Question:

We consider a string consisting of one or more lowercase English alphabetic letters ([a-z]), digits ([0-9]), colons (:), forward slashes (/), and backward slashes (\) to be *adorable* if the following conditions are satisfied:

* The first letter of the string is a lowercase English letter.
* Next, it contains a sequence of *zero or more* of the following characters: lowercase English letters, digits, and colons.
* Next, it contains a forward slash.
* Next, it contains a sequence of *one or more* of the following characters: lowercase English letters and digits.
* Next, it contains a backward slash.
* Next, it contains a sequence of *one or more* lowercase English letters.

Given some string, *s*, we define the following:

* *s[i..j]* is a substring consisting of all the characters in the inclusive range between index *i* and index *j* (i.e., *s[i]*, *s[i + 1]*, *s[i + 2]*, …, *s[j]*).
* Two substrings, *s[i1..j1]* and *s[i2..j2]*, are said to be *distinct* if either *i1 ≠ i2* or *j1 ≠ j2*.

Complete the *adorableCount* function in the editor below. It has one parameter: an array of *n* strings, *words*. The function must return an array of *n* positive integers where the value at each index *i* denotes the total number of *distinct*, *adorable substrings* in *wordsi*.

**Input Format**

Locked stub code in the editor reads the following input from stdin and passes it to the function:

The first line contains an integer, *n*, denoting the number of elements in *words*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains a string describing *wordsi*.

**Constraints**

* *1 ≤ n ≤ 50*
* Each *wordsi* consists of one or more of the following characters: lowercase English alphabetic letters ([a-z]), digits ([0-9]), colons (:), forward slashes (/), and backward slashes (\) only.
* The length of each *wordsi* is no more than *5 × 105*.

**Output Format**

The function must return an array of *n* positive integers where the integer at each index *i* denotes the total number of *distinct*, *adorable* substrings in *wordsi*. This is printed to stdout by locked stub code in the editor.

**Sample Input 0**

6

w\\//a/b

w\\//a\b

w\\/a\b

w:://a\b

w::/a\b

w:/a\bc::/12\xyz

**Sample Output 0**

0

0

0

0

1

8

**Explanation 0**

Let's call our return array *ret*. We fill *ret* as follows:

* *word = "w\\//a/b"* has no adorable substring, so *ret[0] = 0*.
* *word = "w\\//a\b"* has no adorable substring, so *ret[1] = 0*.
* *word = "w\\/a\b"* has no adorable substring, so *ret[2] = 0*.
* *word = "w:://a\b"* has no adorable substring, so *ret[3] = 0*.
* *word = "w::/a\b"* has one adorable substring, *word[0..6] = "w::/a\b"*, so *ret[4] = 1*.
* *word = "w:/a\bc::/12\xyz"* has the following eight adorable substrings:
  1. *word[0..5] = w:/a\b*
  2. *word[0..6] = w:/a\bc*
  3. *word[5..13] = bc::/12\x*
  4. *word[5..14] = bc::/12\xy*
  5. *word[5..15] = bc::/12\xyz*
  6. *word[6..13] = c::/12\x*
  7. *word[6..14] = c::/12\xy*
  8. *word[6..15] = c::/12\xyz*

This means *ret[5] = 8*.

We then return *ret = [0, 0, 0, 0, 1, 8]*.

Solution Details:

* Given question requires possible number of substrings matching the specified criteria. So, I have iterated the string for every index as start to search for possible outcomes.
* As mentioned in the question since length of string can be as long as 5 × 10 5 , to process different strings concurrently I have created thread pool to process them and add respective output at index.
* As string length can be more it was taking time to process larger strings. To reduce the same, I created recursive task which would process string from particular start index to end index only. If string length is greater than threshold it would created more recursive tasks to get the count using fork and join framework.