Code Jam 2019 - Round 3

Zillionim

Problem

Zillionim is a turn-based game for two players. Initially, 10^{12} coins are arranged end-to-end in a single line, numbered from 1 to 10^{12} from left to right. During a turn, a player must select 10^{10} consecutive coins and remove them. Two coins that were not originally consecutive do not become consecutive even if all of the coins in between them are removed.

On their turn, a player makes a valid move if possible, and then it is their opponent's turn. If a player cannot make a valid move on their turn, they lose the game (and the opponent wins the game).

Because our engineers are still hard at work training our machine learning model to play Zillionim, we have created a simple AI that plays Zillionim by making random moves. The AI always gets the first turn. On each of the AI's turns, the AI determines all valid moves and chooses one of them uniformly at random.

Can you beat this AI... at least most of the time?

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**, the number of test cases, and **W**, the minimum number of games you need to win for your solution to be considered correct. Then, you need to process **T** test cases, each of which is a single game of Zillionim.

Each test case is processed by making exchanges with the judge until one player wins the game. For each exchange, the judge first outputs a single line with a single integer **P**, to be interpreted as follows:

- If $1 \leq \mathbf{P} \leq 10^{12} - 10^{10} + 1$, then the AI has removed coins numbered **P**, **P** + 1, ..., **P** + 10^{10} - 1 and it is your turn. Note that this means there is at least one valid move remaining for you to play. The AI always plays a valid move.
- If **P** = `-2`, your last move caused you to win the current game.
- If **P** = `-3`, the AI has made a move that caused it to win the current game. Notice that in this case, the judge does not send you the AI's last move.
- If **P** = `-1`, the last information you sent to the judge was malformed data or an invalid move (out of range or attempting to remove coins that were not there), meaning that you will get a Wrong Answer verdict for not playing correctly (more below).

After receiving a positive integer **P**, you should send back a single line with a positive integer Q ($1 \leq Q \leq 10^{12} - 10^{10} + 1$) representing that you are removing coins numbered Q, Q + 1, ..., Q + 10^{10} - 1. Each of these coins must not have been previously removed during the current game.

After the judge sends a `-2` or `-3`, if it was the last game, the judge will terminate and so should your program. Otherwise, the judge will proceed to send data corresponding to the first exchange of the next game. The judge will not check how many games you have won or lost until all games have been processed correctly. For example, if you win **T** - 1 games and then send malformed data during the last game, you will receive a Wrong Answer verdict, regardless of the value of **W**.

After receiving a `-1`, your program should terminate to receive a Wrong Answer verdict. 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 normally and within the time limit to receive a Wrong Answer verdict instead of a Runtime Error or Time Limit Exceeded.

The seed for the random generator is predetermined (and is different) for each game. This means that two submissions that make the exact same sequence of moves in a given game will receive the exact same sequence of moves from the AI for that game. It also means the play of the AI in a game does not depend, even in the pseudo-random generation sense, on the plays made in previous games within the same test set.

Limits

Time limit: 50 seconds per test set.
Memory limit: 1GB.
T = 500.
$-3 \leq \mathbf{P} \leq 10^{12} - 10^{10} + 1$.
$\mathbf{P} \neq 0$.
P represents a valid play or valid information about the game's status, as explained above.

Test set 1 (Visible)

W = 300.

Test set 2 (Visible)

W = 475.

Test set 3 (Visible)

W = 499.

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

For simplicity, the following interaction uses 50 coins in total instead of 10^{12}, and each move removes 10 consecutive coins instead of 10^{10}. The rules are otherwise the same.

```
t, w = readline_int_list()    // reads 500 into t and 300 into w
p = readline_int()            // reads 23 into p; this is the beginning of the first game. The
                              //   AI has taken coins 23 through 32, inclusive.
printline 38 to stdout        // we decide to take coins 38 through 47, inclusive
flush stdout
p = readline_int()            // reads 3 into p. The AI has taken coins 3 through 12, inclusive.
printline 13 to stdout        // we decide to take coins 13 through 22, inclusive
                              //  (and this is our only remaining move!)
flush stdout
p = readline_int()            // reads -2 into p. We won the first game since the AI had no move.
p = readline_int()            // reads 32 into p; this is the beginning of the second game. The
                              //   AI has taken coins 32 through 41, inclusive.
printline 13 to stdout        // we decide to take coins 13 through 22, inclusive
flush stdout
p = readline_int()            // reads -3 into p. We don't know the AI's move, but it left us
                              //   with no valid move, so we lost the second game.
p = readline_int()            // reads 10 into p; this is the beginning of the third game. The
                              //   AI has taken coins 10 through 19, inclusive.
printline 0 to stdout         // we select an invalid index (coin numbering starts at 1!)
flush stdout
p = readline_int()            // reads -1 into p -- we have made a mistake!
exit                          // exits to avoid an ambiguous TLE error
```