Problem F
Erin Needs Cake
Time Limit: 3 Seconds

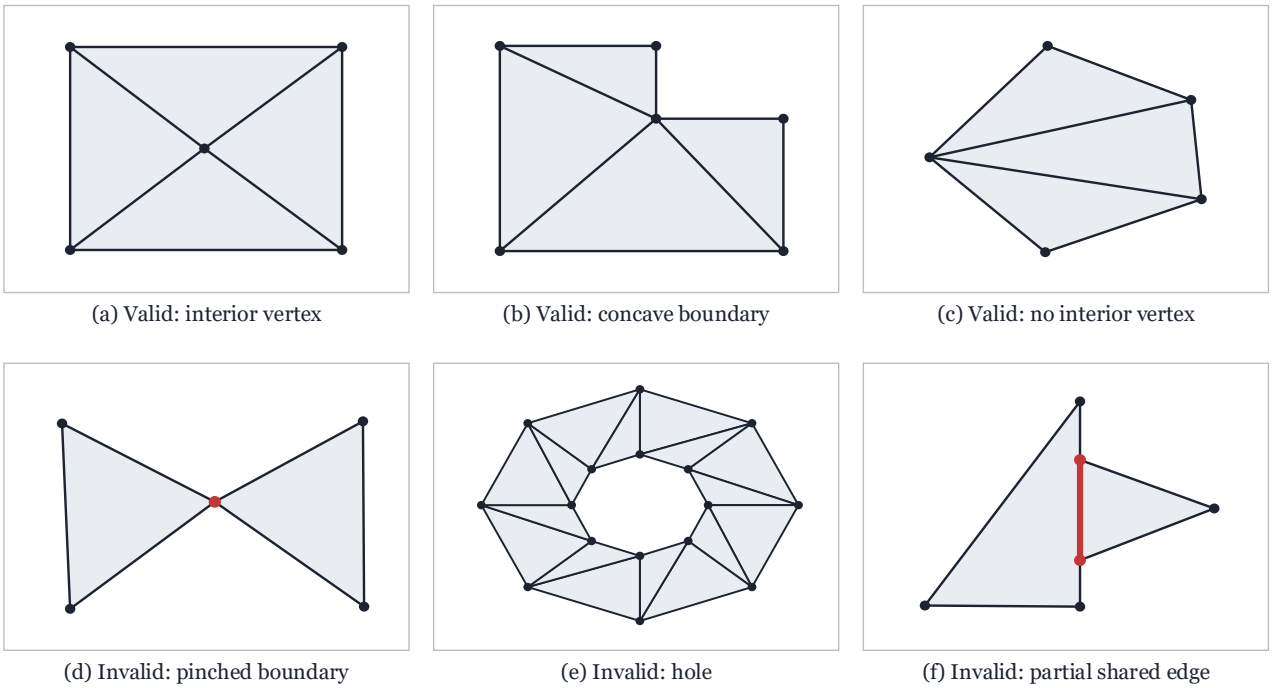


Erin, the queen of the fairies in *Elias*, had a cake she was saving for later. To keep anyone else from eating it, she hid it in a secret place. Unfortunately, she hid it so well that she has now forgotten where it is.

Fortunately, Erin left a locator charm with her map. The map is divided into triangular regions, and the cake is hidden strictly inside exactly one of them, never on a road or a vertex. Trace a closed loop along the map's roads, and the charm will tell you whether the cake is inside the loop.

Each triangle has positive area, and distinct triangles have disjoint interiors. Together, the triangles form a single simple polygon without holes. In particular, the outer boundary is a single simple cycle, so no vertex appears more than once along the boundary. Every vertex belongs to at least one triangle. Each triangle side is a road; a side shared by two triangles is a single road. Distinct roads meet only at common endpoints. The outer boundary need **not** be convex, and vertices may lie strictly inside it.

Figure 1 shows examples of valid and invalid maps.



(a) Valid: interior vertex
(b) Valid: concave boundary
(c) Valid: no interior vertex
(d) Invalid: pinched boundary
(e) Invalid: hole
(f) Invalid: partial shared edge

Figure 1. Examples of valid and invalid maps.

Help Erin identify the triangular region containing her cake using at most $\lceil \log_2 n \rceil$ queries, where n is the number of regions. Here, $\lceil x \rceil$ denotes the smallest integer greater than or equal to x.

Interaction

This is an interactive problem. Your submitted program will interact with an interactor inside the grading server, which reads input from and writes output to your program. Your program must gather information by printing queries and reading the interactor's replies. After every line of output, your program must immediately flush the output.

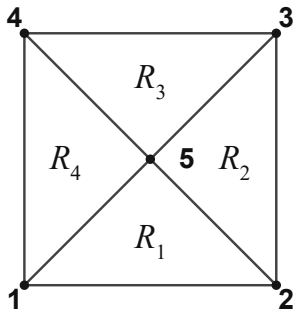
Initial Input

The first line contains two integers n and k: the number of triangular regions and the number of vertices, respectively ($2 \le n \le 2^{18}$; $4 \le k \le n + 2$).

Vertices are numbered from 1 to k, and triangular regions are numbered from 1 to n.

The next k lines describe the vertices. For each i ($1 \le i \le k$), the i-th of these lines contains two integers x_i and y_i ($-10^9 \le x_i, y_i \le 10^9$), the coordinates of vertex i. All vertices have distinct coordinates.

The next n lines describe the triangular regions. For each i ($1 \le i \le n$), the i-th of these lines contains three distinct integers u_i, v_i, and w_i ($1 \le u_i, v_i, w_i \le k$), the vertices of region i.



Black dots are vertices. Bold numbers label vertices; R_1–R_4 label triangular regions.

Region	Vertices in the input
R_1	$1, 2, 5$
R_2	$2, 3, 5$
R_3	$3, 4, 5$
R_4	$4, 1, 5$

Here $n = 4$ and $k = 5$. Vertex 5 is at $(2, 2)$, inside the map. The area outside the square is not a region.

Figure 2. The map in Sample Interaction 1, drawn from its input coordinates.

Queries

Your program may repeatedly output exactly one of the following commands until the interactor outputs "`correct`" or "`wrong`". Every number and special character ("`?`", "`!`") used in the query should be separated by a single space.

- "`?` l p_1 p_2 ... p_l"
  - Ask whether the cake is inside the polygon whose boundary follows vertices $p_1, p_2, \ldots, p_l$ in that order and then returns to p_1.
  - Here $3 \le l \le k$, and $p_1, p_2, \ldots, p_l$ must be **distinct** vertex indices. Every consecutive pair, including $\{p_l, p_1\}$ must be joined by a road. The boundary must be one simple loop: it cannot cross itself or visit a vertex twice. Either direction is allowed. Do **not** repeat p_1 at the end; the closing road is implicit. A straight path through another vertex is **not** a single road.
- "`!` a"
  - Report the region number a containing the cake ($1 \le a \le n$).
  - Flush this command, read the interactor's final reply, and terminate.

Each command must be printed on its own line and immediately flushed. You may ask at most $\lceil \log_2 n \rceil$ queries starting with "`?`". The final "`!`" command does not count toward this limit and must be issued exactly once, unless you have already received "`wrong`".

Interactor Responses

After each command, the interactor responds as follows:

- For a valid "`?`" query, the interactor outputs "`1`" if the cake is inside the chosen polygon, and "`0`" otherwise.
- For a query "`! a`", the interactor outputs "`correct`" if you have successfully identified the hidden region, and "`wrong`" otherwise.
- The interactor immediately terminates the interaction after printing "`wrong`" if any rule is violated — for example, a malformed command, an out-of-range index, or exceeding $\lceil \log_2 n \rceil$ queries starting with "`?`".

The interactor **may be adaptive**. It need not choose a hiding place in advance, but all replies together must be consistent with at least one fixed region. The cake does not move. Your answer must be forced by the replies: guessing one of several still-possible regions is not sufficient.

The time and memory used by the interactor are also included in the calculation of your program's execution time and memory usage. You can assume that the maximum time used by the interactor is 1 second and the maximum amount of memory is 64 MiB.

The following interaction uses the map in Figure 2.

Sample Interaction 1

Read (Interactor's Response)	**Write (Your Command)**

`4 5`
`0 0`
`4 0`
`4 4`
`0 4`
`2 2`
`1 2 5`
`2 3 5`
`3 4 5`
`4 1 5`	

	`? 4 1 2 3 5`
`0`	
	`? 3 3 4 5`
`1`	
	`! 3`
`correct`	

To flush, your program needs to do the following right after writing a command and a newline:

- `fflush(stdout)` in C;
- `std::cout << std::flush` in C++;
- `System.out.flush()` in Java or Kotlin;
- `sys.stdout.flush()` in Python.

A testing tool is provided to help you develop your solution, so it is not mandatory to use it. If you want to use it, you can download the attachment `testing_tool.py` from the *DOMjudge Problemset* page. You can run the testing tool to see how your program interacts with the interactor on a specific map and a fixed hiding region for the cake. **This testing tool can be used regardless of the language of your program as follows.** The same explanation is also included in the comments of `testing_tool.py`.

Usage: `python3 testing_tool.py -f <inputfile> -- <program>`

Use the `-f` parameter to specify the input file, e.g. `input.in`.

Format of the input file: The first line contains two integers n and k. For each i ($1 \leq i \leq k$), the i-th of the next k lines contains two integers x_i and y_i. For each i ($1 \leq i \leq n$), the i-th of the next n lines contains three integers u_i, v_i, and w_i. The last line contains an integer h ($1 \leq h \leq n$), indicating the index of the region where the cake is hidden.

The final integer h is used only by the testing tool and is not provided to your program.

Example: The map from the sample interaction is represented as follows.

```
4 5
0 0
4 0
4 4
0 4
2 2
1 2 5
2 3 5
3 4 5
4 1 5
3
```

The tool is provided as-is, and you should feel free to make whatever alterations or augmentations you like to it. **Note that it is not guaranteed that a program that passes the testing tool will be accepted**. For example, the testing tool uses a **fixed hiding region** specified in the input file, whereas the grading interactor may respond adaptively.

If you have a C++ solution stored in a file called "`sol.cpp`", you must first compile using "`g++ sol.cpp -o sol`" and then invoke the testing tool with:

```
python3 testing_tool.py -f input.in -- ./sol
```

If you have a Python solution that you would run using "`pypy3 solution.py`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in -- pypy3 solution.py
```

If you have a Java solution that you would run using "`java MyClass`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in -- java MyClass
```

If you have a Kotlin solution that you would run using "`kotlin SolutionKt`", you can invoke the testing tool with:

```
python3 testing_tool.py -f input.in -- kotlin SolutionKt
```