536 Tree Recovery

Little Valentine liked playing with binary trees very much. Her favorite game was constructing randomly looking binary trees with capital letters in the nodes.

This is an example of one of her creations:

```
                   D
                  / \
                 /   \
                B     E
               / \     \
              /   \     \
             A     C     G
                        /
                       /
                      F
```

To record her trees for future generations, she wrote down two strings for each tree: a preorder traversal (root, left subtree, right subtree) and an inorder traversal (left subtree, root, right subtree).

For the tree drawn above the preorder traversal is `DBACEGF` and the inorder traversal is `ABCDEFG`.

She thought that such a pair of strings would give enough information to reconstruct the tree later (but she never tried it).

Now, years later, looking again at the strings, she realized that reconstructing the trees was indeed possible, but only because she never had used the same letter twice in the same tree.

However, doing the reconstruction by hand, soon turned out to be tedious.

So now she asks you to write a program that does the job for her!

Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing two strings preord and inord, representing the preorder traversal and inorder traversal of a binary tree. Both strings consist of unique capital letters. (Thus they are not longer than 26 characters.)

Input is terminated by end of file.

Output Specification

For each test case, recover Valentine's binary tree and print one line containing the tree's postorder traversal (left subtree, right subtree, root).

Sample Input

```
DBACEGF ABCDEFG
BCAD CBAD
```

Sample Output

```
ACBFGED
CDAB
```