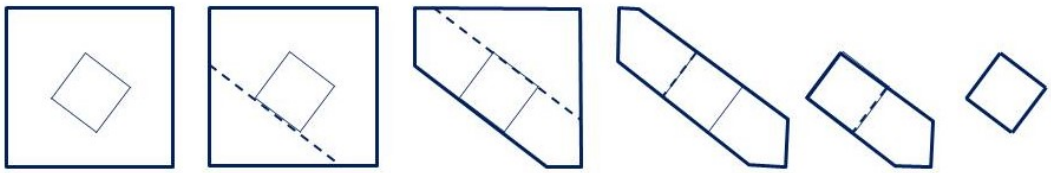
Problem A. Cut It Out!

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

You are given two convex polygons A and B. It is guaranteed that B is strictly contained inside of A.

You would like to make a sequence of cuts to cut out B from A. To do this, you draw a straight line completely through A that is incident to one of the edges of B, which separates A into two pieces. You cut along this line and discard the piece that doesn't contain B. You repeat this until the piece that you have left is exactly B.



The cost of making a cut is equal to the length of the cut (i.e. the length of the line through the remainder of A). Given A and B, find the minimum cost needed to cut B out.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing a single integer a ($3 \le a \le 200$), which is the number of points in polygon A. Each of the next a lines will contain two integers x and y ($-10^6 \le x, y \le 10^6$), which are the vertices of polygon A, in clockwise order. It is guaranteed that polygon A will be convex.

The next line will contain a single integer b ($3 \le b \le 200$), which is the number of points in polygon B. Each of the next b lines will contain two integers x and y ($-10^6 < x, y < 10^6$), which are the vertices of polygon B, in clockwise order. It is guaranteed that polygon B will be convex. It is also guaranteed that polygon B will reside entirely within the interior of polygon A.

No three points, within a polygon or across polygons, will be collinear.

Output

Output a single floating point number, which is the minimum cost to cut B out of A. To be considered correct, this number must be within a relative error of 10^{-6} of the judges' answer.

Examples

standard input	standard output
4 0 0 0 14 15 14 15 0 4 8 3 4 6 7 10 11 7	40.0000000000
4 -100 -100 -100 100 100 100 100 -100 8 -1 -2 -2 -1 -2 1 -1 2 1 2 2 1 2 -1 1 -2	322.1421356237

Problem B. Double Clique

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

You are given an undirected graph G with n nodes and m edges. The set of vertices is V and the set of edges is E.

Let the *Complement* of G be G'. The *Complement* of a graph is a graph with all of the same nodes, but if there's no edge between nodes a and b in G, then there is an edge between a and b in G', and if there is an edge between a and b in G, then there is no edge between a and b in G'.

A *Clique* is a subset of nodes that have an edge between every pair. A subset of nodes S is called a *Double Clique* if S forms a clique in G, and $V - S$ forms a clique in G'. Note that an empty set of nodes is considered a clique.

Given a graph, count the number of double cliques in the graph modulo $10^9 + 7$.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n and m ($1 \le n, m \le 2 \times 10^5$), where n is the number of nodes and m is the number of edges in the graph. The nodes are numbered $1..n$. Each of the next m lines will contain two integers a and b ($1 \le a < b \le n$), representing an edge between nodes a and b. The edges are guaranteed to be unique.

Output

Output a single integer, which is the number of Double Cliques in the graph modulo $10^9 + 7$.

Examples

standard input	standard output
3 3 1 3 1 2 2 3	4

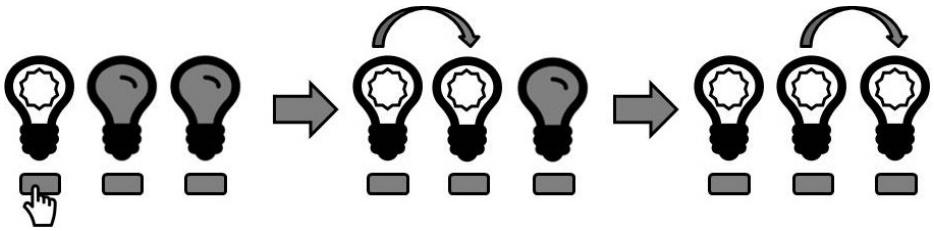
Problem C. Flashing Fluorescents

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

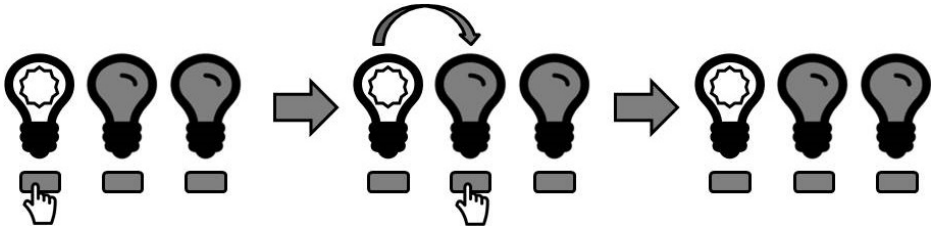
You have n lights, each with its own button, in a line. Pressing a light's button will toggle that light's state; if the light is on, it will turn off, and if the light is off, it will turn on. The lights change at 1 second timesteps. You can press a button at any time, but it will not take effect until the next timestep. Before each timestep, you may choose to push at most one button (you may also choose to not press any button).

Pushing a button will affect not just the light in question, but all lights down the line. More specifically, if you choose to press the i-th button right before the k-th timestep, then the $(i+m)$-th light will toggle on the $(k+m)$-th timestep (with $i+m \leq n$). For example, if you press button 5 just before time 19, then light 5 will toggle at time 19, light 6 will toggle at time 20, light 7 will toggle at time 21, and so on. If you push a button that will take effect at the same time as its light would have toggled due to an earlier button press, then the two cancel each other out, including subsequent toggles.

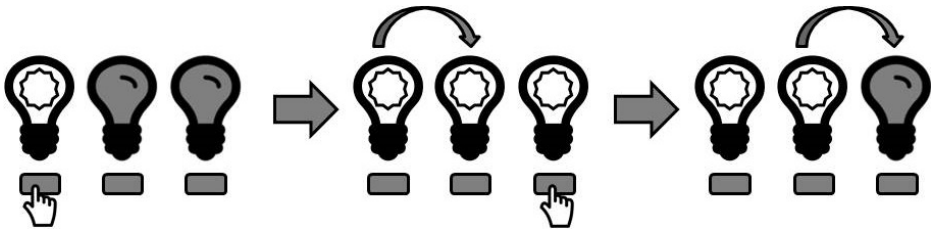
Suppose there are three lights, all of which are off at the start. If you press the first button before the first timestep, this will happen in three timesteps:



Now, suppose you press the first button before the first timestep, and then the second button between the first and second timesteps. The button press will cancel out the propagation, and this will happen (note that the propagation will go no further):



Now, suppose you press the first button before the first timestep, and then the third button between the first and second timesteps. All three lights will be on at the second timestep (but not the third):



You wish to turn on all the lights. What is the earliest time you could possibly see all of the lights turned on? Note that if the lights are all on at time t but not at time $t+1$ due to this propagation, t is still the correct answer.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of a single string S ($1 \leq |S| \leq 16$). The string S will contain only the characters 1 and 0, where 1 represents that that light is initially on, and 0 represents that that light is initially off. The first character is light 1, the next is light 2, and so on.

Output

Output a single integer, which is the earliest time at which all of the lights are on.

Examples

standard input	standard output
1101	1
1	0
000	2

Problem D. Missing Gnomes

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

A family of n gnomes likes to line up for a group picture. Each gnome can be uniquely identified by a number $1..n$ written on their hat.

Suppose there are 5 gnomes. The gnomes could line up like so: $1, 3, 4, 2, 5$.

Now, an evil magician will remove some of the gnomes from the lineup and wipe your memory of the order of the gnomes. The result is a subsequence, perhaps like so: $1, 4, 2$.

He then tells you that if you ordered all permutations of $1..n$ in lexicographical order, the original sequence of gnomes is the first such permutation which contains the remaining subsequence. Your task is to find the original sequence of gnomes.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n and then m ($1 \le m \le n \le 10^5$), where n is the number of gnomes originally, and m is the number of gnomes remaining after the evil magician pulls his trick. Each of the next m lines will contain a single integer g ($1 \le g \le n$). These are the remaining gnomes, in order. The values of g are guaranteed to be unique.

Output

Output n lines, each containing a single integer, representing the first permutation of gnomes that could contain the remaining gnomes in order.

Examples

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

Problem E. Prefix Free Code

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Consider n initial strings of lower case letters, where no initial string is a prefix of any other initial string. Now, consider choosing k of the strings (no string more than once), and concatenating them together. You can make this many such composite strings:

$$n \times (n-1) \times (n-2) \times \ldots \times (n-k+1)$$

Consider sorting all of the composite strings you can get via this process in alphabetical order. You are given a test composite string, which is guaranteed to belong on this list. Find the position of this test composite string in the alphabetized list of all composite strings, modulo $10^9 + 7$. The first composite string in the list is at position 1.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers, first n and then k ($1 \le k \le n$), where n is the number of initial strings, and k is the number of initial strings you choose to form composite strings. The upper bounds of n and k are limited by the constraints on the strings, in the following paragraphs.

Each of the next n lines will contain a string, which will consist of one or more lower case letters $a..z$. These are the n initial strings. It is guaranteed that none of the initial strings will be a prefix of any other of the initial strings.

Finally, the last line will contain another string, consisting of only lower case letters $a..z$. This is the test composite string, the position of which in the sorted list you must find. This test composite string is guaranteed to be a concatenation of k unique initial strings.

The sum of the lengths of all input strings, including the test string, will not exceed 10^6 letters.

Output

Output a single integer, which is the position in the list of sorted composite strings where the test composite string occurs. Output this number modulo $10^9 + 7$.

Examples

standard input	standard output
5 3 a b c d e cad	26
8 8 font lewin darko deon vanb johnb chuckr tgr deonjohnbdarkotgrvanbchuckrfontlewin	12451

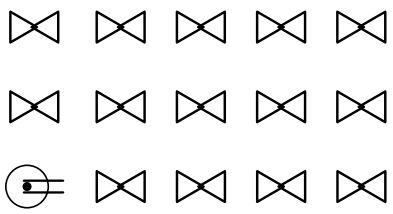
Problem F. Probe Droids

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

After being stationed on Hoth, you've decided that this is the worst decision you have ever made. The planet is cold, there is nothing to do, and to make matters worse, the Empire keeps sending probe droids down to see if anyone is hiding on the planet. At least you can do something about the probe droids!

The field near your base is covered with these droids in a uniform fashion. Your base has a turret in the southwest (bottom-left) corner of the field, at location $(1, 1)$. Each remaining lattice point (points with integer coordinates) on the grid of $n \times m$ points contains a droid.

An example 3×5 grid:



Note that row 1 is at the bottom, row n is at the top, column 1 on the left and column m on the right.

Your base is at coordinate $(1, 1)$, while the droids are at all positive coordinates in the range $(1..n,1..m)$, excluding $(1, 1)$ of course. Your base has a turret on it facing east, towards higher-number columns in row 1. You've programmed the turret to operate in the following way: If a droid is its line of sight, then destroy the droid. Otherwise, rotate counter-clockwise until the turret can see a droid. Keep going until all droids are destroyed.

Your task is to figure out the i-th droid destroyed when the turret executes your algorithm.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with three integers n, m ($1 \le n, m \le 10^6$, m and n cannot both be 1) and q ($1 \le q \le 100$), where the grid has n rows and m columns, and there will be q queries to answer. Each of the next q lines will have a single integer i ($1 \le i < n \times m$), which is a query for the i-th droid destroyed.

Output

Output q lines, corresponding to the q queries in order. For each query, output two integers r and c, separated by a single space, which represent the row (r) and column (c) of the i-th droid destroyed.

Examples

standard input	standard output
3 5 3 1 14 8	1 2 3 1 3 5

Problem G. Rainbow Graph

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

Roy and Biv have an undirected graph with n nodes, numbered $1..n$. Each edge of the graph has a weight and a color. Weights are positive integers. There are exactly three colors of edges: Red, Green, and Blue. There may be multiple edges between the same two vertices, and there may even be self-loops in the graph.

For a fixed positive integer k, consider the following task: Roy and Biv want to choose exactly k edges of their graph and erase all other edges. They want to do this in such a way that when they look at the graph with just the k edges they have chosen, they will both see that the entire graph is connected.

However, there is a catch: Roy cannot see the color Red and Biv cannot see the color Blue. Therefore, they have to choose the edges in such a way that the Blue and Green edges are enough to connect all nodes, and also the Red and Green edges are enough to connect all nodes. What is the minimum combined weights for all of the edges which will allow both Roy and Biv to see the graph as connected, for all values of k from 1 to the total number of edges?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers, n and m ($1 \le n, m \le 100$), where n is the number of nodes in the graph, and m is the number of edges.

Each of the next m lines will contain three integers and a character, a, b ($1 \le a, b \le n$), w ($1 \le w \le 1\,000$) and c ($c \in \{R, G, B\}$), which represents an edge between nodes a and b with weight w, and color c (R=Red, G=Green, B=Blue).

Output

Output m lines, with each line representing a result for $k = 1..m$ in order. If it is not possible for both Roy and Biv to see the graph as connected, output -1 for that value of k. Otherwise, output the minimum sum of edge weights for a subset of k edges which will allow both Roy and Biv to see the graph as connected.

Examples

standard input	standard output
5 8 1 5 1 R 2 1 2 R 5 4 5 R 4 5 3 G 1 3 3 G 4 3 5 G 5 4 1 B 1 2 2 B	-1 -1 -1 -1 15 14 17 22

Problem H. Recovery

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

Consider an $n \times m$ matrix of ones and zeros. For example, this 4×4:

$$\begin{matrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{matrix}$$

We can compute even parity for each row, and each column. In this case, the row parities are $[0, 1, 1, 0]$ and the column parities are $[1, 0, 0, 1]$ (the parity is 1 if there is an odd number of 1s in the row or column, 0 if the number of 1s is even). Note that the top row is row 1, the bottom row is row n, the leftmost column is column 1, and the rightmost column is column m.

Suppose we lost the original matrix, and only have the row and column parities. Can we recover the original matrix? Unfortunately, we cannot uniquely recover the original matrix, but with some constraints, we can uniquely recover a matrix that fits the bill. Firstly, the recovered matrix must contain as many 1's as possible. Secondly, of all possible recovered matrices with the most 1's, use the one which has the smallest binary value when you start with row 1, concatenate row 2 to the end of row 1, then append row 3, row 4, and so on.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will consist of exactly two lines. The first line will contain a string R ($1 \le |R| \le 50$), consisting only of the characters 0 and 1. These are the row parities, in order. The second line will contain a string C ($1 \le |C| \le 50$), consisting only of the characters 0 and 1. These are the column parities, in order.

Output

If it is possible to recover the original matrix with the given constraints, then output the matrix as $|R|$ lines of exactly $|C|$ characters, consisting only of 0's and 1's. If it is not possible to recover the original matrix, output -1.

Examples

standard input	standard output
0110 1001	1111 0111 1110 1111
0 1	-1
11 0110	1011 1101

Problem I. Red Black Tree

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a rooted tree with n nodes. The nodes are numbered $1..n$. The root is node 1, and m of the nodes are colored red, the rest are black.

You would like to choose a subset of nodes such that there is no node in your subset which is an ancestor of any other node in your subset. For example, if A is the parent of B and B is the parent of C, then you could have at most one of A, B or C in your subset. In addition, you would like exactly k of your chosen nodes to be red.

If exactly m of the nodes are red, then for all $k = 0..m$, figure out how many ways you can choose subsets with k red nodes, and no node is an ancestor of any other node.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n ($1 \le n \le 2 \times 10^5$) and m ($0 \le m \le min(10^3,\ n)$), where n is the number of nodes in the tree, and m is the number of nodes which are red. The nodes are numbered $1..n$.

Each of the next $n-1$ lines will contain a single integer p ($1 \le p \le n$), which is the number of the parent of this node. The nodes are listed in order, starting with node 2, then node 3, and so on. Node 1 is skipped, since it is the root. It is guaranteed that the nodes form a single tree, with a single root at node 1 and no cycles.

Each of the next m lines will contain single integer r ($1 \le r \le n$). These are the numbers of the red nodes. No value of r will be repeated.

Output

Output $m + 1$ lines, corresponding to the number of subsets satisfying the given criteria with a number of red nodes equal to $k = 0..m$, in that order. Output this number modulo $10^9 + 7$.

Examples

standard input	standard output
4 1 1 1 1 3	5 4
4 4 1 1 1 1 2 3 4	1 4 3 1 0
14 4 1 2 1 2 3 4 5 5 13 8 10 4 4 8 3 12 13	100 169 90 16 0

Problem J. Winter Festival

Input file:	*standard input*
Output file:	*standard output*
Time limit:	7 seconds
Memory limit:	512 mebibytes

Peter's favorite season is winter. Caught up in the festivities, Peter would like to decorate his city.

The city has many areas for things like ice skating, skiing, and snowball fights. There are roads connecting some pairs of areas. Peter would like to put a special decoration along each of these roads. There are three types of decorations of costs 0, 1, and 2, respectively.

Peter would like his decoration to meet some properties:

- For any pair of distinct roads adjacent to an area, let a and b be the costs of decorations along those roads. It must hold that $(a + b) \bmod 3 \neq 1$.
- The sum of costs along any cycle must be an odd number.

A cycle is a sequence of areas connected by roads that form a loop. Each area may appear exactly once in the loop.

Peter would like to know: What is the cheapest amount he can pay to decorate his city the way he wants?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line with two integers n and m ($1 \leq n, m \leq 10^5$), where n is the number of areas and m is the number of roads. The areas are numbered $1..n$.

Each of the next m lines will each contain two integers a and b ($1 \leq a < b \leq n$), indicating that there is a road directly connecting areas a and b. No two roads will connect the same two areas. It may or may not be possible to get from every area to every other area along the roads.

Output

Output a single integer, which is the minimum cost of decorating the city, or -1 if it isn't possible to decorate the city according to Peter's properties.

Examples

standard input	standard output
5 8 1 4 4 5 1 5 1 2 1 3 2 3 3 5 2 5	-1
6 5 2 4 3 5 1 5 3 6 1 6	5
10 10 5 8 2 6 3 9 1 4 9 10 4 6 5 9 7 8 7 10 2 3	5

Problem K. Zoning Houses

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Given a registry of all houses in your state or province, you would like to know the minimum size of an axis-aligned square zone such that every house in a range of addresses lies in the zone or on its border. The zoning is a bit lenient and you can ignore any one house from the range to make the zone smaller.

The addresses are given as integers from $1..n$. Zoning requests are given as a consecutive range of houses. A valid zone is the smallest axis-aligned square that contains all of the points in the range, ignoring at most one.

Given the (x, y) locations of houses in your state or province, and a list of zoning requests, you must figure out for each request: What is the length of a side of the smallest axis-aligned square zone that contains all of the houses in the zoning request, possibly ignoring one house?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. Each test case will begin with a line containing two integers n and q ($1 \le n, q \le 10^5$), where n is the number of houses, and q is the number of zoning requests.

The next n lines will each contain two integers, x and y ($-10^9 \le x, y \le 10^9$), which are the (x,y) coordinates of a house in your state or province. The address of this house corresponds with the order in the input. The first house has address 1, the second house has address 2, and so on. No two houses will be at the same location.

The next q lines will contain two integers a and b ($1 \le a < b \le n$), which represents a zoning request for houses with addresses in the range $[a..b]$ inclusive.

Output

Output q lines. On each line print the answer to one of the zoning requests, in order: the side length of the smallest axis-aligned square that contains all of the points of houses with those addresses, if at most one house can be ignored.

Examples

standard input	standard output
3 2 1 0 0 1 1000 1 1 3 2 3	1 0
4 2 0 0 1000 1000 300 300 1 1 1 3 2 4	300 299