Source

An important problem of software maintenance is to track down duplication in a large software system. One would like to find not only exact matches between sections of code, but parameterized matches, where a parameterized match between two sections of code means that one section can be transformed into the other by replacing the parameter names (e.g., identifiers and constants) of one section by the parameter names of the other via a one-to-one (bijective) function.

Let Σ and Π be two alphabets: Σ is the upper case English alphabet and Π is the lower case English alphabet. Each symbol in Σ represents a token and each symbol in Π represents a parameter.

A string can consist of any combinations of tokens and parameters from Σ and Π. Two strings *A* and *B* are said to *p-match* if and only if

1. *A* and *B* have the same length (more formally: *length(A) = length(B)*)
2. Each token in *A* matches a token in *B* and each parameter in *A* matches a parameter in *B* (more formally: (A_i is a token and B_i is a token) or (A_i is a parameter and B_i is a parameter) for any 1 <= *i* <= *length(A)*)
3. The alignment of tokens in *A* and *B* is a perfect match (more formally: if A_i is a token then $A_i = B_i$ for any 1 <= *i* <= *length(A)*)
4. The alignment of parameters in *A* and *B* defines a one-to-one correspondence between parameter names in *A* and parameter names in *B* (more formally: there exists a one-to-one (bijective) function *f*: Π → Π such that if A_i is a parameter $f(A_i) = B_i$ for any 1 <= *i* <= *length(A)*)

A token represents a part of the program that cannot be changed, whereas a parameter represents a program's variable, which can be renamed as long as all occurrences of the variable are renamed consistently. Thus if *A* and *B* *p-match*, then the variable names in *A* could be changed to the corresponding variable names in *B*, making the two programs identical. If these two problems were part of a larger program, then they could both be replaced by a call to a single subroutine.

Task

Given a text *T* and a pattern *P*, each a string over Σ and Π, find all substrings of *T* that *p-match P*.

Input data

The first line of the input file `source.in` contains the string *P* followed by the second line containing the string *T*.

Output data

The output file `source.out` must contain on the first line the number of substrings of *T* that *p-match P*. On the second line the offsets of those substrings are to be printed separated by a space (by convention, the prefixes of *T* have offset 1).

Limits and restrictions

- $1 \leq length(T) \leq 1{,}000{,}000$
- $1 \leq length(P) \leq 10{,}000$
- Time limit: 0.07 seconds
- Memory limit: 16 MB

Example

source.in	source.out	Explanation
XYabCaCXZddbW xXYdxCdCXZccxW	1 2	The pattern XYabCaCXZddbW *p-matchs* the 2-offset text substring XYdxCdCXZccxW but does not *p-match* the 1-offset text substring xXYdxCdCXZccx. Notice that when the *p-match* occurs, the following one-to-one (bijective) function *f* is used: *f*(a) = d *f*(b) = x *f*(d) = c