Given below is a set of rules that can be used to transform one string into another:

1. ‘t’ → ‘i’ // change ‘t’ to ‘i’
2. {‘a’ – ‘z’} → ‘m’ // change ‘a’, or ‘b’, or ‘c’, ..., or ‘z’ to ‘m’
3. ‘s’ → ‘i’
4. ‘a’ → ‘aa’
5. {‘a’ – ‘z’} → ‘’ // delete ‘a’, or ‘b’, or ‘c’, ... , or ‘z’
6. ‘t’ → ‘r’
7. {‘a’ – ‘g’} → ‘h’
8. ‘o’ → ‘a’
9. ‘t’ → ‘s’
10. {‘t’ – ‘z’} → ‘u’

Sets are always given as a range, i.e. {‘a’ – ‘c’} is the set with the characters {‘a’,‘b’,‘c’}

Write a program that reads two strings, and prints the steps required to transform the first string into the second.

Input

The input file contains several test cases with a blank line between two consecutive.

Output

For each test case, print the steps required to transform the first string into the second.

The steps in the transformation must be printed in the form:

rule: *origString* -> *newString*

Note that several transformations may exist, your program only needs to find one. It is also possible that the first string cannot be transformed into the second. In this case, your program should simply output the word ‘no’

It must be a blank line between two consecutive outputs.

Sample input

```
thinner
rih

a
x
```

Sample output

```
6: thinner -> rhinner
5: rhinner -> rinner
7: rinner -> rinnhr
5: rinnhr -> rinhr
5: rinhr -> rihr
5: rihr -> rih

no
```