le r \le n$, $1 \le c \le m$).

This represents a sequence of operations where on the i-th step ($1 \le i \le nm$), you color in cell (r_i, c_i). The sequence you output must be valid.

If there are multiple answers, you can output any.

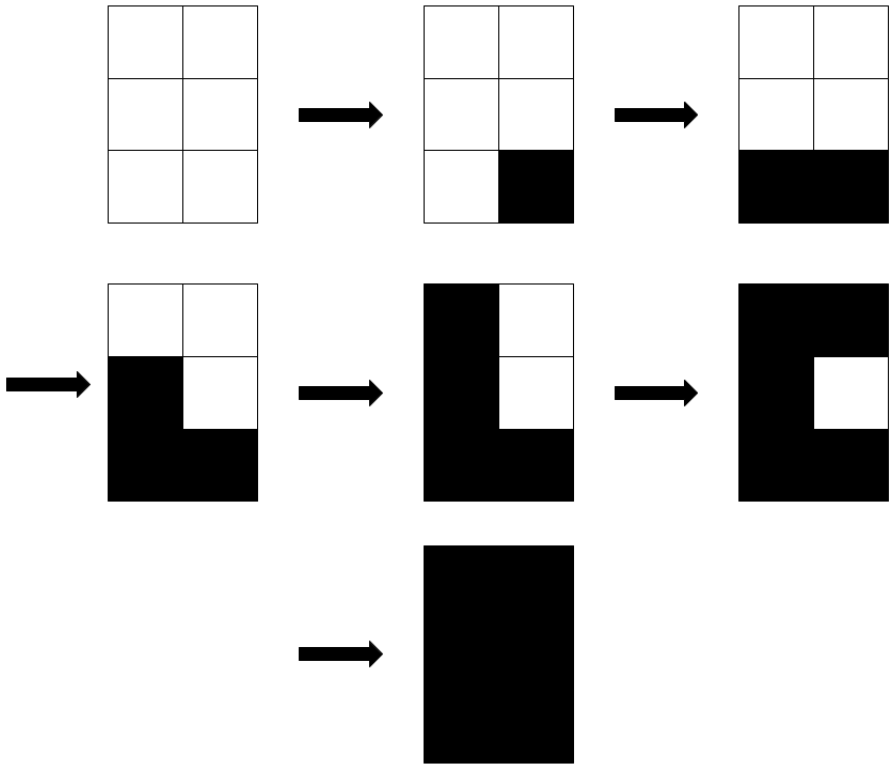
We guarantee that the total size of the correct output across all tests does not exceed $5 \cdot 10^5$ lines.

Example

standard input	standard output
2 3 2 12 34	3 2 3 1 2 1 1 1 1 2 2 2 -1

Note

In the first test, here is a picture of the operations:



In all steps except for the first and last, the cell being colored in has 1 neighbor. In the last step, it has 3 neighbors.

In the second test, we can show there is no answer.

Problem J. Jovial Jaunt

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Jack the jolly jester lives on an undirected tree with n vertices, conveniently numbered 1 through n. At the i-th ($1 \le i \le n$) vertex, there is a jazz jellyfish with a *jubilance value* of a_i.

Jack plans to take a jovial jaunt throughout the joyful junctions. He will choose a starting vertex s and an ending vertex e, then walk along the unique path from s to e. It is allowed for $s = e$.

Let the vertices he visits be $v_1, v_2, \ldots, v_\ell$ in order (so $v_1 = s$ and $v_\ell = e$). Because Jack is a perfectly normal human, the total *jubilance* he gets from walking through this path is defined in the most natural possible way. It will be $f(a_{v_1}, a_{v_2}, \ldots, a_{v_\ell})$, where f is defined as follows:

- $f(x_1) = x_1$.
- $f(x_1, x_2) = \max(x_1, x_2) + \left\lfloor \sqrt{\min(x_1, x_2)} \right\rfloor$.
- For $k > 2$, $f(x_1, x_2, \ldots, x_k) = f(f(x_1, \ldots, x_{k-1}), x_k)$.

Note that the path from s to e may have a different weight from the path from e to s — we consider these paths to be distinct.

Find the maximum possible jubilance over all paths in the tree.

Input

The first line contains a single integer n, the number of vertices in the tree ($2 \le n \le 3 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$, the labels of the vertices ($1 \le a_i \le 10^9$).

The next $n - 1$ lines contain two integers u_i and v_i, the edges of the graph ($1 \le u_i, v_i \le n$).

Output

Print a single integer — the maximum jubilance of a path in the tree.

Examples

standard input	standard output
3 3 2 3 1 2 1 3	5
5 2 3 3 2 3 3 2 5 2 3 4 4 1	7

Note

In the first test case, the path from vertex 2 to vertex 3 has jubilance 5. We can show that no path with greater weight exists.

In the second test case, the path from vertex 5 to vertex 1 has jubilance 7. We can show that no path with greater weight exists.

This page is intentionally left blank.

Problem K. Kickball

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

There are n schoolchildren playing kickball on an infinite grid. Of course, everyone wants to be the kicker, so they plan to line up. However, they have not figured out how to make a line.

Instead of lining up in an orderly fashion, they wander around an infinite 2-d grid, doing the following every minute:

- First, each person currently on the grid counts the total number of people on the line perpendicular to the line formed by the direction they are facing in and their current position (including other people at the same position as them). If they count an odd number of people, they will turn 90° to their right (in the clockwise direction).
- Next, every person on the grid will move 1 unit in the direction they are facing. Note that multiple people can be on the same point on the grid at the same time.

The i-th person enters the grid at the beginning of minute t_i, facing north, at position (x_i, y_i). Since recess ends after m minutes, you need to determine the location of each person at the beginning of minute m.

Input

The first line contains two integers n and m — the number of people and the duration of recess ($1 \le n, m \le 1000$).

The next n lines contain three integers x_i, y_i, and t_i each, the location and time at which each person joins ($1 \le x_i, y_i \le 10^8, 0 \le t_i < m$).

It is guaranteed that $t_i \le t_{i+1}$ for all $1 \le i < n$.

Output

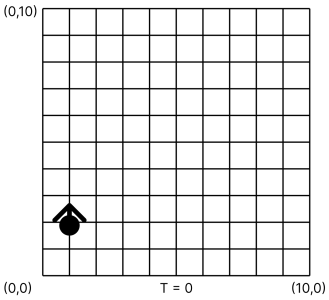
Print n lines consisting of two integers x_i and y_i — the location of the i-th person at the beginning of the m-th minute.

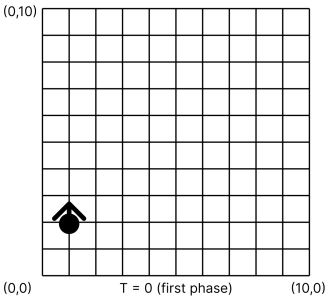
Examples

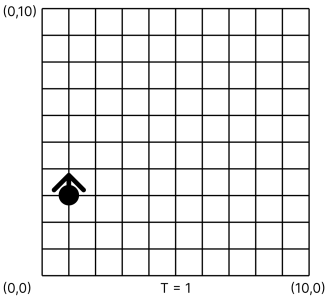
standard input	standard output
3 5 1 2 0 4 4 2 5 6 4	4 4 7 4 5 7
5 5 1 1 1 5 1 2 1 2 2 3 5 4 4 5 4	1 1 5 4 1 1 4 5 5 5

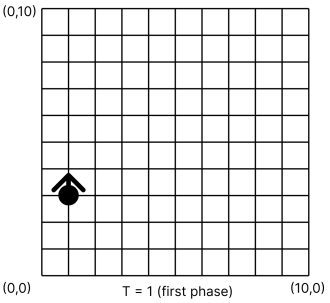
Note

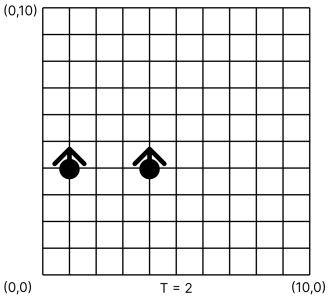
The first sample case is shown in the following images:

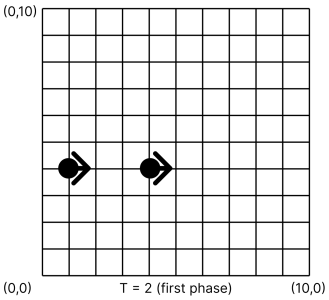

(0,10)
(0,0)
T = 0
(10,0)

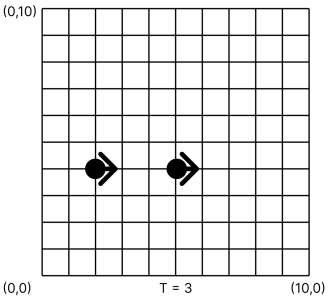

(0,10)
(0,0)
T = 0 (first phase)
(10,0)

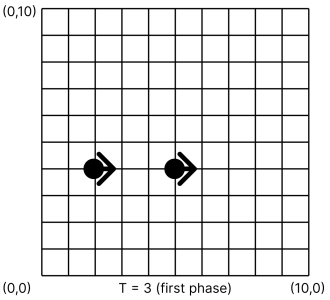

(0,10)
(0,0)
T = 1
(10,0)

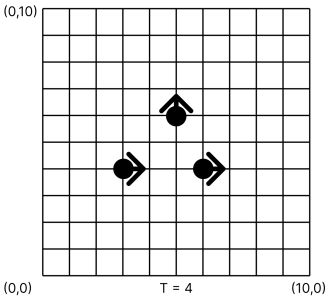

(0,10)
(0,0)
T = 1 (first phase)
(10,0)

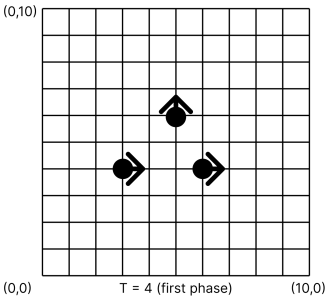

(0,10)
(0,0)
T = 2
(10,0)

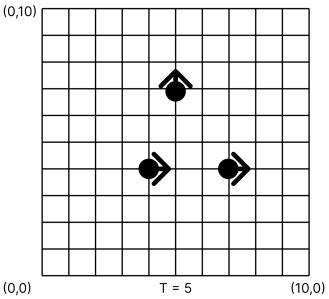

(0,10)
(0,0)
T = 2 (first phase)
(10,0)


(0,10)
(0,0)
T = 3
(10,0)


(0,10)
(0,0)
T = 3 (first phase)
(10,0)


(0,10)
(0,0)
T = 4
(10,0)


(0,10)
(0,0)
T = 4 (first phase)
(10,0)


(0,10)
(0,0)
T = 5
(10,0)

Problem L. Legendary Gyrating Mill

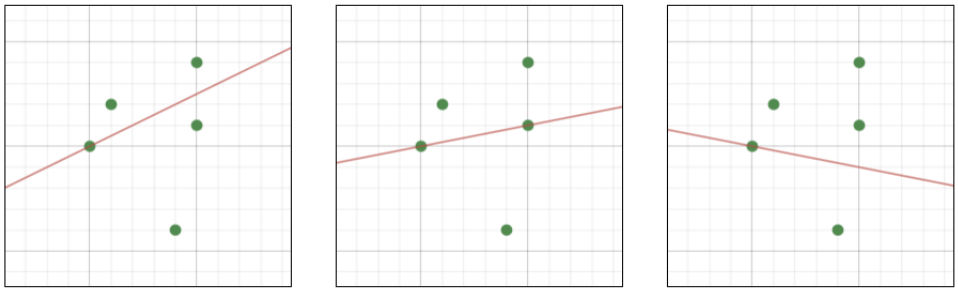
Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

This problem shares a setup with Windmill from the 2011 *IMO, popularized in* 2019 *by 3b1b on YouTube.*

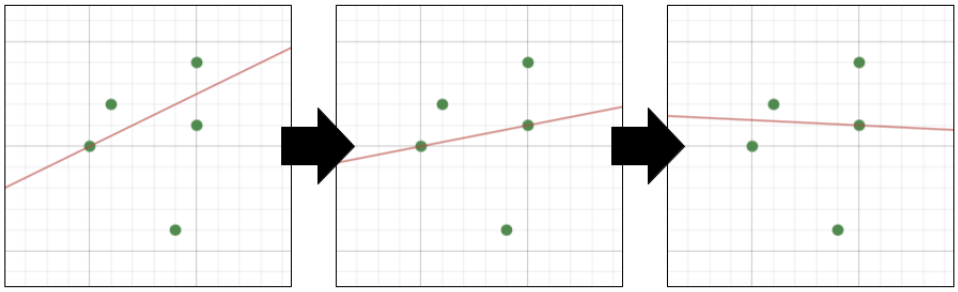
You are given n distinct points $P_1, P_2, \ldots, P_n$ on the 2-d plane. No three points are collinear.

Consider the following process. First you choose a *valid starting position*, described by a pair (P_i, ℓ), where P_i is a point and ℓ is an infinite line that intersects P_i, and does not intersect any points other than P_i.

For example, the leftmost and rightmost pictures below are valid starting positions, but the middle one is not (ℓ passes through two points).



ℓ will then rotate clockwise using P_i as a pivot until it meets another point. The new point, denoted P_j, will then become the new pivot. This process continues indefinitely.



An example of a pivot switch.

Now, consider an undirected graph G on n vertices. Initially G has no edges. Whenever the pivot switches (say, from point P_i to P_j), then we will add an edge in G between vertices (i, j), *as long as doing so would not create a cycle*. This means that G will end up as a spanning forest. We denote the spanning forest generated by a starting position (P_i, ℓ) as $S(P_i, \ell)$.

Of course, you are a programmer, so just calculating $S(P_i, \ell)$ for a fixed valid starting position would be too easy. So you need to calculate $S(P_i, \ell)$ across *all possible valid starting positions*.

There may be infinitely many starting positions, so we will say that two valid starting positions (P_i, ℓ) and (P_i', ℓ') are different if at least one of the following holds:

- $P_i \neq P'_i$.
- Let P_k be the first new pivot we encounter from the starting position (P_i, ℓ), and define P'_k similarly. Then, $P_k \neq P'_k$.

Still, there might be many starting positions, and outputting edges for each of them might be too slow. So, let the edges in some spanning forest $S(P_i, \ell)$ be $(a_1, b_1), (a_2, b_2), \ldots, (a_\gamma, b_\gamma)$. Then the *hash* of $S(P_i, \ell)$ will be the following:

$$(a_1 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus \ldots \oplus (a_\gamma \cdot b_\gamma)$$

Here $\oplus$ denotes the bitwise `XOR` operation.

You need to output the sum of the hashes of all $S(P_i, \ell)$.

Input

The first line contains an integer n ($3 \leq n \leq 2\,000$).

The following n lines contain two integers x_i, y_i each ($1 \leq x_i, y_i \leq 10^9$). It is guaranteed that no three points are collinear, and that all points are distinct.

Output

Output the answer, in the format described above.

Examples

standard input	standard output
3 1 1 3 1 2 2	20
6 2 1 1 2 2 2 3 3 3 4 4 3	492

Note

The first test is a triangle, with 6 distinct starting positions. Note that there are 3 possible spanning trees on a 3-vertex graph.

- The graph with edges $(1, 2)$ and $(2, 3)$ has a hash of $(1 \cdot 2) \oplus (2 \cdot 3) = 4$.
- The graph with edges $(1, 3)$ and $(2, 3)$ has a hash of $(1 \cdot 3) \oplus (2 \cdot 3) = 5$.
- The graph with edges $(1, 2)$ and $(1, 3)$ has a hash of $(1 \cdot 2) \oplus (1 \cdot 3) = 1$.

In fact, we can show that in this test, all of the possible spanning trees are each produced by exactly 2 starting positions. So taking the sum, we get $4 + 5 + 1 + 4 + 5 + 1 = 20$.

In the second test, note that the spanning forest is not necessarily a spanning tree. For example, if we choose the starting position with P_2 and ℓ being a vertical line, the generated spanning forest $S(P_2, \ell)$ will only have edges $(1, 2)$, $(2, 5)$, and $(5, 6)$.

Problem M. Methodical Mixing

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Agastya and Bathan have been messaging each other very frequently, discussing problem ideas for BAPC and other secret things. Unfortunately, it has come to their attention that Ezra is spying on their communications! As part of an effort to more securely encrypt their messages, Agastya needs to generate a random permutation.

Agastya starts with an integer sequence $a_1, a_2, \ldots, a_n$ of length n. Initially, $a_i = i$ for all $1 \leq i \leq n$.

He will perform m *actions* on a, described by a length-m sequence $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ of pairs of integers ($1 \leq x_i < y_i \leq n$).

In the p-th action ($1 \leq p \leq m$):

- Agastya swaps the elements a_{x_p} and a_{y_p}.
- Then he right-shifts a by one. More formally, he replaces a_j with a_{j-1} for each $2 \leq j \leq n$, and replaces a_1 with a_n. For example, right-shifting $[1, 2, 3, 4, 5]$ would give you $[5, 1, 2, 3, 4]$.

Bathan is impressed, of course, but a bit concerned about the security of the permutation. Specifically, he thinks there are q vulnerabilities. The i-th vulnerability ($1 \leq i \leq q$) is described by an integer v_i and a sequence $b_1, b_2, \ldots, b_n$ of length n.

If it is possible to do the following operation **at most once** so that after the v_i-th step, $a = b$, then Ezra will be able to exploit this vulnerability:

- Choose some $1 \leq j \leq m$. Replace (x_j, y_j) with any (x'_j, y'_j), as long as $1 \leq x'_j < y'_j \leq n$.

Help the BAPC organizers determine which vulnerabilities Ezra is able to exploit!

Input

Input consists of multiple tests. The first line contains t ($1 \leq t \leq 10^4$).

The first line of each test contains n, m, and q ($2 \leq n, m \leq 10^5$, $1 \leq q \leq 10$).

The next m lines contain 2 integers x_i and y_i, describing the operations ($1 \leq x_i < y_i \leq n$).

The next q lines contain $n + 1$ integers each: v_i and an integer sequence $b_1, b_2, \ldots, b_n$ ($1 \leq v_i \leq m$, $1 \leq b_i \leq n$).

It is guaranteed that in each vulnerability, $b_1, b_2, \ldots, b_n$ is a permutation of integers 1 through n.

It is also guaranteed that the sum of n over all tests does not exceed 10^5, and the sum of m over all tests does not exceed 10^5. **Note that there is no bound on the sum of q across test cases.**

Output

For each query, output `YES` if Ezra can exploit the vulnerability, and `NO` otherwise.

Example

standard input	standard output
2 5 3 5 3 4 1 2 1 5 3 3 4 1 5 2 2 1 2 3 4 5 2 3 2 1 5 4 2 3 5 1 2 4 3 4 3 2 5 1 5 10 10 2 4 1 2 1 5 2 3 1 5 1 5 3 4 2 5 2 4 2 4 8 4 1 5 3 2 3 3 2 1 5 4 9 1 2 3 5 4 6 4 3 2 1 5 10 1 2 5 3 4 5 2 5 4 3 1 10 5 1 2 3 4 4 4 2 3 1 5 5 5 4 2 1 3 2 3 5 1 4 2	YES NO YES NO YES YES NO YES YES YES YES YES NO NO YES

Note

In the first test:

- After the first action, $a = [5, 1, 2, 4, 3]$.
- After the second action, $a = [3, 1, 5, 2, 4]$.
- After the third action, $a = [3, 4, 1, 5, 2]$.

In the first query, since b_1 is already equal to a after the third action, we output `YES`.

In the second query, we can show that it is impossible to change at most one (x_j, y_j) to make a equal to b_2 after the second action.

In the third query, we can, for example, change (x_2, y_2) from $(1, 2)$ into $(1, 3)$. With this change, the second action will first swap a_1 and a_3 to get $a = [2, 1, 5, 4, 3]$, then right-shift to get $a = [3, 2, 1, 5, 4] = b_3$. So we output `YES`.

In the fourth query, note that if we changed (x_2, y_2) from $(1, 2)$ into something like $(1, 1)$, then we would get $a = [3, 5, 1, 2, 4] = b_4$ after the second action. But we must maintain $x_j < y_j$, so this is not allowed.

In the fifth query, we can change (x_1, y_1) from $(3, 4)$ into $(1, 2)$.

Reference Sheet

If you are unfamiliar with some of the terminology in this contest, you can check this sheet.

Bitwise Operations

The *binary representation* of a non-negative integer *n* is the result of writing it out in base 2. For example:

- The binary representation of 13 is 1101, since $13 = 2^3 + 2^2 + 2^0$.
- The binary representation of 67 is 1000011, since $67 = 2^6 + 2^1 + 2^0$.
- The binary representation of 31 is 11111, since $31 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0$.
- The binary representation of 0 is 0.

Since any non-negative integer can be written uniquely as a sum of powers of 2, this binary representation is unique.

For a non-negative integer *i*, we say that the i-th bit in the binary representation of *n* is *set* if and only if there is a 1 at the corresponding position. For example:

- The 0-th, 2-nd, and 3-rd bits of 13 are set, but the 1-st and 4-th are not.
- The 0-th, 1-st, and 6-th bits of 67 are set, but the 2-nd, 3-rd, 4-th, 5-th, and 7-th are not.
- The 1000-th bit of 67 is not set.

The *bitwise and* of two positive integers *n* and *m*, denoted `n&m`, is computed as follows. The i-th bit in `n&m` will be set if and only if *n* and *m* both have the i-th bit set. For example:

- The *bitwise and* of 13 and 31 is 1101, or 13 in decimal.
- The *bitwise and* of 67 and 31 is 11, or 3 in decimal.
- The *bitwise and* of 0 and 31 is 0.

Similarly, the *bitwise or* of *n* and *m*, written as `n|m`, has its i-th bit set if either *n* or *m* has its i-th bit set. For example:

- The *bitwise or* of 13 and 31 is 11111, or 31 in decimal.
- The *bitwise or* of 67 and 31 is 1011111, or 95 in decimal.
- The *bitwise or* of 0 and 31 is 11111, or 31 in decimal.

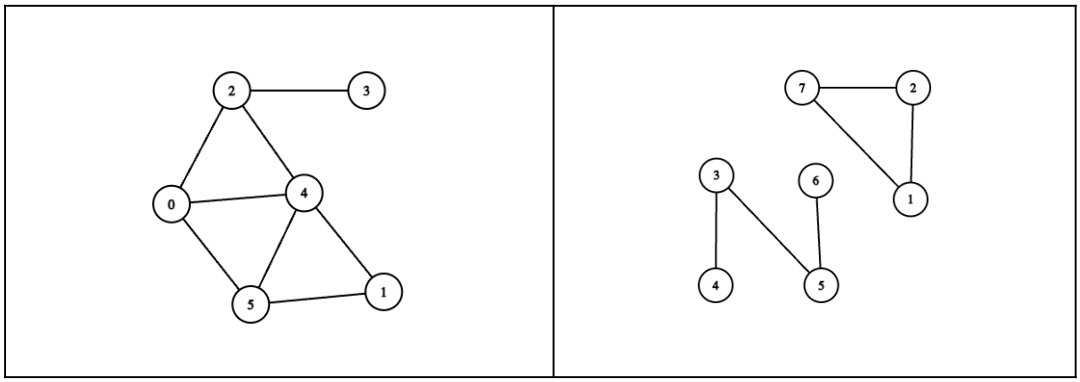
Finally, the *bitwise XOR* of *n* and *m*, written `n^m`, has its i-th bit set if exactly one of *n* and *m* has its i-th bit set. For example:

- The *bitwise XOR* of 13 and 31 is 10010, or 18 in decimal.
- The *bitwise XOR* of 67 and 31 is 1011100, or 92 in decimal.
- The *bitwise XOR* of 0 and 31 is 11111, or 31 in decimal.

In most programming languages, you can compute these operations using the same notation: `n&m`, `n|m`, and `n^m` for AND, OR, XOR respectively.

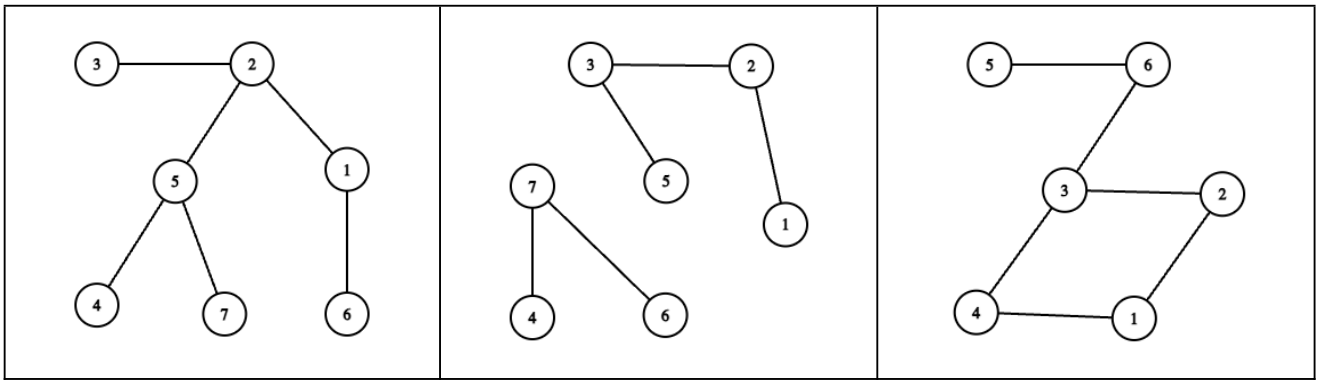
Graphs

A *graph* is a set of nodes, along with a set of edges. Informally, you can think of nodes as being “cities” and edges as being “roads” between the cities.



We say a graph is *connected* if it is possible to reach any node from any other node by traversing edges. For example, the graph to the left above is *connected*, but the graph on the right is not, because, for example, you cannot reach node 7 from node 6.

A *tree* is a connected graph, such that there is exactly one simple path between any two vertices. A *simple path* is a path that visits each vertex at most once. For example, the graph on the left below is a *tree*, but the one in the middle is not (it is not connected) and neither is the one on the right (it has multiple paths between vertices 1 and 3).



A *spanning tree* of a connected graph is a subset of its edges that form a tree. For example, the graph in the middle below is a spanning tree of the graph on the left, but the one on the right is not (it contains an edge that was not in the original graph).