Toy Design

Problem Name	ToyDesign
Input File	Interactive Task
Output File	Interactive Task
Time limit	1 second
Memory limit	256 megabytes

You are working for a company that designs toys. A new toy that is being created works like this: There are n pins, numbered from 1 to n, sticking out of a box. Some pairs of pins are connected with wires inside the box. (In other words, the pins and the wires form an undirected graph, where pins are the vertices and wires are the edges.) The wires are not visible from outside, and the only way to find out something about them is to use a **tester** on the pins: We can pick two pins i and j such that $i \neq j$, and the tester will tell if those two pins are connected inside the box, either directly or indirectly. (Thus, the tester tells whether there is a path between those pins in the graph.)

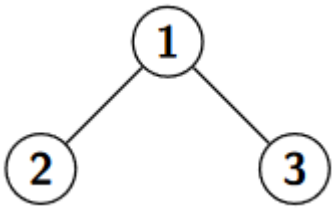
We will call the set of connections inside the box the **design** of the toy.

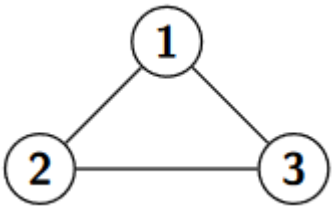
You are using a specialized software to query and create these designs. This software works like this: It starts with some design of the toy which we denote as "design 0". It does not show you the connections inside the box for this design. Instead, you can repeatedly perform the following three-step operation:

1. You pick a design number a and two pin numbers i and j such that $i \neq j$.
2. The software tells you what would happen if we use the tester on those two pins. In other words, it tells you if pins i and j are (directly or indirectly) connected in design a.
3. Also, if the pins were not directly or indirectly connected in design a, then it creates a new design which has all connections from design a plus one additional direct connection between i and j. This design is given the next available design number. (So, the first design created in this way will have number 1, then number 2, and so on.) Note that this does not change design a, just creates a new design that has the additional connection.

Your goal is to learn as much as possible about design 0 by using this operation.

Note that it is not always possible to determine the exact set of connections for design 0, because there is no way to distinguish direct and indirect connections. For example, consider the following two designs with $n = 3$:





The tester would report any pair of pins as connected for both designs, so we will not be able to distinguish them using the software described above.

Your goal is to determine any design that is equivalent to design 0. Two designs are **equivalent** if the tester reports the same result in both designs for all pairs of pins.

Implementation

This is an interactive problem. You must implement a function

```
void ToyDesign(int n, int max_ops);
```

that determines a design that is *equivalent* to design 0. Your implementation should achieve this goal by calling two functions as described below. The first function you can call is:

```
int Connected(int a, int i, int j);
```

where $1 \leq i, j \leq n$, $i \neq j$, $a \geq 0$, and a must not exceed the number of designs created so far. If pins i and j are (directly or indirectly) connected in design a, then it will return a back. Otherwise, it will return the number of designs created so far plus one, which becomes the number assigned to the new design which has all the connections of design a plus the connection between i and j. The function `Connected` can be called at most `max_ops` times.

When your program is done with the `Connected` operations, it should describe a design that is equivalent to design 0. To describe a design, the program should call:

```
void DescribeDesign(std::vector<std::pair<int,int>> result);
```

The parameter `result` is a vector of integer pairs describing the direct connections between the pins. Each pair corresponds to one connection and should contain the two pin numbers of the connection. There must be at most one direct connection between each (unordered) pair of pins, and no direct connections between a pin and itself. Calling this function terminates the execution of your program.

Constraints

- $2 \leq n \leq 200$

Scoring

- Subtask 1 (10 points): $n \leq 200$, *max_ops* $= 20\,000$
- Subtask 2 (20 points): $n \leq 8$, *max_ops* $= 20$
- Subtask 3 (35 points): $n \leq 200$, *max_ops* $= 2\,000$
- Subtask 4 (35 points): $n \leq 200$, *max_ops* $= 1\,350$

Sample Interaction

Contestant action	Grader action	Explanation
	`ToyDesign(4, 20)`	There are 4 pins in the toy. You need to determine any design that is equivalent to design 0 by calling `Connected` at most 20 times.
`Connected(0, 1, 2)`	`Returns 1.`	Pins 1 and 2 are not connected directly or indirectly in design 0. New design 1 is created.
`Connected(1, 3, 2)`	`Returns 2.`	Pins 3 and 2 are not connected directly or indirectly in design 1. New design 2 is created.
`Connected(0, 3, 4)`	`Returns 0.`	Pins 3 and 4 are connected directly or indirectly in design 0. No new design is created.
`DescribeDesign({{3, 4}})`	-	We describe a design that has only one connection: Pins 3 and 4.

Sample Grader

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

- The first line contains the number of pins, n, the number of direct connections, m and *max_ops*

- The following m lines contain direct connections as tuples of pins.

The sample grader reads the input and calls the `ToyDesign` function in user's solution. The grader will output one of the following messages, based on the behaviour of your solution:

- "Wrong answer: Number of operations exceeds the limit.", if the number of calls to `Connected` exceeds *max_ops*
- "Wrong answer: Wrong design id.", if the parameter a to a call to `Connected` is the number of a design that does not exist at the moment the call was made.
- "Wrong answer: Incorrect design.", if the design described via `DescribeDesign` is not equivalent to design 0.
- "OK!" if the design described via `DescribeDesign` is equivalent to design 0.

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
```