Problem F
A Turing Machine
Time Limit: *5 Seconds*

Problem Description

You encounter an erratic professor who's been trapped inside their office for too long with a computer that no longer works. He begins ranting: "Programs these days, don't know their origins. That's the Turing machine! It's what makes the whole computer work." You try to explain that Turing machines are merely models of computation, but he cuts you off. "No, see, that's just what they want you to think. Ultimately, inside every CPU, there's a Turing machine driving the whole thing! Too bad this one's broken. We're doomed!"

You ask how you can help. "Well, unfortunately, the only way to get the computer running again would be to create a whole new Turing machine from scratch, but there's no way you can-" He notices the look on your face, gives you a curious glance, shrugs, and goes back to sweeping the floor.

You find the Turing machine blueprints (your puzzle input) on a tablet in a nearby pile of debris. Looking back up at the broken Turing machine above, you can start to identify its parts:

- A tape which contains 0 repeated infinitely to the left and right.
- A cursor, which can move left or right along the tape and read or write values at its current position.
- A set of states, each containing rules about what to do based on the current value under the cursor.
  - Each slot on the tape has two possible values: 0 (the starting value for all slots) and 1.
  - Based on whether the cursor is pointing at a 0 or a 1, the current state says what value to write at the current position of the cursor, whether to move the cursor left or right one slot, and which state to use next.

The CPU can confirm that the Turing machine is working by taking a *diagnostic checksum* after a specific number of steps (given in the blueprint). Once the specified number of steps have been executed, the Turing machine should pause; once it does, count the number of times 1 appears on the tape.

Recreate the Turing machine and save the computer! What is the *diagnostic checksum* it produces once it's working again?

Input File Format

The input format is exactly as a "blueprint" file. Essentially, it is a modified YAML format (see sample input below). State names will always be a single character. Movements will either be "left" or "right". The maximum number of steps specified will be 15,000,000

Output Format

Print out the diagnostic checksum from running the Turing machine the specified number of steps.

Sample Input

```
Begin in state A.
Perform a diagnostic checksum after 6 steps.

In state A:
  If the current value is 0:
    - Write the value 1.
    - Move one slot to the right.
    - Continue with state B.
  If the current value is 1:
    - Write the value 0.
    - Move one slot to the left.
    - Continue with state B.

In state B:
  If the current value is 0:
    - Write the value 1.
    - Move one slot to the left.
    - Continue with state A.
  If the current value is 1:
    - Write the value 1.
    - Move one slot to the right.
    - Continue with state A.
```

Output for the Sample Input

```
3
```

Trace of Sample Input when simulated

Running it until the number of steps required to take the listed *diagnostic checksum* would result in the following tape configurations (with the *cursor* marked in square brackets):

```
... 0  0  0 [0] 0  0 ... (before any steps; about to run state A)
... 0  0  0  1 [0] 0 ... (after 1 step;     about to run state B)
... 0  0  0 [1] 1  0 ... (after 2 steps;    about to run state A)
... 0  0 [0] 0  1  0 ... (after 3 steps;    about to run state B)
... 0 [0] 1  0  1  0 ... (after 4 steps;    about to run state A)
... 0  1 [1] 0  1  0 ... (after 5 steps;    about to run state B)
... 0  1  1 [0] 1  0 ... (after 6 steps;    about to run state A)
```