Code Jam 2018 - Round 1C

Lollipop Shop

Problem

You own a lollipop shop. At the start of the day, you make **N** lollipops, each of a single unique flavor, like huckleberry, cherry or lime. **N** customers come into the shop during the day, one at a time. Each customer gives you a list of which of your lollipop flavors they like. You can sell them one lollipop of any of those flavors, as long as you have not already sold someone else the same flavor earlier in the day (since there is only one lollipop of each flavor). If none of the flavors they like are still available, you cannot sell them a lollipop, and they leave your shop disappointed.

You do not know what each customer's flavor preferences are going to be until they arrive. Each customer decides if they like or dislike each flavor randomly, independently of whether they like any other flavor, or what flavors anyone else likes. However, your market studies have shown that some flavors may have a higher probability of being liked in general! For example, the lime flavor might have a 10% chance of being liked by any particular customer, whereas that chance might be 1% for the cherry flavor. These values are always chosen independently and uniformly at random from the interval [0.005, 0.1].

Obviously, you want to sell lollipops to as many of the **N** customers as possible! But, since you do not know what flavors your customers will ask for ahead of time, you cannot always make an optimal decision — sometimes you might sell a customer one flavor, and then later wish you had sold them another.

Suppose that you somehow knew all the customers' preferences in advance and could plan ahead; then there is some maximum number M of lollipops that you could possibly sell. You do not know the customers' preferences in advance, but we require you to sell a number of lollipops that is at least 90% of M for each input case.

Input and output

This problem is interactive, which means that the concepts of input and output are different than in standard Code Jam problems. You will interact with a separate process that both provides you with information and evaluates your responses. All information comes into your program via standard input; anything that you need to communicate should be sent via standard output. Remember that many programming languages buffer the output by default, so make sure your output actually goes out (for instance, by flushing the buffer) before blocking to wait for a response. See the FAQ for an explanation of what it means to flush the buffer. Anything your program sends through standard error is ignored, but it might consume some memory and be counted against your memory limit, so do not overflow it.

Initially, your program should read a single line containing a single integer **T** indicating the number of test cases. Then, you need to process **T** test cases.

For each test case, your program should read a single line with one integer **N**, the number of lollipops (which is the same as the number of customers).

Then, for each of the customers, your program should read a single line, which will contain space-separated integers. The first integer is **D**, the number of flavors that customer likes. Then, **D** integers follow, the ID numbers of those flavors, in strictly increasing order. Flavors have unique ID numbers in the range [0, **N** - 1]. Note that **D** might be zero for some or all customers.

After each of these lines, your program must write a single line to standard output, containing the ID number of one of the **D** flavors to sell to the customer, or `-1` if no lollipop is to be sold to the customer. After you have written the **N**th line for the test case, your program should terminate if it was the last test case; otherwise, it should start reading data for the next test case.

If your program gets something wrong (e.g., tries to sell a customer a flavor that was already sold, or tries to sell a customer a flavor they don't like, or uses the wrong output format, or outputs an out-of-bounds value), the judge will send `-1` to your input stream and it will not send any other output after that. 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 the appropriate verdict (Wrong Answer, Runtime Error, etc.) instead of a Time Limit Exceeded error. As usual, if the total time or memory is exceeded, or your program gets a runtime error, you will receive the appropriate verdict. Not selling enough lollipops for a test case will not cause you to get the `-1` message.

You should not send additional information to the judge after processing all test cases. In other words, if your program keeps printing to standard output after the last test case, you will get a Wrong Answer judgment.

A note on judge behavior

At the start of each test case, the judge will determine all customers' preferences. That is, it will use a (hidden) list of probabilities P_i between 0.005 and 0.1, one for each flavor; each customer has a probability P_i of liking the i-th flavor. That is, the random variables indicating whether customer j likes flavor i are independent and identically distributed. These preferences are constant throughout the test and will not be modified e.g. in response to your choices.

Test set 1 (Visible)

T = 50.
N = 200.
0 ≤ **D** ≤ **N**.
Time limit (for the entire test set): 25 seconds.
Memory limit: 1GB.

Sample Interaction

Note that this sample interaction has smaller values of **T** and **N** than the real data. The local testing tool also uses smaller cases.

```
t = readline_int()           // reads 10 into t
n = readline_int()           // reads 4 into n (four customers & flavors)
prefs = readline_int_list()  // reads 1 2 (customer only likes flavor 2)
printline 2 to stdout        // sells this customer flavor 2
flush stdout
prefs = readline_int_list()  // reads 0 (customer likes nothing)
printline -1 to stdout       // no flavor to sell to the customer!
flush stdout
prefs = readline_int_list()  // reads 1 2 (customer only likes flavor 2)
printline -1 to stdout       // already used flavor 2, so no flavor to sell
flush stdout
prefs = readline_int_list()  // reads 2 1 3 (customer likes 1 and 3)
printline 3 to stdout        // note: we could have also sold flavor 1
flush stdout
n = readline_int()           // (start of case 2) reads 1
prefs = readline_int_list()  // reads 1 0
printline -1 to stdout       // non-optimal but legal choice
flush stdout
n = readline_int()           // (start of case 3) reads 5
prefs = readline_int_list()  // reads 2 1 3
printline 1 to stdout
flush stdout
prefs = readline_int_list()  // reads 2 1 2
printline 1 to stdout        // error -- tried to give same flavor twice!
```

```
flush stdout
prefs = readline_int_list()  // reads -1 (judge has given up on us)
exit                         // exits to avoid an ambiguous TLE error
```

The pseudocode above demonstrates the following scenario.

- In the first test case, the program sells a total of two lollipops. It would not have been possible to sell more than two, so the actual number sold is definitely at least 90% of the maximum possible number sold.
- In the second test case, the program chooses (for the sake of demonstration here) not to sell the customer a lollipop, although it could have. It sells a total of 0 when it could have sold a total of 1. So, the program will not pass this test set, but note that this does not cause the judge to stop sending input.
- In the third case, the program makes an error (again for the sake of demonstration) that causes the judge to stop sending input. The program recognizes this and terminates. The user will see a Wrong Answer judgment.

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