Problem A. Stone Enthusiast

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	2 seconds
Memory limit:	512 megabytes

Once there was a boy who loved stones very much, so he could spend whole years studying them. Over these long years, he realized that each stone has a type, denoted by a letter of the Latin alphabet. The boy believes that everyone should know about stones, so he works in a museum where he showcases them.

In the museum, there are n exhibits of stones arranged in the order specified by the string s, where the stone at the i-th stand is of type s_i. m visitors have come to the boy, and each visitor wants to see only certain stones, the types of which are specified by the string t. The boy will guide the visitor through the stands in order from 1 to n, and at each stand, he can choose to either show the stone at that stand to the visitor or skip it. The boy must fulfill the visitor's wish, but he wants to make the stone viewing as challenging as possible.

Let's say the boy shows the stands with numbers $1 \le i_1 < i_2 < \cdots < i_k \le n$, then these stones, in the order shown, must form the string t. In other words, it must hold that $k = |t|$, and for all $1 \le j \le k$, it must hold that $s_{i_j} = t_j$. The difficulty of the viewing is defined as the minimum distance between adjacent shown stands, that is, $\min(i_2 - i_1, i_3 - i_2, \ldots, i_k - i_{k-1})$. If it is impossible to choose such k stands, then the viewing is considered impossible; otherwise, the boy will choose the viewing with the maximum possible difficulty.

Determine for each visitor whether their viewing is possible, and if so, what the maximum difficulty of viewing the stones could be.

Input

The first line contains two integers n and m ($1 \le n, m \le 200\,000$) — the number of exhibits in the museum and the number of visitors.

The second line contains the string s of length n, consisting of lowercase Latin letters — the types of stones on the exhibits in the museum.

The following m lines describe the visitors. The i-th line contains the string t_i, consisting of lowercase Latin letters ($2 \le |t_i| \le 200\,000$) — the types of stones that visitor i wants to see.

It is guaranteed that the sum of all $|t_i|$ does not exceed $400\,000$.

Output

For each of the m visitors, output the maximum difficulty of the tour on a separate line, or -1 if the tour is impossible.

Examples

standard input	standard output
7 3 abacaba aa acb ba	6 2 5
12 2 openolympiad oli goal	4 -1
8 5 abbaabba bab baba bbbb aaaa abbaabba	2 2 1 1 1

Note

Consider the first example.

For the first visitor, it is optimal to choose the stands with numbers $[1, 7]$, then the difficulty of the viewing will be $7 - 1 = 6$.

For the second visitor, it is optimal to choose the stands with numbers $[1, 4, 6]$, then the difficulty of the viewing will be $\min(4 - 1, 6 - 4) = 2$.

For the third visitor, it is optimal to choose the stands with numbers $[2, 7]$, then the difficulty of the viewing will be $7 - 2 = 5$.

Scoring

The tests for this problem consist of seven groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups.

Let T denote the total length of all strings t.

Group	Points	Additional constraints			Required groups	Comment
		n	m	T		
0	0	–	–	–	–	Samples.
1	16	$n \le 20$	–	$T \le 1000$	0	
2	14	$n \le 500$	–	$T \le 500$	0	
3	12	–	–	–	–	s and t consist only of the letter ‘a’
4	16	–	$m = 1$	–	–	s and t consist only of letters ‘a’ and ‘b’
5	11	–	$m = 1$	–	4	–
6	17	–	–	–	3, 4	s and t consist only of letters ‘a’ and ‘b’
7	14	–	–	–	0 – 6	

Problem B. Distinctive Features

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	1 second
Memory limit:	512 megabytes

You are developing a system to assist consultants in smartphone stores.

All smartphones in the store are arranged in a row and numbered with integers from 1 to n in the order they are placed.

Each smartphone has certain **distinctive features**, such as durability or a large battery. There are a total of m different distinctive features, and they are numbered with integers from 1 to m.

The most common question that consultants have to answer is: how does this smartphone differ from those next to it? We formalize this question as follows:

Given a segment of smartphones numbered from l_i to r_i inclusive, and the number of a certain smartphone p_i in this segment ($l_i \le p_i \le r_i$), determine how many distinctive features are present in the given smartphone p_i but not in any other smartphone in the segment $[l_i, r_i]$.

To make the consultants' work easier, you have been tasked with developing a system capable of efficiently answering such queries.

Input

The first line contains three integers n, m, and g ($1 \le n, m \le 500\,000$, $0 \le g \le 9$) — the number of smartphones in the store, the number of different distinctive features, and the group number for this test.

Each of the following n lines describes the distinctive features of the next smartphone in the following format:

At the beginning of the line is an integer k_i ($0 \le k_i \le m$) — the number of distinctive features of the i-th smartphone. Then in the same line are k_i integers $1 \le a_{i,1} < \ldots < a_{i,k_i} \le m$ — the distinctive features of the i-th smartphone in increasing order.

The next line contains an integer q ($1 \le q \le 500\,000$) — the number of queries.

The following q lines describe the queries. In the i-th line, there are three integers l_i, r_i, and p_i ($1 \le l_i \le p_i \le r_i \le n$) — the parameters of the i-th query.

Let s denote the total number of distinctive features across all smartphones ($s = \sum_{i=1}^{n} k_i$). It is guaranteed that $n, m \le s \le 500\,000$.

Output

Output q integers — the number of distinctive features for each query.

Example

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

Note

Consider the example.

In the first query, the second smartphone has no distinctive features, so the answer to the query is zero.

In the second query, the segment consists of a single element, so all distinctive features of the fourth smartphone apply.

In the third query, the distinctive features of the third smartphone are $[1, 4]$, the fourth smartphone has $[2, 3, 4]$, and the fifth smartphone has $[1, 2]$. Among all the distinctive features of the fourth smartphone, only feature 3 is unique to it, so the answer to the query is 1.

Scoring

The tests for this problem consist of nine groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Group	Points	Additional Constraints	Required Groups	Comment
		n, m, q, s		
0	0	–	–	Samples.
1	10	$n, m, q \le 500$	0	
2	7	$q \le 5000, s \le 10\,000$	0	
3	13	$q, s \le 100\,000, m \le 500$	0	
4	19	–	–	$p_i = l_i$
5	7	–	–	$l_i = 1$
6	10	–	–	$l_i \le l_{i+1}, r_i \le r_{i+1}$
7	12	$q, s \le 100\,000$	0, 2, 3	–
8	7	$q, s \le 200\,000$	0, 2, 3, 7	
9	15	–	0 – 8	**Offline-testing.**

Problem C. Road Lighting

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	2 seconds
Memory limit:	512 megabytes

Berland is a country with a not very developed road system. There are a total of n cities in Berland and $n-1$ bidirectional roads, the i-th of which connects cities v_i and u_i. It is known that between every pair of cities, it is possible to reach one another using only these roads.

It is currently night in Berland, and some roads need to have their lighting turned on. There is a law, created for the purpose of saving electricity, that prohibits turning on the lighting simultaneously on two roads that share a common endpoint. The residents of Berland are interested in the maximum number of roads on which lighting can be turned on without violating the law, so they have turned to you for help.

Unfortunately, in Berland, some roads may be affected by a heavy snowstorm, which can disrupt the connectivity between some pairs of cities. Initially, no road is affected by a snowstorm. There are q queries of two types:

1. Change the weather on road e_i ($1 \le e_i \le n-1$): if there is currently no snowstorm on road e_i, a snowstorm begins, and vice versa.
2. It is required to turn on the lighting on the maximum number of roads such that both endpoints can be reached from city x_i, using only the roads that are not affected by a snowstorm. Of course, lighting can be turned on without violating the law, meaning there should not be a pair of roads with lighting turned on that exit from the same city. In other words, the original problem needs to be solved while considering only the cities reachable from x_i via roads that are not affected by a snowstorm.

Input

The first line contains a single integer g ($0 \le g \le 7$) — the group number for this test.

The second line contains a single integer n ($2 \le n \le 300\,000$) — the number of cities.

The next $n-1$ lines describe the roads. The i-th line contains two integers v_i and u_i ($1 \le v_i, u_i \le n$) — the numbers of the cities connected by the i-th road. It is guaranteed that between every pair of cities, it is possible to reach one another using only these roads.

The following line contains a single integer q ($1 \le q \le 300\,000$) — the number of queries.

In the next q lines, the queries are specified. The i-th line begins with an integer t_i ($1 \le t_i \le 2$).

- If $t_i = 1$, then this is a query of the first type, followed by a single integer e_i ($1 \le e_i \le n-1$) — the number of the road on which the weather changes.
- If $t_i = 2$, then this is a query of the second type, followed by a single integer x_i ($1 \le x_i \le n$). In this case, the original problem needs to be solved while considering only the cities reachable from x_i via roads that are not affected by a snowstorm.

Output

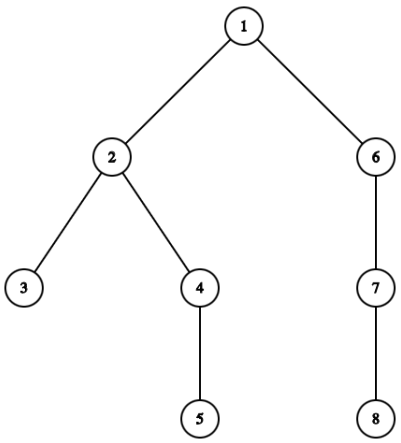
For each query of the second type, output the maximum number of roads connecting cities reachable from x_i on which lighting can be turned on without violating the law.

Example

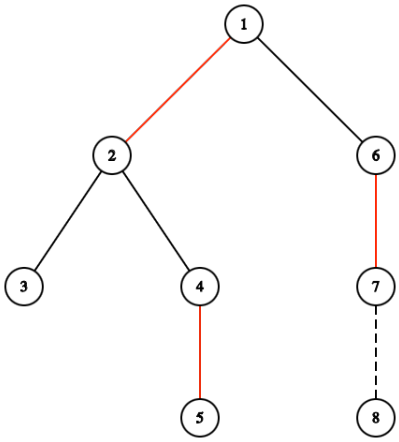
standard input	standard output
0 8 1 2 2 3 2 4 4 5 1 6 6 7 7 8 8 1 7 2 1 1 7 2 1 1 3 2 3 1 5 2 6	3 4 3 1

Note

In the test example, Berland initially has the following structure:

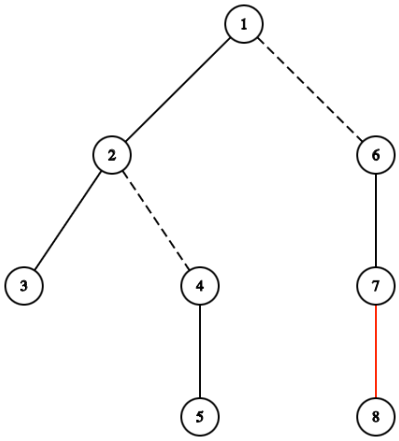


After the first query, a snowstorm begins on the road between cities 7 and 8. Then a query comes regarding city 1. In this case, we consider all cities except for city 8, as it is not reachable from city 1. Below is one of the optimal ways to turn on the lighting on the roads (the roads with lighting turned on are marked in red):



The next query indicates that the snowstorm on the road between cities 7 and 8 ends, meaning everything returns to its original state.

In the very last query, from vertex 6, only vertices 6, 7, and 8 are reachable, so lighting can be turned on at most on one road, for example:



Scoring

The tests for this problem consist of seven groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Group	Points	Additional Constraints		Required Groups	Comment
		n	q		
0	0	–	–	–	Samples.
1	14	$n \le 100$	$q \le 100$	0	
2	13	$n \le 100\,000$	$q \le 100\,000$	–	$v_i = i$, $u_i = i + 1$
3	10	$n \le 100\,000$	$q = 1$	–	
4	12	$n \le 100\,000$	$q \le 100\,000$	–	$v_i = i + 1$, $u_i = \lfloor \frac{v_i}{2} \rfloor$
5	19	$n \le 100\,000$	$q \le 100\,000$	3	The snowstorm cannot end
6	20	$n \le 100\,000$	$q \le 100\,000$	0 – 5	
7	12	–	–	0 – 6	**Offline-testing.**

Problem D. Repainting the Table

Time limit: 1.5 seconds
Memory limit: 512 megabytes

This problem can only be solved using C++ or Python3.

Zakhar loves to play chess. He has grown bored of the usual black-and-white chessboards, so he thought about how to diversify the game.

A chessboard is a square table with n rows and n columns. We number the rows with integers from 0 to $n-1$ from top to bottom, and the columns with integers from 0 to $n-1$ from left to right. Zakhar calls the board k-colored if a cell is painted in a color equal to the remainder of the division of the number of the diagonal going up-right from the cell by k. The diagonals are numbered with consecutive integers starting from zero, beginning at the top left corner of the table. In other words, the number of the diagonal to which the cell (i, j) belongs is equal to $(i + j) \mod k$.

	0	1	2	3	4
0	0	1	2	0	1
1	1	2	0	1	2
2	2	0	1	2	0
3	0	1	2	0	1
4	1	2	0	1	2

Figure of a k-colored board for $n = 5, k = 3$.

Zakhar found a board at his dacha and now wants to obtain a k-colored board for some positive integer value of k. Some t cells of the table are already painted, with cell $(r[i], c[i])$ painted in color $v[i]$, and Zakhar may need to repaint them to the correct color. Unpainted cells do not concern Zakhar because applying paint is easier than repainting an already existing color.

Determine the minimum number of repaints that Zakhar will need to obtain a k-colored board for some positive integer value of k.

Solution Format

This is an unusual problem. It has a testing format with a grader, where you only need to implement the function `solve` in your solution. This function will be called by the testing program of the jury (the grader), and the returned value of the function will be accepted as the solution to the problem.

In particular, this means that there should be **no input or output** in the code you submit. If you code in C++, your code **must not** contain a function `main`. If necessary, you can implement any number of helper functions, structures, classes, and global variables, but all the code of your solution must be in one file.

For a solution in C++, you must implement the following function:

```
int solve(int n, int t, std::vector<int> r, std::vector<int> c, std::vector<int> v)
```

For a solution in Python3 or Pypy3, you must implement the following function:

```
def solve(n, t, r, c, v):
```

Here n and t are integers, and r, c, and v are lists of integers of length t.

Note that the jury does not guarantee the possibility of achieving full points on Python3 or Pypy3.

In both languages, the function `solve` takes the following arguments:

- n ($1 \le n \le 10^6$) — the number of rows and columns in the table.

- t ($1 \le t \le 10^6$) — the number of painted cells.
- r ($0 \le r[i] \le n-1$) — an array of size t consisting of the row numbers of the painted cells.
- c ($0 \le c[i] \le n-1$) — an array of size t consisting of the column numbers of the painted cells.
- v ($0 \le v[i] \le 10^9$) — an array of size t consisting of the colors of the painted cells.

All painted cells are numbered in 0-indexing, so for any i ($0 \le i < t$), the i-th painted cell is located at the intersection of the $r[i]$-th row and the $c[i]$-th column and has color $v[i]$. It is guaranteed that all pairs $(r[i], c[i])$ are distinct.

The function `solve` should return a single integer — the minimum number of originally painted cells that will need to be repainted to make the board k-colored.

It is guaranteed that the function `solve` will be called exactly once during the program's execution.

Testing

You are provided with template files where you can write your solutions: `table.cpp` and `table.py`. Also, in C++, you have been provided with a header file `table.h` containing the definition of the function `solve`.

For your convenience, graders are provided — files `grader.cpp` and `grader.py`. In these files, reading the input from the standard input, the execution of the function `solve`, and the output of the returned value of the function `solve` to the standard output are implemented. In the testing system, these grader files may differ.

To compile your C++ code written in the file `table.cpp`, use the command

```
g++ -std=c++20 grader.cpp table.cpp -o grader
```

After executing this command, an executable file named `grader` or `grader.exe` will be created, depending on your operating system, which can be run to input a test in the specified format.

To run your Python code written in the file `table.py`, use the command `python3 grader.py`, and then you can input a test in the specified format.

If compilation via commands causes you difficulties, for local testing, you can copy the implementation of input and output from the files `grader` into the files `table` and run the files `table.cpp` or `table.py`. However, before submitting your solution to the testing system, you will need to remove these input and output implementations (in particular, you will need to remove the function `main` if you code in C++).

Input

The grader provided to you reads input data in the following format:

The first line contains two integers n and t ($1 \le n, t \le 10^6$) — the number of rows and columns in the table, as well as the number of already painted cells.

The second line contains t integers $r[i]$ ($0 \le r[i] \le n-1$) — the row numbers of the painted cells.

The third line contains t integers $c[i]$ ($0 \le c[i] \le n-1$) — the column numbers of the painted cells.

The fourth line contains t integers $v[i]$ ($0 \le v[i] \le 10^9$) — the colors of the already painted cells.

Output

The grader outputs the returned value of the function `solve`.

Example

test	answer
5 4 0 0 1 4 0 1 0 0 2 1 4 1	2

Note

The example shows a sample input for the grader provided to you. Initially, four cells are painted: $(0, 0)$ in color 2, $(0, 1)$ in color 1, $(1, 0)$ in color 4, and $(4, 0)$ in color 1. It is optimal to choose $k = 3$, and the drawing of the table is provided in the problem statement.

Two cells need to be repainted: $(0, 0)$ from color 2 to color 0, and $(1, 0)$ from color 4 to color 1.

Scoring

The tests for this problem consist of four groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed.

Group	Points	Additional constraints		Required groups	Comment
		n	t		
0	0	–	–	–	Samples.
1	17	$n \le 100$	$t \le 100$	0	
2	21	$n \le 1\,000$	–	0, 1	
3	34	$n \le 100\,000$	$t \le 100\,000$	0, 1	
4	28	–	–	0 – 3	

Problem E. Egor's Gaming Addiction

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	2 seconds
Memory limit:	512 megabytes

The boy Egor constantly plays the trendy game "TOTA" which features n types of heroes. A line of heroes of length k is defined as a sequence of heroes with types $b_1, b_2, \ldots, b_k$. Egor calls a line of heroes of length k *terrible* if each hero type from 1 to k appears in this line exactly once. In other words, a line of length k is terrible for Egor if for every $1 \le i \le k$, there exists a position in the line j such that $b_j = i$.

Egor has a gaming addiction, so every day he plays the game with his friend Timofey. Since the game is still under development, each day the same sequence of heroes of length n is available for constructing lines, where the i-th hero has type a_i. Timofey will play with Egor for q days. Each day, a certain number l_i is fixed—the index of the first hero to be taken in the line. Accordingly, on the i-th day, Timofey can assemble a line of heroes against Egor with indices $a_{l_i}, a_{l_i+1}, \ldots a_{r_i}$ for any r_i such that $l_i \le r_i \le n$.

Timofey really wants Egor to stop playing "TOTA" all the time and instead focus on studying. To do this, Timofey wants to deprive Egor of all the joy of playing. Each day, Egor has x_i joy, which Timofey wants to take away by setting a terrible line against him. It is known that Egor's joy will be taken away if the length of the terrible line is at least the amount of joy, meaning that on day i, Egor will lose all his joy if a line of length at least x_i is set against him.

Timofey wants to deprive Egor of joy as often as possible, so for each i from 1 to q, he wants to know the minimum length of the line that will deprive Egor of joy on day i, as well as the number of possible line configurations on day i that will take away Egor's joy. Unfortunately, due to frequent playing of "TOTA" Timofey's cognitive abilities have diminished, and he will need your help to solve this problem.

Input

The first line contains a single integer g ($0 \le g \le 8$) — the group number for this test.

The second line contains two integers n and q ($1 \le n, q \le 10^6$) — the number of heroes and the number of days, respectively.

The third line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the types of heroes in the initial list.

In each of the following q lines, there are two integers l_i and x_i ($1 \le l_i \le n, 1 \le x_i \le n - l_i + 1$) — the index of the first hero in the line on day i and the amount of joy Egor has on day i.

Output

Output q lines, in the i-th line output the answer for day i — the minimum length of the terrible line and the number of terrible lines that will deprive Egor of joy on day i. If no lines on day i will deprive Egor of joy, output `-1 0`.

Examples

standard input	standard output
0 6 3 1 4 2 3 1 4 1 1 2 1 3 1	1 2 4 1 3 2
0 3 1 3 3 1 1 1	-1 0

Note

Consider the first example.

On the first day, Egor has a joy level of $x_1 = 1$, so to deprive him of joy, a terrible line of at least length 1 is required, but starting from hero $l_1 = 1$, there are two terrible lines:

- 1
- $1, 4, 2, 3$

The minimum length of a terrible line is 1, and their count is two.

On the second day, there is a new number $l_2 = 2$, for which there is only one terrible line: $4, 2, 3, 1$. Note that the line $4, 2, 3, 1, 4$ is not terrible because hero type 4 appears twice in it.

On the third day, lines are formed starting from hero number $l_3 = 3$, and on this day, there are two terrible lines:

- $2, 3, 1$
- $2, 3, 1, 4$

Scoring

The tests for this problem consist of eight groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Group	Points	Additional constraints		Required groups	Comment
		n	q		
0	0	–	–	–	Samples.
1	8	$n \leq 1\,000$	$q \leq 1\,000$	0	
2	7	$n \leq 5\,000$	$q \leq 5\,000$	0, 1	
3	9	$n \leq 200\,000$	$q \leq 200\,000$	–	$a_i = (i-1) \mod x + 1$ for a fixed $x > 0$
4	12	$n \leq 200\,000$	$q \leq 200\,000$	–	a is a permutation of numbers from 1 to n
5	19	$n \leq 200\,000$	$q \leq 200\,000$	–	$x_i = 1$ for all queries
6	10	$n \leq 200\,000$	$q \leq 200\,000$	0 – 5	
7	14	$n \leq 500\,000$	$q \leq 500\,000$	0 – 6	
8	21	–	–	0 – 7	**Offline-testing.**

Problem F. Alien Homophones

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

Okarun is obsessed with the idea that aliens exist. In his childhood, he tried various ways to contact them, but to no avail. This is not surprising, as aliens speak a completely different language! One day, he stumbled upon an article stating that aliens actually use lowercase Latin letters for writing, but read words in a completely different way.

In the alien language, there is a set of $n+26$ different sounds, each represented by a string of Latin letters s_i. It is known that for any $1 \le i \le 26$, the i-th sound is represented by a string of length one, consisting of the i-th letter of the Latin alphabet. For any $i > 26$, the sound with number i is represented by a string consisting of at least two Latin letters.

When an alien wants to read the word x, he starts reading it from the first position. When the alien is at position i, he looks for a sound s_j in the set of sounds such that it appears as a substring[†] in x, starting from position i. If there are multiple such sounds, he chooses the sound s_j with the **maximum** length. Then he pronounces that sound and moves to position $i + |s_j|$ and continues reading the word until he has read it completely. Note that we can always read any word because all Latin letters are sounds.

It also turned out that some alien sounds actually sound the same but are written differently. This means that there are some pairs of sound strings s_i and s_j $(i \ne j)$ such that $s_i \ne s_j$, but the sounds represented by these strings are the same. Note that if s_i and s_j are considered the same sound, and s_j and s_l are considered the same sound, then s_i and s_l are also considered the same sound.

Aliens call some words *homophones* — words that sound the same, and their spelling may either match or differ. In other words, if we have two words v and w, aliens call them homophones if the sequence of sounds pronounced by aliens while reading word v matches the sequence of sounds pronounced by aliens while reading word w.

Okarun wrote some text t, consisting of lowercase Latin letters. At one point, he became interested in examining q pairs of substrings of the text. In each such pair of substrings, the first substring is from the a_i-th to the b_i-th character in the text t, inclusive, and the second is from the c_i-th to the d_i-th characters in the text t, inclusive. For each pair of substrings, Okarun wants to know if these substrings are homophones when read as words in the alien language.

[†] A substring of a string s is a string that can be obtained by removing some number of characters (possibly zero) from the beginning of the string s and some number of characters (possibly zero) from the end of the string s.

Input

The first line contains a non-empty string t $(1 \le |t| \le 500\,000)$, consisting of lowercase Latin letters — the text written by Okarun.

The second line contains two integers n and k $(0 \le n \le 500\,000,\ 0 \le k < n+26)$ — the number of sounds in the set that have a length of at least two and the number of pairs of identical sounds noted by Okarun.

The next n lines describe the sounds numbered from the 27th. The i-th of them contains a non-empty string s_{i+26} $(2 \le |s_{i+26}| \le 10^6)$, consisting of lowercase Latin letters — the representation of the $(i+26)$-th sound. It is guaranteed that all sound strings are different. Note that there are no strings from "a"to "z"in the input; however, in each test, they are considered sounds numbered from 1 to 26.

The next k lines describe pairs of identical sounds originally noted by Okarun. Each of them contains a pair of integers x_i and y_i $(1 \le x_i, y_i \le n+26;\ x_i \ne y_i)$ — a pair of sound numbers that Okarun considers identical in sound. It is guaranteed that each pair of numbers appears no more than once. Note that the equality of some other pairs of sounds may follow from this set.

The next line contains a single integer q ($1 \le q \le 300\,000$) — the number of queries for pairs of substrings of the text, for which it is necessary to determine whether they are alien homophones.

In the next q lines, the i-th of them contains four integers a_i, b_i, c_i, d_i ($1 \le a_i \le b_i \le |t|$, $1 \le c_i \le d_i \le |t|$), which specify the substrings of the text — the first substring from position a_i to position b_i (inclusive) in the text t, the second from position c_i to position d_i (inclusive) in the text t.

Let S denote the sum of the lengths of the sounds $\sum |s_i|$ (excluding the sounds "a" to "z"). It is guaranteed that $S \le 10^6$.

Output

Output q lines. If the pair of strings from the i-th query are identical in sound, output "`Yes`" (without quotes) in the i-th line; otherwise, output "`No`" (without quotes) in the i-th line.

Example

standard input	standard output
abracadabra 2 3 cada ca 1 27 1 28 1 4 4 5 11 1 4 4 6 5 7 5 7 5 8 2 5 2 5	Yes Yes No Yes

Note

In the first query:

- The first substring **cadabra** is read as sounds: **cada**, **b**, **r**, **a**;
- The second substring **abra** is read as sounds: **a**, **b**, **r**, **a**.

The sounds **cada** and **a** are noted as identical (pair (1, 27)), therefore these two substrings are alien homophones.

In the second query:

- The first substring **aca** is read as sounds: **a**, **ca**;
- The second substring **cad** is read as sounds: **ca**, **d**.

The sounds **a** and **ca** are noted as identical (pair (1, 28)), the sounds **ca** and **d** are also identical (since the sounds with numbers (1, 4) and (1, 28) are identical, then the sounds with numbers 4 and 28 are also identical), therefore these two substrings are also alien homophones.

In the third query:

- The first substring **cad** is read as sounds: **ca**, **d**;
- The second substring **cada** is read as sounds: **cada**.

The number of sounds does not match, so these two substrings are definitely not alien homophones.

In the fourth query, two identical substrings are given, so they are read the same and are alien homophones.

Scoring

The tests for this problem consist of ten groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Group	Points	Additional constraints			Required groups	Comment
		$\lvert t\rvert$	S	q		
0	0	–	–	–	–	Samples.
1	8	$\lvert t\rvert \le 500$	$S \le 500$	$q \le 500$	0	
2	7	$\lvert t\rvert \le 6\,000$	$S \le 6\,000$	$q \le 6\,000$	–	$b_i = d_i = \lvert t\rvert$
3	9	$\lvert t\rvert \le 6\,000$	$S \le 6\,000$	$q \le 200\,000$	2	$b_i = d_i = \lvert t\rvert$
4	15	$\lvert t\rvert \le 200\,000$	$S \le 200\,000$	$q \le 200\,000$	2, 3	$b_i = d_i = \lvert t\rvert$
5	8	$\lvert t\rvert \le 6\,000$	$S \le 6\,000$	$q \le 200\,000$	–	$a_i = c_i = 1$
6	6	$\lvert t\rvert \le 500$	$S \le 500$	$q \le 200\,000$	0, 1	
7	7	$\lvert t\rvert \le 6\,000$	$S \le 6\,000$	$q \le 6\,000$	0 – 2	
8	10	$\lvert t\rvert \le 6\,000$	$S \le 200\,000$	$q \le 6\,000$	0 – 2, 7	
9	19	$\lvert t\rvert \le 200\,000$	$S \le 400\,000$	$q \le 200\,000$	0 – 8	
10	11	–	–	–	0 – 9	**Offline-testing.**

Problem G. Ultrafast train

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	1 second
Memory limit:	512 megabytes

Berland, 2224. In Berland, there are n cities, connected by m railways. Each railway connects two cities and can be used to go in both directions. Also, the railway network includes all cities, so that you can go from any city to any other city by using railways.

Berland railway company (BRC) wants to launch a train that departs with passengers from city 1 and arrives at city n. Since this ride can take a very long time, BRC has decided to speed this train up by installing a time machine.

The time machine works as follows: before departing from any city, it can be turned off or on. If it's turned off, then the ride will take exactly 1 hour. However, if it's turned on, then the ride will take exactly -1 hour, and the train will arrive at the next city earlier than it departed.

BRC wants to plan a route for this train. The route must follow these conditions:

1. The route must start at city 1 and end at city n. All adjacent cities in the route must be connected by railways. It's OK that cities 1 and n could be in the middle of the route.
2. The route must take a nonnegative amount of hours at any point, in order to avoid time-space paradoxes.
3. The route must have at most $3n + 2$ cities, since otherwise the ride is considered complex and unattractive.
4. The route must take the minimal amount of time, by following the conditions above.

Please help BRC to plan the most optimal route. If there are multiple optimal routes, then output any.

Input

The first line of the input contains two integers n and m ($2 \le n \le 10^5, n - 1 \le m \le 10^5$) — the numbers of cities and railways.

The next m lines contain two integers u and v ($1 \le u, v \le n$) — numbers of cities connected by railway.

It is guaranteed that every city is reachable from another city by railways. It's also guaranteed that there are no railways from city to each other and multiple railways between the same cities.

Output

The first line of the output must contain two integers t and k ($t \ge 0, 2 \le k \le 3n + 2$) — minimal time of ride and amount of cities in the route, including first and last cities.

The next line contains $2k - 1$ tokens, depicting the train route. Each 1-st, 3-rd, ..., $(2k - 1)$-th element must be an integer, depicting a city. Each 2-nd, 4-th, ..., $(2k - 2)$-th element must be `+`, if it's needed to turn off the time machine on this railway, or `-`, if it's needed to turn on the time machine.

Examples

standard input	standard output
5 4 2 1 3 2 3 5 4 2	1 4 1 + 2 + 3 - 5
5 5 1 2 2 3 3 4 4 5 5 2	0 9 1 + 2 + 3 + 4 + 5 - 2 - 3 - 4 - 5

Note

In the first example, the train will arrive in city 2 in one hour, after which it will arrive in city 3 for another hour. After that, the time machine will be turned on and the train will arrive at city 4 in -1 hour. As a result, the ride takes $1 + 1 - 1 = 1$ hour.

In the second example, the train will arrive at city 5 in 4 hours, after which it will go back to city 2, and from there to city 5 in -4 hours. As a result, the ride takes $4 - 4 = 0$ hours.

Scoring

The tests for this problem consist of five groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups.

If the country is **2-colorable**, then each city can be painted in one of two colors, such that each railway connects cities of two different colors.

Group	Points	Additional constraints		Required Groups	Comment
		n	m		
0	0	–	–	–	Samples.
1	17	$n \le 10$	$m = n - 1$	–	
2	13	–	$m = n - 1$	1	
3	12	–	–	0 – 2	Berland is 2-colorable.
4	26	–	–	–	$u_i = i, v_i = i + 1$ for $1 \le i \le n - 1$.
5	32	–	–	0 – 4	

Problem H. Classic Tree Problem

Time limit: 4 seconds
Memory limit: 1024 megabytes

This problem can only be solved using C++ or Python3.

One morning, Egor went outside and saw on the asphalt classic hopscotch squares — n squares, each containing some numbers, with the i-th square containing the number a_i. But these were not ordinary squares; each square was connected to some other squares.

The game of hopscotch involves starting from a square and jumping to another square connected to the current one, with the restriction that you cannot jump to the same square more than once. The game can end at any moment, and the sequence of visited squares $v_0, v_1, \ldots, v_{k-1}$ will be called the jump sequence.

However, the squares on the asphalt were drawn in the form of a tree, meaning there is a unique way to reach any square from another through some sequence of jumps. The "coolness" of the jump sequence is defined as the number of indices i such that $a_{v_i} = i$.

Egor wanted to find out the coolness of the jump sequences for some pairs of squares u, v, where $v_0 = u$ and $v_{k-1} = v$, to determine whether it is worth making such a sequence of jumps.

Egor wanted to show off his coolness, but he is constantly being interrupted. Therefore, he sometimes erases the number in the s-th square and writes a new number x. Since counting numbers is a difficult task for Egor, he turned to you for help.

In the next q minutes, you need to assist Egor with his questions about the coolness of the jumps. In minute i, Egor can perform one of two actions:

1. Erase the number in square s and write the number x in its place.
2. Ask about the coolness of the jump sequence from square u to square v.

You are required to answer each of Egor's questions.

Solution Format

This is an unusual problem. It has a testing format with a grader, where you only need to implement the function `solve` in your solution. This function will be called by the testing program of the jury (the grader), and the returned value of the function will be accepted as the solution to the problem.

In particular, this means that there should be **no input or output** in the code you submit. If you code in C++, your code **must not** contain a function `main`. If necessary, you can implement any number of helper functions, structures, classes, and global variables, but all the code of your solution must be in one file.

For a solution in C++, you must implement the following function:

```
std::vector<int> solve(int n, int q, std::vector<int> a, std::vector<int> p,
                       std::vector<int> qt, std::vector<int> qx, std::vector<int> qy);
```

For a solution in Python3 or Pypy3, you must implement the following function:

```
def solve(n, q, a, p, qt, qx, qy):
```

Here, n and q are integers; a and p are lists of integers of length n; qt, qx, and qy are lists of integers of length q.

Note that the jury does not guarantee the possibility of achieving full points on Python3 or Pypy3.

In both languages, the function `solve` takes the following arguments:

- n ($1 \le n \le 10^6$) — the number of squares.

- q ($1 \le q \le 10^6$) — the number of queries.
- a ($0 \le a[i] \le n$) — an array of size n, consisting of the numbers initially written in the squares.
- p ($-1 \le p[i] < n$) — an array of size n, where $p[i]$ is the parent number of square i if the hopscotch is represented as a tree. For the root of the tree, $p[i]$ will be -1. It is guaranteed that the array defines a valid tree.
- qt ($1 \le qt[i] \le 2$) — an array of size q, consisting of the types of queries.
- qx and qy — arrays defining the queries.

For each $0 \le i < q$, the i-th query is defined as follows:

- If $qt[i] = 1$, then in the i-th query, it requires erasing the number in the square with index $qx[i]$ and writing the number $qy[i]$ in its place. In this case, the following constraints hold: $0 \le qx[i] < n$, $0 \le qy[i] \le n$.
- If $qt[i] = 2$, then in the i-th query, it requires finding the coolness of the jump sequence from square $qx[i]$ to square $qy[i]$. In this case, the following constraints hold: $0 \le qx[i], qy[i] < n$.

All squares and queries are numbered in 0-indexing.

The function `solve` returns an array containing the answers to the second type of queries. This array should have a length equal to the number of second-type queries. In C++, this array is returned as a type `vector<int>`, and in Python3, it is returned as a list of integers.

It is guaranteed that the function `solve` will be called exactly once during the program's execution.

Testing

You are provided with template files where you can write your solutions: `tree.cpp` and `tree.py`. Also, in C++, you have been provided with a header file `tree.h` containing the definition of the function `solve`.

For your convenience, graders are provided — files `grader.cpp` and `grader.py`. In these files, reading the input from the standard input, the execution of the function `solve`, and the output of the returned value of the function `solve` to the standard output are implemented. In the testing system, these grader files may differ.

To compile your C++ code written in the file `tree.cpp`, use the command

```
g++ -std=c++20 grader.cpp tree.cpp -o grader
```

After executing this command, an executable file named `grader` or `grader.exe` will be created, depending on your operating system, which can be run to input a test in the specified format.

To run your Python code written in the file `tree.py`, use the command `python3 grader.py`, and then you can input a test in the specified format.

If compilation via commands causes you difficulties, for local testing, you can copy the implementation of input and output from the files `grader` into the files `tree` and run the files `tree.cpp` or `tree.py`. However, before submitting your solution to the testing system, you will need to remove these input and output implementations (in particular, you will need to remove the function `main` if you code in C++).

Input

The grader reads the test in the following format:

The first line specifies two integers n, q ($1 \le n, q \le 10^6$) — the number of squares in the hopscotch and the number of actions by Egor.

The next line specifies n integers $a_0, a_1, \ldots, a_{n-1}$ ($0 \le a_i \le n$) — the initial numbers in the squares.

The following line specifies n integers $p_0, p_1, \ldots p_{n-1}$ ($-1 \le p_i < n$), defining the parents of the nodes in the tree. If $p_i = -1$, then node i is the root of the tree; otherwise, node p_i is the parent of node i in the tree. It is guaranteed that this array of ancestors forms a valid rooted tree.

In the next q lines, queries are specified. For any $0 \le i < q$, in the i-th query, depending on the type of query, it is specified in the following format:

1 s x ($0 \le s < n, 0 \le x \le n$) — Egor erases the number in square s and writes the number x there. In the function `solve` arguments, it holds that $qt[i] = 1, qx[i] = s, qy[i] = x$.

2 u v ($0 \le u, v < n$) — Egor asks about the coolness of the jump sequence from u to v. In the function `solve` arguments, it holds that $qt[i] = 2, qx[i] = u, qy[i] = v$.

Output

The grader outputs the results of the function `solve` — the answers to the second type of queries.

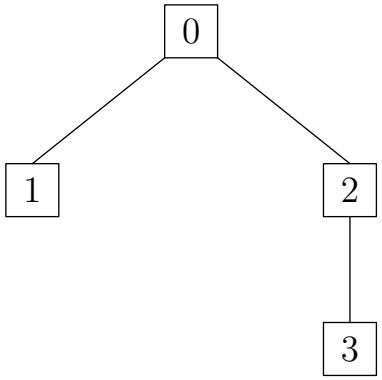
Examples

test	answer
4 3 1 1 2 2 -1 0 0 2 2 0 3 1 2 1 2 0 3	1 2
5 5 0 1 2 3 4 1 2 -1 2 3 2 0 4 1 0 1 1 1 0 2 0 4 2 1 0	5 3 2

Note

In the examples, the input and output data of the grader are specified.

In the first example, the tree of squares looks as follows:



For the first query, consider the jump sequence from 0 to 3:

1. $v_0 = 0, a_0 = 1 \ne 0$, so this square does not add coolness to Egor.
2. $v_1 = 2, a_2 = 2 \ne 1$, so this square also does not add coolness to the jump sequence.

3. $v_2 = 3, a_3 = 2 = 2$, this jump is cool.

Thus, the coolness in the first query is 1.

After the second query, the number in square number 2 is now 1. Therefore, in the second query, jumping to square number 2 becomes cool, and now the answer to this query is 2.

Scoring

The tests for this problem consist of fourteen groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Let c_i be the number of squares connected to the i-th square.

Group	Points	Additional constraints		Required groups	Comment
		n	q		
0	0	–	–	–	Samples.
1	10	$n \le 1\,000$	$q \le 1\,000$	0	
2	11	$n \le 200\,000$	$q \le 5\,000$	0, 1	
3	14	$n \le 200\,000$	$q \le 200\,000$	–	No first-type queries
4	5	$n \le 200\,000$	$q \le 200\,000$	–	For exactly one square $c_i = 2$, for the others $c_i \le 3$
5	11	$n \le 200\,000$	$q \le 200\,000$	–	$c_i \le 2$
6	10	$n \le 200\,000$	$q \le 200\,000$		For all second-type queries $v_i = 0$
7	9	$n \le 100\,000$	$q \le 100\,000$	0, 1	
8	8	$n \le 200\,000$	$q \le 200\,000$	0 – 7	
9	17	$n \le 500\,000$	$q \le 500\,000$	0 – 8	
10	1	$n \le 600\,000$	$q \le 600\,000$	0 – 9	**Offline-testing.**
11	1	$n \le 700\,000$	$q \le 700\,000$	0 – 10	**Offline-testing.**
12	1	$n \le 800\,000$	$q \le 800\,000$	0 – 11	**Offline-testing.**
13	1	$n \le 900\,000$	$q \le 900\,000$	0 – 12	**Offline-testing.**
14	1	–	–	0 – 13	**Offline-testing.**

Problem I. Colorful Diameter

Input file:	`standard input` or `input.txt`
Output file:	`standard output` or `output.txt`
Time limit:	7 seconds
Memory limit:	512 megabytes

On a plane, there are n points with coordinates (x_i, y_i); each point has a color c_i. Each point moves at a constant speed along the vector (dx_i, dy_i) per second. This means that every second, the value dx_i is added to the x-coordinate of point i, and the value dy_i is added to the y-coordinate of the point.

For all integer moments in time during the first T seconds, find the minimum of the maximum distance between points of different colors. In other words, find such an integer moment in time t ($0 \le t \le T$) that the maximum distance between points of different colors at that moment is minimized compared to any other integer time from 0 to T.

Input

Each test consists of several sets of input data. The first line contains one integer t ($1 \le t \le 100\,000$) — the number of input data sets. The description of the input data sets follows.

The first line of each input data set contains two integers n and T ($2 \le n \le 200\,000$, $0 \le T \le 10^8$) — the number of points and the time interval considered.

Each of the following n lines describes the points. The i-th line contains five integers x_i, y_i, dx_i, dy_i, and c_i ($-10^8 \le x_i, y_i, dx_i, dy_i \le 10^8$, $1 \le c_i \le n$) — the coordinates, speed in each coordinate, and the color of point i.

It is guaranteed that there are two points with different colors.

It is guaranteed that the sum of n across all input data sets in each test does not exceed 200 000, and the sum of T does not exceed 10^8.

It is guaranteed that any coordinates of the points at any moment in time up to T inclusive do not exceed 10^9 in absolute value.

Output

For each input data set, output one number on a separate line — the minimum of the maximum distance between points of different colors during the first T seconds.

Your answer is considered correct if its absolute or relative error does not exceed 10^{-9}.

Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if $\frac{|a-b|}{\max(1,|b|)} \le 10^{-9}$.

Example

standard input	standard output
3 3 0 1 1 0 0 1 2 2 0 0 2 4 4 0 0 1 2 2 0 0 1 1 1 2 2 -1 -1 2 4 100 0 0 0 1 1 4 0 -1 0 2 4 4 0 -1 1 0 4 1 0 2	2.8284271247461900976 0.00000000000000000000 2.8284271247461900976

Note

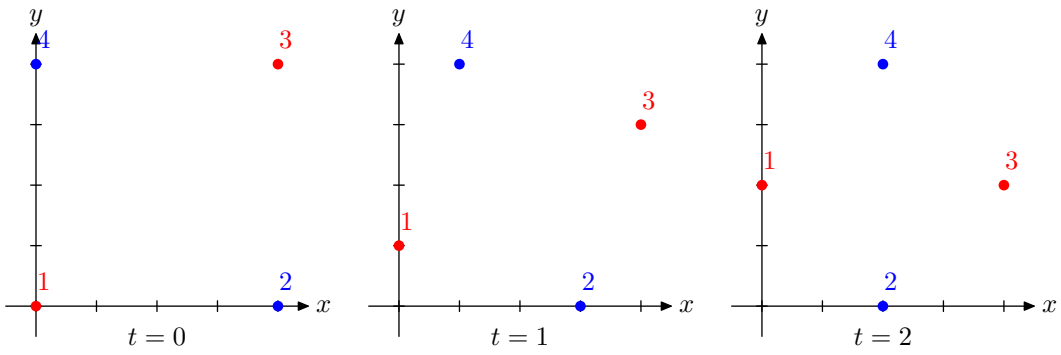
Consider the example from the statement.

In the first input data set, only one moment in time is considered. The distance between the first and second points is $\sqrt{2}$, and between the second and third points is $2\sqrt{2}$. Therefore, the answer is $2\sqrt{2}$.

In the second input data set, at time 1, the points will coincide, and the distance between them will be zero. Therefore, the answer is 0.

In the third input data set, it can be shown that the minimum maximum distance between points of different colors is achieved at time 2 and equals $2\sqrt{2}$.

The figure shows the third input data set for moments in time $t = 0, 1, 2$. Points of color 1 are marked in red, and points of color 2 are marked in blue.



Scoring

The tests for this problem consist of nine groups. Points for each group are awarded only if all tests of the group and all tests of some of the previous groups are passed. Note that passing the samples is not required for some groups. **Offline-testing** means that the results of testing your solution on this group will only be available after the competition ends.

Let $\sum n$ be the sum of n across all input data sets of this test.

Let $\sum T$ be the sum of T across all input data sets of this test.

Group	Points	Additional constraints		Required groups	Comment
		$\sum n$	$\sum T$		
0	0	–	–	–	Samples.
1	10	$\sum n \le 100$	$\sum T \le 10$	0	
2	8	$\sum n \le 10\,000$	$\sum T \le 10$	0, 1	
3	11	$\sum n \le 100$	–	0, 1	
4	14	$\sum n \le 100\,000$	$\sum T \le 10$	–	Three are exactly 2 different colors.
5	8	$\sum n \le 100\,000$	–	4	There are exactly 2 different colors.
6	11	$\sum n \le 100\,000$	$\sum T \le 10$	–	All colors are different.
7	8	$\sum n \le 100\,000$	–	6	All colors are different.
8	19	$\sum n \le 100\,000$	–	0 – 7	
9	11	–	–	0 – 8	**Offline-testing.**