Char Array & Strings - 2

Q1 → Remove all adjacent duplicates in string → 1047 leetcode

i/p → a b b a c a

⇒ a a c a

⇒ c a c out of string

return the ans.

a z x x z y

⇒ a z z y

⇒ a y // ans

string | a | b | b | a | c | a |

ans = " " → empty string

can we push 'a' in the ans string? → Yes, because the last character is not 'a'.

ans ⇒ "a"

can we push 'b'? → Yes

ans ⇒ "ab"

can we push 'b' → NO, because the last charcter we pushed is 'b'. And we have to remove that also.

ans → "a"

ans → " " → because we can't push a and we have to remove a from ans string too.

ans → "c"

ans → "ca" no element left, this is the ans.

i						
a	z	x	x	z	y	

ans = " "

= "a"

= "az"

⇒ "azx"

⇒ "az"

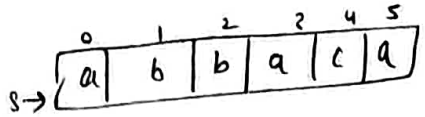
⇒ "a"

⇒ "ay"

- code -

```
string removeDuplicates (string s){
    string ans = "";
    int i = 0;
    while ( i < s.length()){
        if (ans.length() > 0){
            if(ans[ans.length()-1] == s[i]){
                ans.pop_back();
            }
            else{
                ans.push_back (s[i]);
            }
        else{
            ans.push_back (s[i]);
        }
    } return ans;
}
```

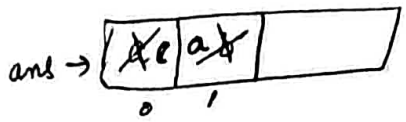
dry runs



s.length = 6

i = 0

0 < 6 → T



if(ans.length())
0 > 0 → F

↳ ans.push_back(s[i]) a

i = 1 ans.length = 1

1 < 6 → T

s[i] == ans[1-1] ans[1-1] → F

↳ insert b.

i = 2

2 > 6 → T

s[i] == ans[2-1] → T

↳ pop_back(). → b is removed

i = 3

3 < 6 → T

s[i] == ans[0] → T

↳ pop_back(); → a is removed

i = 4

4 < 6 → T

ans.length() > 0 → F

push_back(s[i]) → c is inserted

i = 5

5 < 6 → T

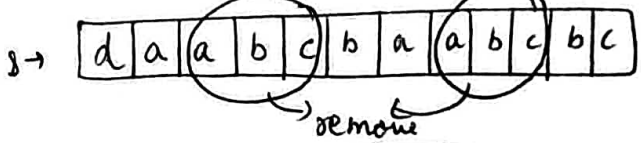
s[i] == ans[0] → F

↳ push_back[s[i]) → a is inserted.

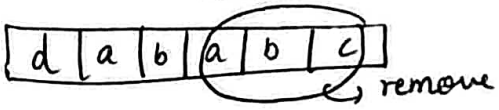
i = 6, 6 < 6 → F return ans.

Q2 → Remove all occurrences of a substring:-

(910)



part = abc

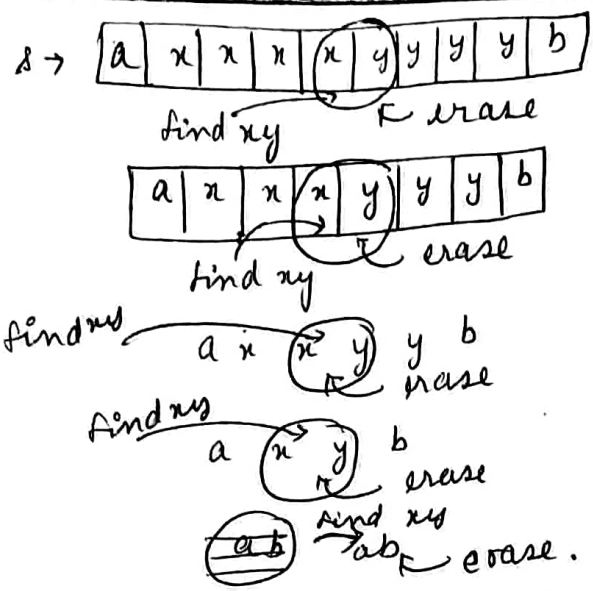


⇒ d a b ans

```
string removeOccurrences(string s, string part)
{
    int pos = s.find(part);
    while(pos != npos){
        s.erase(
        pos = s.find(part);
    }
    return s;
}
```

s.find(part)
returns
① either a valid ~~index~~ starting index
② or npos

H.W → Implement s.find() and s.erase() functions.



Q3 → Valid Palindrome - II → return true if s can be palindrome string after deleting at most 1 character.

GDO leetcode

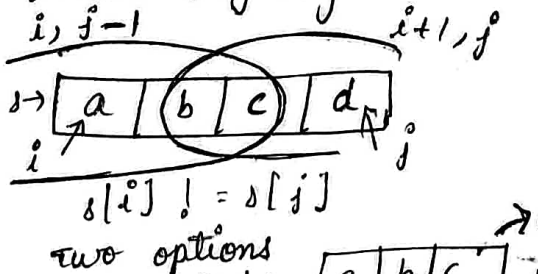
⇒ s → a b c a
i j

if(s[i] == s[j]) → no need to remove
↳ i++, j--, check for next elements

if(s[i] != s[j]) → Two options.

OR condition — या तो s[i] को remove करो, या s[j] को remove करो।

```
bool ~~Palin~~ ValidPalindrome(string s){
    int i=0;
    int n = s.length();
    int j = n-1;
    while(i<j){
        if(s[i] != s[j])   // या तो आप i वाला element हटा दो या फिर j वाला element remove करो।
            return checkPalindrome(s, i+1, j) || checkPalindrome(s, i, j-1);
        else{ // no need to remove anything.
            i++;
            j--;
        }
    }
    return true;
}
```



two options
① ↳ a b c → check Palindrome
② ↳ b c d → check Palindrome.

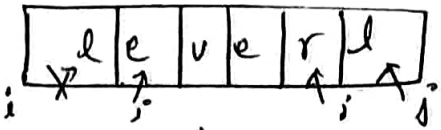
```
bool checkPalindrome(string s,
    int i, int j){
    while(i<j){
        if(s[i] != s[j]){
            return false;
        }
        i++;
        j--;
    }return false;
}
```

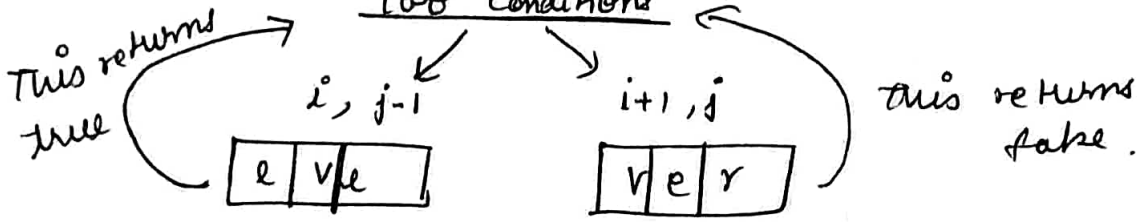
Two cases →

0 - removal → level } True
1 removal → leveel, leevel

if more than 1 removals

eg → abcd, leevel } false.





⇒ true || false → true answer

Another method -

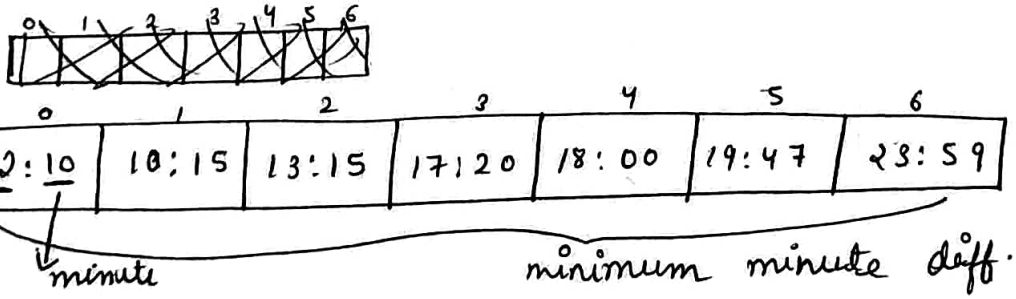
⇒ s → a b b a a
i 0 1 2 3 4 j

0 removal → abbaa → checkPalindrom → false

1 removal → bbaa, abaa, abaa, abba, abba
true false false true true

If one of any these is true, we will return true.

Q4 Minimum Time Difference
539

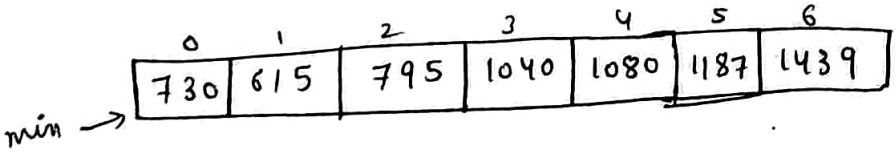


12:10 → 13:10 min. minute diff = 60 min

12:10 → 12:35 min. minute diff = 25 min

We will first convert these times into minutes.

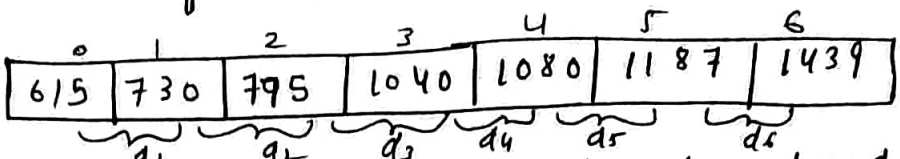
To convert a string into integer → stoi() func is used.



~~int findMinDifference(vector<string~~

sort min. array.

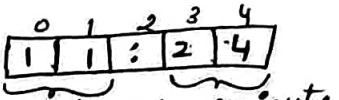
↳ To reduce no. of comparison.



Now compare only with adjacent element and find the difference and return minimum diffuence.

```
int findMinDifference(vector<string>& timePoints){
    vector<int> minutes;
    for(int i=0; i<timePoints.size(); i++){
        string curr = timePoints[i];
        int hours = stoi(curr.substr(0,2));
        int minute = stoi(curr.substr(3,2));
        int totalMinutes = hours*60 + minutes;
        minutes.push_back(totalMinutes);
    }
    // Step2 → sort minute array.
    sort(minutes.begin(), minutes.end());
    // Step3 → difference and calculate min difference
    int mini = INT_MAX;
    int n = minutes.size();
    for(int i=0; i<n; i++){
        int diff = minutes[i+1] - minutes[i];
        mini = min(mini, diff);
    }
    // return mini;
    // Special missing case →
    int lastDiff = (minutes[0]+1440) - minutes[n-1];
    mini = min(mini, lastDiff);
}
```

// Step1 → convert time string into minute integer value.



One edge case will be failed here.

[11:59, 00:00] → it's output = 1

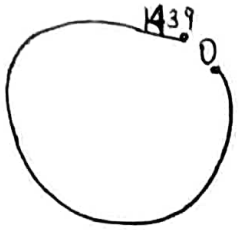
our output = 1439

minutes | 1439 | 0 |

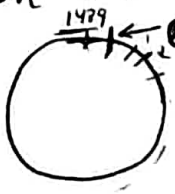
after sorting | 0 | 1439 |

diff = 1439

But time is circular.



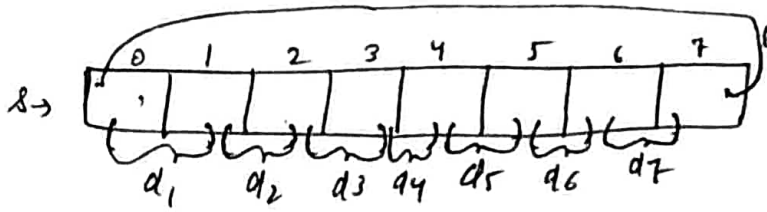
In 24 hours ⇒ 1440 minutes



or add 24:00

so we will add 1440 to the first element and also compare it with the last element.

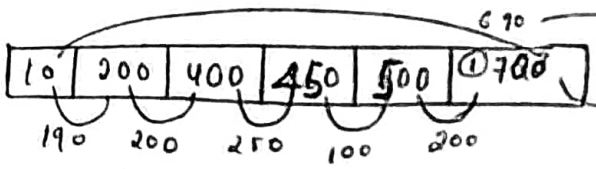
s[0] and s[7]



11:59 → 1 min → 00:00 = 0

= 1439

1439 min ✗



→ and 10+1440-700 = 750

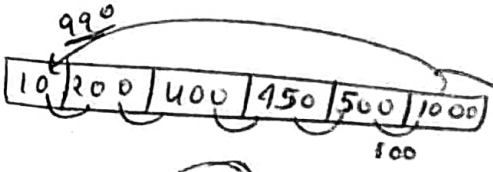
→ 3 cases

① 700

② 1300

③ 1000

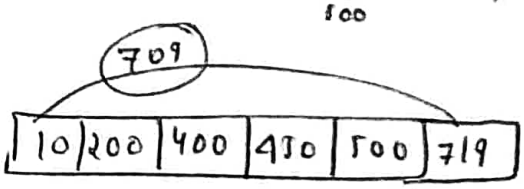
③



10+1440-1000 = 1450-1000

= 450

714

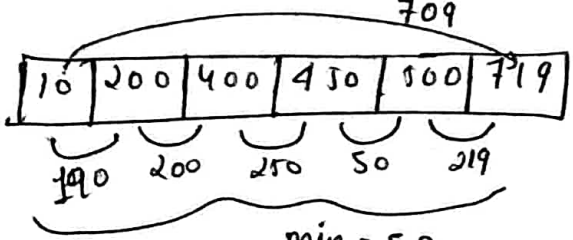


1440+10-719

= 1450-719

= 731

If we are confused whether we have to add 1440 in first element or not, so we will do both and get our answer.



10+1440-719

= 731

min (709, 731)

= 709

min (50, 709)

= 50 ans.

Now slight changes in our code →

```
// Special case →
    int lastDiff1 = (minutes[0] + 1440) - minutes[n-1];
    int lastDiff2 = minutes[n-1] - minutes[0];
    int lastDiff = min(lastDiff1, lastDiff2);
    mini = min(mini, lastDiff);

    return mini;
// we can replace this piece of code with the
   code we wrote previously at special case section.
```

(P) → Number of Palindromic Substring →

647 leetcode

↳ continuous sequence of character within the string.

i/p → a b c

substrings- a, ab, abc, bc, b, c } 3 palindromic

i/p → a a a

substrings → a, aa, aaa, a, aa, a } 3

i/p → n o o n

substrings → n, o, o, n, no, oo, on, noo, oon, noon } 4 palindromic substrings.

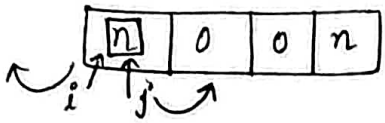
Approch 1 →

① find all substrings. } ——————→ O(n²)

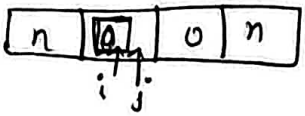
② Check palindrome and if it is palindrome } → O(n)
increment counter by 1.

T.C = O(n³)

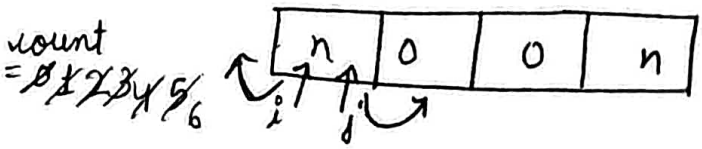
Approach 2-

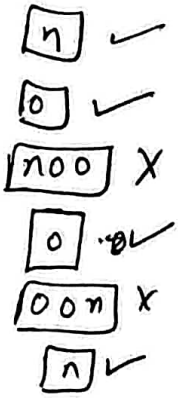


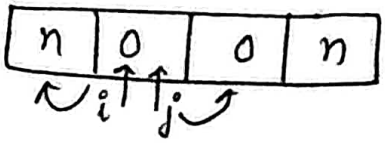
substring → n, o

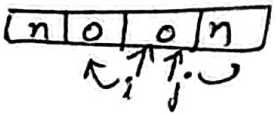


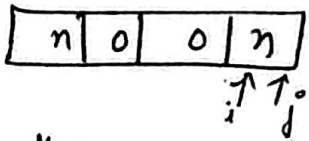
In odd length we start i and j from same index or in even length they both starts from different strings.



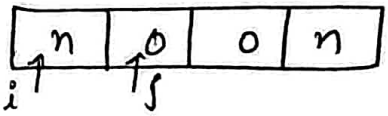


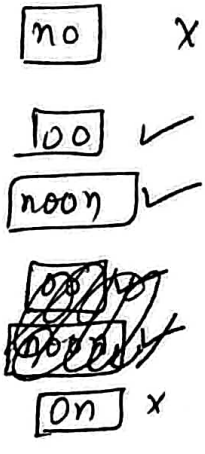


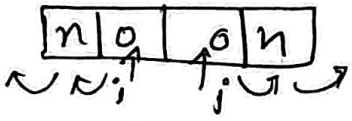


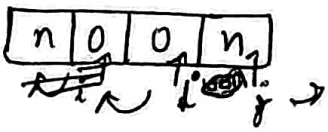


now even length →



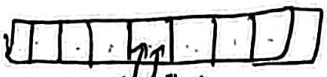




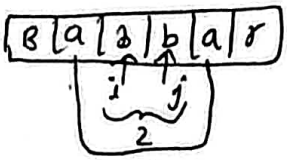


Substring

Odd length — Even length



length → 1, 3, 5, 7, 9 } odd length



length = 2, 4, 6 even length.

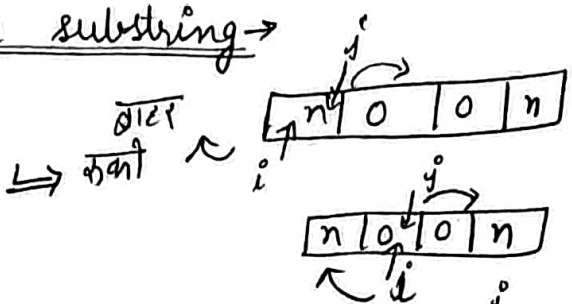
once again →

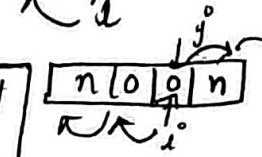
n	o	o	n

count = ~~0 1 2 3 4 5~~ 6

odd length substring →

n
o
noo
oon

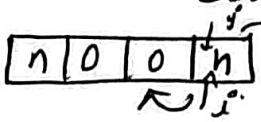




n ✓
o ✓
noo ✗
o ✓
oon ✗
n ✓

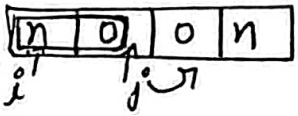
match → count increment
i--, j++
no match → go to next case

STEP
↳ go to next case.



even length substring →

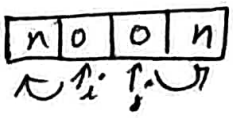
no
oo
on
noon



↳ next case.

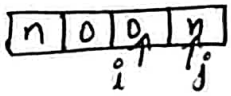
not matched
↳ next case

no ✗
oo ✓



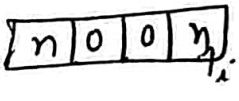
↳ matched

noon ✓



⇒ not matched

on ✗



• code →

```
int countSubstring (string s){
        int count = 0;
        int n = s.length();
        for (int i=0; i<n; i++){
                //odd
                int oddKaAns = expandAroundIndex(s, i, i);
                count += oddKaAns;
                //even
                int evenKaAns = expandAroundIndex(s, i, i+1);
                count += evenKaAns;
        }
        return ~~ans~~ count;
```

```
int expandAroundIndex(String s, int i, int j) {
    int count = 0;
    while(i>=0 && j<s.length() && s[i]==s[j]){
        count++;        Jab tak match karega, tab tak
        i--;            count increment kardo and
        j++;            i piche or j aage kardo.
    }
    return count;
}
```

$T.C. = O(n^2)$