Problem A. Simple Game

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

Alice and Bob like playing games. Today they play on a special grid with 2 rows and n columns. Alice and Bob each have a chess piece, starting at $(1, 1)$ and $(2, n)$, respectively. They will take turns moving their chess piece, with Alice going first.

In each turn, they can choose to stay still or move the chess piece to any point adjacent horizontally or vertically which has not been visited by the other's piece. The game will end after $10^{10^{10}}$ turns.

Every point in this grid has a non-negative weight. For each player, the score that he/she gets is the sum of the weights of all the points his/her chess piece has visited. The weight is counted only once, even if the piece visited a point multiple times.

Both Alice and Bob want to maximize the score that they get. As a spectator, you want to know the score that Alice gets if both of them play optimally.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 5 \cdot 10^4$). The test cases follow.

The first line of each test case contains a single integer n representing the size of the grid ($1 \le n \le 10^5$).

The second line of each test case contains n integers $a_{1,1}, a_{1,2}, \ldots, a_{1,n}$. The i-th of them represents the weight of point $(1, i)$.

The third line of each test case contains n integers $a_{2,1}, a_{2,2}, \ldots, a_{2,n}$. The i-th of them represents the weight of point $(2, i)$.

It is guaranteed that $0 \le a_{i,j} \le 10^9$, and the sum of n across all test cases will not exceed $2.5 \cdot 10^5$.

Output

For each test case, print a line with a single integer: the score that Alice gets if both players play optimally.

Example

standard input	*standard output*
4 2 1 4 2 1 3 1 1 4 5 1 4 4 1 9 4 9 1 0 0 1 7 3 1 4 1 5 9 2 6 5 3 5 8 9 8	5 6 15 25

Problem B. Painting the Roads

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

You are the king of the Pigeon Kingdom. The Pigeon Kingdom consists of n cities and $n-1$ two-way roads, each connecting a pair of cities. It is guaranteed that it is possible to traverse between any two cities through the roads.

Each road has a color, black or white, and a length ℓ_i. Initially, each road is white. However, you think that it is too boring this way. So you have decided to assign m robots to m cities to paint the roads to your favorite color pattern. Different robots **can** be assigned to the same city. Robot i starts at city p_i, travels through some roads (possibly none), and then stops. A robot **cannot** travel through one road multiple times. When a robot travels through a road, it flips the color of the road (if it was white, it turns black, and vice versa). The robots are independent, which means that they will not interfere with each other in the travel. We assume that different robots don't paint any road simultaneously. Also, different robots **can** stop in the same city. The cost of a robot's travel is defined as the sum of lengths of all the roads on its path.

As the king of the Pigeon Kingdom, you want to minimize the total cost of all the travels. If it is impossible to paint the roads to the desired pattern with the m robots, print -1 instead.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 5000$). The test cases follow.

The first line of each test case contains two integers n and m ($2 \le n \le 5000$ and $1 \le m \le 5000$), denoting the number of cities and the number of robots, respectively.

Each of the next $n-1$ lines contains four integers u_i, v_i, ℓ_i, c_i ($1 \le u_i < v_i \le n$; $1 \le \ell_i \le 10$; $c_i = 0$ or $c_i = 1$), denoting a road of length ℓ_i connecting cities u_i and v_i. If $c_i = 0$, you should paint it white in the desired pattern; otherwise, you should paint it black. It is guaranteed that it is possible to traverse between any pair of cities through the given roads.

Then a single line contains m integers p_j ($1 \le p_j \le n$), denoting the starting city for each robot.

It is guaranteed that the of sum of n over all test cases will not exceed 5000, and the sum of m over all test cases will not exceed 5000.

Output

For each test case, print one line containing a single integer: the minimal total cost to paint all the roads to the desired pattern. If it is impossible to do so, print -1 instead.

Example

standard input	*standard output*
5 3 2 1 2 1 1 2 3 2 1 1 3 4 2 1 2 3 1 2 3 1 0 3 4 4 1 1 2 5 4 1 2 3 0 2 3 1 1 3 4 2 0 4 5 2 1 1 1 1 1 5 2 1 2 2 1 1 3 3 0 1 5 2 1 3 4 1 1 1 2 10 5 1 2 10 1 2 3 3 1 3 4 4 0 4 5 4 1 5 6 2 1 2 7 8 0 2 8 9 1 4 9 1 0 1 10 4 0 10 10 2 1 8	3 9 21 -1 42

Problem C. Too Many Edges

Input file:	*standard input*
Output file:	*standard output*
Time limit:	15 seconds
Memory limit:	512 mebibytes

This is an interactive problem. You have to use the `flush` *operation right after printing each line. For example, you can use the function* `fflush(stdout)` *for C or C++,* `System.out.flush()` *for Java,* `flush(output)` *for Pascal, and* `sys.stdout.flush()` *for Python.*

You are an ordinary employee of a data analysis department. However, it seems that your colleagues are geniuses at making trouble. Just now they made some trouble again.

They downloaded a large graph from the remote server and wrote a program to analyze the graph. They nearly finished the analysis and thought the graph should be no longer useful. So they let their program add some new edges to the graph without any backups. But then they changed their mind and want to compute the longest path in the original graph. Downloading the graph again costs too much time. Now it is your time to save the day.

The original graph on the remote server is a directed acyclic graph $G = (V, E)$. All edges are unit length. The input of your program is the modified version $G' = (V, E')$. It is guaranteed that $E \subseteq E'$, **and G' is a directed acyclic graph**.

Your program should output $\ell(G)$, the length of the longest path in G. The length of a path equals the number of edges in the path.

To test whether an edge belongs to the original graph, your program can make queries to the remote server. To query the existence of edge $u \to v$, output "`? u v`". Flush the output stream after printing each query. The remote server will respond `1` if edge $u \to v$ belongs to the original graph, and `0` otherwise.

After your program gets the answer, print "`! ans`", where $ans = \ell(G)$, and **terminate your program normally** immediately after flushing the output stream.

Your program is allowed to make no more than $(|E'| - |E| + 1) \cdot (\ell(G) + 1)$ queries (not including printing the answer) to the remote server, although $|E|$ and $\ell(G)$ are unknown to you.

Input

Use standard input to read the responses to the queries.

The first line contains two integers n and m ($1 \le n \le 5 \cdot 10^4$; $1 \le m \le 10^5$): the number of vertices and edges of graph G'.

Each of the next m lines contains two integers u and v ($1 \le u, v \le n$) specifying a directed edge $u \to v$ in graph G'. No edge appears more than once.

It is guaranteed that $0 \le |E'| - |E| \le 200$.

The following lines will contain responses to your queries. Each response is either "`0`" or "`1`". The i-th of these lines is a response to your i-th query.

After answering $(|E'| - |E| + 1) \cdot (\ell(G) + 1)$ queries, the remote server no longer responds.

The testing system will allow you to read the response to a query only after your program prints the query and performs the `flush` operation.

Output

To make the queries, your program must use standard output.

Your program must print the queries in the form of "`? u v`", one query per line. Do not forget to end the line after each query. Your program must guarantee that $1 \le u, v \le n$. After printing each line your program must perform the `flush` operation.

The response to the query will be given in the standard input after you flush the output. In case your program finds the answer, print a line "! *ans*", where $ans = \ell(G)$, and terminate your program.

Example

standard input	*standard output*
5 5 1 2 1 3 2 5 3 4 4 5 1 1 1 0 1	? 1 2 ? 1 3 ? 2 5 ? 3 4 ? 4 5 ! 2

Problem D. HearthStone

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

Alice loves playing HearthStone! She loves the hero class of Warlock, who can cast the spell named Defile.

When cast, Defile deals 1 unit of damage to the health of all minions. If any minion dies, Defile will be cast again automatically. Importantly, if two or more minions die simultaneously, it still causes a single Defile cast. That, in turn, may kill other minions, causing Defile to be cast again, and so on.

The health of each minion is a nonnegative integer. A minion dies when their health becomes zero. If a minion dies, it will disappear. It will not die twice.

Now there are n minions. Before casting Defile, Alice can make zero or more steps. In each step, Alice changes a single minion's health by one. That is to say, if the health of a minion is x, Alice can change it to $x - 1$ or $x + 1$.

Alice wants to know the minimum number of steps such that, after these steps, she can cast a single Defile to kill all the minions.

Input

The first line contains a single integer n ($1 \le n \le 10^6$).

The next line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$), the health of the n minions.

Output

Print one integer: the minimum number of steps before Alice can cast a single Defile to kill all the minions.

Example

standard input	*standard output*
6 4 6 8 9 2 4	12

Problem E. Billiard

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

There is a table with length n and width m.

A billiard ball begins to move from one corner with an angle of 45 degrees.

When will the ball bounce back to where it starts?

Formally, you are given n and m, and you need to calculate the return value of the following function.

```
int64_t check(int n, int m) {
  int x = 0, y = 0;
  int dx = 1, dy = 1;
  int64_t t = 0;
  while (1) {
    if (x + dx < 0) dx *= -1;
    if (x + dx > n) dx *= -1;
    if (y + dy < 0) dy *= -1;
    if (y + dy > m) dy *= -1;
    x += dx;
    y += dy;
    ++t;
    if (x == 0 && y == 0) break;
  }
  return t;
}
```

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 10^5$). The test cases follow.

Each test case is described by a single line containing two integers n and m ($2 \le n, m \le 10^9$).

Output

For each test case, output a line containing one integer: the answer to the problem.

Example

standard input	*standard output*
5 2 2 2 3 2 4 2 5 2 6	4 12 8 20 12

Problem F. StrCartesian

Input file:	*standard input*
Output file:	*standard output*
Time limit:	13 seconds
Memory limit:	768 mebibytes

Given are two sets of strings $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$. Define a sequence of $n \cdot m$ pairwise concatenations of a_i and b_j:

$$S = (a_1b_1, a_1b_2, \ldots, a_1b_m, a_2b_1, a_2b_2, \ldots, a_2b_m, \ldots, a_nb_1, a_nb_2, \ldots, a_nb_m).$$

Now sort the sequence S lexicographically, and let the sorted sequence be $C = (c_1, c_2, \ldots, c_{n \cdot m})$.

We want to know the sequence C, but it is too large. So we make q queries to your program, and the i-th query asks for c_{k_i}.

However, c_{k_i} is still too long to output. If the answer equals $c = a_f + b_s$, then your program only needs to output the pair (f, s).

Input

The first line contains two integers n and m ($1 \le n, m \le 5 \cdot 10^4$), the sizes of sets A and set B.

The following n lines contain n **distinct** non-empty strings $a_1, a_2, \ldots, a_n$.

The total length of strings in set A does not exceed 10^6.

The following m lines contain m **distinct** non-empty strings $b_1, b_2, \ldots, b_m$.

The total length of strings in set B does not exceed 10^6.

All strings consist of lowercase English letters.

The next line contains one integer q ($1 \le q \le 1000$), the number of queries.

In the following q lines, the i-th line contains an integer k_i ($1 \le k_i \le n \cdot m$), specifying that the query asks for the k_i-th element of C.

Output

Print q lines. The i-th line must contain two integers f_i and s_i ($1 \le f_i \le n$; $1 \le s_i \le m$) specifying that the answer c_{k_i} equals to $a_{f_i}b_{s_i}$. If there are multiple correct answers, your program may output any one of them.

Example

standard input	*standard output*
2 3 a ab a aa ba 2 3 4	2 1 1 3

Problem G. Ald

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

You are given a tree. The tree has n vertices, labeled from 1 to n.

Let us denote the path between vertices a and b as (a, b). Let the d-set of a path be the set of vertices on the tree located within a distance $\le d$ from at least one vertex of the path. For example, the 0-set of a path is the set of its vertices. The distance between vertices is the number of edges on the path between these vertices.

Let S be a multiset of tree paths. Initially, S is empty. Your task is to process the following queries:

- "`1 u v`": add path (u, v) into S $(1 \le u, v \le n)$.
- "`2 u v`": delete a single path (u, v) from S $(1 \le u, v \le n)$. Note that (u, v) and (v, u) denote the same path. For example, if $S = \{(2, 3), (2, 3)\}$, then after a query "`2 3 2`", we will have $S = \{(2, 3)\}$. Before this query, it is guaranteed that at least one path (u, v) or (v, u) is present in S.
- "`3 d`": print the size of intersection of d-sets of all paths from S $(0 \le d \le n)$. If S is empty, print 0.

Input

The first line contains an integer t, the number of test cases $(1 \le t \le 10^4)$. The test cases follow.

The first line of each test case contains two integers n and q $(1 \le n, q \le 10^5)$, the number of vertices in the tree and the number of queries.

Each of the following $n - 1$ lines contains two integers u_i and v_i: indices of vertices connected by the i-th edge of the tree $(1 \le u_i, v_i \le n)$.

The following q lines contain queries in the format described in the statement.

The sum of n over all test cases does not exceed 10^5. The sum of q over all test cases does not exceed 10^5.

Output

For each query of the third type, output a single line with the answer.

Example

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

Problem H. Fast Debugger

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

Recently you received an old computer. This old computer has an 8-bit CPU with four registers: `ax`, `bx`, `cx`, `dx`, and only supports some simple instructions. For those who are not familiar with assembly language, here is a programming guide.

This CPU has four 8-bit registers: `ax`, `bx`, `cx`, `dx`. You can treat them as variables storing integers within $[0, 255]$.

Only three kinds of bitwise operation are supported by this CPU, bitwise-or/and/xor. Each bitwise operation has two kinds of instructions.

Type 1: Both operands are registers, written as "`or r1 r2`", where both `r1` and `r2` are one of the register names `ax`, `bx`, `cx`, `dx` (`r1` and `r2` may refer to the same register). This instruction will set the value of `r1` to the result of `r1 bitwise-or r2`. (Similarly, bitwise-and and bitwise-xor instructions are written as "`and r1 r2`" and "`xor r1 r2`").

Type 2: One of the operands is immediate, written as "`ori r imm`", where `r` is one of the register names `ax`, `bx`, `cx`, `dx`, and `imm` is a constant in $[0, 255]$ given in the instruction. This instruction will set the value of `r` to the result of `r bitwise-or imm`. (Similarly, bitwise-and and bitwise-xor instructions are written as "`andi r imm`" and "`xori r imm`").

Loops are not supported by the CPU, but the assembler implemented an easy loop for programmers. If the assembler sees a "`repeat` m" statement, it will automatically repeat the contents of the repeat block, a total of m times. The format is shown below.

```
repeat m
<repeat block>
end
```

Here, m is a constant in $[2, 255]$ given in the statement, and `<repeat block>` consists of one or more statements that can be either bitwise instructions or repeat–end statements.

Now you want to write a simulator on your new laptop which is much faster than the old computer.

Your simulator will be given a valid program and q queries. Each query consists of five integers k, a_0, b_0, c_0, d_0. Initially, the registers are set to the given values: `ax` $= a_0$, `bx` $= b_0$, `cx` $= c_0$, `dx` $= d_0$. You should output the values of registers `ax`, `bx`, `cx`, `dx` after the program executes k bitwise instructions.

Input

The first line of input contains two integers n and q ($1 \le n \le 12\,000$; $1 \le q \le 10\,000$), denoting the number of instructions and the number of queries.

Then follow n lines. Each line is an instruction. The format is described above.

Each of the following q lines contains five integers k, a_0, b_0, c_0, d_0 ($1 \le k \le 10^9$; $0 \le a_0, b_0, c_0, d_0 \le 255$), denoting a query for the value of registers after evaluating k bitwise operations. It is guaranteed that the program does not terminate before executing k bitwise instructions.

Output

Output q lines. The i-th line must contain four integers a_k, b_k, c_k, d_k, denoting the values of registers `ax`, `bx`, `cx`, `dx` after the program executes k bitwise instructions.

Example

standard input	*standard output*
6 2 repeat 5 xor ax bx xori ax 3 and cx ax xor cx dx end 10 1 2 4 3 8 4 1 2 3	0 2 2 3 4 1 3 3

Problem I. 01tree

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

There is a tree with n nodes. Every node has a value of 0 or 1.

In one second, you can choose two adjacent nodes with the same value and flip both values.

Given some starting state and some ending state, you will always spend the least number of seconds transforming the starting state into the ending state. If it is impossible to transform the starting state into the ending state, you just skip it (so you spend 0 seconds).

The issue is that, for some nodes, you do not remember the value on them (in either the starting state, the ending state, or both). Over all pairs of (starting state, ending state) that are consistent with your memory, find the total amount of time that it will take to transform from the starting state to the ending state. Print this value modulo $10^9 + 7$.

Input

The first line contains an integer t, the number of test cases ($1 \le t \le 1000$). The test cases follow.

The first line of each test case contains one integer n ($2 \le n \le 10^5$) denoting the size of the tree.

Then $n - 1$ lines follow, each containing two integers u and v, denoting the edge that connects u and v in the tree.

The following line contains a string s of length n consisting of characters "0", "1", and "?". This string denotes your memory of the starting state: "0" and "1" represent the value of the node, and "?" represents that you do not remember the value of the node.

The following line contains a string t of length n denoting your memory of the ending state. It follows the same format as the starting state.

It is guaranteed that the sum of n over all test cases does not exceed 10^5.

Output

For each test case, print a line with a single integer: the answer modulo $10^9 + 7$.

Example

standard input	*standard output*
3 2 1 2 00 11 3 1 2 2 3 ??? ??? 3 1 2 2 3 ??1 0?0	1 16 1

Problem J. Independent Set

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

This is an interactive problem. You have to use the ***flush*** *operation right after printing each line. For example, you can use the function* ***fflush(stdout)*** *for C or C++,* ***System.out.flush()*** *for Java,* ***flush(output)*** *for Pascal, and* ***sys.stdout.flush()*** *for Python.*

An independent set in a graph is a set of vertices such that, for every two vertices in the set, there is no edge connecting them.

There is an algorithm to construct an independent set in the graph.

The algorithm receives a sequence of vertices. Suppose the sequence is $a_1, a_2, \ldots, a_\ell$. Initially, there is an empty set S and a sequence $cnt_1, cnt_2, \ldots, cnt_\ell$ that satisfies $cnt_1 = cnt_2 = \ldots = cnt_\ell = 0$.

Then for i from 1 to ℓ:

- Look at every edge in the graph, and increment $cnt_i = cnt_i + 1$ every time when the edge connects a_i and a vertex in S.
- If $cnt_i = 0$ after searching, insert a_i into S.

Obviously, S will be an independent set after the algorithm.

Now, you only know the number of vertices n in the graph, and you want to find all its edges. There is an interactor to help you. The interactor receives a sequence of vertices, runs the algorithm above, and returns the sequence $cnt_1, cnt_2, \ldots, cnt_\ell$.

You want to achieve your goal with the total length of the sequences that you feed no more than 176 000.

Note that there may be multiple edges and self-loops in the graph.

Input

The first line contains a single integer n, the number of vertices in the graph.

Suppose m is the number of edges in the graph. It is guaranteed that $1 \le n \le 4000$ and $0 \le m \le 10\,000$.

Interaction Protocol

To feed a sequence $a_1, a_2, \ldots, a_k$ to the interactor, print a single line formatted as "`?` k a_1 a_2 ... a_k" ($k \ge 1$). Then you should read a single line with the answer, which is formatted as "cnt_1 cnt_2 ... cnt_k".

If you have found all the edges in the graph, print a single line formatted as "`!` m x_1 y_1 x_2 y_2 ... x_m y_m". The edges (x_1, y_1), (x_2, y_2), ..., (x_m, y_m) denote your answer. You can print the edges in any order.

After printing each line, your program must perform the `flush` operation.

Example

standard input	*standard output*
4 0 2 1 0 1 0 0 1 1 0 0	? 6 1 2 3 1 3 4 ? 5 4 4 4 2 3 ! 4 1 2 1 2 1 3 4 4

Problem K. Planar Graph

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

There are n base points on the plane, with some segments connecting them.

It is guaranteed that every two base points do not coincide, every three base points are not collinear, and segments only intersect at endpoints.

There are also m source points on the plane.

It is guaranteed that every source point is different from each of the n base points and does not lie on any segment.

The question is, for each segment, whether you can draw a curve from some source point to the midpoint of this segment without intersecting any other segments.

Input

The first line contains three integers n, m, e ($1 \le n, m \le 100$; $0 \le e \le 300$), denoting the number of base points, source points, and segments.

Each line of the next n lines contains two integers x, y ($|x|, |y| \le 10^9$), denoting a base point (x, y).

Each line of the next m lines contains two integers x, y ($|x|, |y| \le 10^9$), denoting a source point (x, y).

Each line of the next e lines contains two integers i, j, denoting a segment connecting the i-th base point and the j-th base point ($1 \le i < j \le n$).

Output

Print a string of length e. The i-th character of the string must be "1" if you can draw a curve from some source point to the midpoint of the i-th segment without intersecting any other segments, or "0" otherwise.

Example

standard input	*standard output*
4 1 3 -2 0 0 2 2 0 0 1 0 3 1 2 2 3 1 3	111