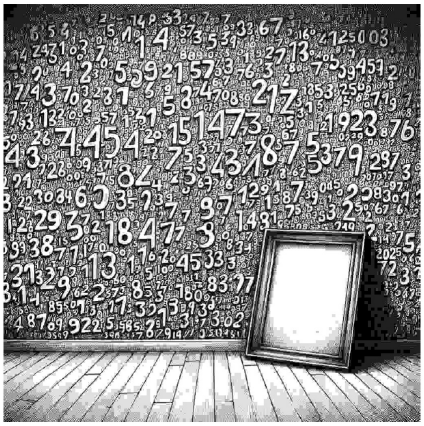
Problem A. AN2DL

Source file name:	AN2DL.c, AN2DL.cpp, AN2DL.java, AN2DL.py
Input:	`Standard`
Output:	`Standard`



While wandering around Building 21, you came across a wall completely covered with numbers, arranged in a table of n rows and m columns. Soon you noticed that there was a frame leaning against the wall large enough to frame r rows and s columns of the table on the wall. And next to the frame you found a pencil and a piece of paper containing an empty table.

You are sad that the table on the piece of paper is empty, so you decided to play around with the frame to fill it.

You leaned the frame against the wall so that the number in the i-th row and j-th column is in the upper left corner, and the borders of the frame are parallel to the edges of the wall. Considering the numbers inside the frame, and since you like large numbers, you have decided to write the largest among them in the i-th row and j-th column of the table on the piece of paper.

You repeated the process for every possible position of the frame on the wall (such that the frame is entirely on the wall, and that there are exactly $r \times s$ numbers inside it), making sure that the edges of the frame are parallel to the edges of the wall.

When you were done, the table on the piece of paper was even more beautiful than the one on the wall. What numbers are in the table on the piece of paper?

Input

The first line contains two integers n and m ($1 \leq n, m \leq 4000$), the number of rows and columns of the table on the wall.

Each of the following n lines contain m integers $a_{i,j}$ ($|a_{i,j}| \leq 10000$), where a_i,j is the number in the i-th row and j-th column of the table on the wall.

The last line contains two integers r and s ($1 \leq r \leq n,\ 1 \leq s \leq m$), the size of the frame.

Output

Output the numbers written in the table on the piece of paper.

Example

Input	Output
3 3 1 1 2 2 3 4 4 3 2 3 3	4
3 3 1 1 2 2 3 4 4 3 2 2 1	2 3 4 4 3 4
5 5 -1 -3 -4 -2 4 -8 -7 -9 -10 11 5 2 -8 -2 1 13 -3 -2 -6 -9 11 6 2 7 4 2 3	-1 -2 11 5 2 11 13 2 1 13 7 7

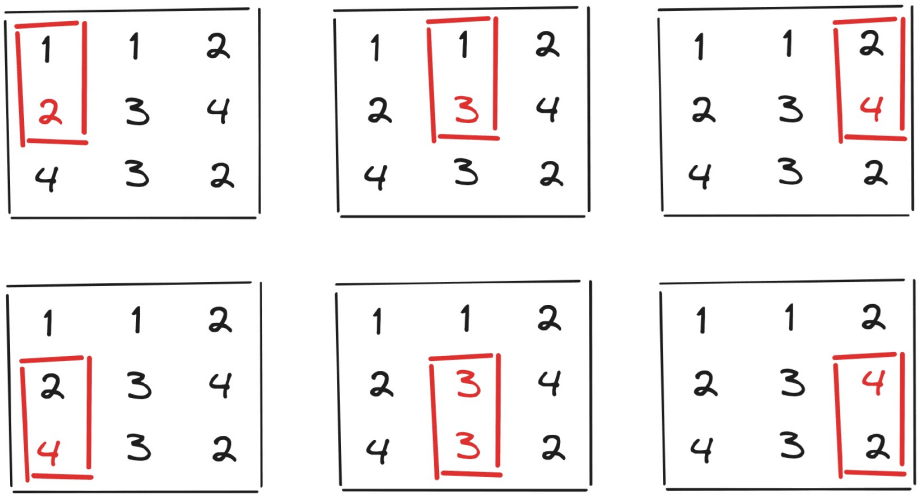
Explanation

Clarification of the first example:

The frame is big enough to fit the entire table on the wall. The largest number inside the frame is 4, so that is the only number written on the table on the piece of paper.

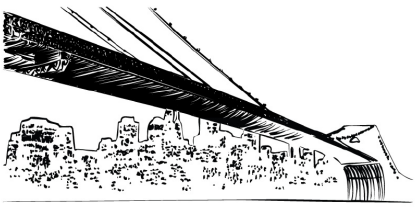
Clarification of the second example:

All possible frame positions are shown in the picture below. The largest number for each of the positions is written in red.



Problem B. Bridge

Source file name:	Bridge.c, Bridge.cpp, Bridge.java, Bridge.py
Input:	`Standard`
Output:	`Standard`



When Leonhard Euler resolved the famous Königsberg bridge problem, he had no clue he had discovered a whole new area of mathematics - graph theory!

Unfortunately, the Königsberg bridge problem is far too easy for the programmers of this era, so Euler came up with another problem - the Zagreb bridge problem!

The bridges of Zagreb form a graph with n nodes and m edges where the edges represent the bridges and the nodes represent the riverine islands. The graph is connected, in other words, it's possible to get from any node to any other by traveling across the edges. Now Euler asked, how many edges are there such that after their removal the graph becomes disconnected?

Again, Euler didn't know that this problem is also famous today (those damn Codeforces blogs). So the author of this problem decided to give you an even harder one, how many edges are there such that after the removal of the nodes which it connects, the remaining $n-2$ nodes become disconnected?

Input

The first line contains integers n and m ($4 \leq n \leq 100000,\ n-1 \leq m \leq 300000$) - the number of nodes and edges respectively.

Each of the next m lines contains integers a_i and b_i ($1 \leq a_i, b_i \leq n$) - this means nodes a_i and b_i are connected with an edge.

There are no loops or multiple edges.

Output

In a single line output the number of edges with the given property.

Example

Input	Output
4 5 1 2 2 3 3 4 4 1 1 3	1
6 7 1 2 2 4 2 6 3 5 6 1 4 3 2 5	4

Explanation

Clarification of the first example:

By removing edge (1, 3) and corresponding nodes 1 and 1, the remaining graph has two connected components, node 2 and node 4. In other words, it is not connected. It is easy to check it is the only edge with this property.

Clarification of the second example:

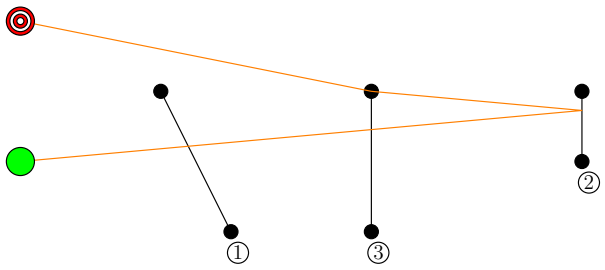
The edges with the given property are (1, 2), (2, 4), (2, 6) and (2, 5).

Problem C. Cross Country

Source file name:	Cross.c, Cross.cpp, Cross.java, Cross.py
Input:	`Standard`
Output:	`Standard`

Cross-country running is a sport in which contestants run a race on an open-air course over natural terrain. To record contestants' progress, the organisers set up RFID checkpoints that each span a line across part of the course.

A contestant has finished the race once they go through all of the checkpoints in order from 1 to n. Crossing a checkpoint out of order conveys no advantage or penalty to a runner, as they simply have to cross it again later at the right time. Thus, for example, a runner may choose to cross a checkpoint once and then immediately cross it again in another direction if it leads to a quicker finish.



Optimal running route for the course given in example input 3.

Your objective is to find the shortest distance one has to run to finish the race, so that we can use this as the official distance of the course.

Input

- One line containing the number of checkpoints, n ($1 \leq n \leq 16$).
- One line containing the start coordinate of the race, x_s and y_s ($-10^6 \leq x, y \leq 10^6$).
- n further lines, the ith of which contains the two integer coordinate of the ith checkpoint's endpoints, $x_{ai} y_a x_b y_b$ ($-10^6 \leq x, y \leq 10^6$).
- One line containing the end coordinate of the race, x_t and y_t ($-10^6 \leq x, y \leq 10^6$).

All of the checkpoints have non-zero length; however, they may overlap either with each other or with the start and finish points.

Output

Output the shortest distance you can run to go visit all of the checkpoints in the right order, regardless of whether you touch some of the checkpoints multiple times or in the wrong order along the way.

The output must be accurate to an absolute or relative error of at most 10^{-6}.

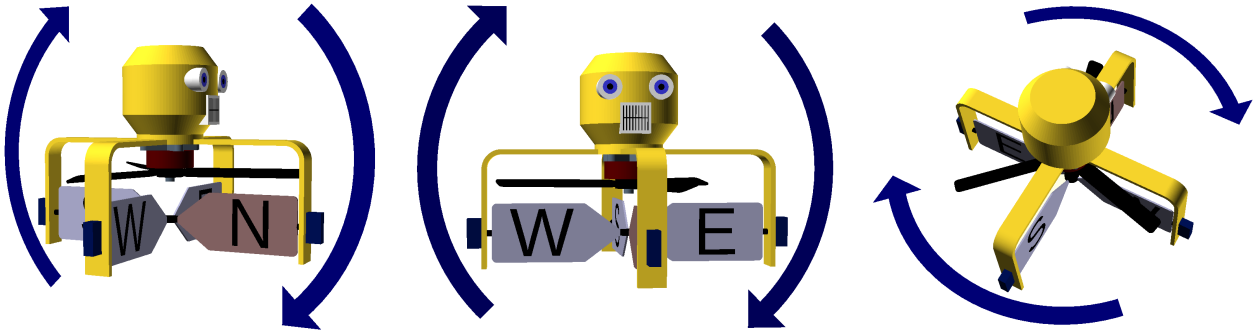
Example

Input	Output
2 0 1 10 0 10 2 20 2 20 0 30 1	30
4 5 5 10 1 8 -1 12 3 13 0 18 3 17 0 20 1 22 -1 25 5	22.80624847
3 0 0 3 -1 2 1 8 0 8 1 5 -1 5 1 0 2	16.144380531

Problem D. Drone Control

Source file name:	Drone.c, Drone.cpp, Drone.java, Drone.py
Input:	`Standard`
Output:	`Standard`

You are designing a controller for an interesting aircraft called the *Single Copter*. It only has one propeller, but the outgoing air flow is further shaped by four *flaps* that control three Euler angles (pitch, roll and yaw) that help maintain the requested orientation of the craft. Each of these flaps can assume any angle requested by the flight controller, and the effects of the flaps being at certain angles should translate to exerting the requested forces on pitch, roll and yaw.



Pitch Roll Yaw

Pitch, roll and yaw on a *Single Copter*

Define the angles of the flaps to be n, e, s, and w (for "north", "east", "south" and "west" respectively). The forces in the directions of pitch, roll and yaw are defined by the following equations:

$$p = e - w$$
$$r = n - s$$
$$y = n + e + s + w$$

As there are four variables and three constraints, you decided that, from the perspective of aerodynamics, it makes sense to make the maximum of the flap angles as small as possible, that is, you additionally want to minimise $\max\{|n|, |e|, |s|, |w|\}$.

Find the best parameters to send to the *Single Copter* to achieve the desired pitch, yaw, and roll.

Input

- One line containing the number q $\left(1 \leq q \leq 10^4\right)$, the number of requests to follow.
- i further lines, each containing three real numbers p_i, r_i, y_i $(-1 \leq p_i, r_i, y_i \leq +1)$.

Output

Output q lines. In the i-th line, output the solution for the i-th request, four numbers n_i, e_i, s_i, w_i, separated by whitespace.

Your answer will be considered correct if the resulting pitch, roll and yaw differ by at most 10^{-6} from the requested ones, and the maximum of the absolute values of flap outputs does not exceed the true value by more than 10^{-6}.

Example

Input	Output
8 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 1 1	0.0 0.0 0.0 0.0 0.0 0.5 0.0 -0.5 0.5 0.0 -0.5 0.0 0.25 0.25 0.25 0.25 0.5 0.5 -0.5 -0.5 0.5 0.5 0.5 -0.5 0.5 0.5 -0.5 0.5 0.75 0.75 -0.25 -0.25

Problem E. Eradication Sort

Source file name:	Eradication.c, Eradication.cpp, Eradication.java, Eradication.py
Input:	`Standard`
Output:	`Standard`

The members of the No-Weather-too-Extreme Recreational Climbing society completed their first successful summit seven years ago to this day!

At the time, we took a picture of all the members standing together in one row. However, the photograph looks messy, as the climbers were not standing in order of height, and we have no way to reorder them.

We will need to cut some of the climbers out of the picture.

This picture of 7 (formerly 11) climbers was edited to solve Example Input 3.

An optimal solution minimises the size and number of visible gaps in the photo. We define the *cost* as the sum of the squares of the lengths of gaps left in the edited photo. For example, if two individual climbers are removed from the photo and one pair of adjacent climbers are removed, the total *cost* is $1^2 + 1^2 + 2^2 = 6$.

Find the minimum possible cost you can reach by removing climbers.

Input

- The number of people in the photo n $\left(1 \leq n \leq 10^6\right)$.
- n integers representing the heights of people in the photo, $h_1 \ldots h_n$ $\left(0 \leq h \leq 10^6\right)$.

Output

Output the minimum *cost* achieved by removing climbers from the photo, such that the remaining climbers in the photo make a **non-decreasing** sequence.

Example

Input	Output
7 1 2 3 0 5 6 7	1
9 4 5 6 4 2 3 6 6 6	8
11 3 6 12 7 7 7 6 8 10 5 5	6

Problem F. Finding Suspicious Proteins

Source file name:	Finding.c, Finding.cpp, Finding.java, Finding.py
Input:	`Standard`
Output:	`Standard`

Little Claire studies proteins, which are sequences of amino acids. There are 20 amino acids from which proteins are built. While amino acids all have proper names, such as *alanine* or *glycine*, they are often denoted by single letters, so that proteins can be seen as sequences of different lengths, such as `DTASDAAAAAALTAABAAAAAKLTABBAAAAAAATAA`, `TIFLQQQQQQQQQQ` or even maybe `RICKRQLL`.

Comparing two proteins can be difficult, because they may contain active sites, which determine their function in a cell, and less important parts of the sequence. Recent advances in artificial neural networks made it possible to train a network that, given a protein, outputs a sequence of l numbers, where each number roughly corresponds to a feature of a protein that correlates with its possible functions in a cell. Such a sequence is called an *embedding*.

Claire is particularly interested in *suspicious* proteins, those which are really different from others. For this purpose, she considers the so-called *Manhattan distance* between embeddings of proteins. For two protein embeddings p and q of length l, the distance $\mathcal{D}(p, q)$ is computed as follows:

$$\mathcal{D}(p, q) = \sum_{i=1}^{l} |p_i - q_i|,$$

where p_i is the i-th element of the embedding p.

Claire wants to find k suspicious proteins in the given list of n proteins. As a baseline for her studies, Claire wants to use the following greedy algorithm:

- Find a protein $p^{(1)}$ which is the most distant from the first protein in the list.
- The second protein, $p^{(2)}$, is chosen as the most distant protein from $p^{(1)}$.
- The third one, $p^{(3)}$, is chosen so that $\min\{\mathcal{D}\left(p^{(1)}, p^{(3)}\right), \mathcal{D}\left(p^{(2)}, p^{(3)}\right)\}$ is maximum possible. That is, it must be far away from *both* previously chosen proteins.
- All subsequent proteins $p^{(i)}$, $4 \leq i \leq k$, are chosen in a similar way: the minimum of the distances to all the previously chosen proteins should be maximum possible.

Note that, in the case of ties, the first matching protein in the list must be chosen.

Claire's implementation works nicely for small numbers n and k, but becomes too slow as they increase. You must find a way to optimise this.

Input

The first line contains three numbers n ($3 \leq n \leq 10^4$), l ($1 \leq l \leq 100$) and k ($2 \leq k \leq \min\{n, 256\}$): the overall number of proteins, the length of each protein embedding, and the number of proteins to choose.

Each of the following n lines starts with a protein identifier, which is a sequence of at least one and most ten capital letters and/or numbers. Then, separated by whitespace, come l single-digit integer numbers $v_{1\ldots l}$ ($0 \leq v \leq 9$), which define the embedding of the protein. All protein identifiers will be different.

Output

Output the identifiers of k chosen proteins, one per line, in their respective order ($p^{(1)}$ to $p^{(k)}$).

Example

Input	Output
4 2 2 FIRST 3 4 SECOND 1 2 THIRD 8 7 FOURTH 5 6	THIRD SECOND
6 5 3 1OGLOBIN 1 1 1 1 1 GLU10 9 9 9 9 9 8EIN 8 9 8 9 9 COLLA6EN 6 5 4 3 2 7ILK 3 4 5 6 7 OLBUMIN 1 2 0 2 1	GLU10 1OGLOBIN 7ILK

Problem G. Word Search

Source file name:	Word.c, Word.cpp, Word.java, Word.py
Input:	`Standard`
Output:	`Standard`

Find parts of a 2d grid matching a 2d word.

Input

- One line containing the number of rows and columns in the search key, r_k and c_k ($1 \leq r, c \leq 2000$).
- r_k further lines, each containing c_k Latin characters comprising a row of the search key.
- One line containing the number of rows and columns in the haystack, r_h and c_h ($r_k \leq r_h \leq 2000$, $c_k \leq c_h \leq 2000$).
- r_h further lines, each containing c_h Latin characters comprising a row of the search key.

Output

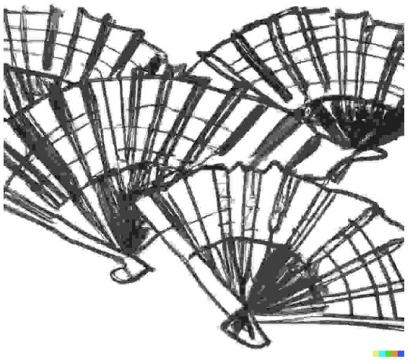
Illustrate the matching areas of the haystack by printing a grid of the same size. In locations that are part of at least one match, print the original character from the haystack. In other cases, print a full-stop "." character.

Example

Input	Output
3 3 ghi lmn qrs 5 5 abcde fghij klmno pqrst uvwxy	..... .ghi. .lmn. .qrs. .....
1 2 ab 6 4 abba baab abba baab abba baab	ab.. ..ab ab.. ..ab ab.. ..ab
4 1 n a n a 7 6 ananan nanana ananan nanana ananan nanana batman	.n.n.n nanana ananan nanana ananan .a.ana ....a.
2 2 oo oo 5 5 xoooo oxooo ooxoo oooxo oooox	..ooo ..ooo oo.oo ooo.. ooo..

Problem H. Hand Fan

Source file name:	Handfan.c, Handfan.cpp, Handfan.java, Handfan.py
Input:	`Standard`
Output:	`Standard`



Little Fran received a wooden frame in the shape of a regular polygon as a gift. As polygon has n vertices, he also received $\frac{n(n-3)}{2}$ wooden sticks that match each possible diagonal. Vertices of the polygon are labelled with integers from 1 to n in counterclockwise order. In the beginning, Fran arranged $n-3$ sticks inside the frame in such a way that every stick touches two non-neighboring vertive of the frame, and no two sticks cross each other. In other words, he made a triangulation. As that was not interesting enough for him, he decided to play with this configuration by applying a particular operation that consists of two steps:

1. Remove a stick.
2. Add a new stick in such a way that we obtain a new triangulation.

We characterize the operation with an ordered pair of unordered pairs $((a,\ b),\ (c,\ d))$ which signifies that little Fran removed a stick touching vertices a and b, and added a stick touching vertices c and d.

Fran loves hand fans so, while doing these operations, he sometimes asks himself: *"How many operations is needed to transform this triangulation into a "fan" triangulation in vertex x, and, in how many ways is this achievable?"*.

Since he is busy doing operations and having fun, he asks for your help!

"Fan" triangulation in vertex x is a triangulation where all diagonals have a common endpoint, namely vertex x.

Let the number of needed operations be m. Let $f_1,\ f_2,\ \ldots,\ f_m$ be a sequence of operations that, when applied in given order, achieves wanted triangulation, thus representing one way of getting there. Let $s_1,\ s_2,\ \ldots,\ s_m$ be another such sequence. Two sequences are distinct if there exists an index i such that $f_i \neq s_i$.

As the number of such sequences can be huge, little Fran is only interested in its remainder modulo 10^9+7.

Input

In the first line are integeres n and q $\left(4 \leq n \leq 2 \cdot 10^5,\ 1 \leq q \leq 2 \cdot 10^5\right)$, the number of vertices and the number of events.

In each of the next $n-3$ lines there are integers $x_i,\ y_i$ $(1 \leq x_i, y_i \leq n)$, the labels of vertices that the i-th stick touches.

In each of the next q lines there is the integer t_i $(1 \leq t_i \leq 2)$ that represents the type of event.

If $t_i = 1$, it is followed by 4 integers $a_i,\ b_i,\ c_i,\ d_i$ $(1 \leq a_i, b_i, c_i, d_i \leq n)$ that signify an operation $((a_i,\ b_i),\ (c_i,\ d_i))$ is being made at that moment. It is guaranteed that given operation can be realized.

If $t_i = 2$, it is followed by an integer x_i $(1 \leq x_i \leq n)$, which means that little Fran is interested in data for the "fan" triangulation at vertex x_i in relation to the current triangulation.

Output

For every event of type 2, in order they came in input, output two integers, minimal number of operations

needed and number of ways to get to the target triangulation using minimal number of operations.

Example

Input	Output
4 3 1 3 2 1 1 1 3 2 4 2 1	0 1 1 1
5 4 1 3 3 5 2 1 2 2 1 1 3 2 5 2 2	1 1 2 1 1 1
9 3 1 5 1 7 2 4 2 5 5 7 7 9 2 1 1 2 5 1 4 2 1	4 12 3 6

Explanation

Clarification of the first example:

Starting triangulation is already a "fan" triangulation in vertex 1, so little Fran must do no operations, which there is one way of doing so. After executing a given operation, there is now only one way to get it to the previous state and that is by applying operation ((2, 4), (1, 3)).

Clarification of the second example:

Only sequence of operations for the first query: ((3, 5), (1, 4)). Only sequence of operations for the second query: ((1, 3), (2, 5)), ((3, 5), (2, 4)). Only sequence of operations for the third query: ((3, 5), (2, 5)).

Problem I. Die Hard

Source file name:	Diehard.c, Diehard.cpp, Diehard.java, Diehard.py
Input:	`Standard`
Output:	`Standard`

John and Hans are playing a game involving 3 dice. Even though they are all 6-sided, they are not guaranteed to be identical.

First John picks one of the dice and then Hans picks one of the remaining two. Then they both roll their chosen die. If they roll the same number, they both re-roll their die. Otherwise the winner is the one who rolled the highest number.

In case neither John or Hans can win with their chosen dice, they do not bother to re-roll the dice indefinitely and no winner is declared.

Can you help John pick a die that guarantees that he wins with a probability of at least $\frac{1}{2}$?

Miwin's dice by Dr.M.Winkelmann, public domain

Input

The input consists of three lines. Line i contains 6 positive integers x_j ($1 \leq x_j \leq 1000$), describing the sides of the i'th die.

Output

Output the smallest $i \in \{1, 2, 3\}$, such that John can pick die i and be guaranteed to win with probability at least $\frac{1}{2}$. If no such die exists, output "No dice".

Example

Input	Output
1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6	1
1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3	3
2 2 4 4 9 9 1 1 6 6 8 8 7 7 5 5 3 3	No dice
1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2	No dice

Problem J. Jabber Network

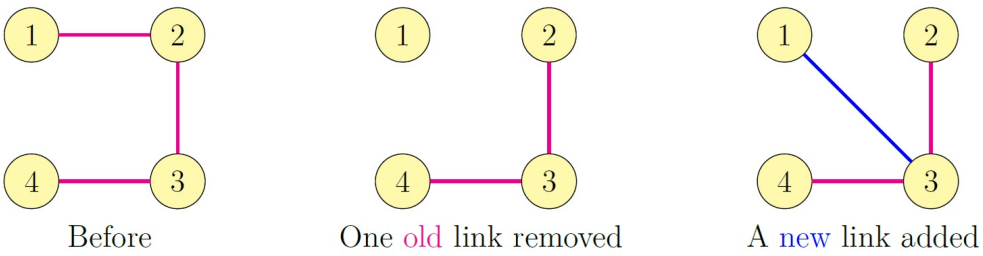
Source file name:	Jabber.c, Jabber.cpp, Jabber.java, Jabber.py
Input:	`Standard`
Output:	`Standard`

Dave, an old Computer Science professor, still maintains a local community computer network even after retirement. Each community member has a computer with three networking cards, and some of these cards may be connected by a cable. They form a connected network, and, following a long resource-saving tradition, the number of cables is kept to the minimum possible.

The habits of all the community members are quite stable: for every two computers the number of packets per second between them is known exactlyc. However, the network was first assembled a long time ago, so the connections are not necessarily be optimal any more. For two computers numbered i and j we define d_{ij} the shortest path between them, measured in the number of cables, and c_{ij} the number of packets per second that should be transferred from i to j. The *commutation stress* is defined to be the sum of $c_{ij} \cdot d_{ij}$ for all $i < j$, and one would like to minimise it.

Dave realised that it is finally the time to upgrade the cables — after all, they do degrade with time. He wants to take this opportunity to also optimise the network, such that the *commutation stress* becomes smaller. However, he is no longer as quick as in his youth, and his friends may get dissatisfied if too much disruption happens at once. So he decided that he will perform the upgrade using the following scenario. For each of the old cables, he will do the following:

1. Remove the old cable.
2. Connect the network back using a new cable, choosing the computers to connect in such a way that the resulting *commutation stress* is minimum possible.
3. If there are many ways to do this, break ties by choosing the computers with the smallest numbers: if (u_1, u_2) and (v_1, v_2) result in the same *commutation stress*, but $u_1 < v_1$ (or $u_1 = v_1$ and $u_2 < v_2$), then (u_1, u_2) should be chosen.



A single reconnection operation (the first one in the example input)

Note that, since each computer only has three network cards, Dave cannot connect two arbitrary computers on the second step: if one of them is already connected to three other computers, it is impossible to connect it to yet another computer. Fortunately, it is not hard to show that it is always possible to find two computers to connect: for instance, Dave can choose the two just-disconnected computers.

Unfortunately, the task appeared to be more difficult than it seemed initially. Could you help Dave?

Input

The first line of the input file contains an integer n ($2 \le n \le 2 \cdot 10^3$), the number of computers in the network.

The following $n-1$ lines contain two integers each: a_i, b_i, where $(1 \leq a_i < b_i \leq n)$ are the numbers of the computers initially connected by an old cable number i. The cables are to be removed and replaced in the order they are given in the input file. It is guaranteed that it is possible to reach any computer from any other computer using old cables (that is, the network is initially connected), and that no computer is connected with more than three other computers.

The next line contains an integer d $(2 \leq d \leq 10^4)$, the number of computer pairs that are known to transmit data to each other.

The following d lines contain three integers each: s_i, t_i and d_i, where s_i and t_i are the numbers of the computers which transmit data to each other $(1 \leq s_i < t_i \leq n)$, and d_i $(1 \leq d_i \leq 10^9)$ is the number of packets per second to be transmitted.

Output

Output $n-1$ lines containing two integers each: x_i, y_i, where $(1 \leq x_i < y_i \leq n)$, should be the numbers of the computers connected by a new cable at step i.

Note that, due to the tie-breaking rule detailed above, the correct output is unique.

Example

Input	Output
4 1 2 2 3 3 4 6 1 2 1 1 3 10 1 4 1 2 3 10 2 4 1 3 4 10	1 3 2 3 3 4

Problem K. Kindong

Source file name:	Kindong.c, Kindong.cpp, Kindong.java, Kindong.py
Input:	`Standard`
Output:	`Standard`

In a certain kingdom, the king wants to protect his citizens by deploying guards. He has recruited a number of guards, and has outfitted them with heavy armor for protection from bandits, foreign knights, and other ne'er-do-wells. His guards are tough, but unfortunately they aren't very bright and will attack anyone wearing armor, even each other!

The kingdom consists of a number of villages, connected by roads. All of the roads are of poor quality. Some are muddy, some have rickety bridges. None of them can support a guard in full armor. So, the king must decide which roads to improve so that his guards can reach the entire kingdom. The roads are bidirectional. Each guard can only be deployed to a single village in a certain subset of the kingdom's villages.

The king needs to minimize the cost of improving roads, while satisfying several other constraints:

- Every guard must be deployed; none must be left out.
- Every guard must be deployed in their subset of villages.
- Every village must be reachable by exactly one guard. If two guards can reach each other, they'll fight.

Help the king determine the minimum cost of improving the roads of his kingdom while satisfying the above constraints.

Input

The first line of input contains three integers n ($1 \leq n \leq 300$), r $\left(0 \leq r \leq \frac{n(n-1)}{2}\right)$ and g ($1 \leq g \leq n$), where n is the number of villages, r is the number of roads, and g is the number of guards. The villages are numbered 1 through n.

Each of the next r lines contains three integers a, b ($1 \leq a < b \leq n$) and c ($1 \leq c \leq 1000$). Each line describes a road between village a and village b, costing c to improve. The roads are bidirectional; a guard can go from a to b or from b to a. Every pair of villages has at most one road between them.

Each of the next g lines starts with an integer k ($1 \leq k \leq n$), and then contains k integers v ($1 \leq v \leq n$). Each line describes the villages comprising the subset where one particular guard may be placed. The subsets may overlap.

Output

Output a single integer, which is the minimum cost the king must pay to improve enough roads so that every village is reachable by exactly one guard, and every guard is deployed. Output -1 if it isn't possible to deploy the guards in a way that satisfies all of the constraints.

Example

Input	Output
5 6 2 1 2 1 1 3 4 2 4 2 2 5 5 3 4 7 4 5 3 2 1 2 2 2 4	8

Problem L. Lego Cubes

Source file name:	Lego.c, Lego.cpp, Lego.java, Lego.py
Input:	`Standard`
Output:	`Standard`

For his thirteenth birthday, Donald's parents bought him a brand-new set of Lego cubes. In the set, there are n cubes of equal size, where the i-th cube came in color i. Using these cubes he decided to build a wall.

Donald will build his wall on a row-like Lego base that has k places where cubes can be put in. He puts the cubes in the following way:

- First, he puts the cube with color 1 on an arbitrary spot on the base.
- For each cube from 2 to n, he places it in a spot neighboring the previously placed cube. If that spot isn't empty, he puts the new cube on top of all the others.

After he built the wall, Donald wrote on a piece of paper a sequence of length k: on the i-th position in the sequence he wrote the color of the top cube in the i-th place, or 0 if there isn't a cube in that place.

He immediately asked himself how many different sequences could he have written on the piece of paper. Two sequences are considered different if there exists a position in which they differ. After some time, he has managed to calculate the solution, but he is not sure whether it is correct, so he asks for your help.

Input

The only line has integers n and k ($2 \leq n, k \leq 5000$), the number of cubes, and the length of the base.

Output

In the only line, print the answer to Donald's question, modulo $10^9 + 7$.

Example

Input	Output
4 3	8
3 5	14
100 200	410783331

Explanation

Clarification of the first example:

All possible sequences are: (0, 3, 4), (2, 3, 4), (0, 4, 3), (1, 4, 3), (4, 3, 0), (4, 3, 2), (3, 4, 0), (3, 4, 1).

Clarification of the second example:

One of the possible sequences is (0, 3, 2, 0, 0). Donald can achieve that by putting the first cube on the second place, second cube on the third place, and third cube on the second place (on top of the first cube).