→ Ques 1047. (leetcode) 26-02-2023

Date..................

Ques Remove all adjacent duplicates in a string.

In this we've given a string consist of lowercase english letters. A duplicate consists of choosing 2 adjacent & eaual letters & removing them. Example :-

Input : " abbaca "
output : " ca "

Explanation:- In "abbaca", we could remove "bb" as they are duplicates as well as adjacent, and this is the only possible move. The result of this after removing "bb" is that the string is "aaca" of which "aa" is duplicate & adjacent. So, the final string is "ca".

↙ this is the resulting string.

Approach: The approach is that, we create an empty string to store the final string. Then we follow two pointer approach, i.e., i = 0, j = s.length() - 1. while (i <= j), we check if last character of "ans" string is matched with current character of "s" string, it means they are duplicate & adjacent. So,

Teacher's Sign ....

we remove the character from ans string. If they are not equal then we push_back the current character of "s" string into the "ans" string, & after every condition i will incremented by one.

code:-

```
#include<iostream>
using namespace std;
string removeDuplicate(strings)
{
    string ans = " ";
    int i = 0;
    int j = s.length();
    while (i <= j)
    {
        if(ans.length > 0 && ans[
            ans.length() - 1] == s[i])
        {
            ans.pop_back();
        }
        else
        {
            ans.push_back(s[i]);
        }
        i++;
    }
    return ans;
}
```

```
int main()
{
    string s = "abbacyycabbafu";
    string ans = removeDuplicate(s);
    cout << "Final string : " << ans;
    return 0;
}
```

→ Ques 1910 (leetcode)

Date.................

Ques Remove all occurrence of a substring.

we've given two strings, i.e., "str" and "part". Perform the operations on "str" until all occurrences of the substring "part" are removed.

Example:-

str Input:- "daabcbaabcbc"

part Input:- "abc"

Output:- "dab"

Explanation:-

str = "daabcbaabcbc", remove "abc" starting at index 2. So now str = "dabaabcbc", remove "abc" starting at position 4. So now, str = "dababc", remove "abc" starting at position 3. So now, str = "dab" & str has no occurrence of "abc"

↳ this is the resulting string.

Approach: The approach is that, first we find the index where "part" is starting in "str" from find() function. Then, we erase the part from that position using erase() method.

code :-

```
#include <iostream>
using namespace std;

string removeOccurrence (string
                    str, string part)
{
    int pos = str.find(part);
    while (pos != string::npos)
    {
        str.erase (pos, part.length
                              ());
        pos = str.find(part);
    }
    return str;
}

int main()
{
    string str = "daabcbaabcbc";
    string part = "abc";
    string ans = removeOccurrence
                  (str, part);
    cout << "Final string is:" <<
                  ans << endl;
    return 0;
}
```

→ Ques 680 (leetcode)

Date..................

Ques: valid palindrome II.
we've given a string "str".
If str can be a valid palindrome after deleting atmost one character from it, then return true else return false.

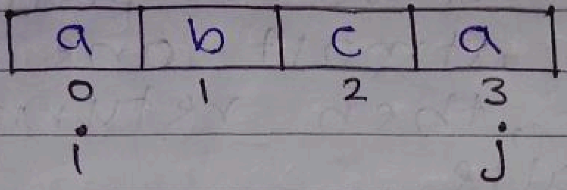
Example:-
Input:- "aba"
Output:- true. ↘ after removing "b", the string can be a valid palindrome.

Input:- "abca"
Output:- true. ↘ after removing "c" or "b", the string can be a valid palindrome.

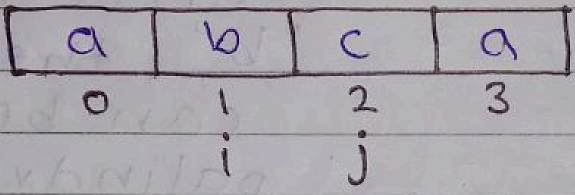
Input:- "abc"
Output:- false.

Approach:- Here we follow the same palindrome two pointer approach. If str[i] == str[j], we normally increment i and decrement j. If str[i] & str[j] is not equal, means we have to remove either str[i] or str[j] but we don't know that which one to remove. So, once we remove str[i] & check the string for palindrome, then remove str[j] & check the string for

palindrome.
For example:-



Firstly, str[i] = str[j]. So we, increment i & decrement j; i.e., i++ & j--.



Now, str[i] != str[j]; it means we have to remove one of them, but we don't know which one to remove makes the string palindrome. So, once we remove str[i] & check the the rest of the string, i.e., i+1 to j Then once we remove str[j] & check the rest of the string, i.e., i to j-1 If any one from these makes a valid palindrome then we return True else we return False.

code:-

```
#include<iostream>
using namespace std;
bool checkPalindrome(string s,
                     int i, int j)
{
    while(i<=j){
        if(s[i] != s[j]){
            return false;
        }
         i++;
         j--;
    }
    return true;
}
bool validPalindrome(string s)
{
    int i = 0;
    int j = s.length() - 1;
    while(i<=j){
        if(s[i] != s[j]){
            return checkPalindrome(s,
                i+1, j) || checkPalindrome
                              (s, i, j-1);
        }
        else{
            i++;
            j--;
        }
    }
    return true;
}
```

```
int main()
{
    string str = "acda";
    bool ans = validPalindrome(str);
    if(ans) {
        cout << "valid";
    }
    else {
        cout << "Invalid";
    }
    return 0;
}
```

** very important & Ques539 (leetcode)
→ tricky Question. Date...............

Ques Minimum Time Difference.
: we've given a list of 24 Hour clock time points in "HH:MM" format, we have to return the minimum minutes difference between any two time points in the list.

Example:-
Input:- time poinst given,

["23:59", "00:00"]

output:- 1

Example:-
Input:- time points given,

["23:59", "00:00", "00:00"]

output:- 0

The first noticeable thing is that, we want the difference in minutes. so, first we convert our given time string into minutes integer values.
After converting the string we store it in new integer vector,

```
vector<int> minutes;
```

Loop through each element, as each element contains "23:59", first number is Hours & second number is minutes. "23:59" is a complete string, so we retrieve 2 substrings, one for Hours & one for minutes. Then calculate the total minutes from them.

```
for(int i = 0; i < timePoints.size();
                                  i++)
{
    string curr = timePoints[i];
    int hours = stoi(curr.substr
                          (0,2));
    int min = stoi(curr.substr
                        (3,2));
    int totalMinutes = hours * 60 +
                          min;
    minutes.push_back(totalMin
                         -utes);
}
```

taking ith element from the given string.

fetching substring, last 2 characters from 3 index are for minutes.

In end, push_back calculated minutes in our "minutes" vector

fetching substring, first 2 character from 0 index are for Hours

calculate total minutes, i.e., hours x 60 + minutes

Now, we want to find the minimum difference between any two values. So, we can use sorting, because sorting gives the minimum to maximum values, then we can easily compare first values with next values, sort our minutes vector which contains all the hours in the minutes,

```
sort(minutes.begin(),
                  minutes.end());
```

After sorting, then compare each adjacent elements & find out the differences of all the elements & after that find out the minimum difference in all that differences,

```
int mini = INT_MAX;
int n = minutes.size();
for(int i = 0; i < n-1; i++)
{
     int diff = minutes[i+1] -
                       minutes[i];
   mini = min(mini, diff);
}
```

calculating the difference between i+1 element & ith element.

then finding the mininum of all the elements.

Now, we have find out the minimum of all the differences but we have to find out the difference between the first element & the last element, also because of circular nature of clock.

For example :-

00:00	23:59

Difference between 00:00 to 23:59 is 1439 minutes.

But, difference between 23:59 to 00:00 is 1 minute. So, we have to return 1 in this case.

So, we add 24 hours in our first element then subtract the last element from that.

Means,

```
int lastDiff = (minutes[0] + 1440) -
                         minutes[n-1];
```

Then we compare this with our minimum value again,

```
mini = min(mini, lastDiff);
```

code:-

```
int findMinDifference(vector<string>
                          &timePoints){
   //convert time strings into
     minutes integer value.
   vector<int> minutes;
   for(int i=0; i< timePoints.size();
                                  i++)
   {
       string curr = timePoints[i];
       int hours = stoi(curr.substr
                              (0, 2));
       int min = stoi(curr.substr
                            (3, 2));
       int totalMinutes = hours * 60 +
                            min;
       minutes.push_back(totalMinu
                              -tes);
   }
  //sort minutes vector.
  sort(minutes.begin(), minutes.
                             end());
  //difference find & calculate
    the minimum difference
  int mini = INT_MAX;
  int n = minutes.size();
  for(int i=0; i<n-1; i++){
      int diff = minutes[i+1] -
                       minutes[i];
      mini = min(mini, diff);
  }
//tricky part - THIS IS THE GAME
int lastDiff = (minutes[0] + 1440)
                 - minutes[n-1];
mini = min(mini, lastDiff);

    return mini;
}
```

another approach for this part,

```
int lastDiff1 = (minutes[0] +
                      1440) -
                        minutes[n-1];
int lastDiffe2 = minutes[n-1] -
                   minutes[0];
int lastDiff = min(lastDiff1,
                   lastDiffe2);
mini = min(mini, lastDiff);
return mini;
```

→ Ques 647(leetcode)

Date..................

Ques. count the no. of palindromic substring. (V.V.Imp).

we have given a string "str" & we have to return the no. of palindromic substring in the given string.

Example:-

Input :- "abc"

output :- 3

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

input :- "aaa"

ouput :- 6

Explanation :- six palindromic strings are :- "a", "a", "a", "aa", "aa", "aaa".

input :- "noon"

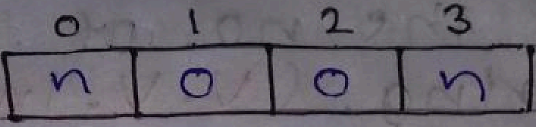
output :- 6

Explanation :- six palindromic string are :- "n", "o", "o", "n", "oo", "noon".

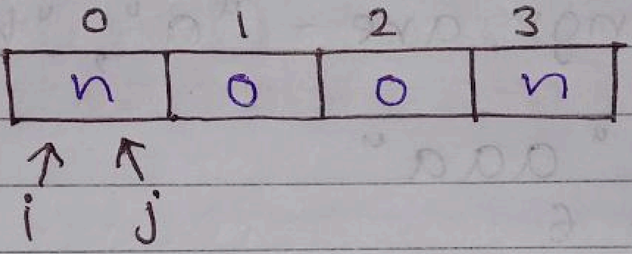
Here we can observe that either the substring is of even length or is of odd length.

So, we can separately calculate both the strings, first we calculate even length, then we calculate odd length.

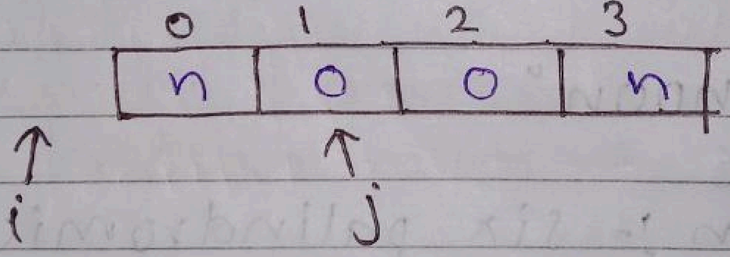
for example:-



First if we want to find the odd strings then we place our both the pointers at same index, i.e., i = 0 & j = 0.
we will stop if either i or j will go out of bound, & arr[i] != arr[j], & shift to next iteration.

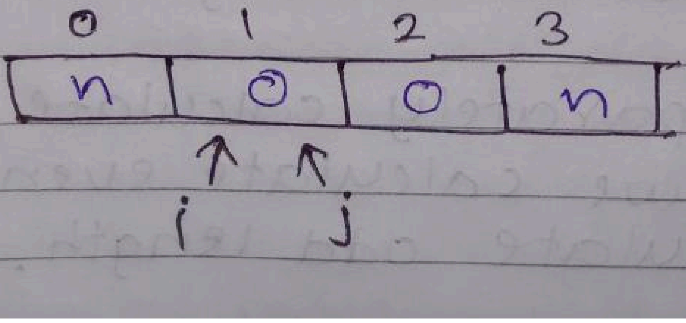


if arr[i] == arr[j], then decrement i & increment j, i.e., i-- & j++, & along with that increment "count".

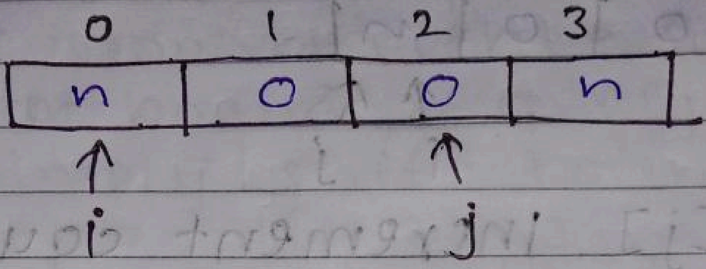


'i' is out of bound. Means we have to stop.

Now, set i & j both to 1 index.

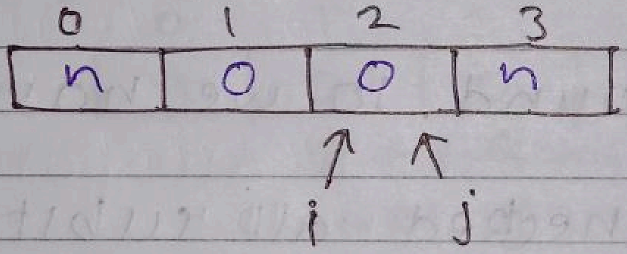


arr[i] == arr[j]. Increase count value , decrement i & increment j.

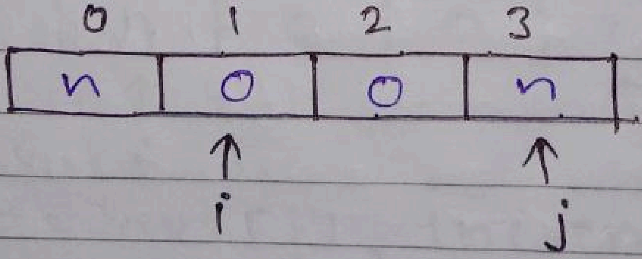


Now, we check arr[i] != arr[j]. So we stop here.

set i & j both to 2 index,

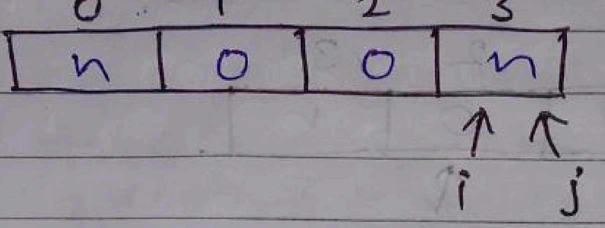


arr[i] == arr[j]. Increment count, decrement i & increment j.

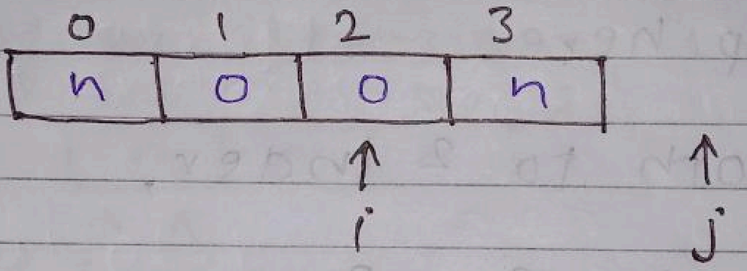


arr[i] != arr[j]. So we stop here.

set i & j both to 3 index.



arr[i] == arr[j]. Increment count, decrement i & increment j.



'j' is out of bound. so we have to stop.

Now, we have checked all substrings of odd length.

$\boxed{\text{count : 4}}$

Now, we find the even length strings so we place our both the pointers at adjacent places, means i = 0, j = 1. we will stop if either i or j will go out of bound & arr[i] != arr[j] & we simply shift to next indexes.

0	1	2	3
n	o	o	n
↑	↑		
i	j		

arr[i] != arr[j]. So we stop here & move to next indexes. Means, i = 1 & j = 2.

0	1	2	3
n	o	o	n
	↑	↑	
	i	j	

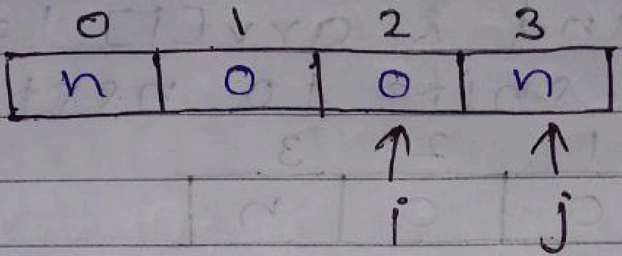
arr[i] == arr[j], Increment count, decrement i & increment j.

0	1	2	3
n	o	o	n
↑			↑
i			j

arr[i] == arr[j], Increment count, decrement i & increment j

	0	1	2	3	
	n	o	o	n	
↑					↑
i					j

'i' & 'j' both are out of bound. so we have to stop, & move to next indexes. Means i=2 & j=3.



arr[i] != arr[j]. so stop here & move to next indexes. Means, i = 3 & j=4.

"j" is out of bound. so we stop the loop.

Now, we have checked all possible substrings of even lengths.

$\boxed{\text{count} = 6}$

After checking all the subitrings, our count contains $\boxed{6}$.
Means, total no. of palindromic substrings are 6.

code:-

```
#include<iostream>
using namespace std;

int expandIndexes(string s, int i,
                             int j)
{
    int count = 0;
    while (i>= 0 && j < s.size() &&
                   s[i] == s[j])
    {
        count++;
        i--;
        j++;
    }
    return count;
}

int countSubstrings (string s)
{
    int count = 0;
    int n = s.size();
    for(int i = 0; i < n; i++)
    {
        int oddAns = expandIndexes(s,
                                  i, i);
        count += oddAns;
        int evenAns = expandIndexes(s,
                                  i, i+1);
        count += evenAns;
    }
    return count;
}
```

```
int main()
{
   string s = "noon";
   int count = countSubstrings(s);
   cout << "total palindromic
      substring in"<< s <<"is:"<<
                           count;
   return 0;
}
```