Challenge 2 - Did you mean...?

You work in a company that's main product is a search engine. You have noticed that users make a lot of mistakes while typing :) And that the most common mistake is that they type a very similar word instead of the one they are looking for. So what you have decided is to make suggestions to the users while they are typing. You want to suggest valid words from the dictionary that contain the same letters (and the same number of each letter) but in different orders.

For example, a user types "act", you might want to suggest "cat". In the same way, if user types "elvis", your suggestion should be "lives".

Your goal is to find all possible suggestions for a set of words and a given dictionary. Assume that all the words in the dictionary are correct.

Input

The input consists of comments (lines starting with '#'), name of the dictionary file to use, number of words that require suggestions and the list of words that require suggestions. Comments describe and separate each part of the intput:

```
#Dictionary file
name of the file containting the dictionary
#Suggestion numbers
Integer N, representing the number of words provided, one per line, that require
suggestions
#Find the suggestions
List of words for which you need to find the suggestions, each one in a different
line
```

The dictionary file is nothing else than a list of words.

Output

The output must be a list of

```
word -> suggestion1 suggestion2 ...
```

each one on a different line. The suggestions for each word, if more than one, need to be **in alphabetical order.**

If no suggestion for a word can be found, then output

```
word ->
```

for that word.

Sample input

```
#Dictionary file
```

```
dictionary
#Suggestion numbers
3
#Find the suggestions
elvis
lactoprotein
nosuggestion
```

Sample output

```
elvis -> lives velis
lactoprotein -> protectional
nosuggestion ->
```

Sample dictionary

```
gainly
laying
protectional
lactoprotein
elvis
lives
velis
nosuggestion
```

The dictionaries can be found here and you will need to unzip it in the folder where you run your program.

Submit & test your code

To test and submit code we provide a set of tools to help you. Download contest tools if you haven't already done that. You will then be able to test your solution to this challenge with the challenge tokens.

```
Challenge tokens: CHALLENGE_2, CHALLENGE_SUBMIT_2
```

To test your program

```
./test_challenge CHALLENGE_2 path/program
```

A nice output will tell you if your program got the right solution or not. You can try as many times as you need.

To test your program against the input provided in the submit phase

```
./test_challenge CHALLENGE_SUBMIT_2 path/program
```

During the submit phase, in some problems, we might give your program harder inputs. As with the test token, a nice output will tell you if your program got the right solution or not. You can try as many times as you need.

In the actual contest you first need to solve the test phase before submitting the code, you must provide the source code used to solve the challenge and you can only submit once (once your solution is submitted you won't be able to amend it to fix issues or make it faster).

If you have any doubts, please check the info section.

Problem stats

Completion time:	**10 percentile:** 1:03 h **90 percentile:** 26:19 h
Submit exec time:	**min:** 0.57 s **10 percentile:** 1.26 s **90 percentile:** 932.00 s **max:** 16207.05 s
Test tries:	**min:** 1 **10 percentile:** 2 **90 percentile:** 36 **max:** 187
# of completions:	420

« Prev Next »

Tweet about this! #TuentiChallenge3

Share Follow @Tuentieng