Code Jam 2019 - Round 1A

Golf Gophers

Problem

Last year, a bunch of pesky gophers took up residence in our orchard. We tried to change our line of work by opening up a miniature golf course, but it looks like the gophers have followed us here! Once again, we need to figure out how many gophers there are, but we cannot observe them directly because they are secretive and nocturnal, whereas we like to sleep at night. We do know there are between 1 and **M** gophers, inclusive.

Our mini golf course is famous for having a small electronic windmill on each of its 18 holes. The i-th windmill has $2 \leq \mathbf{B}_i \leq 18$ blades, which are numbered from 0 to $\mathbf{B}_i$-1, clockwise. Each night, before going to sleep, we turn off the windmills and set each one such that blade 0 is pointing downward, which is important so that the windmills can charge up properly for the next day. However, we have noticed that when we wake up, the windmills have been disturbed. Since our mini golf course is in a windless area, we think the mischievous gophers must be responsible!

We know that every night, all of the gophers emerge, one by one; each of them chooses one of the windmills independently and uniformly at random and rotates it counterclockwise by one blade. So, for example, for a windmill with 3 blades for which 0 is pointing downward, the first gopher to interact with it turns it so that 1 is pointing downward, and then the next gophers to interact with that windmill make the downward-pointing blade have number 2, then 0, then 1, and so on.

We have devised a plan. We designed our windmills so that we can easily change the number of blades (to modulate the difficulty of our course), and we will now take advantage of this! Each night, before going to sleep, we can choose the number of blades on each of the 18 windmills, within the given limits; we do not have to use the same number of blades on each windmill, or make the same choices every night. In the morning, we will observe the number on each windmill's downward-pointing blade.

We have **N** nights in which to figure out G, the number of gophers. Can you help us?

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially, your program should read a single line containing three integers **T**, **N** and **M**, the number of test cases, the number of nights allowed per test case and the maximum number of gophers, respectively. Then, you need to process **T** test cases.

In each test case, your program processes up to **N** + 1 exchanges with our judge. You may make up to **N** exchanges of the following form:

- Your program outputs one line with eighteen integers between 2 and 18, inclusive; the i-th of these represents the number of blades you want the i-th windmill to have on that night.
- The judge responds with one line with eighteen integers; the i-th of these represents the number on the downward-pointing blade of the i-th windmill in the morning, after the gophers have worked their mischief. If you sent invalid data (e.g., a number out of range, or a malformed line), the judge instead responds with `-1`.

On each night, for each gopher, the choice of which windmill the gopher turns is uniform at (pseudo)-random, and independent of any other choice by any gopher (including itself) on any night.

After making between 0 and **N** exchanges as explained above, you must make one more exchange of the following form:

- Your program outputs one integer: your guess for G, the number of gophers.
- The judge responds with one line with a single integer: `1` if your answer is correct, and `-1` if it is not (or you have provided a malformed line).

After the judge sends `-1` to your input stream (because of either invalid data or an incorrect answer), it will not send any other output. If your program continues to wait for the judge after receiving `-1`, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

Limits

$1 \le \mathbf{T} \le 20$.
Time limit: 20 seconds per test set.
Memory limit: 1GB.

Test set 1 (Visible)

N = 365.
M = 100.

Test set 2 (Hidden)

N = 7.
M = 10^6.

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Sample Interaction

This interaction corresponds to Test set 1. Suppose that, unbeknownst to us, the judge has decided that there are 10 gophers.

```
t, n, m = readline_int_list()   // Reads 20 into t, 365 into n and 100 into m.
// Choose numbers of blades for day 1.
printline 2 2 2 2 18 3 3 3 3 3 3 4 4 4 4 5 2 2 to stdout
flush stdout
// Reads 0 0 0 0 0 0 1 2 1 0 1 2 0 0 0 0 1 0 into res.
res = readline_int_list()
// Choose numbers of blades for day 2.
printline 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 to stdout
flush stdout
// Reads 0 1 1 2 0 0 1 0 0 0 0 0 0 1 0 0 0 0 into res.
res = readline_int_list()
printline 8 to stdout        // We make a wrong guess even though we could
flush stdout                 // have investigated for up to 363 more nights.
verdict = readline_int()     // Reads -1 into verdict (judge has decided our
                             //   solution is incorrect)
exit                         // Exits to avoid an ambiguous TLE error
```

Notice that even though the guess was consistent with the information we received from the judge, we were still wrong because we did not find the correct value.