Code Jam 2020 - Qualification Round

ESAb ATAd

Problem

Last year, a research consortium had some trouble with a distributed database system that sometimes lost pieces of the data. You do not need to read or understand that problem in order to solve this one!

The consortium has decided that distributed systems are too complicated, so they are storing **B** bits of important information in a single array on one awesome machine. As an additional layer of security, they have made it difficult to obtain the information quickly; the user must query for a bit position between 1 and **B**, and then they receive that bit of the stored array as a response.

Unfortunately, this ultra-modern machine is subject to random quantum fluctuations! Specifically, after every 1st, 11th, 21st, 31st... etc. query is sent, *but before the response is given*, quantum fluctuation causes exactly one of the following four effects, with equal probability:

- 25% of the time, the array is complemented: every `0` becomes a `1`, and vice versa.
- 25% of the time, the array is reversed: the first bit swaps with the last bit, the second bit swaps with the second-to-last bit, and so on.
- 25% of the time, both of the things above (complementation and reversal) happen to the array. (Notice that the order in which they happen does not matter.)
- 25% of the time, nothing happens to the array.

Moreover, there is no indication of what effect the quantum fluctuation has had each time. The consortium is now concerned, and it has hired you to get its precious data back, in whatever form it is in! Can you find the entire array, such that your answer is accurate *as of the time that you give it*? Answering does not count as a query, so if you answer after your 30th query, for example, the array will be the same as it was after your 21st through 30th queries.

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 two integers **T** and **B**: the number of test cases and the number of bits in the array, respectively. Note that **B** is the same for every test case.

Then, you need to process **T** test cases. In each case, the judge begins with a predetermined **B**-bit array; note that this array can vary from test case to test case, and is not necessarily chosen at random. Then, you may make up to 150 queries of the following form:

- Your program outputs one line containing a single integer P between 1 and **B**, inclusive, indicating which position in the array you wish to look at.
- If the number of queries you have made so far ends with a 1, the judge chooses one of the four possibilities described above (complementation, reversal, complementation + reversal, or nothing), uniformly at random and independently of all other choices, and alters the stored array accordingly. (Notice that this will happen on the very first query you make.)
- The judge responds with one line containing a single character `0` or `1`, the value it currently has stored at bit position P, or `N` if you provided a malformed line (e.g., an invalid position).

Then, after you have made as many of the 150 queries above as you want, you must make one more exchange of the following form:

- Your program outputs one line containing a string of **B** characters, each of which is `0` or `1`, representing the bits *currently* stored in the array (which will not necessarily match the bits that were initially present!)

- The judge responds with one line containing a single letter: uppercase `Y` if your answer was correct, and uppercase `N` if it was not (or you provided a malformed line). If you receive `Y`, you should begin the next test case, or stop sending input if there are no more test cases.

After the judge sends `N` to your input stream, it will not send any other output. If your program continues to wait for the judge after receiving `N`, 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

Time limit: 40 seconds per test set.
Memory limit: 1GB.
1 ≤ **T** ≤ 100.

Test set 1 (Visible Verdict)

B = 10.

Test set 2 (Visible Verdict)

B = 20.

Test set 3 (Hidden Verdict)

B = 100.

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. **The interactive runner was changed after the 2019 contest. Be sure to download the latest version.** For more information, read the Interactive Problems section of the FAQ.

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 corresponds to Test Set 1.

```
t, b = readline_int_list()      // reads 100 into t and 10 into b.
// The judge starts with the predetermined array for this test case:
// 0001101111. (Note: the actual Test Set 1 will not necessarily
// use this array.)
printline 1 to stdout   // we ask about position 1.
flush stdout
```

```
// Since this is our 1st query, and 1 is 1 mod 10, the judge secretly and
// randomly chooses one of the four possible quantum fluctuation effects, as
// described above. It happens to choose complementation + reversal, so now
// the stored value is 0000100111.
r = readline_chr()      // reads 0.
printline 6 to stdout   // we ask about position 6.
flush stdout
// Since this is our 2nd query, and 2 is 2 mod 10, the judge does not choose
// a quantum fluctuation effect.
r = readline_chr()      // reads 0.
...
// We have omitted the third through tenth queries in this example.
...
printline 1 to stdout   // we decide to ask about position 1 again.
flush stdout
// Since this is our 11th query, and 11 is 1 mod 10, the judge secretly and
// randomly chooses a quantum fluctuation effect, and happens to get
// reversal, so now the stored value is 1110010000.
r = readline_chr()      // reads 1.
printline 1110110000 to stdout   // we try to answer. why?!?!
flush stdout
ok = readline_chr()     // reads N -- we have made a mistake!
exit                    // exits to avoid an ambiguous TLE error
```