

1. The Lower Bovonian Intelligence Service employs a rather simple-minded cipher with which to encode its missives. They encode a piece of plaintext,  $p_1p_2\dots p_n$ , as  $c_1c_2\dots c_n$ , where

$$c_i = \begin{cases} 21, & \text{if } i = 0 \\ (\alpha \cdot c_{i-1} + \beta \cdot p_i + \gamma) \bmod 89, & \text{if } i > 0. \end{cases}$$

where  $\alpha$ ,  $\beta$ , and  $\gamma$  are integers with  $0 < \alpha, \beta < 89$  and  $0 \leq \gamma < 89$ . The characters are encoded as integers in the range  $[0..88]$  such that the value  $x$  encodes the character whose ASCII code is  $x + 34$  (thus 0 is ‘’, 1 is ‘#’, 63 is ‘a’, etc.)

This isn’t a good cipher, and the Lower Bovonians don’t use it all that well either. They tend to change the values of the key values  $\alpha$ ,  $\beta$ , and  $\gamma$  only about once a day, and to send some particularly predictable messages each day as well. All of this is good for you, a member of the Upper Bovonian Intelligence Service. You’d like to take advantage of the Lower Bovonian’s naïvety by writing a program that, given an intercepted encrypted message, its (known or guessed-at) plaintext translation, and a second encrypted message (whose plaintext is unknown) and decrypts the second message, using the same values of  $\alpha$ ,  $\beta$ , and  $\gamma$  that decode the first message. For example, given the two encrypted messages

```
BNi [DH21i [2C(\ki.G=2]Ei:5S?8(V/Tsa#sni3EpX#^
<=K;0\Y$fF]--zy?RrAy$ZK GK;]8h10V_
```

and assuming that you somehow know or guess that the first one translates as

```
The_quick_brown_fox_jumps_over_the_lazy_dog.
```

you can calculate that  $\alpha = 7$ ,  $\beta = 19$ ,  $\gamma = 3$ , and that the second encrypted message decrypts to

```
One_if_by_land_and_two_if_by_sea.
```

The input to your program will consist of a sequence of groups of three messages in free format (none of the encodable characters is whitespace). The second line in each group will be the decryption of the first. The third line is the line you are to decrypt. The input is encoded as ASCII as usual, so you will have to translate back and forth to the Bovonian character set. You may assume that each ciphertext/plaintext pair of messages gives you enough information to decode the unknown ciphertext message in each group of three. Print your decryption using the format shown in the example below.

**Example.**

Input	Output
<pre>BNi [DH21i [2C(\ki.G=2]Ei:5S?8(V/Tsa#sni3EpX#^ The_quick_brown_fox_jumps_over_the_lazy_dog. &lt;=K;0\Y\$fF]--zy?RrAy\$ZKGK;]8h10V_  &lt;=K;0\Y\$fF]--zy?RrAy\$ZKGK;]8h10V_ One_if_by_land_and_two_if_by_sea. 0B;i(c7a&lt;IeX0[jbC)K4LAta#Yi[Xb*XIEjh9A&amp;s</pre>	<pre>Message #1: One_if_by_land_and_two_if_by_sea. Message #2: England_expects_each_man_to_do_his_duty.</pre>

2. [From the UVa Online Judge]

Consider the situation of an ideal forest, where trees grow on a regular finite euclidean lattice (that is, where all points have integral coordinates). At every site only one tree grows, and it can be of one among  $n$  species. Each species is denoted by a single non-whitespace character (e.g.,  $\{A, B, C, 0, x\}$  are all valid species names). Two trees of the same species are considered neighbors if the maximum absolute difference between their coordinates is one.

Families of (rather specialized) monkeys are released, one family at a time, in this forest onto the next unoccupied tree scanning left to right, top to bottom, until all trees are occupied. Upon release, each family will immediately spread to all neighboring trees of a single species that have not been taken yet by another family.

Given the map of the forest, build the map of the monkey families, starting with family "1" and numbering them consecutively. The input contains a sequence of matrices. Each matrix consists of one or more rows, one per line. Each row contains a sequence of single characters, indicating tree species, separated by whitespace. All rows of a matrix will have the same number of entries. The last row is followed by a blank line.

For each matrix, print a line of integers for each row, giving the number of the family that ends up sitting in that tree. Use the format shown in the example, separating the output matrices with blank lines.

**Example.**

Input	Output
A B D E C C D	1 2 3 4 5 5 3
F F W D D D D	6 6 7 3 3 3 3
P W E W W W W	8 7 9 7 7 7 7
a A b B c d E t	1 2 3 4 5 6 7 8
a a a a a c c t	1 1 1 1 1 5 5 8
e f g h c a a t	9 10 11 12 5 1 1 8

**3.** [From the UVa Online Judge] This problem involves describing blood relationships between people. If two people, say Mary and Fred, have a common ancestor, Jack, and if Mary is  $m + 1$  generations removed from Jack, while Fred is  $m + 1 + n$  generations removed ( $m, n \geq 0$ ), we say that Mary and Fred are “ $m^{\text{th}}$  cousins  $n$  times removed,” which we’ll write as **cousin- $m-n$** . By this definition, siblings are  $0^{\text{th}}$  cousins (0 times removed). We’ll write the relationship between Jack and Mary as **descendant- $m + 1$** , and between Jack and Fred as **descendant- $m + 1 + n$** .

A relationship **cousin- $m_1-n_1$**  is *closer* than a relationship **cousin- $m_2-n_2$**  if  $m_1 < m_2$  or else  $m_1 = m_2$  and  $n_1 < n_2$ . A relationship **descendant- $p_1$**  is closer than a relationship **descendant- $p_2$**  if  $p_1 < p_2$ . A descendant relationship is always closer than a cousin relationship.

Write a program that accepts definitions of simple relationships between individuals and displays the closest cousin or descendant relationship, if any, that exists between arbitrary pairs of individuals.

The input, in free format, consists of a sequence of datasets. Each dataset consists of a sequence of relationships and queries. A relationship has the form

R *NAME1* *NAME2* *N*

where *NAME1* and *NAME2* are distinct names (containing no whitespace), and  $N \geq 0$  indicates the distance of the relationship ( $N = 0$  for siblings,  $N = 1$  if *NAME1* is a child of *NAME2*,  $N = 2$  if *NAME1* is a grandchild of *NAME2*, and so forth). If two people are siblings, they are children of someone, even if that person is not mentioned by name anywhere in the data. A query has the form

F *NAME1* *NAME2*

and inquires as to the relationship between the two named individuals. A single letter E marks the end of each dataset.

Label the results for each dataset as shown in the example. For each F query,

F *NAME1* *NAME2*

your program should print the closest relationship between *NAME1* and *NAME2* using one of these formats, as appropriate:

*NAME1* and *NAME2* are descendant- $p$ .

*NAME1* and *NAME2* are cousin- $m-n$ .

*NAME1* and *NAME2* are not related.

Answer each query using only the information in the R entries that precede it in the dataset. Assume that a relationship between two people exists only if it necessarily follows from the data. For example, knowing only that Jack and Jill are siblings (R Jack Jill 0) and Jill is a child of Ethan (R Jill Ethan 1) does not tell you that Jack is a child of Ethan (Jack could be Jill’s half-brother). Therefore, you must assume that Jack is *not* Ethan’s son, absent more specific information.

**Example.**

Input	Output
R Fred Joe 1 R Fran Fred 2 R Jake Fred 1 R Bill Joe 1 R Bill Sue 1 R Jean Sue 1 R Jean Don 1 R Phil Jean 3 R Stan Jean 1 R John Jean 1 R Mary Don 1 R Susan Mary 4 R Peg Mary 2 F Fred Joe F Jean Jake F Phil Bill F Phil Susan F Jake Bill F Don Sue F Stan John F Peg John F Jean Susan F Fran Peg F John Avram R Avram Stan 99 F John Avram F Avram Phil E F Jake Bill E	Dataset #1: Fred and Joe are descendant-1. Jean and Jake are not related. Phil and Bill are cousin-0-3. Phil and Susan are cousin-3-1. Jake and Bill are cousin-0-1. Don and Sue are not related. Stan and John are cousin-0-0. Peg and John are cousin-1-1. Jean and Susan are cousin-0-4. Fran and Peg are not related. John and Avram are not related. John and Avram are cousin-0-99. Avram and Phil are cousin-2-97.  Dataset #2: Jake and Bill are not related.