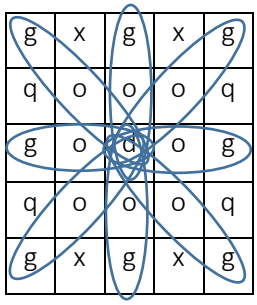
Word Search

The Academy of Young Europeans (AYE) is organizing a Word Search competition. In this competition, competitors are given a grid of letters and a target word. They are supposed to find as many occurrences of the word as possible within a time limit. The word can be in any direction be it horizontal, vertical, diagonal, forwards or backwards. They get 1 point for every occurrence they find and the team with the most points wins!



To help organize the competition, the AYE has employed you. They assign you one job: you need to write a program to quickly find what is the maximum score any competitor can get given a grid and a target word. That means, you want to count how many times the target word appears in the grid in any direction.

Input

The first line of input is the target word which will be a string of at least 2 and at most 10 characters in length. The string will only consist of English lower-case letters from 'a' to 'z'. Furthermore, the string will not be a palindrome, which means it will not be the same word when you reverse the word. The next line will be **blank**.

The second line of input contains a single integer **N** (1 <= **N** <= 250) representing the size of the grid.

The next **N** lines of input will represent the character grid and each line will be a string of **N** English lower-case letters from 'a' to 'z'.

Output

Print a single integer representing the number of times the target character appears in the provided grid. Your output must contain a newline character.

Sample Input

```
dog

5
gxgxg
qoooq
godog
qoooq
gxgxg
```

Sample Output

```
8
```

Explanation

All 8 occurrences of the word 'dog' start from the centre of the grid and span out in the 8 possible directions of the word, as shown in the figure above.

Skeleton

You are given the skeleton file WordSearch.java. You should see a non-empty file when opening it, otherwise you are in the wrong directory.

Notes:

1. You should develop your program in the subdirectory `ex1` and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm.
3. You do not need to use OOP for this sit-in lab.
4. You are free to define your own helper methods. **Remember to use private methods whenever possible.**
5. Please be reminded that the marking scheme is:

   Input : 10%
   Output : 10%
   Correctness : 50%
   Programming Style : 30%, which consists of:
   - Meaningful comments (pre- and post- conditions, comments inside the code): 10%
   - Modularity (incremental programming, proper modifiers [public / private]): 10%
   - Proper Indentation: 5%
   - Meaningful Identifiers (for both method and variable names): 5%

   Compilation Error : Deduction of **50% of the total marks obtained**.