

Problem D. Hidden Anagrams

Input file: *standard input*
Output file: *standard output*
Time limit: 20 seconds
Memory limit: 512 mebibytes

An *anagram* is a word or a phrase that is formed by rearranging the letters of another. For instance, by rearranging the letters of “William Shakespeare,” we can have its anagrams “I am a weakish speller,” “I’ll make a wise phrase,” and so on. Note that when A is an anagram of B , B is an anagram of A .

In the above examples, differences in letter cases are ignored, and word spaces and punctuation symbols are freely inserted and/or removed. These rules are common but not applied here; only exact matching of the letters is considered.

For two strings s_1 and s_2 of letters, if a substring s'_1 of s_1 is an anagram of a substring s'_2 of s_2 , we call s'_1 a *hidden anagram* of the two strings, s_1 and s_2 . Of course, s'_2 is also a *hidden anagram* of them.

Your task is to write a program that, for given two strings, computes the length of the longest hidden anagrams of them.

Suppose, for instance, that “anagram” and “grandmother” are given. Their substrings “nagr” and “gran” are hidden anagrams since by moving letters you can have one from the other. They are the longest since any substrings of “grandmother” of lengths five or more must contain ‘d’ or ‘o’ that “anagram” does not. In this case, therefore, the length of the longest hidden anagrams is four. Note that a substring must be a sequence of letters occurring *consecutively* in the original string and so “nagrm” and “granm” are not hidden anagrams.

Input

Input consists of two strings s_1 and s_2 consisting of lowercase letters (‘a’ through ‘z’) and their lengths are between 1 and 4000, inclusive.

Output

Output the length of the longest hidden anagrams of s_1 and s_2 . If there are no hidden anagrams, print a zero.

Examples

standard input	standard output
anagram grandmother	4
williamshakespeare iamaweakishspeller	18
aaaaaaaaabbbbbbbb xxxxxabababxxxxxabab	6
abababacdcdcd efefefghghghghgh	0