Problem A
Does Not Compute
Input file: compute.in

In some science fiction stories and films, when computers encounter an illogical situtation, their response is often "DOES NOT COMPUTE." For this problem, you will write a computer program which can simulate this behavior whenever it sees arithmetic statements that are incorrect.

Input

The input file contains multiple test cases, each of which describes an arithmetic statement. Each statement is of the form "*A* OPERATOR *B* = *C*", where *A, B, C* are integers, and OPERATOR is "+", "-", or "*" indicating addition, subtraction, or multiplication. The integers *A* and *B* are both in the range 1 to 999, and all tokens are separated by whitespace (spaces or newlines). The last test case is followed by a line containing a single zero.

Output

For each test case, print the case number (beginning with 1) followed by either `OKAY` (if the statement is correct), or `DOES NOT COMPUTE` (if the statement is incorrect).

Follow the format of the sample output.

Sample Input	Output for the Sample Input
1 + 1 = 2 3 * 5 = 15 100 - 10 = 10 0	Case 1: OKAY Case 2: OKAY Case 3: DOES NOT COMPUTE

Problem B
Formatting Text
Input file: format.in

Computers are great tools for storing, editing, and presenting textual information. When presenting text, one of the most important factors is where to place newlines. For this problem, you should write a program that re-formats text according to a simple set of rules and special markers in the source text.

Input

The input is a sequence of tokens separated by whitespace (each separator is a single space or newline). All non-whitespace tokens are considered normal except for two special tokens: "N" and "END". The "N" token indicates that a newline should be inserted into the text, and the "END" token indicates the end of input.

Output

Print out each normal token followed by a space. For each occurrence of the special token "N", print a newline.

Follow the format of the sample output.

Sample Input	Output for the Sample Input
A man provided with paper, N pencil, N and rubber, N and subject to strict discipline, N is in effect a universal machine. N N A. M. Turing N END	A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine. A. M. Turing

Problem C
Genetic Search
Input file: genetic.in

Technology has greatly changed the field of biology in the last decade, since biological information can be digitized and analyzed by computer. One of the most basic analysis tasks is counting the number of occurrences (or near-occurrences) of a search string *S* inside of another string *L* that is at least as long as *S*.

For this problem, you are given pairs of strings *S* and *L*. Both strings contain only the uppercase characters A, G, C, and T. For each of the following types of search strings, count the number of times that type occurs as an exact substring of *L*:

- Type 1: *S*, without any changes.
- Type 2: All unique strings that can be made by deleting one character from *S* (for example, AGC can become AG, AC, or GC).
- Type 3: All unique strings that can be made by inserting one character in *S* (for example, AG can become any of the following: AAG, GAG, CAG, TAG, AGG, ACG, ATG, AGA, AGC, or AGT).

If two or more different modifications of *S* result in the same string, count only the occurrences of that string once.

Input

The input file contains multiple test cases, each of which contains two strings: *S* followed by *L*. The length of *S* is at least 2 characters and at most the length of *L*. The length of *L* is at most 100 characters. The last test case is followed by a line containing a single zero.

Output

For each test case, print the case number (beginning with 1) followed by the number of occurrences of Type 1, then Type 2, then Type 3.

Follow the format of the sample output.

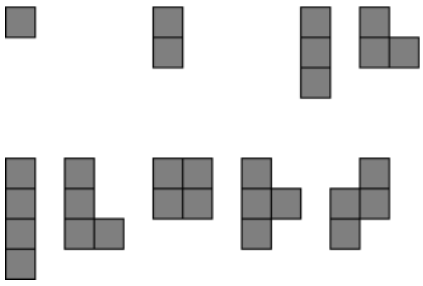
Sample Input	Output for the Sample Input
AGCT AGCTAGCT AAA AAAAAAAAAA AGC TTTTGT 0	Case 1: 2 4 2 Case 2: 8 9 7 Case 3: 0 0 0

Problem D
Unique Shapes
Input file: There is no input for this problem.

Imagine you have a lot of square blocks, and you can put them together into different shapes. You can make all kinds of shapes – but how many can you make that are truly unique?

We will consider a shape to be a connected group of blocks placed on a regular grid, where each block is connected to another block above it, below it, to its left, or to its right. Two shapes are considered unique if there is **no** sequence of rotations, translations, or reflections that can cause them to be identical.

With one block, only one unique shape can be made. With two blocks, the same is true. With three blocks, two unique shapes are possible, and with four blocks, five unique shapes are possible. The figure below gives illustrations of these cases:



Input

There is no input for this problem.

Output

Print out the number of unique shapes that can be constructed using 8 square blocks, as a decimal integer.

Sample Input	Output for the Sample Input
There is no input for this problem.	There is no sample output for this problem.