Problem K
Up a Tree
Time Limit: 6 seconds

Anatoly Cheng McDougal is a typical student in many ways. Whenever possible he tries to cut and paste code instead of writing it from scratch. Unavoidably this approach causes him problems. For example, when he first learned about preorder, inorder and postorder traversals of trees, and was given code for a preorder print of a tree (shown on the left below), he simply cut and pasted the code, then moved the print statement to the correct location and renamed the procedure. However, he forgot to rename the procedure calls inside the code, resulting in the defective inorder print and postorder print code shown below.

```
void prePrint(TNode t)
{
  output(t.value);
  if (t.left != null)
    prePrint(t.left);
  if (t.right != null)
    prePrint(t.right);
}
```

```
void inPrint(TNode t)
{
  if (t.left != null)
    prePrint(t.left);
  output(t.value);
  if (t.right != null)
    prePrint(t.right);
}
```

```
void postPrint(TNode t)
{
  if (t.left != null)
    prePrint(t.left);
  if (t.right != null)
    prePrint(t.right);
  output(t.value);
}
```

At this point, Anatoly did not behave like a typical student. He actually tested his code! Unfortunately, when the results were not correct, he reverted back to typical student behavior. He panicked and started randomly changing calls in all three procedures, hoping to get things right. Needless to say, the situation became even worse now than when he started.

Anatoly's professor tested the code on a random tree of characters. When she looked at the output of his three print routines, she correctly guessed what had happened. However, instead of going directly to his code, she decided to try to reconstruct Anatoly's code just by observing the output. In order to do this, she correctly made the following assumptions:

1. The output statement in each print routine is in the correct location (for example, between the two recursive calls in the `inPrint` routine).
2. Among the six recursive calls made by the three routines, exactly two calls are to `prePrint`, exactly two are to `inPrint`, and exactly two are to `postPrint`, though potentially in the wrong routines.

Soon the professor realized that reconstructing Anatoly's code and the test tree from his output was not a simple task and that the result might be ambiguous. You will have to help her find all possible reconstructions of Anatoly's code. In addition, for each such reconstruction, you are to find the alphabetically first tree (as described in the output section) giving the observed output.

Input

The input consists of a single test case. A test case consists of three strings on three separate lines: the observed output of Anatoly's `prePrint`, `inPrint` and `postPrint` routines (in that order) on some test tree. Each of these strings consists of n uppercase letters ($4 \leq n \leq 26$), with no repeated letters in any string. The test case is guaranteed to have at least one solution.

Output

Display all possible reconstructions for the test case, ordered as described in the last paragraph below. The output for each reconstruction consists of two parts. The first part is a single line and describes the six calls in Anatoly's routines: first the two (recursive) calls in Anatoly's `prePrint` routine, followed by the calls in his `inPrint` routine, and finally the calls in his `postPrint` routine. The calls are described by the words `Pre`, `In`, and `Post`, separated by spaces. For example, if Anatoly's routines were correct, the resulting output of the first part of the reconstruction would be `Pre Pre In In Post Post`.

The second part consists of three lines and describes the first test tree that could have generated the observed outputs. The first line is the *correct* preorder print of the tree, and the second and third lines contain the correct inorder and postorder prints, respectively. The first tree is the one with the alphabetically first preorder print. If there are multiple such trees, the first of these is the one with the alphabetically first inorder print.

Every reconstruction is a sequence of 6 tokens chosen from `Pre`, `In`, and `Post`. The ordering of reconstructions is lexicographic with respect to the following ordering of tokens: `Pre < In < Post`.

Sample Input 1	Sample Output 1
HFBIGEDCJA BIGEDCJFAH BIGEDCJFAH	Pre Post In Post In Pre HFBJCDEGIA BIGEDCJFAH IGEDCJBAFH

Sample Input 2	Sample Output 2
BNLFAGHPEDOCMJIK NLBGAPHCODEIJMKF NLFAGHPEDOCMJIKB	In Pre In Post Post Pre BLNFKMEHAGPCODIJ NLBAGHPEODCMIJKF NLGAPHDOCEJIMKFB Post Pre In In Post Pre BLNFKICPGAHEODMJ NLBGAPHCODEIJMKF NLAGHPDOECJMIKFB