CLASS-15 - 25/09/2023
https://www.linkedin.com/in/manojoffcialmj/

CHAR ARRAYS & STRINGS LEVEL-2

Program 01: Remove All Adjacent Duplicates In String (Leetcode-1047)

Input | a | b | b | a | c | a | output: ca

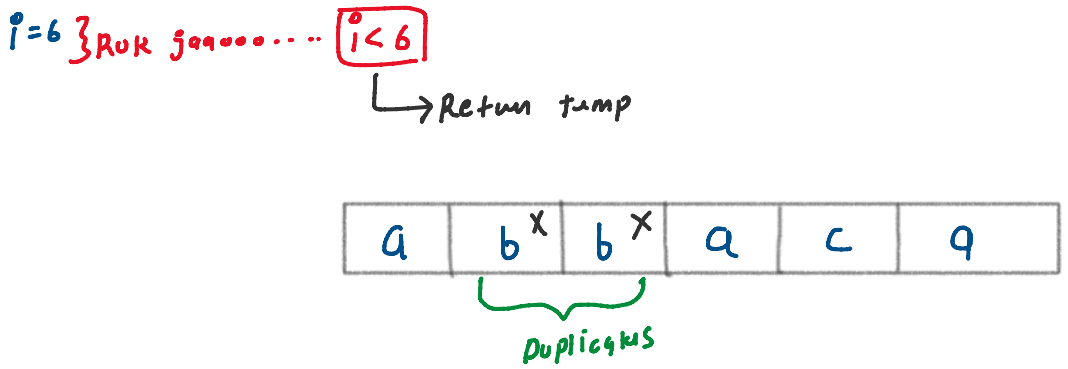
Approach:
Step 01: create a temp string
Step 02: compare input string's element with right most element of temp string
- When both elements are different then push element in temp
- When both elements are same then pop element from temp

DRY RUN

s	a	b	b	a	c	a
	0	1	2	3	4	5

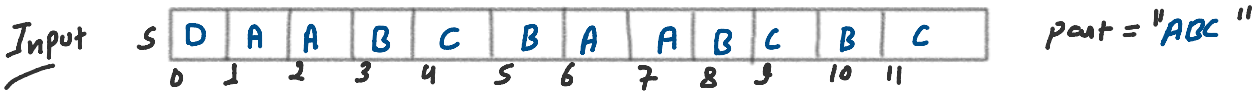
Step 01: create a temp string

i = 0	temp = ""	s[0] = a	temp: [a] (Right most Element)	push a
i = 1	temp = "a"	s[1] = b	temp: [a][b] (RME)	push b
i = 2	temp = "ab"	s[2] = b	temp: [a] (RME)	pop b
i = 3	temp = "a"	s[3] = a	temp: [EMPTY] (RME)	pop a
i = 4	temp = ""	s[4] = c	temp: [c] (RME)	push c
i = 5	temp = "c"	s[5] = a	temp: [c][a] (RME)	push a
i = 6	RUK jaaaoo....	i < 6		




```cpp
// Program 01: Remove All Adjacent Duplicates In String (Leetcode-1047)
class Solution {
public:
    string removeDuplicates(string s) {
        // Step 01: create a temp string
        string temp = "";
        int n = s.length();
        int i = 0;

        while(i<n){
            // Step 02: compare input string's element with right most element of temp string
            // When both elements are same then pop element from temp
            if(temp.length()>0 && s[i]==temp[temp.length()-1]){
                temp.pop_back();
            }
            // When both elements are different then push element in temp
            else{
                temp.push_back(s[i]);
            }
            i++;
        }
        return temp;
    }
};

/*
Time Complexity: 0(N), where N is length of s
Space Complexity: 0(N), where N is length of temp
*/

/*
Example 1:
Input: s = "abbaca"
Output: "ca"

Example 2:
Input: s = "azxxzy"
Output: "ay"
*/
```

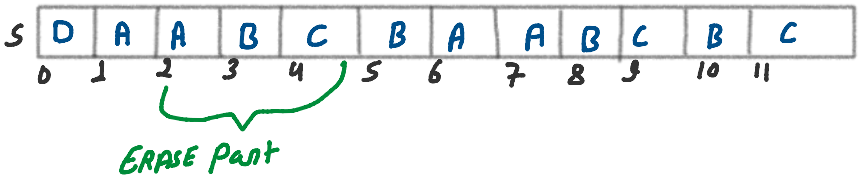
Program 02: Remove All Occurrences of a Substring (Leetcode-1910)



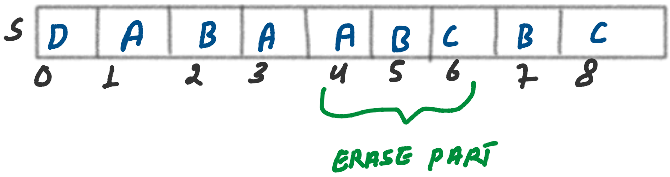
```
Approach:
Step 01: find part from s and remove part till end of last iteration of s
Step 02: return s
```

DRY RUN

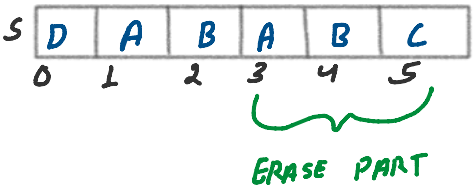
Iteration:01



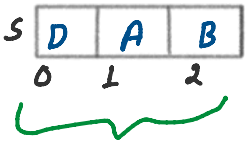
Iteration:02



Iteration:03



Iteration:04



Final output

```cpp
// Program 02: Remove All Occurrences of a Substring (Leetcode-1910)
class Solution {
public:
    string removeOccurrences(string s, string part) {
        // Step 01: Find part from s when part found then erase it from s
        while(s.find(part) != string::npos){
            // if inside loop, it means that part exists in s string
            s.erase(s.find(part),part.length());
        }
        // Step 02: return s
        return s;
    }
};

/*
Time Complexity: 0(N), where N is length of s
Space Complexity: 0(1), no extra space used
*/

/*
Example 1:
Input: s = "daabcbaabcbc", part = "abc"
Output: "dab"

Example 2:
Input: s = "axxxxyyyyb", part = "xy"
Output: "ab"
*/
```

Time complexity of find() function and erase() function is O(N)

Program 03: Valid Palindrome II (Leetcode-680)

EX:01

EX:01

Input s | A | B | A |

output: true

```
Approach: Two Pointer
Step 01: when s[start] and s[end] are equals then start++ and end--
Step 02: when s[start] and s[end] are different then remove at most 1 character
         s[start] or s[end] then check palindrome for remaining string after removal
Step 03: then return true or false
```

DRY RUN
EX:01

Itr:1 s | A | B | A |
0 1 2

CASE: 0 Removal

start = 0
end = 2

s[start] == s[end]
A == A

s++
e--

Itr:2 s | B |
1

start = 1
end = 1

s[start] == s[end]
B == B

s++
e--

start = 2
end = 0

Ruk jaun...
(start < = end)
End

Return true

DRY RUN
EX:2

Itr:01 S | A | B | C | A |
0 1 2 3

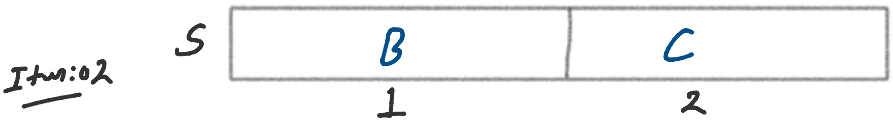
output: true

CASE: 1 Removal

start: 0
end = 3

s[start] == s[end]
A == A

start ++
end --



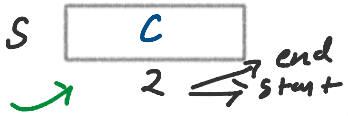
Item:02

output: true

start: 1
end = 2

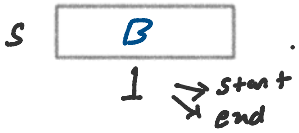
S[start] != S[end]
B != C

case: 1
when I remove B character then Remaining string



This is palindrome then Return true.

case: 2
when I remove C character then Remaining string

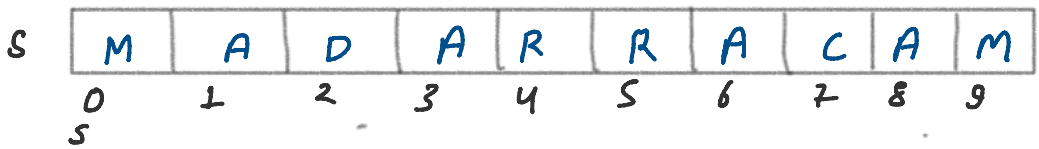


this is palindrome then return true

(return overall ans is true)
return TRUE || TRUE

Ans = True

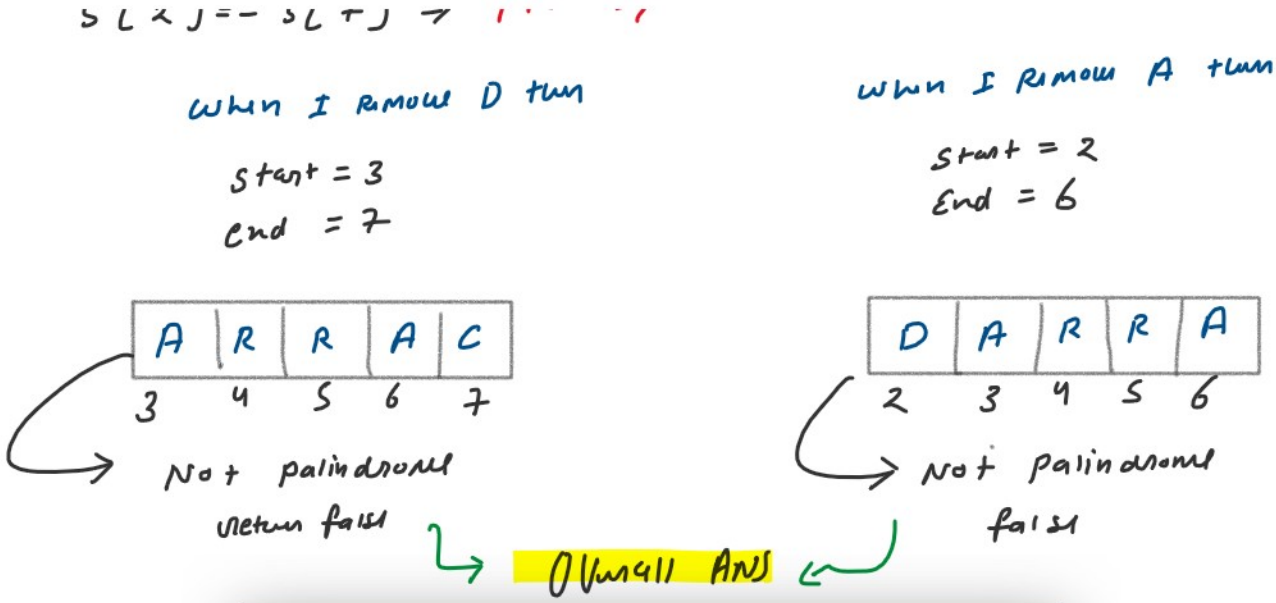
DRY RUN
EX:03



S[0] == S[9] ⇒ true
S[1] == S[8] ⇒ true
S[2] == S[7] ⇒ false ⇒

when I Remove D then

when I Remove A then




```cpp
// Program 03: Valid Palindrome II (Leetcode-680)
class Solution {
public:
    // Palindrome Function
    bool checkPalindrome(string s, int start, int end){
        while(start<=end){
            if(s[start]==s[end]){
                start++;
                end--;
            }
            else{
                return false;
            }
        }
        return true;
    }

    // Valid Palindrome II
    bool validPalindrome(string s) {
        int N = s.length();
        int start = 0;
        int end = N-1;

        while(start<end){
            // Step 01
            if(s[start]==s[end]){
                start++;
                end--;
            }
            // Step 02:
            else{
                // Only one removal allowed
                // check palindrome for remaining string after removal

                //s[start] character --> remove
                bool ans1 = checkPalindrome(s, start+1, end);
                //s[end] character --> remove
                bool ans2 = checkPalindrome(s, start, end-1);

                // Step 03
                return ans1 || ans2;
            }
        }
        // Agar yanha tak pahunche ho to iska mtlb valid palindrome hai
        // iska mtlb --> zero removal hai
        return true;
    }
};

/*
Time Complexity: 0(N), where N is length of s
Space Complexity: 0(1), no extra space use
*/

/*
Example 1:
Input: s = "aba"
Output: true

Example 2:
Input: s = "abca"
Output: true

Example 3:
Input: s = "abc"
Output: false
*/
```

Program 04: Palindromic Substrings (Leetcode-647) (Most Important)

EX:01

Input S | A | B | A |

output : 4

Total substring ⇒ A ✓, AB ✗, ABA ✓, BA ✗, A ✓, B ✓

Total palindrome substrings = 4

Hum yanha par yah absorb kar paa rha hun ki kuch substring ki length odd and kuch ki length even hai

Odd → A, ABA, A, B

Even → AB, BA

Approach: Two Pointer
Step 01: check odd length's substring, how many substrings are palindrome
Step 02: check even length's substring, how many substrings are palindrome
Step 03: then return total number of substrings are palindrome (Odd + Even)

DRY RUN

EX:01

Input S | A | B | A | (0, 1, 2)

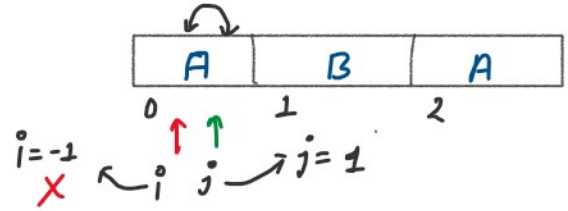
output : 4

STEP:01

ODD Smallest odd = 1

Count = 0

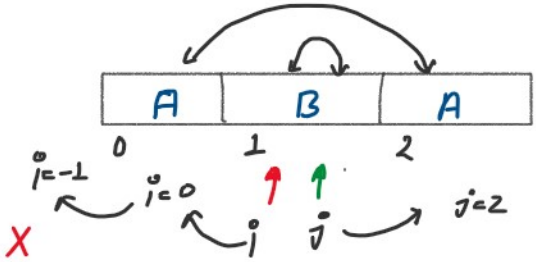
A



⇒ Count = 0 + 1

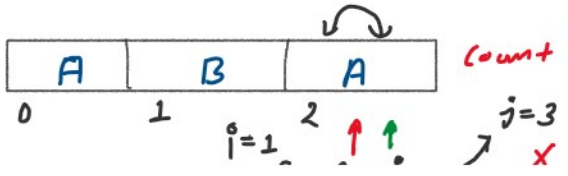
B

ABA



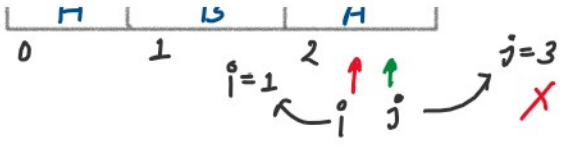
⇒ Count = 0 + 1 + 2

A



Count = 0 + 1 + 2 + 1

ODD Count = 4



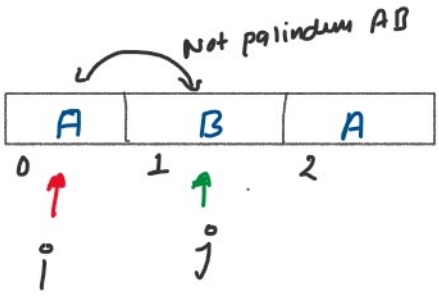
ODD
Count = 4

STEP:02

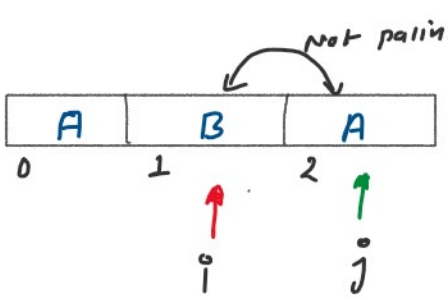
Even

Smallest Even = 2

Count = 0



⇒ Count = 0+0



⇒ Count = 0+0+0

Even
Count = 0

Step:03

TotalCount = odd + Even ⇒ 4+0 = 4

Final output

EXAMPLE:02

S | A | B | C

output: 3

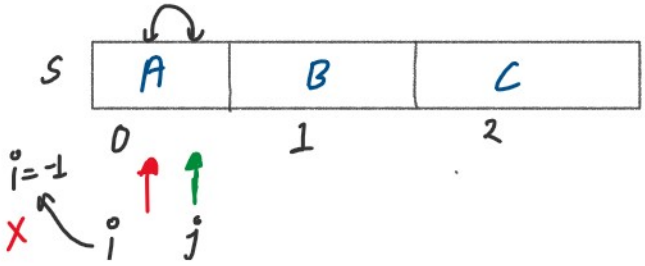
DRY RUN

Step:01

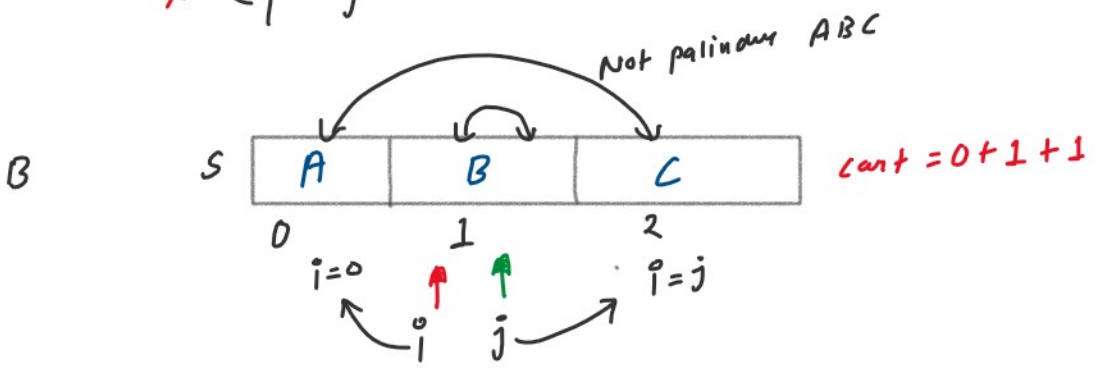
ODD

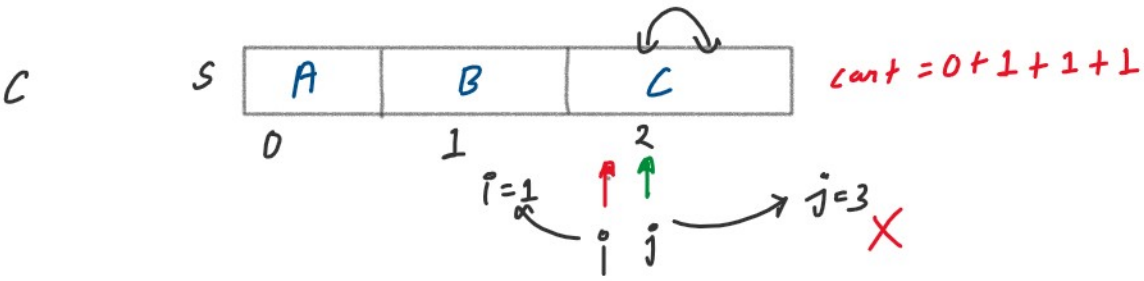
Count = 0

A



cant = 0+1

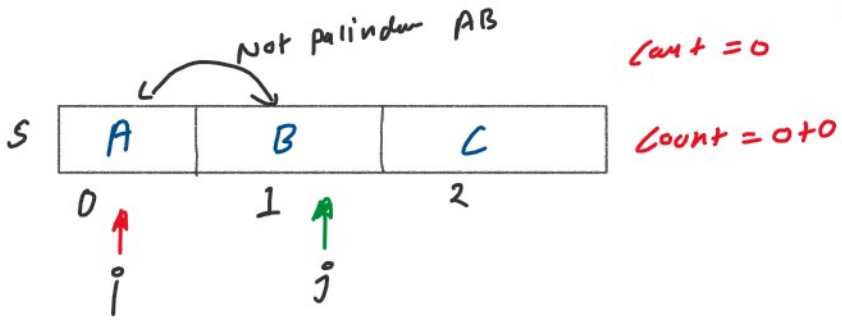


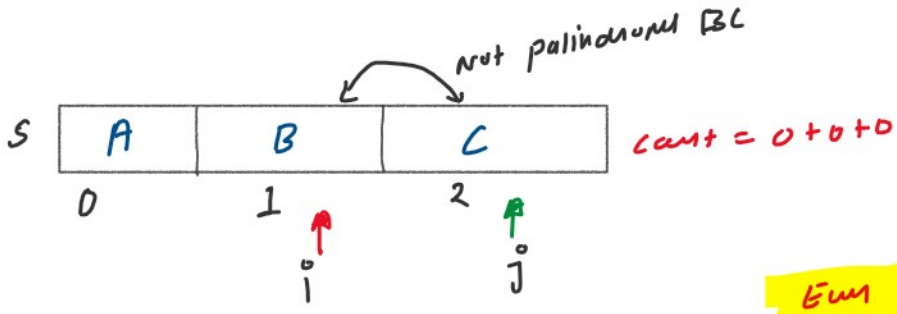


ODD
Count = 3

Step:02

Even





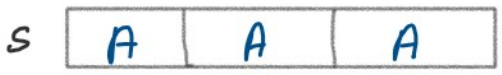
Even
Count = 0

Step:03

Total Count = odd + Even
= 3 + 0
= 3 Final output

Example:03

Example: 03

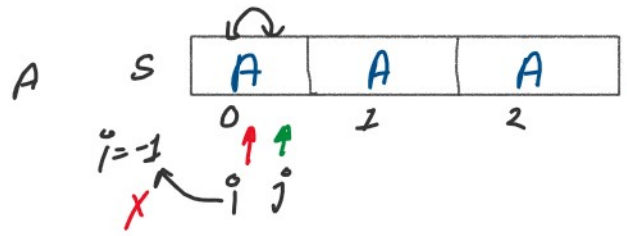


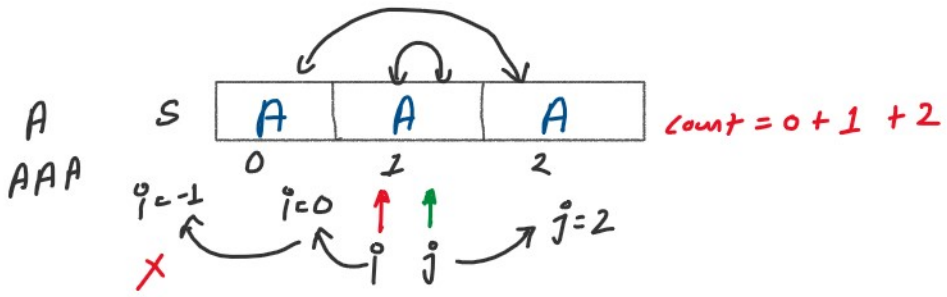
Output : 6

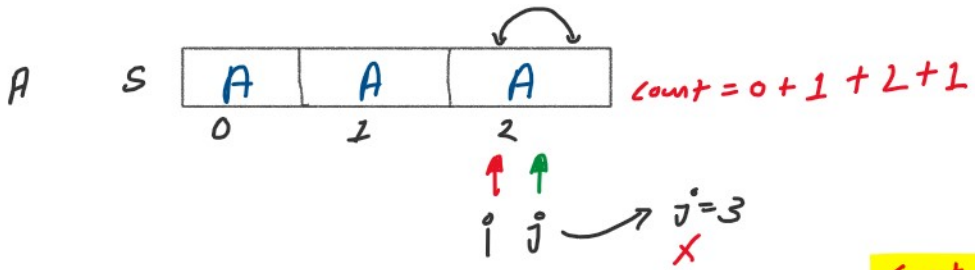
STEP: 01

Odd

count = 0





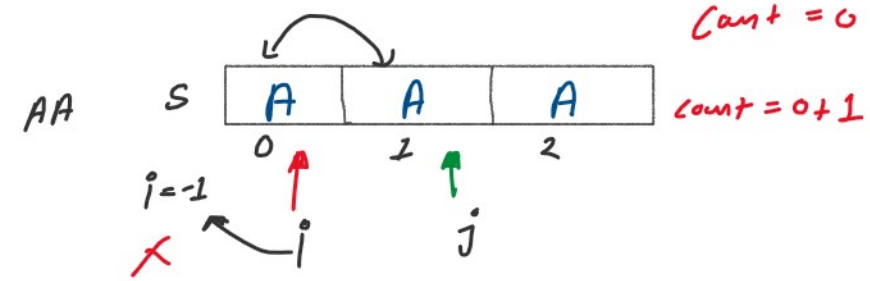


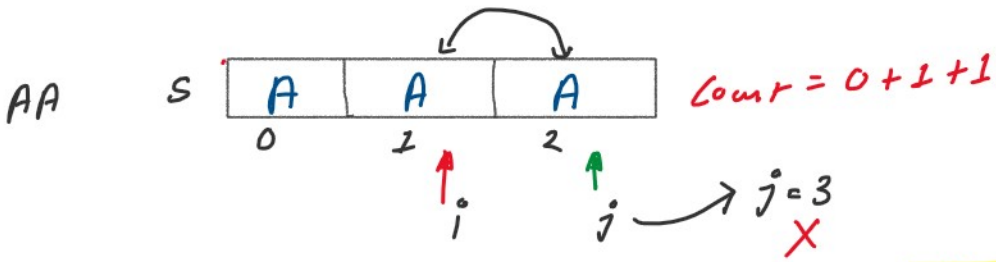
Count Odd = 4

Step: 02

Even

Count = 0





Count Even = 2

Step:03

Total Count = 4 + 2

= 6 Final output

```cpp
// Program 04: Palindromic Substrings (Leetcode-647)
class Solution {
public:
    int expand(string s, int i, int j){
        int count = 0;

        // Jab tak substring palindrome ho tab tak me count increase karunga
        // other wise loop  rok doonga
        while(i>=0 && j<s.length() && s[i]==s[j]){
            count++;
            i--;
            j++;
        }
        return count;
    }
    int countSubstrings(string s) {

        int totalCount = 0;
        int center = 0;

        while(center<s.length()){

            // Step 01: ODD
            int oddCount = expand(s,center,center);

            // Step 02: EVEN
            int evenCount = expand(s,center,center+1);

            // Step 03: Total Count
            totalCount = totalCount + oddCount + evenCount;

            // Increase center
            center++;
        }
        // Return totalCount
        return totalCount;
    }
};

/*
Time Complexity: 0(N), where N is length of s
Space Complexity: 0(1), no extra space use
*/

/*
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".
*/
```