Free the Bunny Prisoners

==========

You need to free the bunny prisoners before Commander Lambda's space station explodes! Unfortunately, the commander was very careful with her highest-value prisoners - they're all held in separate, maximum-security cells. The cells are opened by putting keys into each console, then pressing the open button on each console simultaneously. When the open button is pressed, each key opens its corresponding lock on the cell. So, the union of the keys in all of the consoles must be all of the keys. The scheme may require multiple copies of one key given to different minions.

The consoles are far enough apart that a separate minion is needed for each one. Fortunately, you have already freed some bunnies to aid you - and even better, you were able to steal the keys while you were working as Commander Lambda's assistant. The problem is, you don't know which keys to use at which consoles. The consoles are programmed to know which keys each minion had, to prevent someone from just stealing all of the keys and using them blindly. There are signs by the consoles saying how many minions had some keys for the set of consoles. You suspect that Commander Lambda has a systematic way to decide which keys to give to each minion such that they could use the consoles.

You need to figure out the scheme that Commander Lambda used to distribute the keys. You know how many minions had keys, and how many consoles are by each cell. You know that Command Lambda wouldn't issue more keys than necessary (beyond what the key distribution scheme requires), and that you need as many bunnies with keys as there are consoles to open the cell.

Given the number of bunnies available and the number of locks required to open a cell, write a function solution(num_buns, num_required) which returns a specification of how to distribute the keys such that any num_required bunnies can open the locks, but no group of (num_required - 1) bunnies can.

Each lock is numbered starting from 0. The keys are numbered the same as the lock they open (so for a duplicate key, the number will repeat, since it opens the same lock). For a given bunny, the keys they get is represented as a sorted list of the numbers for the keys. To cover all of the bunnies, the final answer is represented by a sorted list of each individual bunny's list of keys. Find the lexicographically least such key distribution - that is, the first bunny should have keys sequentially starting from 0.

num_buns will always be between 1 and 9, and num_required will always be between 0 and 9 (both inclusive). For example, if you had 3 bunnies and required only 1 of them to open the cell, you would give each bunny the same key such that any of the 3 of them would be able to open it, like so:

[

[0],

[0],

[0],

]

If you had 2 bunnies and required both of them to open the cell, they would receive different keys (otherwise they wouldn't both actually be required), and your answer would be as follows:

[

[0],

[1],

]

Finally, if you had 3 bunnies and required 2 of them to open the cell, then any 2 of the 3 bunnies should have all of the keys necessary to open the cell, but no single bunny would be able to do it. Thus, the answer would be:

[

[0, 1],

[0, 2],

[1, 2],

]

Languages

=========

To provide a Python solution, edit solution.py

To provide a Java solution, edit Solution.java

Test cases

==========

Your code should pass the following test cases.

Note that it may also be run against hidden test cases not shown here.

-- Python cases --

Input:

solution.solution(2, 1)

Output:

[[0], [0]]

Input:

solution.solution(4, 4)

Output:

[[0], [1], [2], [3]]

Input:

solution.solution(5, 3)

Output:

[[0, 1, 2, 3, 4, 5], [0, 1, 2, 6, 7, 8], [0, 3, 4, 6, 7, 9], [1, 3, 5, 6, 8, 9], [2, 4, 5, 7, 8, 9]]

-- Java cases --

Input:

Solution.solution(2, 1)

Output:

[[0], [0]]

Input:

Solution.solution(5, 3)

Output:

[[0, 1, 2, 3, 4, 5], [0, 1, 2, 6, 7, 8], [0, 3, 4, 6, 7, 9], [1, 3, 5, 6, 8, 9], [2, 4, 5, 7, 8, 9]]

Input:

Solution.solution(4, 4)

Output:

[[0], [1], [2], [3]]

Use verify [file] to test your solution and see how it does. When you are finished editing your code, use submit [file] to submit your answer. If your solution passes the test cases, it will be removed from your home folder.