Problem 6: Big Ben's Bullion Bewitches Busy Beaver 6+3+3 Points

Problem ID: `amnesia`
Rank: `2+2+3`

Introduction

Under investigation for tax fraud, Big Ben withdraws his earnings for the 2023 fiscal year from the bank and hastily invests it all into bricks made of various rare materials. As a recently hired special agent for the IRS, Busy Beaver successfully infiltrates his secret lair in the depths of Soda Hall. But while collecting evidence to indict him, Busy Beaver runs into a problem—the bricks are enchanted with the cruel curse of long short term memory loss!

Problem Statement

This is a *special* interactive problem! Please *carefully* read the Special Interaction Format section before attempting to solve this problem!

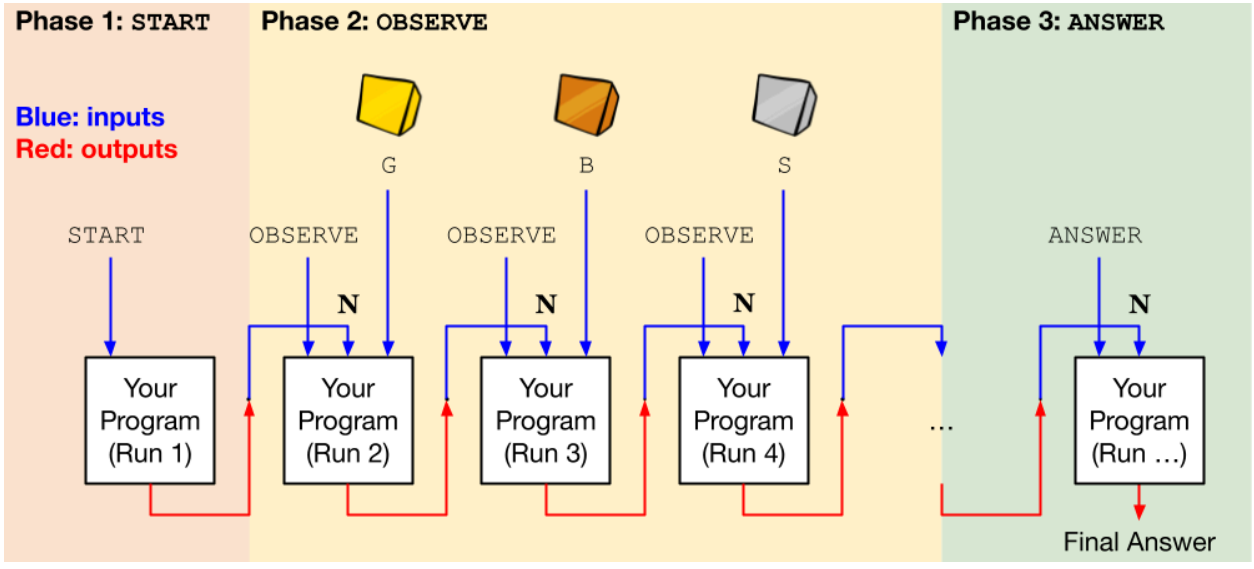
There are 3 types of bricks: bronze, silver, and gold. You don't know exactly how many of each type there are, but you know there are **~50** (between 40 and 60) of some type, **~125** (between 115 and 135) of another, and **~225** (between 215 and 235) of the last.

The goal is to order the brick types from least frequent to most frequent after being given them across multiple runs of your program by **designing a strategy** around using the persistent variable $\mathbf{N}$ to overcome the memory loss of program termination.

Your program will be run multiple times across 3 different phases:

1. In phase 1, your program will be run once to determine the initial value of a persistent fixed-size digit string called $\mathbf{N}$. This is the only variable that will persist because at the end of each run, your program will **terminate and all variables stored in memory will be lost**.
2. In phase 2, your program will be run once for each brick. The bricks will be given as input one at a time in an arbitrary **(not necessarily random)** order. Each time your program is run, the value of $\mathbf{N}$ from the *previous* run will be given as input, and it will be able to output a new value of $\mathbf{N}$ for the next run.
3. In phase 3, your program will be run once to submit the final answer after being given the value of $\mathbf{N}$ from the last run of phase 2 as input.

Below is a diagram that illustrates the flow of inputs/outputs. See next section for details.



Special Interaction Format

Prior to interaction, the judge decides on the quantities of each type of brick and the order in which they will be given. This information will not be directly provided to you. Interaction will then proceed across the following 3 phases in order.

Phase 1: `START`

In phase 1, your program will be run exactly once.

Your program will be given a single line of input containing the string `START`

Your program should output a single line containing a string of digits $\mathbf{N}$, denoting the initial value of the persistent variable. If $\mathbf{N}$ is longer than the longest allowed length for this test set (see constraints below), you will receive a wrong answer verdict.

Your program will then be terminated, and all variables stored in memory will be lost.

Phase 2: `OBSERVE`

During phase 2, your program will be run multiple times—once for each brick.

Your program will be given 3 lines of input:

- The first line contains the string `OBSERVE`
- The second line contains a string of digits $\mathbf{N}$, denoting the value of the persistent variable. This will be the same value your program outputted for each previous run.
- The third line contains a single letter, one of `B`, `S`, or `G`, (bronze, silver, or gold) denoting the color of the current brick being given.

Your program should output a single line containing a string of digits $\mathbf{N}$, denoting the updated persistent value. If the value is a different length than the value given by input (for example, `719361` was given as input but `26491` was output), you will receive a wrong answer verdict.

Your program will then be terminated, and all variables stored in memory will be lost.

Phase 3: `ANSWER`

During phase 3, your program will be run exactly once.

Your program will be given 2 lines of input:

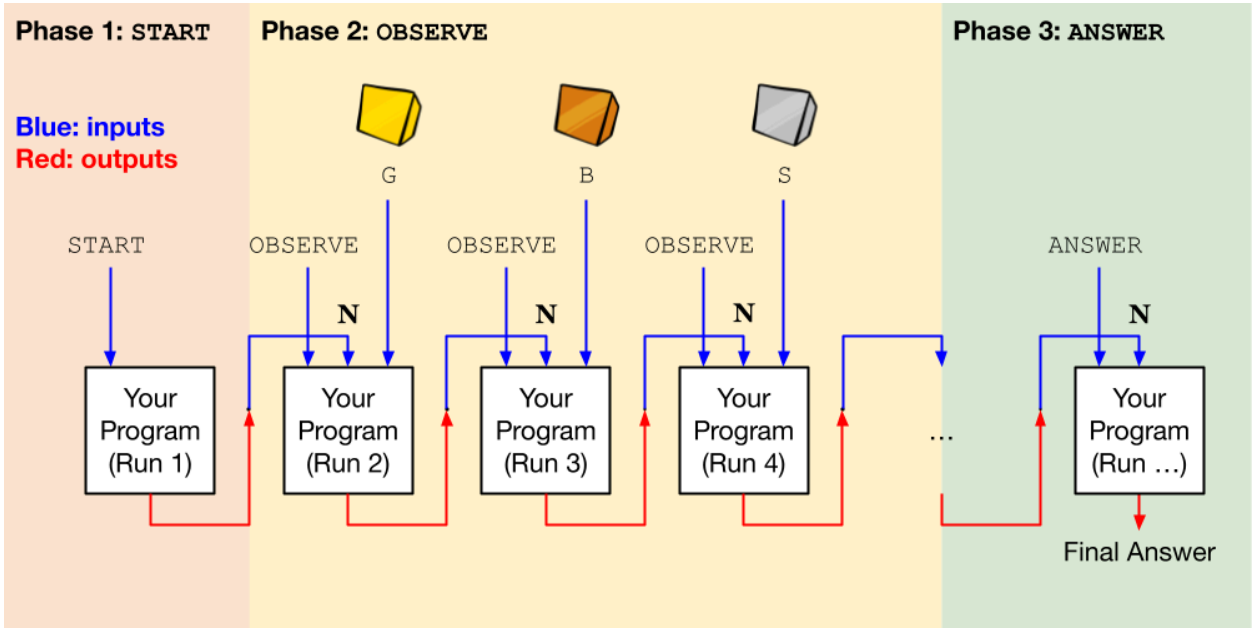
- The first line contains the string `ANSWER`
- The second line contains a string of digits $\mathbf{N}$, denoting the persistent value. This will be the same value outputted from the previous run.

Your program should output a single line containing a single string of length 3 containing each of the characters `B`, `S`, and `G` exactly once, denoting the types from least to most frequent. For example, if silver was the least frequent, gold was in the middle, and bronze was the most frequent, then you should output `SGB`

If the answer you output is correct, you will receive an accepted verdict. Otherwise, you will receive a wrong answer verdict.

If at any point your program deviates from the interaction format (e.g. treats a phase 1 interaction as a phase 2 interaction, outputs an invalid value of $\mathbf{N}$, etc.), your program will be terminated. Your program will not be run any more and you will receive a wrong answer verdict.

Below is the same diagram that illustrates the flow of inputs/outputs:



Constraints

Time Limit: **0.1 seconds per run.**
Due to the nature of submissions taking a long time, **there will only be 11 test cases in total** (1 sample + 10 hidden).

There are always exactly 3 types of bricks: bronze, silver, and gold.
Their quantities are always **~50** (40-60), **~125** (115-135), and **~225** (215-235).
$\mathbf{N}$ can only store the digits (`0123456789`) and no other characters.

Main Test Set

$1 \le |\mathbf{N}| \le 9$; $\mathbf{N}$ can store up to 9 digits.

Bonus Test Set 1

$1 \le |\mathbf{N}| \le 6$; $\mathbf{N}$ can store up to 6 digits.

Bonus Test Set 2

$1 \le |\mathbf{N}| \le 4$; $\mathbf{N}$ can store up to 4 digits.

Important Notes

Due to the nature of running programs many times, It may take a few minutes to run each submission against all test sets, so **please be patient and don't spam**. Spamming will result in your team's submissions getting deprioritized by the judge. Remember that penalty time is taken from the time of submission, not the time that the judging finishes, so **as long as you submit in time, it will be eventually judged**.

Additionally, **Java and Python submissions will take *significantly longer* to test than C/C++**, but this problem is still solvable using Java and Python.

Finally, we'd like to remind contestants that your program should **only send and receive information through standard input/output** as described in the interaction format. Attempting to circumvent this by tampering with the judge system (for example by reading/writing files, modifying environment variables, etc.) is strictly forbidden and **may result in disqualification**. The judge will also attempt to use fake data/rollback attacks to catch suspicious submissions, but this will not count towards your time limit.

Sample Interaction

Sample Input (Run 1) (Phase 1)

```
START
```

Sample Output (Run 1) (Phase 1)

```
12345678
```

Sample Input (Run 2) (Phase 2)

```
OBSERVE
12345678
G
```

Sample Output (Run 2) (Phase 2)

```
87654321
```

Sample Input (Run 3) (Phase 2)

```
OBSERVE
87654321
B
```

Sample Output (Run 3) (Phase 2)

```
12345678
```

Sample Input (Run 4) (Phase 2)

```
OBSERVE
12345678
S
```

Sample Output (Run 4) (Phase 2)

```
69696969
```

391 runs omitted here (runs 5 - 395).

Sample Input (Run 396) (Phase 3)

```
ANSWER
12629834
```

Sample Output (Run 500) (Phase 3)

```
SGB
```

Sample Explanations

Prior to the first run, the judge decides that there will be 218 bronze bricks, 42 silver bricks, and 134 gold bricks. This makes 394 bricks in total. The judge then decides on an ordering such that the first three bricks are gold, bronze, and silver.

On the first run, we begin in the first phase. The judge inputs `START`. The program outputs `12345678` as the first value of $\mathbf{N}$. Since this is an 8 digit number, all future values must also be 8 digit numbers. Additionally, this will only work in the main test set as the bonuses require you to use fewer digits.

On the second run, we begin the second phase, so the judge inputs `OBSERVE`, followed by the value of $\mathbf{N}$ that was outputted in the first run, `12345678`, followed by the type of the first brick, `G`. After inputting all of this information, the program outputs the next value of $\mathbf{N}$ as `87654321`

On the third run, we are still in the second phase, so the same thing happens but with `B` and the value of $\mathbf{N}$ updates from `87654321` to `12345678`. Note that this is the same value of $\mathbf{N}$ as at the start of the second run and values can be repeated across runs.

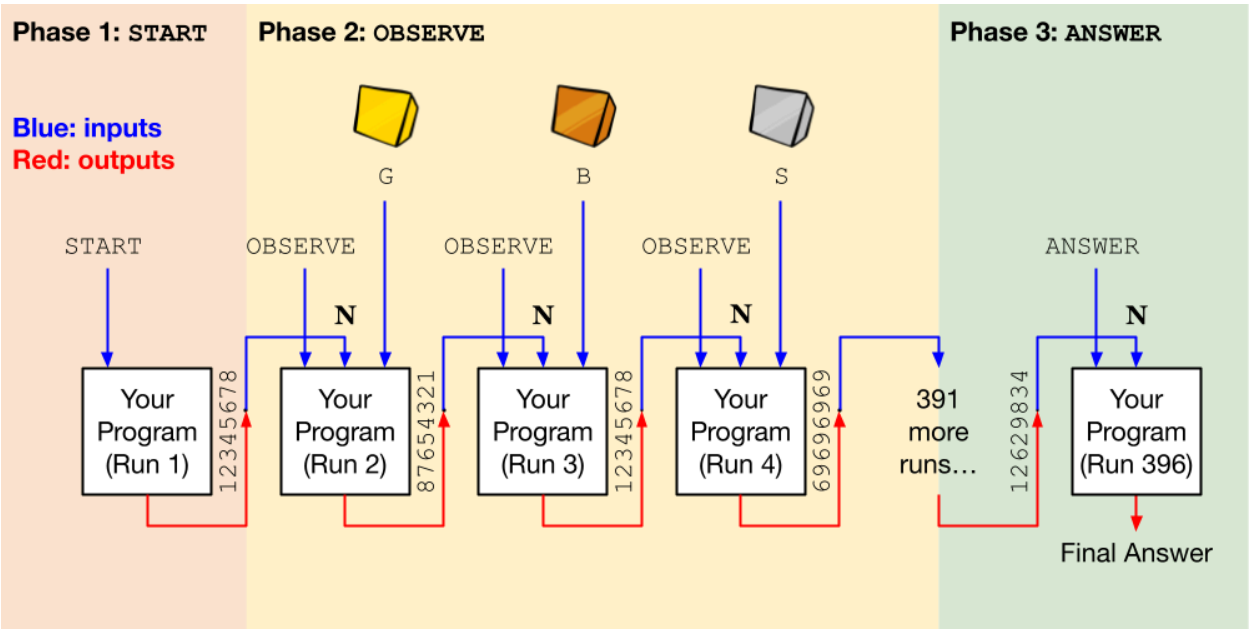
On the fourth run, the same thing happens but with `S`, and $\mathbf{N}$ updates from `12345678` to `69696969`

Between the fifth and the 395th run, the program continues and inputs new bricks and updates $\mathbf{N}$, with the final value after the 395th run being `12629834`

On the 495th run, all bricks have been observed, so we enter the third phase. The judge inputs `ANSWER` followed by the value of $\mathbf{N}$ that was outputted in the 494th run, `12629834`. The program outputs the final answer, `SGB`, claiming that silver is the least frequent, gold is the middle frequent, and bronze is the most frequent.

As the judge decided initially on 218 bronze bricks, 42 silver bricks, and 134 gold bricks, this answer is correct. No more runs are made, and the judge gives an accepted verdict.

Below is a diagram that illustrates this sample interaction with added labels for each $\mathbf{N}$:



This page intentionally left blank to preserve an even page count.