ACM Summer Challenge 2020

String Manipulation

Editorial

Will of Strings

The question was based on Manacher's algorithm. Manacher's algorithm helps us find the longest palindromic substring with the center of that palindrome as the **i^{th}** character in the string.

Refer the below text to understand how we will find the number of palindromic substrings using this concept.

First of all, we need to modify the string so that we can also detect longest even sized palindromes. For that, you need to insert a label between each character in the string, for example, the string '**abc**' will become '**#a#b#c#**'.

We need to maintain two variables, the **c** (the center) and the **right boundary** - the center, **c**, keeps track of the center of the palindrome encountered and the **right boundary** keeps track of the right most position which is a part of the palindrome whose palindromic center is the position stored in the variable **c**.

We also need to maintain an array of size equal to that of the modified string, which will store the size of the longest palindromic substring whose center is the given position. Let's call this array **manacher**.

Now iterate through all the characters of the modified string.
For any position **i** in this modified string,
Find its mirror index keeping **c** as the axis(For example, the mirror index of '**c**' in '**abc**' if '**b**' is the center will be '**a**'). Let's call this **m_index**.
Now, if **i** is less than the **right boundary**, then this means that position **i** is already a part of a palindrome detected earlier, and hence, the length palindrome formed at center **i** can be equal to length of palindrome formed at the **m_index** or it can be **right boundary - i**. Then,

manacher[i] = min(right boundary - i,manacher[m_index])

Now that we got the initial size of the palindrome formed with center **i**, we can check whether this palindrome expands to its left and right side or not.
This can be done by initializing two variables with **rB**, **lB** as **i + manacher[i]** and **i - manacher[i]**, and increasing **rB** and decreasing **lB** until we dont find **modified_string[lB] = modified_string[rB]**.

Now, comes the part where we shift the **c**. The **c** variable needs to be modified only if **i + rB > right boundary** (why? Because if it is not so, then the palindrome detected in the previous step with center **i** was already a part of the palindrome whose center is **c**). Now if this condition is satisfied, then, modify center , **c** and **right boundary** as :

c = i

right boundary = manacher[i] + i

Do this for all positions in the modified string. You will notice the manacher array consists of the size of longest palindromic substring with center as positions of modified string.
For example , For '**#a#b#b#a#**' the manacher array will have values **(0, 1, 0, 1, 4, 1, 0, 1, 0)**.

Now lets store the required value, that is number of palindromic substrings in variable **Ans**.
Then **loop i from 0 to len(modified_string) - 1**:

Ans = Ans + ceil(manacher[i]/2.0)

Thus, you will have obtained your **Ans**.

This is done in **O(N)**.

If you still didn't understand the editorial, refer this youtube video:
https://www.youtube.com/watch?v=l-XCWjps-UQ

Counting

This problem can be solved using Permutations & Combinations.

You can find number of substrings which contains char **c**(c must be Vowel for this problem) using formula

(n-i)*(i+1) , where **i** is the index of char **c**(Indexing is 0-based).

PROOF:
To find the number of substrings , you have to select any one length for the prefix which we can find by **(n-i)** and for the suffix including char **c**, we can find using formula **(i+1)**. So we can find total number of substrings by **(n-i)*(i+1)**.

This can be done in **O(N)**.

Ignore the Comments

This is a good implementation problem.
You have to collect substrings which should not start or end with **(“//”)**.
You can manipulate using many methods. You can store valid strings as problem statements in a vector. You can also solve the problem using **indexes**.
This can be done in **O(N)**.