Problem A. Control Towers

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are an architect tasked with designing the new airport in your city. After you have completed your design, you just realize you forgot to allocate spaces for the control towers.

The layout of the new airport can be represented by a 2D grid of r rows and c columns, with the rows numbered from 1 to r (top to bottom) and the columns numbered from 1 to c (left to right). The cell in row i and column j is denoted by (i, j). Each cell can either be occupied or empty.

You need to place four control towers (numbered from 1 to 4), each in a different empty cell. To allow easier communication between different towers, for all $k = 1, 2, 3$, you want tower k and tower $k + 1$ to be placed either in the same row or in the same column.

You want to calculate the number of ways to place the control towers to satisfy the requirements above. Two ways are considered different if there exists k where control tower k is placed in different cells.

Input

The first line of input contains two integers r and c ($1 \leq r, c \leq 2000$). Each of the next r lines contains a string of c characters. The j-th character in the i-th line is `#` if cell (i, j) is occupied; otherwise, it is `.` (dot).

Output

Output the number of ways to place the control towers to satisfy the requirements above.

Examples

standard input	standard output
3 4 .#.# #... .###	10
4 6 ###### #.#.#. .#.#.# ######	0
1 10	5040
1 10 ##########	0

Note

Explanation for the sample input/output #1

Figure 1 illustrates all 10 possible ways to place the control towers, where the cells numbered 1, 2, 3, and 4 represent control towers 1, 2, 3, and 4 respectively.

Explanation for the sample input/output #2

It is impossible for any control tower in row 2 to be in the same column as any control tower in row 3. Since there are not enough empty cells to place all four control towers in the same row, there is no way to place the control towers to satisfy the requirements above.

1	#	2	#
#	4	3	.
.	#	#	#
.	#	4	#
#	1	3	2
.	#	#	#

1	#	2	#
#	.	3	4
.	#	#	#
3	#	2	#
#	.	1	.
4	#	#	#

.	#	1	#
#	3	2	4
.	#	#	#
.	#	4	#
#	2	3	1
.	#	#	#

.	#	1	#
#	4	2	3
.	#	#	#
4	#	3	#
#	.	2	1
.	#	#	#

4	#	3	#
#	1	2	.
.	#	#	#
2	#	3	#
#	.	4	.
1	#	#	#

Figure 1: All 10 possible ways to place the control towers.

Explanation for the sample input/output #3

All $10 \times 9 \times 8 \times 7 = 5040$ ways of control tower positions satisfy the requirements above.

Explanation for the sample input/output #4

There is no empty cell to place any control tower.

Problem B. Three-Dimensional Embedding

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

An embedding of a graph in a space is a way of placing each vertex at a distinct point in that space and drawing each edge as a simple arc connecting its two vertices, so that no two arcs intersect except at a shared vertex. In this problem, we focus on embeddings in a three-dimensional space under certain conditions.

You are given a simple undirected graph with n vertices and m edges, which means there is at most one edge connecting any pair of vertices and each edge connects different vertices. The vertices are numbered from 1 to n, and the edges are numbered from 1 to m. Edge j connects the two distinct vertices v_j and w_j. Each vertex is incident to at most five edges.

Find an embedding of the graph such that all of the following conditions are satisfied.

- Each vertex i is embedded as a point $(x_i, y_i, 0)$ in the space. The coordinates x_i and y_i must be integers between 0 and 400, inclusive. All points must have distinct coordinates.
- Each edge j is embedded as a polyline (a connected series of line segments) with the embedded points for vertices v_j and w_j as its endpoints. Each segment of the polyline must be parallel to the x-, y-, or z-axis. Each node of the polyline must have integer coordinates between 0 and 400, inclusive. Each polyline must have no more than 30 nodes, counting its endpoints.
- Polylines must not have self-intersections. Distinct polylines must not share any point, except when they correspond to edges incident to the same vertex. In that case, they may share only that single endpoint.

Input

The first line of input contains two integers n and m ($2 \leq n \leq 1600$, $1 \leq m \leq 4000$). The j-th of the following m lines contains two integers v_j and w_j ($1 \leq v_j < w_j \leq n$).

The input guarantees that each vertex is incident to at most five edges. Further, there are no parallel edges; that is, if $j \neq j'$, $(v_j, w_j) \neq (v_{j'}, w_{j'})$ holds.

Output

First, output n lines. The i-th of these lines should contain two integers x_i and y_i, representing the coordinates where vertex i is embedded. Then, output m lines, where the j-th line represents the polyline corresponding to edge j, using the following format:

$$k\ x'_1\ y'_1\ z'_1\ \cdots\ x'_k\ y'_k\ z'_k$$

Here, k is the number of nodes, which must be between 2 and 30, inclusive. The points $(x'_1, y'_1, z'_1), \ldots, (x'_k, y'_k, z'_k)$ are the nodes of the polyline. The first point (x'_1, y'_1, z'_1) must be $(x_{v_j}, y_{v_j}, 0)$, and the last point (x'_k, y'_k, z'_k) must be $(x_{w_j}, y_{w_j}, 0)$. Each pair of consecutive points is connected by a segment to form the polyline. Each segment must have a positive length. Two consecutive segments may have the same orientation; for example, both can be parallel to the x-axis.

The embedding that you output must satisfy all of the conditions mentioned above.

Under the given input constraints, it can be shown that there exists at least one valid output. If there are multiple outputs, any one of them will be accepted.

Example

standard input	standard output
3 3 1 2 1 3 2 3	0 0 400 0 0 399 3 0 0 0 100 0 0 400 0 0 4 0 0 0 0 0 200 0 399 200 0 399 0 3 400 0 0 400 399 0 0 399 0

Note

Notes on special judging:

You are provided with a command-line tool for local testing. You can download the file from the contest materials. The tool has comments at the top to explain its use.

Explanation for the sample input/output #1

Figure 2 illustrates the embedding represented by the sample output.

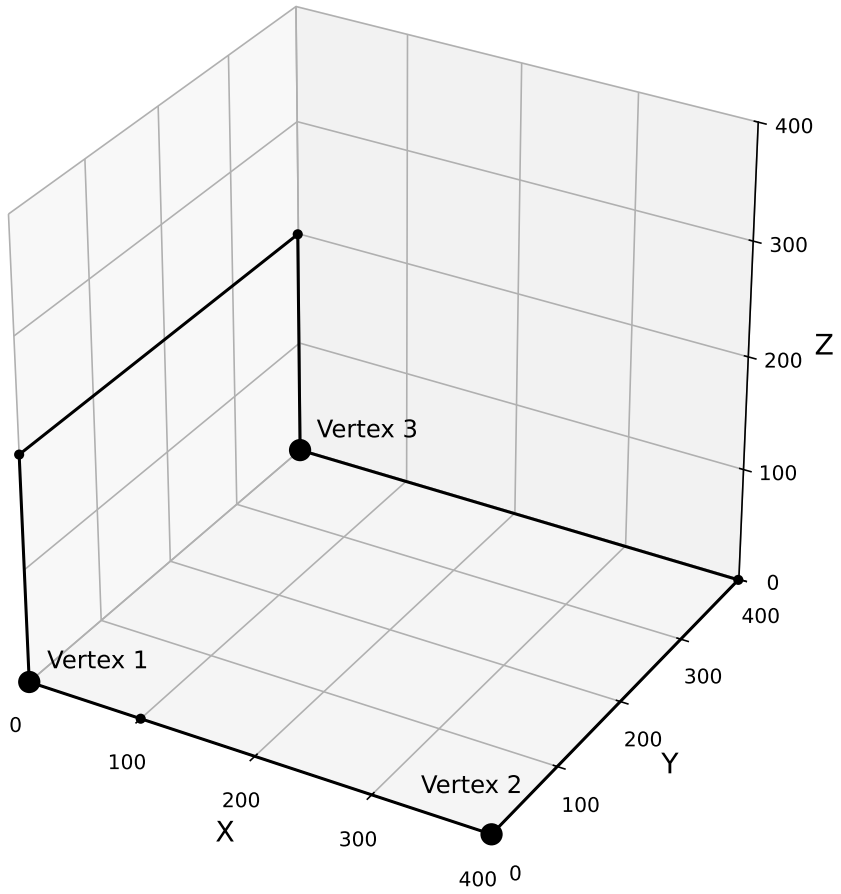


Figure 2: Illustration of the embedding.

Problem C. Cactus Connectivity

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are interested in the topic of graphs and their properties. In this problem, we assume that all graphs are simple undirected graphs, meaning there is at most one edge connecting any pair of vertices and each edge connects different vertices.

A simple cycle of a graph is a sequence of three or more **distinct** vertices $(v_1, v_2, \ldots, v_c)$ such that there exists an edge connecting vertices v_i and v_{i+1} for each $1 \leq i < c$ and there also exists an edge connecting vertices v_1 and v_c. For a simple cycle $(v_1, v_2, \ldots, v_c)$, its edge set is defined as $\{(v_1, v_2), (v_2, v_3), \ldots, (v_{c-1}, v_c), (v_c, v_1)\}$, where (v_i, v_j) represents an edge connecting vertices v_i and v_j. Two simple cycles are the same if they share the same edge set.

A graph is a cactus if any two distinct simple cycles share at most one common vertex. For example, Figure 3 illustrates three graphs. Graph X is a cactus since the two simple cycles $(1, 2, 3)$ and $(3, 4, 5)$ only share vertex 3 as the common vertex. Note that the sequence of vertices $(1, 2, 3, 4, 5, 3)$ does not form a simple cycle since vertex 3 appears twice. Also, the simple cycle $(2, 3, 1)$ is the same simple cycle as the simple cycle $(1, 2, 3)$. Meanwhile, graph Y is not a cactus since the two simple cycles $(1, 2, 4)$ and $(2, 3, 4)$ share vertices 2 and 4 as the common vertices. Graph Z is a cactus since there is only one simple cycle.

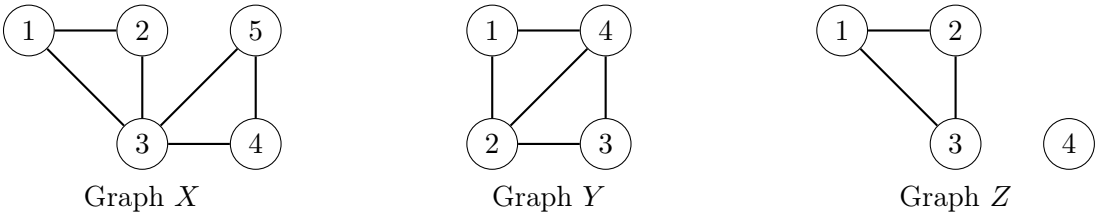


Figure 3: Graphs X and Z are cactii. Graph Y is not a cactus.

A graph is connected if there is a path connecting every pair of vertices. In particular, a graph with one vertex is connected.

For any positive integer k, a graph is k-edge-connected if, for any set of fewer than k edges, removing those edges leaves the graph connected. In particular, all connected graphs are 1-edge-connected. For example, graph Y in Figure 3 is 1-edge-connected and 2-edge-connected, but not 3-edge-connected, since removing both edges incident to vertex 1 does not leave the graph connected.

A graph H is a supergraph of a graph G if H can be obtained by adding zero or more vertices and edges to G without removing any vertices and edges from G. Note that two graphs are the same if they have the same set of vertices and the same set of edges. For example, in Figure 3 above, graph X is a supergraph of graph Z, while graph Y is not a supergraph of graph Z since it is missing the edge connecting vertices 1 and 3.

The connectivity value of a graph G is the smallest positive integer k such that, for any k-edge-connected graph H that is a supergraph of G, removing all the edges of G from H keeps H connected. For example, in Figure 3 above, graph X is a 2-edge-connected supergraph of graph Z, and removing all the edges of Z from X leaves vertices 1 and 2 disconnected from the rest, as illustrated by Figure 4 below. Therefore, the connectivity value of Z is more than 2. It can be shown that the connectivity value of Z is 3.

You are given a cactus with n vertices and m edges, where the vertices are numbered from 1 to n and the edges are 1 to m. Edge i connects vertices u_i and v_i. You are required to compute the connectivity value of the cactus.

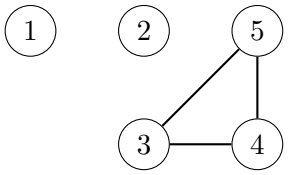


Figure 4: Removing the edges of graph Z from graph X

Input

The first line of input contains two integers n and m ($1 \leq n \leq 100\,000; 0 \leq m \leq 200\,000$). The i-th of the next m lines contains two integers u_i and v_i ($1 \leq u_i < v_i \leq n$). The input represents a cactus with at most one edge connecting any pair of vertices.

Output

Output the connectivity value of the given graph.

Examples

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

Note

Explanation for the sample input/output #1

This corresponds to graph Z from the problem description.

Explanation for the sample input/output #2

Any 1-edge-connected supergraph is a connected graph. Since the given graph has no edges, removing all the edges of the given graph from a connected graph keeps the graph connected.

Problem D. Tower of Hanoi

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

While visiting Hanoi to compete in The ICPC Asia Pacific Championship last year, you learned about the famous Tower of Hanoi problem. In the problem, there are three rods and several disks of distinct radii, which can slide onto any rod. The rods are numbered from 1 to 3. At any point in time, each disk must be stacked on one of the rods, and the disks stacked on each rod must be arranged in increasing order of radius from top to bottom. In one step, you can move the disk on top of one rod to the top of another rod, provided this move does not violate the restriction above. The goal is to move all the disks to rod 1 in the minimum number of steps.

You are solving an extension of this famous problem. You have a sequence of n integers $p_1, p_2, \ldots, p_n$, the initial values of which are given to you.

You are also given q operations. Each operation is either of the following:

Change operation: Two integers x and y are given. This operation requires you to change the value of p_x to y.

Solve operation: Two integers l and r are given. This operation requires you to solve the Tower of Hanoi problem with $r - l + 1$ disks of radii $l, l + 1, \ldots, r$, where the disk of radius i is initially stacked on rod p_i, for each $l \le i \le r$. The order of the disks initially stacked on each rod satisfies the restriction explained earlier. You need to find the minimum number of steps to move all disks to rod 1 modulo 998 244 353.

Your task is to perform all the given operations sequentially.

Input

The first line of input contains two integers n and q ($1 \le n \le 100\,000; 1 \le q \le 100\,000$). The second line contains n integers representing the initial values of $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le 3$). The next q lines represent the operations in the order they are to be performed. Each line is in one of the following formats:

1. "`c` x y" ($1 \le x \le n; 1 \le y \le 3$) to apply a Change operation for the specified integers x and y.
2. "`s` l r" ($1 \le l \le r \le n$) to apply a Solve operation for the specified integers l and r.

The input contains at least one Solve operation.

Output

For each Solve operation, in order, output the minimum number of steps to solve the Tower of Hanoi problem with disks of radii $l, l + 1, \ldots, r$ modulo 998 244 353.

Example

standard input	standard output
4 4 2 3 1 3 s 2 4 s 1 3 c 3 3 s 2 4	6 2 7

Note

Explanation for the sample input/output #1

The first operation requires you to solve the Tower of Hanoi problem with disks of radii 2, 3, and 4 initially stacked on rods 3, 1, and 3 respectively. All disks can be moved to rod 1 in 6 steps as illustrated by Figure 5.

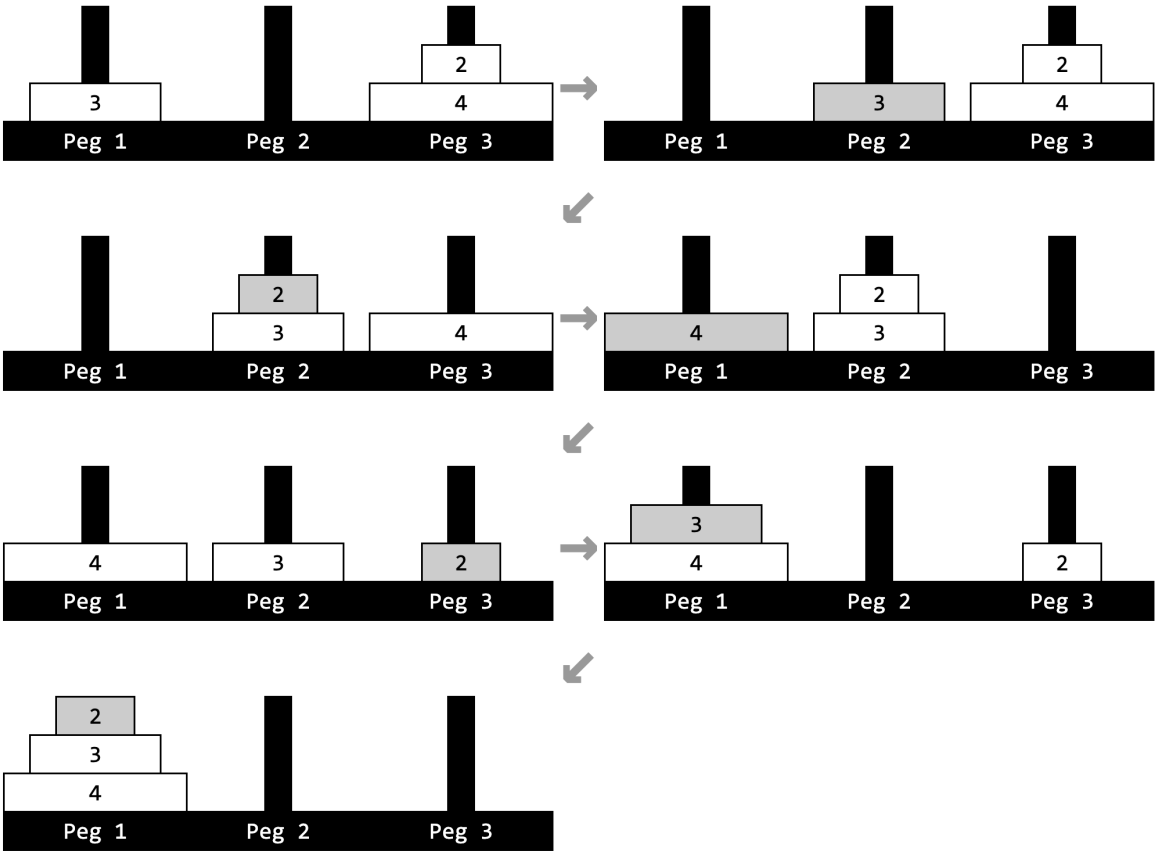


Figure 5: 6 steps to move all disks to rod 1. The shaded rod represents the rod moved on the last step.

The second operation requires you to solve the Tower of Hanoi problem with disks of radii 1, 2, and 3 initially stacked on rods 2, 3, and 1 respectively.

The fourth operation requires you to solve the Tower of Hanoi problem with disks of radii 2, 3, and 4 initially stacked on rods 3, 3, and 3 respectively.

Problem E. Minus Operator

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

The minus operator on binary values a and b is defined by $(a - b) = 1$ if $a = 1$ and $b = 0$; otherwise, $(a - b) = 0$. Also, the syntax of an expression is defined as follows. Here, `x`, parentheses, and the minus are terminal symbols, and E is the start symbol.

$$E ::= \texttt{x} \mid (E - E)$$

The judge program possesses an expression adhering to E. The expression is hidden from you. At the start, you are only provided with the number of terminal symbols `x` in the expression, denoted by n.

Your task is to guess the expression by making a limited number of queries.

In a single query, you specify a binary string S of length n. The judge program then temporarily replaces each occurrence of the i-th terminal symbol `x` from the left with S_i, for $i = 1, \ldots, n$, and evaluates the replaced expression based on the definition of the minus operator. After that, the judge program returns the evaluated value to you, which is either 0 or 1.

Interaction Protocol

The first line of input contains an integer n ($3 \leq n \leq 200$). After reading the integer, your program can start making queries. For each query, your program should write a line of the form "`query` S". Here, S is a binary string of length n. Each character of S must be either `0` or `1`. In response, an input line containing the evaluated value becomes available.

Your program can make up to **500 queries.** If your program makes more than 500 queries, it will be judged as "Wrong Answer."

When your program identifies the expression that the judge program possesses, it should write a line of the form "`answer` T", where T is the expression. After that, the interaction stops and your program should terminate.

Under the constraints above, it can be shown that the expression can be uniquely identified.

Examples

standard input	standard output
3 0	 query 111 answer ((x-x)-x)
4 0 1	 query 0000 query 1001 answer ((x-x)-(x-x))

Note

Notes on interactive judging:

- The evaluation is non-adversarial, meaning that the expression is chosen in advance rather than in response to your queries.
- Do not forget to flush output buffers after writing. See the "Judging Details" document for details.
- You are provided with a command-line tool for local testing, together with input files corresponding to the sample interactions. You can download these files from the contest materials. The tool has comments at the top to explain its use.

Explanation for the sample interaction #1

When $n = 3$, there are only two possible expressions: `((x-x)-x)` or `(x-(x-x))`. For a query $s =$ `111`, the evaluated values for these expressions are,

- $((1 - 1) - 1) = (0 - 1) = 0$, and
- $(1 - (1 - 1)) = (1 - 0) = 1$.

In this example, the judge program returns 0, indicating that the first expression is the correct one.

Problem F. Hold the Star

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are playing a computer game with n rooms, m characters, and one star. The rooms are arranged from left to right and numbered from 1 to n in that order. The characters are numbered from 1 to m. At any time, each character is in one of the rooms and the star is either in one of the rooms or held by one of the characters. The objective of the game is for the star to be held by character m.

You can play the game by performing several actions. Each action costs a certain amount of staracips (the unit of currency in the game), possibly zero. In each action, you choose a character x (let room y be the room the character is currently in) and command the character to do either of the following:

- Move to one of the adjacent rooms ($y-1$ or $y+1$), if such a room exists. If character x is holding the star, then the character continues to hold the star. This action costs s_x staracips. The values of $s_1, s_2, \ldots, s_m$ are given.
- Pick the star up and hold it, if the star is currently in room y and is not held by any character. This action costs 0 staracips.
- Put the star down and release it, if the star is currently held by character x. The star then falls to room y. This action costs 0 staracips.

The game contains q levels, numbered from 1 to q. In all levels, each character i is initially in room r_i and character m must hold the star to win the level. The only difference between the levels is that, in each level j, the star is initially in room l_j.

For each level, you want to compute the minimum total staracips you have to spend to win the level. Note that you don't have to minimize the number of actions.

Input

The first line of input contains three integers n, m, and q ($1 \le n \le 10^9; 1 \le m \le 100\,000; 1 \le q \le 100\,000$). The i-th of the next m lines contains two integers r_i and s_i ($1 \le r_i \le n; 1 \le s_i \le 10^9$). The j-th of the next q lines contains an integer l_j ($1 \le l_j \le n$).

Output

For each level in order, output the minimum total staracips you have to spend to win the level.

Example

standard input	standard output
6 5 4 1 7 3 2 2 3 5 3 2 5 1 2 5 6	5 0 8 14

Note

Explanation for the sample input/output #1

You can win the first level by spending 5 staracips by doing the following actions:

1. Character 5 moves from room 2 to room 1. This action costs 5 staracips.
2. Character 5 picks the star up.

You can win the second level by spending 0 staracips by doing the following actions:

1. Character 5 picks the star up.

You can win the third level by spending 8 staracips by doing the following actions:

1. Character 4 picks the star up.
2. Character 4 moves from room 5 to room 4. This action costs 3 staracips.
3. Character 4 moves from room 4 to room 3. This action costs 3 staracips.
4. Character 4 puts the star down.
5. Character 2 picks the star up.
6. Character 2 moves from room 3 to room 2. This action costs 2 staracips.
7. Character 2 puts the star down.
8. Character 5 picks the star up.

You can win the fourth level by spending 14 staracips by doing the following actions:

1. Character 2 moves from room 3 to room 4, then to room 5, then to room 6. These actions cost $3 \times 2 = 6$ staracips in total.
2. Character 2 picks the star up.
3. Character 2 moves from room 6 to room 5, then to room 4, then to room 3, then to room 2. These actions cost $4 \times 2 = 8$ staracips in total.
4. Character 2 puts the star down.
5. Character 5 picks the star up.

Problem G. Corrupted File

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

The WannaLaugh malware is a new computer malware that is spreading on the internet. If a computer is infected by this malware, then the malware will corrupt all files in the computer. A file in a computer contains zero or more bits. The malware corrupts a file by performing zero or more operations. In one operation, the malware randomly picks two consecutive bits and replaces them with a single bit. The new bit is 1 if both of the replaced bits are 1, or 0 otherwise.

For example, the malware might corrupt a file with bits 11011011 as follows:

1. The malware picks the first and second bits: **11**011011 $\rightarrow$ **1**011011.
2. The malware picks the second and third bits: 1**01**1011 $\rightarrow$ 1**0**1011.
3. The malware picks the third and fourth bits: 10**10**11 $\rightarrow$ 10**0**11.

Alternatively, the malware might first pick the third and fourth bits: 11**01**1011 $\rightarrow$ 11**0**1011.

At the start of the day, you have a file containing n bits, denoted by B. You spend the day surfing the internet, including checking on your favorite programming contest website, just like many ICPC contestants would do. At the end of the day, the same file contains m bits, denoted by C. You want to determine whether this file could have been corrupted by the WannaLaugh malware, or if it must have changed for other reasons.

Input

The first line of input contains one integer t ($1 \le t \le 10\,000$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of input contains two integers n and m ($1 \le m \le n \le 100\,000$). The second line contains a string with n characters, each is either `0` or `1`, representing the bits B. The third line contains a string with m characters, each is either `0` or `1`, representing the bits C.

The sum of n across all test cases in one input file does not exceed 100 000.

Output

For each test case, output `yes` if the file with bits B could have been corrupted by the WannaLaugh malware into bits C, or `no` otherwise.

Example

standard input	standard output
3 8 5 11011011 10011 3 3 101 101 3 2 101 00	yes yes no

Note

Explanation for the sample input/output #1

The first test case corresponds to the example from the problem description.

For the second test case, it is possible that the malware performs zero operations.

For the third test case, it is impossible for the malware to corrupt a file with bits 101 so that it has bits 00.

Problem H. Secret Lilies and Roses

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

There are n flowers arranged in a line from left to right, which are numbered from 1 to n in that order. Each flower is either a lily or a rose. For an integer j between 0 and n, inclusive, let l_j denote the number of lilies among the leftmost j flowers, and let r_j denote the number of roses among the rightmost $n - j$ flowers.

Initially, only the number of flowers n is provided to you. The types of the flowers are hidden. You can obtain information on the flowers by making queries. In one query, you can perform one of the following.

Type query: Specify an integer i between 1 and n, inclusive. You will then receive the type of flower i.

Multiply query: Specify an integer j between 0 and n, inclusive. You will then receive the value of $l_j \times r_j$.

Your task is to find an integer k between 0 and n, inclusive, for which $l_k = r_k$ by making a limited number of queries. You can assume that at least one such integer exists for the arrangement of the flower types. Note that you do not need to identify the type of each flower.

Interaction Protocol

The first line of input contains one integer t ($1 \le t \le 100$) representing the number of test cases. After that, t test cases follow. Each of them is presented as follows.

The first line of input for each test case contains an integer n ($1 \le n \le 100$). After reading it, your program can start making queries. For each query, your program should write a line containing one of the following:

- "`type` i" to perform a Type query by specifying an integer i ($1 \le i \le n$). In response, an input line containing a string either `lily` or `rose` becomes available, indicating the type of flower i.
- "`multi` j" to perform a Multiply query by specifying an integer j ($0 \le j \le n$). In response, an input line containing the integer $l_j \times r_j$ becomes available.

Your program is allowed to make up to **10 queries** for each test case. This means the total number of Type queries and Multiply queries combined must not exceed 10. If your program makes more than 10 queries, it will be judged as "Wrong Answer".

When your program identifies an integer k such that $l_k = r_k$, it should write a line of the form "`answer` k" ($0 \le k \le n$). Note that providing the answer does not count as a query.

The input guarantees that at least one correct output exists. If there are multiple correct outputs, any one of them will be accepted.

After writing the answer line, your program should start processing the next test case. When all t test cases have been processed, the interaction stops and your program should terminate.

Example

standard input	standard output
2 9 lily rose 6 3 3 0 rose	 type 8 type 1 multi 6 multi 3 answer 5 multi 3 type 3 answer 3

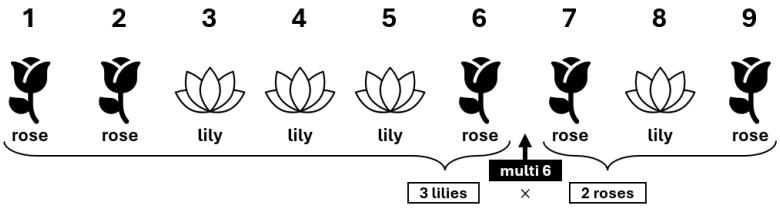
Note

Notes on interactive judging:

- The evaluation is non-adversarial, meaning that the types of the flowers are chosen in advance rather than in response to your queries.
- Do not forget to flush output buffers after writing. See the "Judging Details" document for details.
- You are provided with a command-line tool for local testing, together with input files corresponding to the sample interactions. You can download these files from the contest materials. The tool has comments at the top to explain its use.

Explanation for the sample interaction #1

For the first test case, nine flowers are arranged as shown in the following figure. In response to the third query, $l_6 \times r_6 = 3 \times 2 = 6$ is returned, and in response to the fourth query, $l_3 \times r_3 = 1 \times 3 = 3$ is returned. Since $l_5 = r_5 = 3$, $k = 5$ is a correct output.



For the second test case, all three flowers are assumed to be roses.

Problem I. Squares on Grid Lines

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

You have a square of side length n on a 2D plane, partitioned into a grid of 1×1 square cells, totaling n^2 cells.

Your task is to answer q queries, numbered from 1 to q, described below. In query i, you are given a real number s_i, and you must count the number of ways to place four points on the plane such that

- each point lies on the boundary of a cell (not necessarily the same), and
- the four points form the vertices of a square with area s_i.

Here, the edges of the square formed by these points do **not** need to be parallel to the edges of the cells. If there are infinitely many valid placements, you must report that as your answer.

Two placements are considered different if there exists a point that appears in one placement but not in the other.

Input

The first line of input contains two integers n and q ($1 \le n \le 2000$, $1 \le q \le 100\,000$). The i-th of the next q lines contains a real number s_i ($0.01 \le s_i \le n^2$), given with exactly two digits after the decimal point.

Output

Output q lines. The i-th line should contain the number of valid placements for query i. If infinitely many exist, output `-1` instead.

Examples

standard input	standard output
3 4 6.90 0.26 2.65 1.00	2 4 10 -1
1 5 0.49 0.50 0.51 0.99 1.00	0 1 2 2 1

Note

Explanation for the sample input/output #1

For queries 1 and 2, the valid placements are illustrated in Figure 6. The top two placements correspond to query 1, and the bottom four correspond to query 2. In each placement, the shaded region represents a square formed by the points.

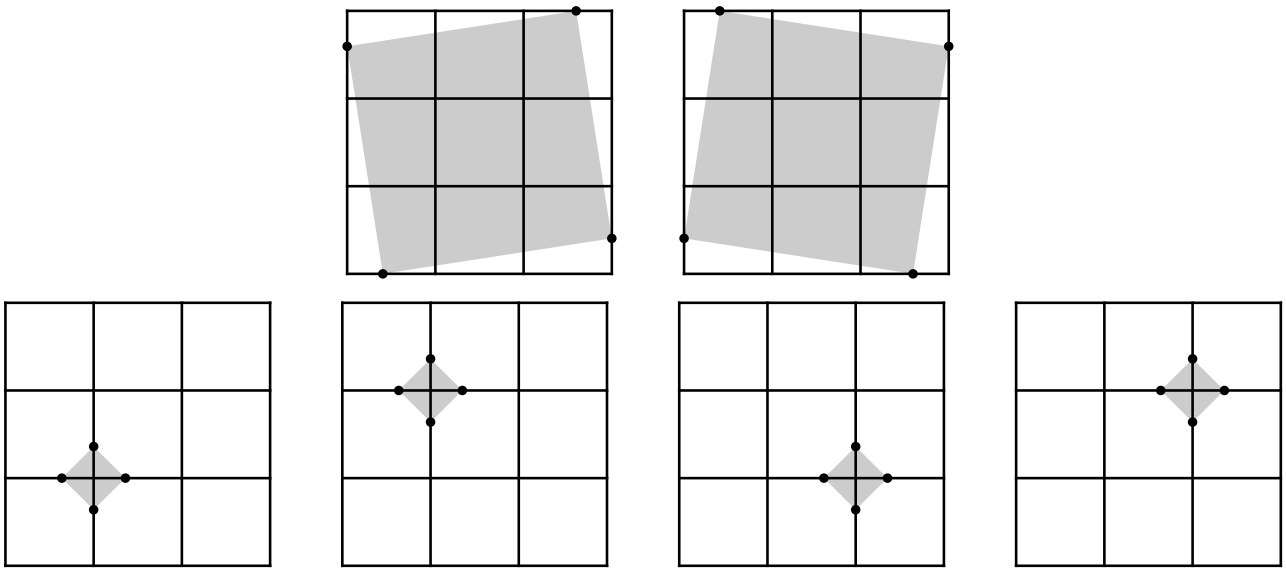


Figure 6: Illustrations of Sample Input #1.

Problem J. Gathering Sharks

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are the leader of a swarm of n sharks living in a one-dimensional ocean. The sharks are positioned from left to right, with each adjacent pair separated by a distance of one unit.

As the leader, you want all the sharks to gather at a common point to form a single group. Initially, no two sharks belong to the same group; for each $i = 1, \ldots, n$, the i-th shark from the left forms its own group, uniquely numbered a_i, consisting of only itself.

To achieve your goal, you can command the sharks to perform the following actions $n - 1$ times.

1. You shout out an integer b that meets both conditions:
   - There exists a group numbered b.
   - There exists at least one group numbered strictly smaller than b.
2. Afterward, letting c be the **largest existing** group number strictly smaller than b, all the sharks in the group numbered b simultaneously move to the position of the group numbered c, and the two groups merge.
3. The merged group is numbered b, and the group numbered c ceases to exist.

All sharks move at a constant speed of one unit distance per unit time. Commands must be executed sequentially, with no overlap in execution. Once a command is completed, the next one can begin immediately.

Compute the minimum time required for all the sharks to gather at a common point by commanding the sharks $n - 1$ times optimally.

Input

The first line of input contains an integer n ($2 \le n \le 500$). The second line contains n pairwise distinct integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$).

Output

Output the minimum time required for all the sharks to gather at a common point.

Examples

standard input	standard output
4 3 2 4 1	4
9 1 2 4 5 7 8 3 6 9	17

Note

Explanation for the sample input/output #1

You can command the sharks to perform the following actions:

1. You shout out 3. The leftmost shark moves to the position of the second-leftmost shark, and they form a group numbered 3. This takes 1 unit of time.

2. You shout out 4. The second rightmost shark moves to the position of the group numbered 3, and they form a group numbered 4. This takes 1 unit of time.
3. You shout out 4. The sharks in the group numbered 4 move to the rightmost position, forming a group of four sharks. This takes 2 units of time.

The total time is $1 + 1 + 2 = 4$. It can be shown that 4 units of time is optimal.

Problem K. Book Sorting

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 megabytes

You have n books arranged from left to right on a bookshelf. These books are uniquely labeled from 1 to n. The i-th book from the left is labeled p_i. You want to sort the books so that their labels are in ascending order from left to right.

In one step, you can perform one of the following actions:

- Choose two adjacent books and swap them.
- Choose one book and move it to the leftmost position.
- Choose one book and move it to the rightmost position.

Compute the minimum number of steps required to sort the books.

Input

The first line of input contains an integer n ($2 \le n \le 500\,000$). The second line contains n pairwise distinct integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$).

Output

Output the minimum number of steps to sort the books in ascending order from left to right by their labels.

Examples

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

Note

Explanation for the sample input/output #1

You can do the following three steps in order: swap the books labeled 2 and 1, swap the books labeled 4 and 3, and move the book labeled 6 to the rightmost position.

$$6\ 2\ 1\ 4\ 3\ 5 \to 6\ 1\ 2\ 4\ 3\ 5 \to 6\ 1\ 2\ 3\ 4\ 5 \to 1\ 2\ 3\ 4\ 5\ 6$$

It can be shown that two or fewer steps are insufficient to sort the books.

This page is intentionally left blank.

Problem L. Boarding Queue

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are on your way to compete in The 2025 ICPC Asia Pacific Championship. Unfortunately, you are in the process of the worst part of flying: waiting in the boarding queue.

You are in a queue with n travelers, numbered from 1 to n ordered from the front of the queue to the back of the queue.

The boarding area is represented by a grid of r rows and c columns, where the rows are numbered from 1 to r (top to bottom) and the columns are numbered from 1 to c (left to right). Each traveler occupies exactly one distinct cell in the grid. Two travelers are adjacent if the cells they are in share an edge. Traveler t and traveler $t-1$ are guaranteed to be adjacent, for any $2 \le t \le n$.

For example, Figure 7 illustrates a possible location of the travelers. In this example, traveler 1 is adjacent to travelers 2 and 10 but not adjacent to traveler 11.

11				
10	1	2		
9		3	4	
8	7	6	5	

Figure 7: Example location of the travelers.

At each boarding step, all of the following happen simultaneously:

- The frontmost traveler in the queue, say traveler t, boards the aircraft and leaves the boarding area.
- For each t' $(t+1 \le t' \le n)$, traveler t' takes the cell that traveler $t'-1$ was occupying immediately before the step.

For example, Figure 8 illustrates the locations of the travelers after the first three boarding steps from the initial location above.

11	2	3		
10		4	5	
9	8	7	6	

	3	4		
11		5	6	
10	9	8	7	

	4	5		
		6	7	
11	10	9	8	

Figure 8: Location of the travelers after 1, 2, and 3 boarding steps respectively.

You are traveler p (that is, there are $p-1$ travelers in front of you). You know that your team coach is somewhere in the queue, but you do not know where. Assuming your team coach is equally likely to be any of travelers 1 to n (except p), you want to calculate the probability that you will be adjacent to your team coach at some point before you board the aircraft. Formally, you will be adjacent to traveler q at

some point before you board the aircraft if there exists an integer s ($0 \leq s < p$) such that traveler p and traveler q are adjacent after s boarding steps.

Input

The first line of input contains four integers r, c, n, and p ($1 \leq r, c \leq 1000; 2 \leq n \leq r \times c; 1 \leq p \leq n$). Each of the next r lines contains c integers. The j-th integer in the i-th line denotes $G_{i,j}$ ($0 \leq G_{i,j} \leq n$), where a non-zero value of $G_{i,j}$ means that traveler $G_{i,j}$ initially occupies the cell at row i and column j of the boarding area, while a zero value of $G_{i,j}$ means that no traveler occupies the cell. Across all pairs (i, j), each of the integers 1 to n appears exactly once in $G_{i,j}$. The input guarantees that traveler t and traveler $t-1$ are adjacent, for all $2 \leq t \leq n$.

Output

Output a fraction in the x/y format indicating the probability that you will be adjacent to your team coach at some point before you board the aircraft. The value of y must be equal to $n-1$. Note that there must not be spaces between the integers and the `/` delimiter.

Examples

standard input	standard output
4 5 11 2 11 0 0 0 0 10 1 2 0 0 9 0 3 4 0 8 7 6 5 0	3/10
1 7 7 6 1 2 3 4 5 6 7	2/6

Note

Explanation for the sample input/output #1

This sample corresponds to the example given in the problem description above. You will be adjacent to your team coach at some point before you board the aircraft if your team coach is either traveler 1, 3, or 11.

Explanation for the sample input/output #2

You will be adjacent to your team coach at some point before you board the aircraft if your team coach is either traveler 5 or 7. Note that the output `1/3` is **not** accepted.

Problem M. Can You Reach There?

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 megabytes

You are given n distinct marked points on a 2D plane, numbered from 1 to n. Marked point i has coordinates (x_i, y_i).

In this problem, you are given q scenarios, numbered from 1 to q. In each scenario k, four integers a_k, b_k, c_k, and d_k are given, indicating that you initially stand at (a_k, b_k) and aim to reach (c_k, d_k) by repeating the steps described below any number of times.

In a single step, you choose two marked points P and Q, which may be identical. Let S denote the point where you are currently standing, and define a point T by

$$\overrightarrow{PT} = \overrightarrow{SQ}.$$

In other words, T is chosen so that the vector from P to T has the same direction and length as the vector from S to Q. You may then move to any point on the segment ST, including the point T itself, and you will stand at that new point.

For each scenario, determine whether the objective can be achieved using the described steps. Note that all scenarios are independent of each other.

Input

The first line of input contains two integers n and q ($1 \le n \le 100\,000$, $1 \le q \le 100\,000$). The i-th of the next n lines contains two integers x_i and y_i ($0 \le x_i, y_i \le 10^9$). The input guarantees that no two marked points have the same coordinates.

The next q lines represent the scenarios. The k-th of these lines contains four integers a_k, b_k, c_k, and d_k ($0 \le a_k, b_k, c_k, d_k \le 10^9$; $(a_k, b_k) \ne (c_k, d_k)$).

Output

Output q lines. The k-th line should contain `yes` if the objective of scenario k is achievable, or `no` otherwise.

Example

standard input	standard output
2 4 10 0 0 10 3 4 6 5 4 0 7 0 4 0 16 0 123 456 789 0	yes yes yes no

Note

Explanation for the sample input/output #1

There are two marked points $(10, 0)$ and $(0, 10)$. In scenario 1, starting from the point $S = (a_1, b_1) = (3, 4)$, the objective is achieved as follows:

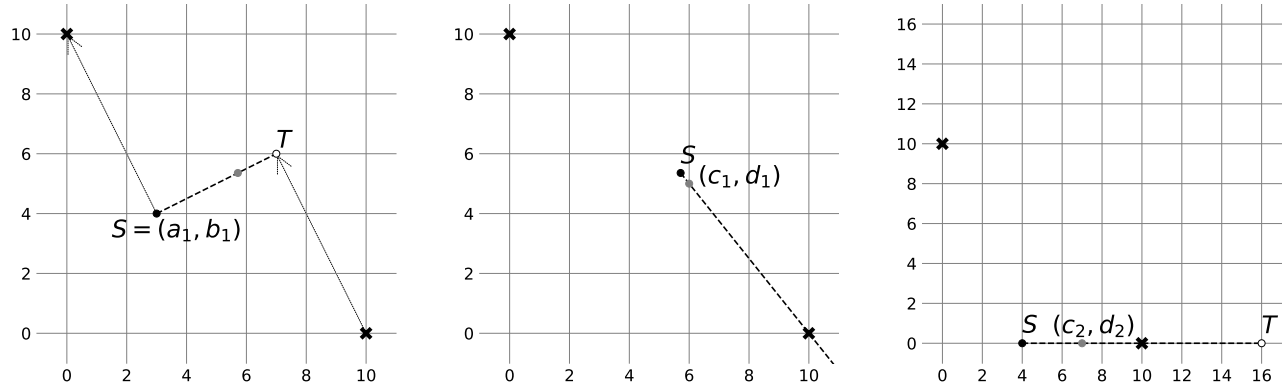
- In the first step, choose $(10, 0)$ as P and $(0, 10)$ as Q. A point T is determined with coordinates $(7, 6)$. Move to a point $(40/7, 75/14)$, which lies on the segment ST. (Figure 9 (a))

- In the next step, choose $(10, 0)$ as both P and Q. A point T is determined with coordinates $(100/7, -75/14)$. From there, you can reach the point $(c_1, d_1) = (6, 5)$. (Figure 9 (b))

In scenario 2, you start from $S = (a_2, b_2) = (4, 0)$. Choose $(10, 0)$ as both P and Q. A point T is determined with coordinates $(16, 0)$. The point $(c_2, d_2) = (7, 0)$ is on the segment ST, allowing you to reach there in a single step. (Figure 9 (c))

In scenario 3, the objective is similarly achievable.

In scenario 4, it can be shown that reaching the point $(c_4, d_4) = (789, 0)$ from $(a_4, b_4) = (123, 456)$ is impossible.



(a) (b) (c)

Figure 9: Illustrations of the scenarios in Sample Input #1.