Golf Gophers (11pts, 21pts)

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 $2 \le \mathbf{B}_i \le 18$ 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

$1 \leq \mathbf{T} \leq 20$.
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 servers. 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 Interactive Problems section of the FAQ.

Local Testing Tool

To better facilitate local testing, we provide you the following script. Instructions are included inside. You are encouraged to add more test cases for better testing. 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 our FAQ to make sure that you are using the same compiler as us.

file_downloadDownload testing_tool.py

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.

Analysis

Test set 1

The difficulty of this problem stems from the fact that the windmills can loop around. If we have a windmill with 5 blades, and we find it with blade number 2 pointing downward in the morning, does that mean that 2 gophers rotated it? or that 7 did? or 12? In general, if a windmill has B blades, and we find it in position P, all that we can say for sure is that some number (P + K × B) of gophers (for some integer K) tampered with it during the night.

Because of this, we might reason that we should put as many blades as possible (that is, 18) on each windmill, and then hope that none of those windmills is visited by more than 17 gophers, in which case the total number of rotations of all blades equals the total number of gophers. This turns out to be a reasonable hope! It is hard to directly calculate the probability that none of the 18 blades will be turned more than 17 times on a given night, but a quick simulation can tell us that even in the worst case of 100 gophers, the probability of this happening is about 0.00017. Since we can run this same experiment 365 times and take the maximum result that we find, the chances of a wrong answer (due to getting a misleading result all 365 times) are infinitesimally small.

Test set 2

In test set 2, we only have 7 nights to work with, and the number of gophers can be quite large, so we cannot reasonably hope that the windmills will not loop. Now what? We might consider using different numbers of blades on each windmill during a given night, but it's hard to see how that buys us anything.

Suppose that, on one night, we try putting two blades on every windmill. At first this doesn't seem to help — shouldn't the resulting data should be almost pure noise? However, we can observe that if the number of gophers is odd, the total number of turns (across all windmills) will be odd, and if the number of gophers is even, that total number will be even. So we have a way of determining the parity of the number of gophers, no matter how they happen to turn the blades!

We can extend this idea to find out how many gophers there are modulo any number of our choice between 2 and 18. Since we only get 7 nights, though, we should choose our numbers carefully. One promising idea is to make them all prime, and there are seven primes in the range we can use: 2, 3, 5, 7, 11, 13, and 17. Then we can try to use the construction suggested by the Chinese remainder theorem to uniquely determine the number of gophers. However, this method would only work for any number of gophers up to 2 × 3 × ... × 17 = 510510; it cannot distinguish 510511 gophers from 1 gopher! We are in trouble, since we might have up to 10^6 gophers.

The final insight that we need is that the Chinese remainder theorem only requires its moduli to all be *relatively* prime, not necessarily prime. So we can use 16 in place of 2, and 9 in place of 3. This lets us identify any number of gophers up to 5 × 7 × 9 × 11 × 13 × 16 × 17 = 12252240, which easily covers what we need. (We can also get away with using the numbers 12 through 18, for example; we leave the details to the reader.)

Notice that we do not really need to do any calculations based on the Chinese remainder theorem; since the number of possible answers is relatively small, we can check all of them until we find the one that has the appropriate residue for each of our chosen moduli.

Test set 1 appendix

To prove that the probability of at least one blade rotating more than 17 times is about 0.00017, we can solve a related problem:

"Count the number of integer arrays of length **L** such that each array element is one of 1, 2, ..., **U** and each number appears at most **B** times in the array."

This related problem is equivalent to counting integer partitions with constraints on the size of each partition.

We can solve this related problem using Dynamic Programming. First, we choose how many times **U** appears in the array; say it appears K times, with 0 ≤ K ≤ min(**B**,**L**). Once we know K, we have to choose where those K values go. Using combinations, we can see that there are `C(L, K)` possible options. For each option, we can treat the **L**-K other spaces as an array that must contain the numbers 1, 2, ..., **U**-1 such that no number appears more than **B** times, which is a subproblem of our original problem, so we may recurse. If we sum over all possible values for K, we have the total number of valid arrays.

If we solve the above problem with **L** = 100, **U** = 18, **B** = 17, we see that there are 33664778324823401186092706362918765459845506244656050183448782053595666316176 253355560987063931385912519171492847625697152000 (or approximately 3.366×10^{125}) arrays such that no number appears more than 17 times. These arrays can represent which windmill the gophers select in a *good*configuration out of the 18^{100} possible configurations. This gives us the probability quoted above.