SubsetMex

Problem Name	Subset Mex
Input file	standard input
Output file	standard output
Time limit	1 second
Memory limit	256 megabytes

A *multiset* is a collection of elements similar to a set, where elements can repeat multiple times. For example, the following is a multiset:

{0, 0, 1, 2, 2, 5, 5, 5, 8}

Given a multiset S defined on non-negative integers, and a target non-negative integer value n such that n does not belong to S, your goal is to insert n into S by using the following 3-step operation, repeatedly:

1. Choose a (possibly empty) subset T of S. Here, T is a set of distinct numbers that appear in S.
2. Erase elements of T from S. (Remove only one copy of each element.)
3. Insert **mex**(T) into S, where **mex**(T) is the smallest non-negative integer that does not belong to T. The name **mex** stands for “minimum excluded” value.

Your goal is to find the minimum number of operations to perform so that n becomes part of S.

Since the size of S may be large, it will be given in the form of a list $(f_0, \ldots, f_{n-1})$ of size n, where f_i represents the number of times that the number i appears in S. (Recall that n is the integer we are trying to insert into S.)

Input

The first line contains a single integer t ($1 \le t \le 200$) — the number of test cases. Each two of the following lines describe a test case:

- The first line of each test case contains a single integer n ($1 \le n \le 50$), representing the integer to be inserted into S.

- The second line of each test case contains n integers $f_0, f_1, \ldots, f_{n-1}$ ($0 \le f_i \le 10^{16}$), representing the multiset S as mentioned above.

Output

For each test case, print a single line containing the minimum number of operations needed to satisfy the condition.

Scoring

Subtask #1 (5 points): $n \le 2$

Subtask #2 (17 points): $n \le 20$

Subtask #3 (7 points): $f_i = 0$

Subtask #4 (9 points): $f_i \le 1$

Subtask #5 (20 points): $f_i \le 2000$

Subtask #6 (9 points): $f_0 \le 10^{16}$ and $f_j = 0$ (for all $j \neq 0$)

Subtask #7 (10 points): There exists a value i for which $f_i \le 10^{16}$ and $f_j = 0$ (for all $j \neq i$)

Subtask #8 (23 points): No additional constraints

Examples

standard input	standard output
2 4 0 3 0 3 5 4 1 0 2 0	4 10

Note

In the first example, initially, $S = \{1, 1, 1, 3, 3, 3\}$ and our goal is to have 4 in S. We can do the following:

1. choose $T = \{\}$ then S becomes $\{0, 1, 1, 1, 3, 3, 3\}$

2. choose $T = \{0, 1, 3\}$ then S becomes $\{1, 1, 2, 3, 3\}$

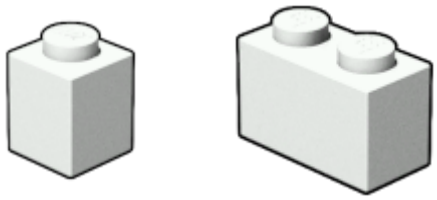
3. choose $T = \{1\}$ then S becomes $\{0, 1, 2, 3, 3\}$

4. choose $T = \{0, 1, 2, 3\}$ then S becomes $\{3, 4\}$

Lego Wall

Problem Name	Lego Wall
Input file	standard input
Output file	standard output
Time limit	3 seconds
Memory limit	256 megabytes

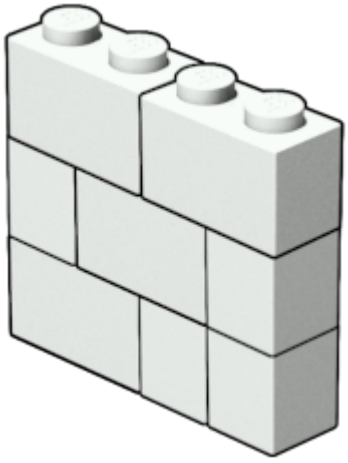
There are two kinds of lego bricks, characterized by their dimensions: 1 × 1 × 1 and 2 × 1 × 1 (width, height and depth, respectively, as shown below). You have an infinite supply of each of them, and within each kind, they are indistinguishable.



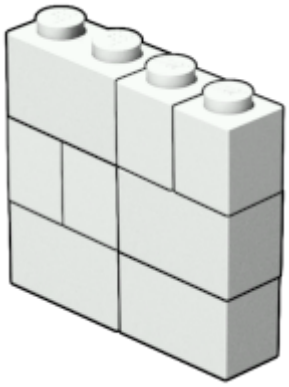
A lego brick is always used in upright position. The faces on the sides are made from identical material and are indistinguishable, except their dimensions.

We consider two lego bricks to be **locked** if one is directly above the other. Two bricks b_0 and b_k are said to be **connected** if there is a sequence of bricks $b_0, b_1, \ldots, b_k$ such that bricks b_{i-1} and b_i are locked for all i such that $1 \le i \le k$. We consider an arrangement of bricks **connected** if each pair of bricks within this arrangement is connected.

You would like to build a thin rectangular wall with width w and height h (and depth 1) such that the wall contains **no holes** and its brick arrangement is **connected**. As an example, below is a such a lego wall of width 4 and height 3:



On the other hand, the following 4 × 3 lego wall is **not** connected, and therefore is not desired:



How many ways of building a **connected** wall **with no holes** are there? Since this number may be large, output it modulo 1 000 000 007.

Note that the mirrored (180-degree rotated) version of a lego wall is considered to be a different wall, unless the mirrored wall looks identical to the original wall.

Input

The input consists of a single line containing two space separated integers w and h ($1 \le w \le 250\,000$, $2 \le h \le 250\,000$, $w \times h \le 500\,000$) – the width and the height of the wall, respectively.

Output

Output a single integer – the number of hole-free connected lego walls of dimensions $w \times h$, modulo 1 000 000 007.

Scoring

Subtask 1 (14 points): $w = 2$.

Subtask 2 (12 points): $h = 2$.

Subtask 3 (18 points): $w, h \leq 100$.

Subtask 4 (30 points): $w \leq 700$.

Subtask 5 (20 points): $h \leq 700$.

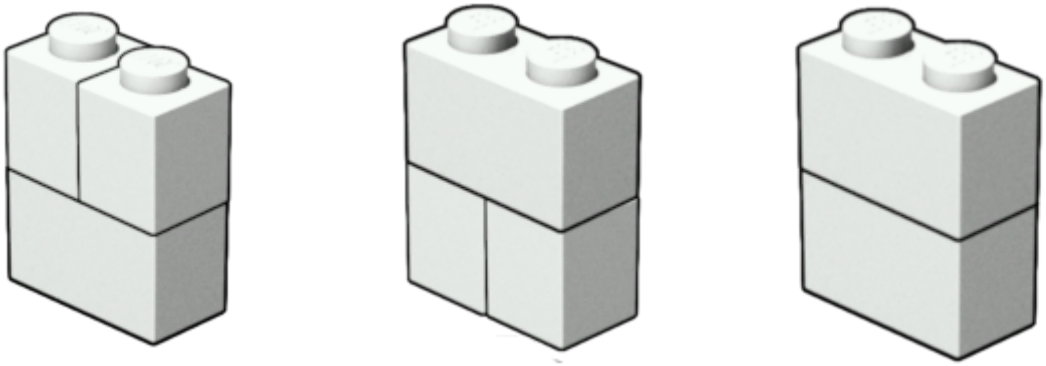
Subtask 6 (6 points): no additional constraints.

Examples

Input	Output
2 2	3
3 3	12
5 7	1436232

Explanation for the first input

The three connected 2 × 2 walls one can build are:



Social Engineering

Problem Name	Social Engineering
Input file	Interactive task
Output file	Interactive task
Time limit	5 seconds
Memory limit	256 megabytes

A social network consists of an undirected connected graph with n vertices and m edges, where each vertex is a person, and two people are friends, if there is an edge between them.
Maria is a member of this social network. She likes challenging her friends to do various things. This means that she first performs some simple task, and then challenges one of her friends to do the same. This friend will then challenge one of their friends, who will challenge one of their friends, and so on. It could happen that the same person gets challenged more than once, but each unordered pair of friends can only take part in the challenge at most once. (Once a person A challenges a person B, then neither person A nor person B can challenge the other again.) In other words, the challenges will be a walk in the graph that never uses an edge more than once.
A person loses the challenge if it is their turn and they cannot challenge any of their friends. The challenges are always started by Maria, and she rarely loses. Now the remaining $n - 1$ people have decided to collaborate in order to make Maria lose the next challenge, and it is your job to coordinate this effort.

Implementation

You must implement a function:

```
void SocialEngineering(int n, int m, vector<pair<int,int>> edges);
```

that plays the game on a graph with n vertices and m edges. This function will be called once by the grader. The list `edges` will contain exactly m pairs of integers (u, v), meaning that an edge goes between vertex u and vertex v. Vertices are numbered from 1 to n. Maria is always vertex 1. Your function can make calls to the following functions:

```
int GetMove();
```

This method should be called whenever it is Maria's turn, such as in the beginning of the game. If

you call this method when it is not Maria's turn, you will get the verdict `Wrong Answer`. The method can return one of the following values:

- an integer v, where $2 \leq v \leq n$. This means that Maria challenges the person with index v. This will always be a legal move.
- 0, if Maria resigns the game. Maria will always resign, if she has no legal moves. When this happens, your program should let the function `SocialEngineering` return, and you will get the verdict `Accepted`.

```
void MakeMove(int v);
```

This method should be called whenever it is not Maria's turn. This means that the person with the turn challenges the person v. If this is not a legal move or if it is Maria's turn at the time of the call, then you will get the verdict `Wrong Answer`.

If Maria has a winning strategy at the start of the game, your program should let `SocialEngineering` return *before* the first call to `GetMove()`. You will then get the verdict `Accepted`.

Constraints

- $2 \leq n \leq 2 \cdot 10^5$.
- $1 \leq m \leq 4 \cdot 10^5$.
- The graph is connected. Every unordered pair of vertices will appear at most once as an edge, and every edge will go between two different vertices.

Subtasks

Maria will always play perfectly in the sense that she will make winning moves whenever she has a winning strategy. If she does not have a winning strategy, then she will try to lure your program into making a mistake, in various clever ways. She will only resign if she has no legal moves, except in Subtask 3.

1. (15 points) $n, m \leq 10$.
2. (15 points) Everyone except Maria has at most 2 friends.
3. (20 points) Maria will resign immediately unless she has a winning strategy.
4. (25 points) $n, m \leq 100$.
5. (25 points) No further constraints.

Sample Interaction

Your action	Grader action	Explanation
-	`SocialEngineering(5, 6, {{1,4}, {1,5}, {2,4}, {2,5}, {2,3}, {3,5}})`	`SocialEngineering` is called with a graph with 5 vertices and 6 edges.
`GetMove()`	`Returns 4`	Maria challenges person 4.
`MakeMove(2)`	-	Person 4 challenges person 2.
`MakeMove(5)`	-	Person 2 challenges person 5.
`MakeMove(1)`	-	Person 5 challenges Maria.
`GetMove()`	`Returns 0`	Maria has no legal moves, so she resigns.
`Returns`	-	You have won the game, and should let `SocialEngineering` return.

Your action	Grader action	Explanation
-	`SocialEngineering(2, 1, {{1,2}})`	`SocialEngineering` is called with a graph with 2 vertices and 1 edge.
`Returns`	-	Maria has a winning strategy on this graph, so you should return without making any `GetMove()` calls to resign.

Sample Grader

The provided sample grader, `grader.cpp`, in the task attachment `SocialEngineering.zip`, reads the input from the standard input in the following format:

- The first line contains the number of vertices, *n*, and the number of edges, *m* in the graph.
- The following *m* lines contains two integers *u* and *v*, indicating that there is an edge between *u* and *v*.

Sample grader reads the input and calls the `SocialEngineering` function in user's solution. Note that the sample grader does not implement Maria's winning strategy and provided only for a sample interaction.
To compile the sample grader with your solution, you may use the following command at the

terminal prompt:

```
g++ -std=gnu++11 -O2 -o solution grader.cpp solution.cpp
```

where `solution.cpp` is your solution file to be submitted to CMS. To run the program with the sample input provided in the attachment, type the following command into the terminal prompt:

```
./solution < input.txt
```

Tourists

Problem Name	Tourists
Input file	standard input
Output file	standard output
Time limit	4 seconds
Memory limit	256 megabytes

There are n cities in Utopia, numbered from 1 through n. There are also $n-1$ two-way roads connecting the cities. It is possible to travel between each pair of cities using only these roads. Because Utopia is very beautiful, there are m tourists, numbered from 1 through m, who are currently visiting this country. Initially, the i^{th} tourist is visiting city a_i. It is possible that multiple tourists are in the same city; that is, it can be that $a_i = a_j$ for a pair i, j such that $i \neq j$.

Each tourist has an opinion on how interesting their current visit in Utopia is, represented as a number. Initially each tourist's opinion is 0. In order to encourage further visits, Utopian government wants to increase the tourists' opinion of the country by organizing events in selected cities. When an event is held in city c, all tourists who are currently staying there will have their opinion increased by d, where d is a value depending on the type of the event.

Some of the tourists have planned to travel between cities during their stay in Utopia. Although traveling from one city to another takes almost no time (thanks to efficient Utopian roads), it still is an inconvenience and thus results in a lower tourist opinion. To be exact, a tourist who traveled a path consisting of k roads will have their opinion decreased by k (tourists will always choose the shortest path between two cities).

You are asked by the Utopian government to track the tourists' opinions, as they travel through the country. As part of this request, you will be given q queries as part of the input. You are supposed to carry out and answer all queries in the order they appear in the input.

Input

The first line contains three integers n, m, q ($2 \leq n \leq 200\ 000$, $1 \leq m, q \leq 200\ 000$) - number of cities, tourists and queries, respectively.

The second line contains m integers $a_1, a_2, ..., a_m$ ($1 \leq a_i \leq n$), where a_i represents the starting city of the i^{th} tourist.

Next $n-1$ lines contain 2 integers each: v_i and w_i ($1 \leq v_i$, $w_i \leq n$, $v_i \neq w_i$) implying that there exists a road between city v_i and w_i.

Next q lines describe queries in the order they are asked. Each line is in one of the following three forms:

- The letter 't' followed by three integers f_i, g_i, c_i ($1 \leq f_i \leq g_i \leq m$, $1 \leq c_i \leq n$), meaning that all tourists with numbers from f_i to g_i (inclusively) travel to city c_i. Those who are already in city c_i do not move, and their opinion does not change.
- The letter 'e' followed by two integers c_i, d_i ($1 \leq c_i \leq n$, $0 \leq d_i \leq 10^9$), meaning that in city c_i, an event is being held that increases the tourists' opinion by d_i.
- The letter 'q' followed by one integer v_i ($1 \leq v_i \leq m$), representing a question about the current opinion of the tourist v_i.

It is guaranteed that there is at least one 'q' query in the input.

Output

Print the answer for all 'q' queries, each in a separate line, in the order they were asked.

Scoring

Subtask 1 (10 points): $n, m, q \leq 200$
Subtask 2 (15 points): $n, m, q \leq 2\,000$
Subtask 3 (25 points): $m, q \leq 2\,000$
Subtask 4 (25 points): No 'e' queries
Subtask 5 (25 points): No additional constraints

Example Input

8 4 11
1 4 8 1
6 4
6 3
3 7
6 5
5 1
1 2
1 8

```
q 4
t 3 4 5
t 2 2 7
q 4
e 5 10
e 1 5
q 4
t 1 1 5
t 2 2 1
q 1
q 2
```

Example Output

```
0
-1
9
4
-7
```