Problem J

Perfect Matching

A palindrome is a word that can be read the same way in either direction (front to back, or back to front). E.g., “RACECAR”, “DEED”, “LEVEL” are palindromes, while “ADAM”, “CAR”, “WATER” are not palindromes. Of course there are trivial palindromes which are words that consist of only one character such as “A”, “B”, “C”, etc.

In this problem, you have N strings and a function named perfect which takes two arguments string A and string B which will return whether concatenation of A and B (A·B) is a palindrome or not.

For example, let there are 4 strings {“race”, “car”, “ac”, ecar”}. We can take any permutation of 2 strings out of those 4 strings:

• perfect(“race”, “car”)	yes, “racecar” is a palindrome.
• perfect(“race”, “ac”)	no, “raceac” is not a palindrome.
• perfect(“race”, “ecar”)	yes, “raceecar” is a palindrome.
• perfect(“car”, “race”)	no, “carrace” is not a palindrome.
• perfect(“car”, “ac”)	yes, “carac” is a palindrome.
• perfect(“car”, “ecar”)	no, “carecar” is not a palindrome.
• perfect(“ac”, “race”)	no, “acrace” is not a palindrome.
• perfect(“ac”, “car”)	no, “accar” is not a palindrome.
• perfect(“ac”, “ecar”)	no, “acecar” is not a palindrome.
• perfect(“ecar”, “race”)	yes, “ecarrace” is a palindrome.
• perfect(“ecar”, “car”)	no, “ecarcar” is not a palindrome.
• perfect(“ecar”, “ac”)	no, “ecarac” is not a palindrome.

Out of those 12 possible 2-permutation of 4 strings, there are 4 which have “yes” as the result: (“race”,”car”), (“race”,”ecar”), (“car”,”ac”), (“ecar”,”race”); and we call those as perfect matching. Note that A·B is considered as different from B·A, you can find in the example above both (“race”,”ecar”) and (“ecar”,”race”) are counted as a different matching although they are composed of a same set of string.

Given N strings, your task is to count how many perfect matching there are in those strings.

Input

The first line of input contains an integer T (T ≤ 50) denoting the number of cases. Each case begins with an integer N (1 ≤ N ≤ 1,000) denoting the number of strings. The next N lines, each contains a string S_i. S_i contains only lowercase alphabets (a-z) and its length is between 1 and 500, inclusive. You may assume that all strings in each case are unique.

Output

For each case, output “`Case #X: Y`”, where `X` is case number starts from 1 and `Y` is the number of perfect matching in the corresponding case.

Sample Input	Output for Sample Input

`4`
`4`
`race`
`car`
`ac`
`ecar`
`3`
`xyz`
`abac`
`aba`
`5`
`abc`
`ba`
`xab`
`x`
`bax`
`5`
`abc`
`ab`
`cba`
`ba`
`c`
	`Case #1: 4`
	`Case #2: 1`
	`Case #3: 4`
	`Case #4: 6`

Explanation for the 1st sample input
This is the example from the problem statement.

Explanation for the 2nd sample input
(“abac”, “aba”) is the only perfect matching.

Explanation for the 3rd sample input
(“abc”, “ba”), (“ba”, “xab”), (“xab”, “bax”), (“bax”, “xab”) are perfect matching.

Explanation for the 4th sample input
(“abc”, “cba”), (“abc”, “ba”), (“ab”, “cba”), (“ab”, “ba”), (“cba”, “abc”), (“ba”, “ab”) are perfect matching.