Problem 5-Roman Ciphers

Lucretius has proposed a novel way to decrypt a message. He just applies the encryption key a number of times until the original message emerges. For example, if A maps to Z, B maps to Y, and so forth, applying the encryption key twice always yields the original message. If A maps to B, B maps to C, C maps to D, and so forth with Z mapping to A, the original message returns after encrypting it 26 times. Imagine you have an alphabet with n letters, where n can be as large as 1000. You are given an encryption cipher, and you are to compute the smallest positive number of times the encryption key must be applied to a message to guarantee that the original message has been restored. HINT: You will never get it if you apply the cipher over and over until you get the original order. This will take too long. Look for cycles in the cipher.

INPUT SPECIFICATION. Each input case begins with a positive decimal integer, n, indicating the length of the alphabet followed by ***<EOLN>***, Then will follow a permutation of the integers 1 through n, each on its own line followed by ***<EOLN>***. It is guaranteed to be valid permutation. Each integer appears once and only once. The interpretation of this list is that 1 encrypts to the first integer in the permutation, 2 encrypts to the second, and so forth. A zero followed by ***<EOLN>*** will follow the last case.

OUTPUT SPECIFICATION. The output cases should appear in the same order as their corresponding input cases. Each output case should be the following: "Case c: The cipher would have to be applied t time(s).***<EOLN>***" where c is the case number and t is the answer to the problem.

SAMPLE INPUT

```
5<EOLN>
1<EOLN>
2<EOLN>
3<EOLN>
4<EOLN>
5<EOLN>
5<EOLN>
3<EOLN>
4<EOLN>
5<EOLN>
2<EOLN>
1<EOLN>
5<EOLN>
2<EOLN>
3<EOLN>
4<EOLN>
5<EOLN>
1<EOLN>
0<EOLN>
<EOF>
```

SAMPLE OUTPUT

```
Case·1:·The·cipher·would·have·to·be·applied·1·time(s).<EOLN>
Case·2:·The·cipher·would·have·to·be·applied·6·time(s).<EOLN>
Case·3:·The·cipher·would·have·to·be·applied·5·time(s).<EOLN>
<EOF>
```