Super Dango Maker

JOI-kun is a professional confectioner making dangos (Japanese dumplings). In JOI-kun's shop, the colors of dangos are specified. There are N colors of dangos, numbered from 1 to N.

A **beautiful dango stick** is a famous item in JOI-kun's shop. A beautiful dango stick is made of N dangos of **different** colors skewered with a stick.

For each of the N colors, JOI-kun has M dangos of that color. Therefore, JOI-kun has $N \times M$ dangos in total. These dangos are numbered from 1 to $N \times M$. Using these dangos and M sticks, JOI-kun wants to make M beautiful skewered dango sticks.

To avoid a mistake about the colors of the dangos, JOI-kun will use a dango checker. If JOI-kun inputs the indices of some dangos, the dango checker answers the maximum number of beautiful dango sticks he can make using the dangos in the input and sufficiently many sticks.

Using the dango checker several times, JOI-kun wants to divide the $N \times M$ dangos into M groups. Every group should consist of N dangos, and contain a dango of each color.

JOI-kun wants to divide the $N \times M$ dangos into M groups using the dango checker at most 50 000 times.

Write a program which, given information of the dangos, implements JOI-kun's strategy to divide the dangos into groups using the the dango checker at most 50 000 times.

Implementation Details

You need to submit one file. The name of the file is `dango3.cpp`. It should implement the following function. The program should include `dango3.h` using the preprocessing directive `#include`.

- `void Solve(int N, int M)`
  This function is called once for each test case.
  - The parameter `N` is the number of colors of dangos N.
  - The parameter `M` is the number of beautiful dango sticks M JOI-kun wants to make.

Your program may call the following functions.

★ `int Query(const std::vector<int> &x)`
Using this function, your program asks questions to the dango checker.
- ◇ The parameter `x` is the list of the indices of the dangos sent to the dango checker.
- ◇ The return value is the maximum number of beautiful dango sticks JOI-kun can make using the dangos specified by the parameter `x` and sufficiently many sticks.
- ◇ Each element of the parameter `x` should be between 1 and $N \times M$, inclusive. If this condition is not satisfied, your program is judged as **Wrong Answer [1]**.
- ◇ The elements of the parameter `x` should be different from each other. If this condition is not satisfied, your program is judged as **Wrong Answer [2]**.
- ◇ It is not allowed to call the function `Query` more than 50 000 times. If the function `Query` is called more than 50 000 times, your program is judged as **Wrong Answer [3]**.

★ `void Answer(const std::vector<int> &a)`
Using this function, your program answers groups of dangos to make beautiful dango sticks.
- ◇ The parameter `a` is the list of the indices of dangos for a beautiful dango stick.
- ◇ The length of the parameter `a` should be N. If this condition is not satisfied, your program is judged as **Wrong Answer [4]**.
- ◇ Every element of the parameter `a` should be between 1 and $N \times M$, inclusive. If this condition is not satisfied, your program is judged as **Wrong Answer [5]**.
- ◇ Throughout the process, no value should appear in a more than once. If this condition is not satisfied, your program is judged as **Wrong Answer [6]**.
- ◇ If it is impossible to make a beautiful dango stick using the dangos specified by `a` and a stick, your program is judged as **Wrong Answer [7]**.
- ◇ The function `Answer` should be called exactly M times. If the number of function calls to `Answer` is

different from M when the function `Solve` terminates, your program is judged as **Wrong Answer [8]**.

Important Notices

- Your program can implement other functions for internal use, or use global variables.
- Your program must not use the standard input and the standard output. Your program must not communicate with other files by any methods. However, your program may output debugging information to the standard error.

Compilation and Test Run

You can download an archive file from the contest webpage which contains the sample grader to test your program. The archive file also contains a sample source file of your program.

The sample grader is the file `grader.cpp`. In order to test your program, put `grader.cpp`, `dango3.cpp`, `dango3.h` in the same directory, and run the following command to compile your programs.

```
g++ -std=gnu++17 -O2 -o grader grader.cpp dango3.cpp
```

When the compilation succeeds, the executable file `grader` is generated.

Note that the actual grader is different from the sample grader. The sample grader will be executed as a single process, which will read input data from the standard input and write the results to the standard output.

Input for the Sample Grader

The sample grader reads the following data from the standard input.

N M
C_1 C_2 $\cdots$ $C_{N\times M}$

Here C_i ($1 \leq i \leq N \times M$) is the color of the dango i. It is an integer between 1 and N, inclusive.

Output of the Sample Grader

The sample grader outputs the following information to the standard output (quotes for clarity).

- If your program is judged as correct, it writes the number of times of the function calls to `Query` as

"`Accepted: 2022`".

- If your program is judged as any one of Wrong Answer, the sample grader writes its type as "`Wrong Answer [4]`".

If your program satisfies the conditions of several types of Wrong Answer, the sample grader reports only one of them.

Constraints

All input data satisfy the following constraints. For the values of C, see **Input for the Sample Grader**.

- $1 \leq C_i \leq N$ $(1 \leq i \leq N \times M)$.
- For each j $(1 \leq j \leq N)$, there are exactly M indices i $(1 \leq i \leq N \times M)$ satisfying $C_i = j$.
- N, M are integers.
- C_i $(1 \leq i \leq N \times M)$ is an integer.

Subtasks

1. (2 points) $N = 4$, $M = 4$.
2. (5 points) $N = 100$, $M = 10$.
3. (15 points) $N = 200$, $M = 25$.
4. (78 points) $N = 400$, $M = 25$.

Sample Communication

Here is a sample input for the sample grader and corresponding function calls.

Sample Input 1	Sample Function Calls		
	Call	Call	Return Value
`3 2` `3 3 1 2 1 2`	`Solve(3, 2)`		
		`Query([])`	`0`
		`Query([4, 2, 1 ,3])`	`1`
		`Query([3, 4, 5])`	`0`
		`Query([2, 6, 5])`	`1`
		`Query([6, 5, 4, 3, 2, 1])`	`2`
		`Answer([1, 6, 5])`	
		`Answer([2, 3, 4])`	

Note that this sample input **does not satisfy the constraints of any subtask.**

Among the files which can be downloaded from the contest webpage, `sample-02.txt` satisfies the constraints of Subtask 1.