Papa and Baby Frog

Input File	Output File	Time Limit	Memory Limit
standard input	standard output	1 second	256 MiB

Papa and Baby frog live among a line of n **equally-spaced** stones, numbered 1 to n, from left to right. Stone i has a positive integer height h_i.

The frogs can jump between stones, and do so, according to the following rules:

- A single jump moves Papa frog to the **closest** stone that has a **strictly greater** height than his current stone. If there are multiple options, he chooses the **rightmost** among these;
- A single jump moves Baby frog to the **closest** stone that has a **strictly smaller** height than her current stone. If there are multiple options, she chooses the **rightmost** among these.

Papa frog would like to choose a stone to be *home*, and a **different** stone to be *school*. He requires that:

- Papa frog can jump from home to school, in no more than k jumps; and
- Baby frog can jump from school to home, in no more than k jumps.

A stone is called a *potential home* if there is **at least one** other stone that can be chosen as school, if this stone is chosen as home.

Your task is to help the frogs determine which stones are potential homes.

Subtasks and Constraints

For all subtasks, you are guaranteed that:

- $n \geq 2$
- $1 \leq k \leq n$
- For all stones i, $1 \leq h_i \leq 1\,000\,000\,000$

Additional constraints for each subtask are given below.

Subtask	Points	n	k
1	10	$n \leq 2000$	$k = 1$
2	10	$n \leq 2000$	$k \leq 5$
3	20	$n \leq 2000$	$k \leq 100$
4	20	$n \leq 2000$	$k \leq n$
5	20	$n \leq 100\,000$	$k \leq 5$
6	10	$n \leq 100\,000$	$k = n$
7	10	$n \leq 100\,000$	$k \leq n$

Input

The first line of input contains the two integers, n and k.

The second line of input contains n integers $h_1, \ldots, h_n$, which are the heights of the stones.

Output

The output should contain a string of n characters on a single line. The i-th of these characters should be `1`, if stone i is a potential home, or `0`, otherwise.

Sample Input 1

```
4 1
4 1 3 2
```

Sample Output 1

```
0001
```

Sample Input 2

```
7 2
3 1 2 1 2 2 3
```

Sample Output 2

```
0101010
```

Sample Input 3

```
10 3
10 5 6 4 7 3 8 2 9 1
```

Sample Output 3

```
0000010001
```

Explanation

In the first example, only stone 4 can be chosen as home, when stone 3 is chosen as school. Hence, stone 4 is the only potential home.

In the second example:

- Stone 2 can be chosen as home, when stone 1 is chosen as school. Papa frog jumps through stones $2 \to 3 \to 1$, whereas Baby frog jumps directly from stone 1 to stone 2.

- Stone 4 can be chosen as home, when stone 5 is chosen as school. Papa and Baby frog both jump directly between home and school, and vice-versa.
- Stone 6 can be chosen as home, when stone 7 is chosen as school. Papa and Baby frog both jump directly between home and school, and vice-versa.

Hence, stones 2, 4 and 6 are potential homes.

In the third example:

- Stone 6 can be chosen as home, when stone 1 is chosen as school. Papa frog jumps through stones $6 \to 7 \to 9 \to 1$, whereas Baby frog jumps through stones $1 \to 2 \to 4 \to 6$.
- Stone 10 can be chosen as home, when stone 9 is chosen as school. Papa and Baby frog both jump directly between home and school, and vice-versa.

Hence, stones 6 and 10 are potential homes.

Coloured Walkway

Input File	Output File	Time Limit	Memory Limit
standard input	standard output	2 seconds	1024 MiB

A new art installation has appeared in the streets of your local mall. It's a long walkway of n tiles, numbered 1 to n from left to right. Each tile is coloured one of c possible colours, numbered from 1 to c. The i-th tile has colour x_i.

You start on the leftmost tile and must get to the rightmost tile. You may only move by making jumps to the right. Using your powerful legs, you can skip any positive number of tiles with each jump.

Things are not so simple, however. The artist insists that you may only jump from one coloured tile to another colour if the colours are *compatible*! You have a list of p pairs of compatible colours (a colour may or may not be compatible with itself). The j-th such pair states that colours a_j and b_j are compatible.

The colours are very pretty and this compels you to count the number of ways you can get from the leftmost tile to the rightmost one by jumping. Two ways are considered different if there is a tile you land on in one way, but not the other. Since the answer can be quite big, give the answer modulo 1 000 000 007.

As this number may be large, you may wish to use 64-bit integer types (such as `long long` in C/++) in your program.

Subtasks and Constraints

For all subtasks:

- $2 \leq n \leq 100\,000$
- $1 \leq c \leq 100\,000$
- $1 \leq x_i \leq c$, for all i
- $0 \leq p \leq 100\,000$
- $1 \leq a_j, b_j \leq c$ for all j (Note that it is possible that $a_j = b_j$)
- $(a_i, b_i) \neq (a_j, b_j)$ and $(a_i, b_i) \neq (b_j, a_j)$ for any $i \neq j$. That is, no compatible pair of colours will appear twice.

Furthermore:

- For Subtask 1 (4 points): $c = 1$
- For Subtask 2 (22 points): $c \leq 100$.
- For Subtask 3 (15 points): $x_i = i$, for all i.
- For Subtask 4 (26 points): $p = c - 1$, $a_j = j$ and $b_j > j$ for all j.
- For Subtask 5 (33 points): No further constraints apply.

Input

The first line of input will contain the two integers n, c. The second line will contain the n integers x_1, x_2, $\ldots$, x_n.

The third line contains the single integer p. Then, p lines follow, describing the pairs of compatible colours. The j-th such pair states that colours a_j and b_j are compatible.

Output

Output a single integer, the total number of ways of getting from the leftmost tile to the rightmost tile, modulo 1 000 000 007.

Sample Input 1

```
5 3
1 2 3 2 3
3
1 2
1 3
2 3
```

Sample Output 1

```
5
```

Sample Input 2

```
4 4
1 2 3 4
2
1 2
2 3
```

Sample Output 2

```
0
```

Sample Input 3

```
6 2
1 1 1 2 2 1
3
1 1
2 1
2 2
```

Sample Output 3

```
16
```

In the first sample case, there are 5 ways, which are listed below by the tile numbers they land on:

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

In the second sample case, it is not possible to reach the rightmost tile, no matter how you jump! Hence the answer is 0.

In the third sample case, every colour is compatible with every other colour (including themselves), so any jump is allowed. There are 16 ways.

Nav

Input File	Output File	Time Limit	Memory Limit
none	**none**	4 seconds	1024 MiB

In the latest video game you are playing, you find yourself in a mysterious world. The world can be represented as a connected graph over n vertices, numbered from 0 to $n-1$. There are m bidirectional edges in the graph, numbered from 0 to $m-1$, each connecting a different pair of vertices. All edges have the same length. You have been told the exact layout of this graph.

To test your vast knowledge, you will have to solve t tests.

In each test, one vertex has been secretly chosen as the *prime* vertex, and your task is to identify which vertex it is. To help you do so, you have been given a device called the *Nav*. The Nav is designed to help you navigate to the prime vertex. It does so by allowing you to query it, in the following manner.

1. You input a vertex of your choosing, call it u. If u is the prime vertex, the Nav will return -1.
2. Otherwise, the Nav computes a **shortest path** from u to the prime vertex. Note that if there is more than one such path, the Nav can choose any of these.
3. The Nav returns an integer between 0 and $m-1$, corresponding to a **single edge** that is a part of the computed shortest path.

Your task is to identify the prime vertex, using as few Nav queries as possible.

The fewer queries you use, the more points you will score for this problem. See the *Subtasks and Constraints* section below for further details.

Subtasks and Constraints

For all subtasks, you are guaranteed that:

- $n \geq 1$
- $n-1 \leq m \leq 100\,000$
- The provided graph will be connected
- Every edge will connect two *different* vertices
- No two edges connect the same pair of vertices.

For each subtask, there is a maximum number q of queries you can make to the Nav **per test**. Your score for a testcase is computed as follows:

- If you make more than q queries for any test, you will receive 0 points. This will be reported as `Not correct` in the contest system.
- If your program exceeds the time limit, you will receive 0 points.
- If your program does not successfully terminate, or fails to identify the prime vertex, you will receive a score of 0 points. This will be reported as `Not correct` in the contest system.
- Otherwise, if your program correctly identifies the prime vertex in all cases:
  - For Subtasks 1 to 3: you will score 100% of the marks for that subtask. This will be reported as `Correct` in the contest system.

- For Subtask 4, your score is computed, based on the number of queries q_max that your program made in any test. Specifically, your score will be:
  - 100%, if $q_{\max} \leq 9$;
  - 70%, if $10 \leq q_{\max} \leq 12$; or
  - $\frac{2}{3} \times \frac{30-q_{\max}}{q_{\max}}$, if $13 \leq q \leq 30$.

Subtask	Points	Maximum t	Maximum n	q	Additional constraints
1	7	750	$n \leq 300$	300	
2	15	250 000	$n \leq 100\,000$	17	$m = n - 1$, and for each $0 \leq j \leq m-1$, edge j connects vertex j and vertex $j+1$. That is, the graph is a line.
3	26	750	$n \leq 1000$	10	$m = n - 1$. That is, the graph is a tree.
4	52	750	$n \leq 300$	30	

Recall that your score for this problem is the sum of your scores for each subtask. Your score for a subtask is the maximum score obtained among any of your submissions. As such, you may wish to solve different subtasks in different submissions.

Input / Output

This task has no input or output files. Instead, your solution must interact with the functions in the header file `"nav.h"`. The provided functions are described in detail in the next section. ***Do not* output *anything* to `stdout`, or you will receive 0% points for the test case.**

Implementation

You **must not** implement a `main` function. Instead, you should `#include "nav.h"` and implement the two functions `init` and `findPrime`, described below:

```
void init(int subtask, int n, int m, std::vector<int> A, std::vector<int>
B);
```

where:

- `subtask` is the subtask number
- `n` is the number of vertices
- `m` is the number of edges
- For every $0 \leq j \leq m-1$, edge j connects vertices `A[j]` and `B[j]`.
- This function will be called first to initialise your program.

```
int findPrime();
```

This function should return an integer between 0 and $n-1$: the number corresponding to the prime vertex.

The grader will call this function t times, once for each test. Your program is not told t.

In your implementation, you may call the following function.

```
int nav(int u)
```

where:

- `u` must be an integer, between 0 and $n-1$, corresponding to the number of a vertex of your choosing.
- If u is the prime vertex, the function returns -1. Otherwise, the function returns an integer between 0 and $m-1$, corresponding to an **edge** along a shortest path between u and the prime vertex.

If `nav` is called with an invalid value of u, the program will terminate, and you will score 0% for that testcase. This will be reported as `Not correct` in the contest system.

You may assume that over the course of a single test, the grader that is used does not change the identity of the prime vertex.

A template `nav.cpp` has been provided for your convenience.

Experimentation

In order to experiment with your code on your own machine, first download the provided files `nav.h` and `grader.cpp`, which should be placed in the same directory as your code. Please note that the grader that is used may have **different** behaviour to the provided grader.

Compile your solution with:

```
g++ -std=c++11 -O2 -Wall -static nav.cpp grader.cpp -o nav
```

This will create an executable `nav`, which you can run with `./nav`. If you have trouble compiling, please send a message in the Communication section of the contest website.

The compiled sample grader will read in the subtask number, n and m on the first line.

It will then read m more lines. From the i-th of these lines, the sample grader wil read two integers `A[i]` and `B[i]`.

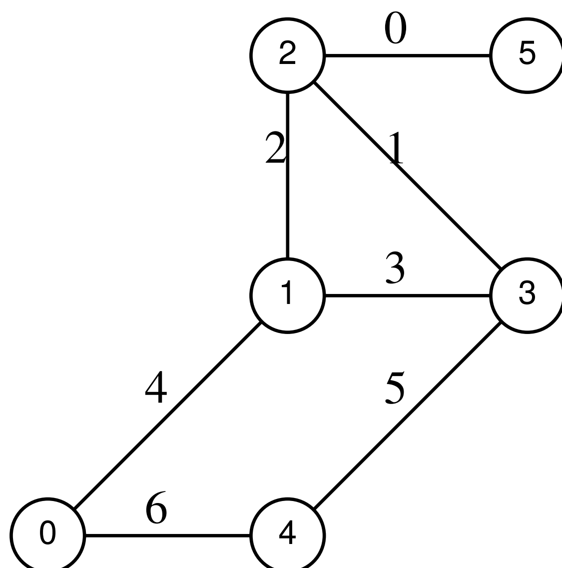
On the next line, it will read in the integer t, the number of tests, followed by t lines. From the j-th of these lines, it will read in the secret prime vertex for the j-th test.

For each test, the sample grader will call `findPrime()` and print out whether your program successfully found the prime vertex, along with the number of queries it made.

Sample Grader Input and Sample Session

```
4 6 7
2 5
2 3
2 1
1 3
0 1
3 4
4 0
3
3
5
3
```

The grader begins by reading in the graph, and then interacts with your program.



One possible interaction is shown below:

Grader	Student	Description
`init(4, 6, 7, [2, 2, 2, 1, 0, 3, 4], [5, 3, 1, 3, 1, 4, 0])`		The grader initialises your program. This sample case is for Subtask 4.
`findPrime()`		The grader starts the first test. The prime vertex is vertex 3
	`nav(0)`	Your program calls `nav` on vertex 0.
`nav` returns 6		The grader picks the shortest path $0 \to 4 \to 3$, and chooses to return edge 6 (the edge between 0 and 4).
	`nav(2)`	Your program calls `nav` on vertex 2.
`nav` returns 1		The grader picks the shortest path $2 \to 3$, and chooses to return edge 1 (the edge between 2 and 3).
	`nav(3)`	Your program calls `nav` on vertex 3.
`nav` returns -1		This is the prime vertex.
	You return 3	Your program guesses the prime vertex is vertex 3. This is correct.
`findPrime()`		The grader starts the second test. The prime vertex is vertex 5
	`nav(4)`	Your program calls `nav` on vertex 4.
`nav` returns 1		The grader picks the shortest path $4 \to 3 \to 2 \to 5$, and chooses to return edge 1 (the edge between 2 and 3).
	You return 1	Your program guesses the prime vertex is vertex 1. This is not correct.
`findPrime()`		The grader starts the third test. The prime vertex is vertex 3 again.

Grader	Student	Description
	You return 3	Your program guesses the prime vertex is vertex 3, without making any calls to `nav`! This is correct.

In this example, you have solved two of the three cases correctly, so you would score 0% points.