Code Jam 2019 - Qualification Round

Dat Bae

Problem

A research consortium has built a new database system for their new data center. The database is made up of one master computer and **N** worker computers, which are given IDs from 0 to **N**-1. Each worker stores exactly one bit of information... which seems rather wasteful, but this is very important data!

You have been hired to evaluate the following instruction for the database:

- `TEST_STORE` <bits>: The master reads in <bits>, which is a string of **N** bits, and sends the i-th bit to the i-th worker for storage. The master will then read the bits back from the workers and return them to the user, in the same order in which they were read in.

During normal operation, `TEST_STORE` should return the same string of bits that it read in, but unfortunately, **B** of the workers are broken!

The broken workers are correctly able to store the bits given to them, but whenever the master tries to read from a broken worker, no bit is returned. This causes the `TEST_STORE` operation to return only **N**-**B** bits, which are the bits stored on the non-broken workers (in ascending order of their IDs). For example, suppose **N** = 5 and the 0th and 3rd workers are broken (so **B** = 2). Then:

- `TEST_STORE 01101` returns `111`.
- `TEST_STORE 00110` returns `010`.
- `TEST_STORE 01010` returns `100`.
- `TEST_STORE 11010` also returns `100`.

For security reasons, the database is hidden in an underground mountain vault, so calls to `TEST_STORE` take a very long time. You have been tasked with working out which workers are broken using at most **F** calls to `TEST_STORE`.

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 a single integer **T** indicating the number of test cases. Then, you need to process **T** test cases.

For each test case, your program will first read a single line containing three integers **N**, **B**, and **F**, indicating the number of workers, the number of broken workers, and the number of lines you may send (as described below).

Then you may send the judge up to **F** lines, each containing a string of exactly **N** characters, each either `0` or `1`. Each time you send a line, the judge will check that you have not made more than **F** calls. If you have, the judge will send you a single line containing a single `-1`, and then finish all communication and wait for your program to finish. Otherwise, the judge will send a string of length **N**-**B**: the string returned by `TEST_STORE`, as described above.

Once your program knows the index of the **B** broken workers, it can finish the test case by sending **B** space-separated integers: the IDs of the broken workers, in sorted order. This does not count as one of your **F** calls.

If the **B** integers are not exactly the IDs of the **B** broken workers, you will receive a Wrong Answer verdict, and the judge will send a single line containing `-1`, and then no additional communication. If

your answer was correct, the judge will send a single line with `1`, followed by the line that begins the next test case (or exit, if that was the last test case).

Limits

Time limit: 20 seconds per test set.
Memory limit: 1GB.
$1 \le \mathbf{T} \le 100$.
$2 \le \mathbf{N} \le 1024$.
$1 \le \mathbf{B} \le \min(15, \mathbf{N}-1)$.

Test set 1 (Visible)

F = 10.

Test set 2 (Hidden)

F = 5.

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

The following interaction meets the limits for Test set 1.

```
t = readline_int()            // Reads 2 into t
n, b, f = readline_int_list()  // Reads 5, 2, 10 into n, b, f
printline 01101 to stdout     // The next four outputs match the example in
                              // the problem statement.
flush stdout
response = readline_str()     // Reads 111 into response. (At this point, we
                              // could determine the answer; the remaining
                              // queries are just examples!)
printline 00110 to stdout
flush stdout
response = readline_str()     // Reads 010 into response
printline 01010 to stdout
flush stdout
response = readline_str()     // Reads 100 into response
printline 11010 to stdout
flush stdout
response = readline_str()     // Reads 100 into response
printline 0 3 to stdout       // Guesses the answer. Notice that we were
                              // not required to use all 10 of our allowed
                              // queries.
flush stdout
```

```
verdict = readline_int()     // Reads 1 into verdict. We got that test case
                             // right!
n, b, f = readline_int_list()  // Reads 2, 1, 10 into n, b, f.
printline 01 to stdout       // 01 is a query, not a guess at the final
                             // answer (if we wanted to guess that just
                             // worker 1 were broken, we would have to
                             // send 1 as we do below)
flush stdout
response = readline_str()    // Reads 1 into response.
printline 1 to stdout        // Makes a (bad) wild guess.
verdict = readline_str()     // Reads -1 into verdict.
exit                         // exits to avoid an ambiguous TLE error
```