East Central North America Regional Contest 2015

ECNA 2015

Practice Session

October 30

acm International Collegiate Programming Contest

IBM event sponsor

Problems

Do not open before the contest has started.

Problem A
Bottled-Up Feelings

Peter is expecting a large shipment of fuel oil, but he has a small problem (doesn't everyone in these programming problems!). The only containers he has are a set of large bottles (each with the same volume) and a set of smaller bottles (also each with the same, but smaller volume). Given the volume of the shipment of oil, he would like to store the oil in the bottles so that

1. all of the oil is stored,
2. each bottle is filled to the top, and
3. the minimum number of bottles is used.

While Peter thinks he has solved this problem for his given bottle sizes, he often spends hours wondering what would happen if his bottles had different volumes (apparently Peter doesn't lead the most exciting life).

Input

The input consists of a single line containing three positive integers s v_1 v_2, where $s \leq 10^6$ is the volume of the shipment, and $v_1, v_2 \leq 10^6$ are the volumes of the two types of bottles, with $v_1 > v_2$.

Output

Output the number of bottles of size v_1 and the number of bottles of size v_2 which satisfy Peter's two conditions. If the conditions cannot be met, output `Impossible`.

Sample Input 1	Sample Output 1
`1000 9 7`	`108 4`

Sample Input 2	Sample Output 2
`1000 900 7`	`Impossible`

Sample Input 3	Sample Output 3
`1000 10 7`	`100 0`

This page is intentionally left blank.

Problem B
Mastering Mastermind

Mastermind is a two-person code breaking game which works as follows. The first person (the *code maker*) creates a sequence of n colored pegs (with duplicate colors allowed) and hides it from view. This sequence of pegs is the *code*.

The second person (the *code breaker*) has the job of trying to determine the code maker's code and she does so by making a series of guesses. Each guess also consists of n colored pegs. After each guess, the code maker gives the code breaker feedback about how close she is. This feedback consists of two number r and s, where

- $r =$ the number of pegs that are identical in both color and position in the code and the guess, and
- $s =$ the number of remaining pegs that are identical in color but not in the same position.

For example, if the code is BACC (where we use different letters to represent colors) and the guess is CABB, then $r = 1$ (the A in the second position of both the code and the guess) and $s = 2$ (a B and C in the remaining three characters). Note that only one of the B's in the guess will "match" with the single B in the code: once pegs in the code and the guess have been "matched" with each other, they can't be matched with any other pegs.

Your job in this problem is to determine r and s given a code and a guess.

Input

The input is a single line containing a positive integer $n \leq 50$ (the length of the code) followed by two strings of length n — the first of these is the code and the second is the guess. Both code and guess are made up of upper-case alphabetic characters.

Output

Output the values of r and s for the given input.

Sample Input 1	Sample Output 1
`4 BACC CABB`	`1 2`

Sample Input 2	Sample Output 2
`13 ABCDEFGHIJKLM NOPQRSTUVWXYZ`	`0 0`

This page is intentionally left blank.

Problem C
Shuffling Along

Most of you have played card games (and if you haven't, why not???) in which the deck of cards is randomized by shuffling it one or more times.

A *perfect shuffle* is a type of shuffle where the initial deck is divided exactly in half, and the two halves are perfectly interleaved. For example, a deck consisting of eight cards ABCDEFGH (where A is the top card of the deck) would be divided into two halves ABCD and EFGH and then interleaved to get AEBFCGDH. Note that in this shuffle the original top card (A) stays on top — this type of perfect shuffle is called an *out-shuffle*. An equally valid perfect shuffle would start with the first card from the second half and result in EAFBGCHD — this is known as an *in-shuffle*.

While normal shuffling does a good job at randomizing a deck, perfect shuffles result in only a small number of possible orderings. For example, if we perform multiple out-shuffles on the deck above, we obtain the following:

$$\text{ABCDEFGH} \rightarrow \text{AEBFCGDH} \rightarrow \text{ACEGBDFH} \rightarrow \text{ABCDEFGH} \rightarrow \cdots$$

So after 3 out-shuffles, the deck is returned to its original state. A similar thing happens if we perform multiple in-shuffles on an 8-card deck, though in this case it would take 6 shuffles before we get back to where we started. With a standard 52 card deck, only 8 out-shuffles are needed before the deck is returned to its original order (talented magicians can make use of this result in many of their tricks). These shuffles can also be used on decks with an odd number of cards, but we have to be a little careful: for out-shuffles, the first half of the deck must have 1 more card than the second half; for in-shuffles, it's the exact opposite. For example, an out-shuffle on the deck ABCDE results in ADBEC, while an in-shuffle results in CADBE.

For this problem you will be given the size of a deck and must determine how many in- or out-shuffles it takes to return the deck to its pre-shuffled order.

Input

The input consists of one line containing a positive integer $n \leq 1000$ (the size of the deck) followed by either the word `in` or `out`, indicating whether you should perform in-shuffles or out-shuffles.

Output

For each test case, output the case number followed by the number of in- or out-shuffles required to return the deck to its original order.

Sample Input 1	Sample Output 1
`8 out`	`3`

Sample Input 2

```
8 in
```

Sample Output 2

```
6
```

Sample Input 3

```
52 out
```

Sample Output 3

```
8
```

Sample Input 4

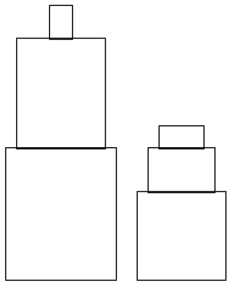
```
53 out
```

Sample Output 4

```
52
```

Problem D
A Towering Problem

You've been put in charge of an art exhibit from the famous minimalist sculptor J (even his name is minimalist!). J's work involves the careful layout of vertically dispositioned orthogonal parallelpipeds in a set of tapering obelisks — in other words, he puts smaller boxes on top of larger boxes. His most recent triumph is called "2 by 3's Decreasing," in which he has various sets of six boxes arranged in two stacks of three boxes each. One such set is shown below:



J has sent you the art exhibit and it is your job to set up each of the six-box sets at various locations throughout the museum. But when the sculptures arrived at the museum, uncultured barbarians (i.e., delivery men) simply dropped each set of six boxes on the floor, not realizing the aesthetic appeal of their original layout. You need to reconstruct each set of two towers, but you have no idea which box goes on top of the other! All you know is the following: for each set of six, you have the heights of the two towers, and you know that in any tower the largest height box is always on the bottom and the smallest height box is on the top. Armed with this information, you hope to be able to figure out which boxes go together before tomorrow night's grand opening gala.

Input

The input consists of eight positive integers. The first six represent the heights of the six boxes. These values will be given in no particular order and no two will be equal.

The last two values (which will never be the same) are the heights of the two towers.

All box heights will be ≤ 100 and the sum of the box heights will equal the sum of the tower heights.

Output

Output the heights of the three boxes in the first tower (i.e., the tower specified by the first tower height in the input), then the heights of the three boxes in the second tower. Each set of boxes should be output in order of decreasing height. Each test case will have a unique answer.

Sample Input 1

```
12 8 2 4 10 3 25 14
```

Sample Output 1

```
12 10 3 8 4 2
```

Sample Input 2

```
12 17 36 37 51 63 92 124
```

Sample Output 2

```
63 17 12 51 37 36
```