Problem D – Oni está o Oni?

This is an **output-only** problem.
Unlike other problems where you need to read data and write the output, in this problem you should only submit a text file with extension `.txt`

An *Oni* is a very powerful creature, associated with controlling the forces of nature, but it's just a legend from Japanese mythology... or at least that's what was thought until last week! Rumors have started to surface that the offspring of an *Oni*, known as Tobias, has been sighted in one of the caves in a dense forest, and since then the inhabitants of the nearest village have not rested until they find it! According to the legend, if the *Oni* is found and fed by a human, the local populations will be protected from disasters for 73,000 years. Can the village population develop a strategy to find it?

The forest can be represented by an undirected graph with N vertices and M edges. Initially, the *Oni* is located at one of these vertices. Every day, the population selects exactly K hunters to search for the *Oni*, and each will be responsible for investigating one of the nodes in the forest to see if it is there or not. To facilitate the coordination of operations, the village population decided that the number of hunters selected daily, K, is fixed, but it is allowed to have multiple hunters investigate the same node (redundantly) if not so many hunters are needed that day. Thus, every day the population selects a multiset of K nodes to visit that day. It is known that if the *Oni* is not found on a certain day, it will move to one of the adjacent nodes the next day, and the search continues.

Determine a strategy for visiting the nodes that will allow the village population to definitely find the *Oni*, regardless of its movements. We define a strategy as a sequence of D multisets of K nodes each, where D is the number of days the hunters will need to find the *Oni*. While the number K of hunters selected daily is chosen by the population, you can choose the number D, as long as it does not exceed 5000.

Constraints

The following limits are guaranteed in all test cases that will be presented to the program:

$2 \leq \boldsymbol{N} \leq 100$	Number of nodes in the graph
$1 \leq \boldsymbol{M} \leq \frac{\boldsymbol{N} \times (\boldsymbol{N}-1)}{2}$	Number of edges in the graph
$1 \leq \boldsymbol{K} \leq \boldsymbol{N}$	Number of hunters selected daily
$1 \leq \boldsymbol{D} \leq 5000$	Maximum number of days

Input Format

Each test case begins with three integers separated by spaces: $\boldsymbol{N}$ (the number of nodes), $\boldsymbol{M}$ (the number of edges), and $\boldsymbol{K}$ (the number of hunters to be selected per day).

Then follow $\boldsymbol{M}$ lines, each with two integers u_i, v_i between 1 and $\boldsymbol{N}$ separated by spaces, representing an edge. It is guaranteed that the graph is connected, that **there are no multiple edges** between the same pair of vertices, and that **all edges connect different vertices**.

Test Cases

There are 9 test cases, organized into 9 groups with different additional restrictions and each with a fixed score (no partial scores for each one). The test cases are as follows (zip file containing all cases)

Group	Points	Additional Constraints
1 (`inp1.txt`)	5	$\boldsymbol{K} = 3$ and the graph is a tree with 6 nodes
2 (`inp2.txt`)	10	$\boldsymbol{K} = 1$ and the graph is a tree with 6 nodes
3 (`inp3.txt`)	5	$\boldsymbol{K} = 3$ and the graph is a line with 100 nodes
4 (`inp4.txt`)	15	$\boldsymbol{K} = 1$ and the graph is a line with 100 nodes
5 (`inp5.txt`)	10	$\boldsymbol{K} = 2$ and the graph is an amoeba (see below) with 100 nodes
6 (`inp6.txt`)	15	$\boldsymbol{K} = 1$ and the graph is an amoeba (see below) with 100 nodes
7 (`inp7.txt`)	10	$\boldsymbol{K} = 6$ and the graph is a 5×5 grid
8 (`inp8.txt`)	15	$\boldsymbol{K} = 6$ and the graph is a 10×10 grid
9 (`inp9.txt`)	15	$\boldsymbol{K} = 7$ and the graph is a tree with 100 nodes

An **amoeba graph** is a tree that satisfies the following property: all vertices are within a distance of 2 from a vertex in the tree's diameter. The diameter of a tree is a set of vertices that forms the longest path between any two vertices in the tree.

Output Format

You should submit a **single** `txt` file (file format is important, your file should end in `.txt`), containing the answer for all test cases. For each test case, in the correct order, the file should contain $\boldsymbol{D}+1$ lines, where $\boldsymbol{D}$ is the number of days your strategy uses for that test case (you can use different $\boldsymbol{D}$ values for different test cases).

The format per test case is as follows:

- one line with the integer $\boldsymbol{D}$,
- followed by $\boldsymbol{D}$ lines, each with $\boldsymbol{K}$ integers $u_1, u_2, \ldots, u_{\boldsymbol{K}}$ between 1 and $\boldsymbol{N}$: the nodes that will be visited that day. They can be in any order and do not have to be distinct, but there must be exactly $\boldsymbol{K}$.

Important Notes:

- If you do not have a solution for one of the test cases, print a line with $\boldsymbol{D} = 0$. You will receive 0 points for that test case, but it will not affect your score for other test cases.
- If you do not respect the mentioned format (for example, by printing more than $\boldsymbol{D}$ lines for a particular test case), it is possible that your score will be 0 points for all test cases, because the evaluator may not read your solutions correctly.
- If any of the outputs is invalid (for example, because it does not contain enough integers, or because the integers are not between 1 and $\boldsymbol{N}$), the result of the submission will be **Wrong Answer** and you will receive 0 points for that test case.

A possible example of a solution file would be as follows (its score is 0 points):

```
0
3
1
2
3
0
0
0
0
0
0
1
1 1 50 50 100 100 100
```

Tips for Generating the Output

Since it is necessary to create file with extension `.txt` for your submission, it may be useful to use the terminal to generate it (this is not mandatory, you can create the file as you wish).

If you use the command `X` to run your program (for example, for C++ this would be something like `./a.out`, after compiling with `g++ code.cpp`), you can use the following command to read the input from a `inp.txt` file and write to a `out.txt` file: `X < inp.txt > out.txt`.

To facilitate the organization of the submission file, it is recommended to generate a `txt` file for each test case (for example, for test case 2 have a `out2.txt`).

To later generate a `out.txt` file for submission, you can, for example, use the following command (assuming you have a file for each of the 9 test cases):

```
cat out1.txt > out.txt; cat out2.txt >> out.txt; cat out3.txt >> out.txt; cat out4.txt
>> out.txt; cat out5.txt >> out.txt; cat out6.txt >> out.txt; cat out7.txt >> out.txt;
cat out8.txt >> out.txt; cat out9.txt >> out.txt
```