<https://leetcode.com/problems/palindromic-substrings/description/>

Given a string s, return *the number of* ***palindromic substrings*** *in it*.

A string is a **palindrome** when it reads the same backward as forward.

A **substring** is a contiguous sequence of characters within the string.

**Example 1:**

Input: s = "abc"

Output: 3

Explanation: Three palindromic strings: "a", "b", "c".

**Example 2:**

Input: s = "aaa"

Output: 6

Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".

**Constraints:**

* 1 <= s.length <= 1000
* s consists of lowercase English letters.

**Attempt 1: 2023-06-27**

**Solution 1:  Brute Force (10 min)**

**Style 1: i is start index, j is (real end index but plus 1) to get substring need s.substring(i, j)**

class Solution {

public int countSubstrings(String s) {

int len = s.length();

int count = 0;

for(int i = 0; i < len; i++) {

for(int j = i + 1; j <= len; j++) {

if(isPalindrome(s, i, j - 1)) {

count++;

}

}

}

return count;

}

private boolean isPalindrome(String s, int i, int j) {

while(i < j) {

if(s.charAt(i) != s.charAt(j)) {

return false;

}

i++;

j--;

}

return true;

}

}

Time Complexity : O(N^3), Here three nested loop creates the time complexity. Where N is the size of the string(s).

Space Complexity : O(1), Constant space.

Solved using string(Three Nested Loop). Brute Force Approach.

**Style 2: i is start index, j is (real end index but plus 1) to get substring need s.substring(i, j)**

class Solution {

public int countSubstrings(String s) {

int len = s.length();

int count = 0;

for(int i = 0; i < len; i++) {

for(int j = i; j < len; j++) {

if(isPalindrome(s, i, j)) {

count++;

}

}

}

return count;

}

private boolean isPalindrome(String s, int i, int j) {

while(i < j) {

if(s.charAt(i) != s.charAt(j)) {

return false;

}

i++;

j--;

}

return true;

}

}

Time Complexity : O(N^3), Here three nested loop creates the time complexity. Where N is the size of the string(s).

Space Complexity : O(1), Constant space.

Solved using string(Three Nested Loop). Brute Force Approach.

**Refer to**

<https://leetcode.com/problems/palindromic-substrings/solutions/105707/java-python-dp-solution-based-on-longest-palindromic-substring/comments/271651>

**When j - i < 3, we don't need to know the result of dp[i + 1][j - 1]** Here is my analysis:

j - i == 0: we are checking a substring of a single character. It is obvious that such substring must be palindromic

j - i == 1: we are checking a substring of two characters. If we can get to there, then we must pass the check s.charAt(i) == s.charAt(j), which means the first character of this string is identical to the second character. The string must be palindromic as well

j - i == 2, we are checking a substring of three characters. If we can get to there, then we must pass the check s.charAt(i) == s.charAt(j), which means the first character of this string is identical to the second character. A string of length 3, with its leftmost character equal to its rightmost character, no matter what its middle character is, must be palindromic as well

**Solution 2:  Spread From Center (10 min)**

**Style 1: Use global variable and initialize as 1**

class Solution {

int count = 1;

public int countSubstrings(String s) {

int len = s.length();

for(int i = 0; i < len - 1; i++) {

isPalindrome(s, i, i);

isPalindrome(s, i, i + 1);

}

return count;

}

private void isPalindrome(String s, int i, int j) {

while(i >= 0 && j < s.length() && s.charAt(i) == s.charAt(j)) {

count++;

i--;

j++;

}

}

}

**Refer to**

<https://leetcode.com/problems/palindromic-substrings/solutions/105688/very-simple-java-solution-with-detail-explanation/>

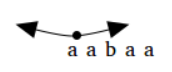
A very easy explanation with an example

Lets take a string "aabaa"

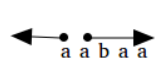
**Step 1:** Start a for loop to point at every single character from where we will trace the palindrome string.checkPalindrome(s,i,i); //To check the palindrome of odd length palindromic sub-stringcheckPalindrome(s,i,i+1); //To check the palindrome of even length palindromic sub-string

**Step 2:** From each character of the string, we will keep checking if the sub-string is a palindrome and increment the palindrome count. To check the palindrome, keep checking the left and right of the character if it is same or not.

**First Loop:**



Palindrome: a (Count=1)

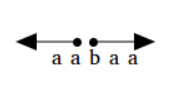


Palindrome: aa (Count=2)

**Second Loop:**

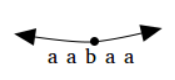


Palindrome: a (Count=3)

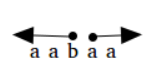


Palindrome: No Palindrome

**Third Loop:**

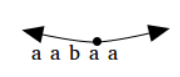


Palindrome: b, aba, aabaa (Count=6)

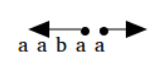


Palindrome: No Palindrome

**Forth Loop:**



Palindrome: a (Count=7)



Palindrome: aa (Count=8)

Count = 9 (For the last character)

**Answer = 9**

int count =1;

public int countSubstrings(String s) {

if(s.length()==0)

return 0;

for(int i=0; i<s.length()-1; i++){

checkPalindrome(s,i,i); //To check the palindrome of odd length palindromic sub-string

checkPalindrome(s,i,i+1); //To check the palindrome of even length palindromic sub-string

}

return count;

}

private void checkPalindrome(String s, int i, int j) {

while(i>=0 && j<s.length() && s.charAt(i)==s.charAt(j)){ //Check for the palindrome string

count++; //Increment the count if palindromin substring found

i--; //To trace string in left direction

j++; //To trace string in right direction

}

}

**Style 2: Use global variable and initialize as 0 (The last character can be handled by the for loop, better to set count = 0 and for(int i = 0; i < s.length(); i++), it works!)**

class Solution {

int count = 0;

public int countSubstrings(String s) {

int len = s.length();

for(int i = 0; i < len; i++) {

isPalindrome(s, i, i);

isPalindrome(s, i, i + 1);

}

return count;

}

private void isPalindrome(String s, int i, int j) {

while(i >= 0 && j < s.length() && s.charAt(i) == s.charAt(j)) {

count++;

i--;

j++;

}

}

}

**Style 3: Use local variable (Multi-threading implementations are difficult when global variables are used)**

class Solution {

int count = 0;

public int countSubstrings(String s) {

int len = s.length();

for(int i = 0; i < len; i++) {

count += isPalindrome(s, i, i);

count += isPalindrome(s, i, i + 1);

}

return count;

}

private int isPalindrome(String s, int i, int j) {

int count = 0;

while(i >= 0 && j < s.length() && s.charAt(i) == s.charAt(j)) {

count++;

i--;

j++;

}

return count;

}

}

**Refer to**

<https://leetcode.com/problems/palindromic-substrings/solutions/105688/very-simple-java-solution-with-detail-explanation/comments/163295>

It uses a global variable which can be easily eliminated:

public int countSubstrings(String str) {

if(str == null || str.length() < 1) return 0;

int count = 0;

for(int i=0;i<str.length();i++){

count += countPalindromes(str, i, i); //Count even sized

count += countPalindromes(str, i, i+1); //Count odd sized

}

return count;

}

private int countPalindromes(String str, int s, int e){

int count = 0;

while(s>=0 && e<str.length() && str.charAt(s) == str.charAt(e)){

s--;

e++;

count++;

}

return count;

}

**Solution 3: DP (10 min)**

class Solution {

public int countSubstrings(String s) {

int len = s.length();

boolean[][] dp = new boolean[len][len];

int count = 0;

for(int i = len - 1; i >= 0; i--) {

for(int j = i; j < len; j++) {

// 1.j - i == 0, only a character is a palindrome,

// 2.j - i == 1 and s.charAt(i) == s.charAt(j), ij is a palindrome,

// 3.j - i == 2 and s.charAt(i) == s.charAt(j), no matter what between i and j, i#j is a palindrome

// 4.and if j - i > 2, then the internal string between i and j must be palindrome

if(j - i <= 2) {

dp[i][j] = (s.charAt(i) == s.charAt(j));

} else {

dp[i][j] = (s.charAt(i) == s.charAt(j) && dp[i + 1][j - 1]);

}

if(dp[i][j]) {

count++;

}

}

}

return count;

}

}