26/02/2023

Q1 You are given a string s consisting of lowercase english letters. We have to remove all adjacent duplicates in the string.

i/p → a b b a c a
o/p → c a

a b b a c a → a a c a → c a (Answer)
Remove Remove

Algorithm

a	z	x	x	z	y

Initially make ans string which will be empty and start adding characters from the i/p string to the ans string

ans = " "
ans = "a"
ans = "az"
ans = "azx"
Now in input string x comes & the previous element added was also x. Remove x from the ans string.
ans = "az"
Now in input string z comes & the last element added in ans is also z & hence remove z from ans string.
ans = "a"
ans = "ay"
Hence we have fully traversed the i/p string & the ans string will be the final answer.

if (ans [ans.length() - 1] == s [i])
    // We have to pop back
else
    // push the character in ans string.

Code

```
string RemoveDuplicates (string s) {
        string ans = " ";
        int i = 0;
        //Traverse whole string
        while (i < s.length()) {
            // Check for valid index
            if (ans.length() > 0) {
                // Pop condition
                if (ans [ans.length()-1] == s[i])
                    ans.pop_back();
                }
                else { // Push condition
                    ans.push_back (s[i]);
                }
            }
            else { //Simply push if ans is empty
                ans.push_back (s[i]);
            }
            i++; //Go to next character of
        }           i/p string
        return ans;
}
```

Time Complexity O(n) → Simply traverse string
Space Complexity O(n) → String ans is made

Q2 Remove all occurences of a substring

i/p → daabcbaabcbc
part = abc

o/p → dab

daabcbaabcbc → dabaabcbc → dab
Remove Remove Remove

We can use find & erase function to solve this question.

Code

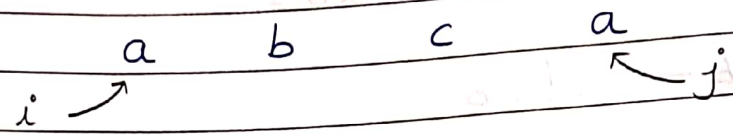
```
                                        part
string removeOccurrences (string s, string p)
{
        //Find index where part is located
        int pos = s.find(part);
//Search  while (pos != string::npos) {
until part is present in the i/p string.
//Erase/remove  s.erase(pos, part.length());
the part        pos = s.find(part); //Update
                                    pos.
    }
    return s;
}
```

Q3 Given a string s, return true if the s can be palindrome after deleting atmost one character from it.

i/p → abca
o/p → True as after removing c, it

becomes a palindrome

Algorithm



1) s[i] == s[j], this means we don't have to remove anything & simply do i++ & j--.

2) s[i] != s[j], either remove element present at i^{th} index or remove element present at the j^{th} index.

Code

```
bool validPalindrome (string s) {

    int i = 0;                          } 2 pointer
    int j = s.length() - 1;             } approach

    while (i <= j) {

        if (s[i] != s[j]) {
        // Either remove ith index or jth index char.
    return checkPalindrome (s, i+1, j) ||
           checkPalindrome (s, i, j-1);

    }
    else {
        i++;                            } Simply increment i &
        j--;                            } decrement j.
    }
```

```
    }
    return true; // 0 removal case.
}

bool checkPalindrome (string s, int start,
                      int end) {

    while (start <= end) {
        if (s[start] != s[end]) {
            return false; //Not Palindrome
        }
        start++;
        end--;
    }
    return true; //Checked full string
}
```

↳ This is the game question (Revise again)

Q4 Given a list of 24-hour clock time points in "HH:MM" format, return the minimum minutes difference blw any 2 points in the list.

i/p → 12:10 10:15 13:15 17:20 18:00 19:47 23:59

o/p →

First of all to make our work easy, convert the time into minutes like

10:15 → 10 X 60 + 15 = 600 + 15 = 615 minutes

Also to convert the string to integers, we use function named stoi. Now we have

got an integer array which contains the minutes of all the time. Now we need to sort the array of minutes so as to reduce the number of comparasion & now just check the adjacent elements & find the minimum difference among all. Here the answer of 23:59 and 00:00 will give the wrong answer & hence we need to handle this case explicitly. We need to compare the first & last elements also & then we will obtain the right answer.

Code

```
int findMinDifference (vector<string>& timepoints)
// Finding the minutes array
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 total = hours * 60 + min;
    minutes.push_back (total);
}
// Sort the minutes array
sort (minutes.begin(), minutes.end());
// Calculate minimum difference
int n = minutes.size();
int mini = INT_MAX;
for (int i=0; i<n-1; i++){
    int diff =
        minutes[i+1] - minutes[i];
```

```
mini = min (mini, diff);
}
// Special case
int lastdiff = (minutes[0] + 1440) - minutes[n-1]
mini = min(mini, lastdiff);
return mini;
}
```

→ This is the game in this whole question. This will solve the edge case.

Note→ 1440 → 24 hours i.e why we added 1440 in the first element & then subtracted last index of minutes array.

Q5 Given a string s, return the number of palindromic substrings in it.

i/p → abc
o/p → 3

"a" "b" "c" } 3 palindromic substrings

No. of substrings in abc

a, ab, abc, b, bc, c → Total 6 substrings out of which 3 are palindromic substrings.

Algorithm

n	o	o	n

Length of substring can be odd or even & while finding the substring we check for

palindrome

Odd case

```
n  o  o  n
↑↑
i j
```

Initially count = 0 , now s[i] == s[j] hence increment count

count = 1 , i-- & j++ . Now i = -1 and hence exit.

```
n  o  o  n
   ↑↑
   i j
```

s[i] == s[j] ⇒ count = 2 , i-- & j++

```
n  o  o  n
↑        ↑
i        j
```

noo → not palindrome & hence count won't be incremented.

i-- & j++ but i becomes -1 & hence exit

```
n  o  o  n
      ↑↑
      i j
```

s[i] == s[j] ⇒ count = 3 , i-- & j++

```
n  o  o  n
↑        ↑
i        j
```

oon → not a palindrome & hence i--, j++ but here j will go out of range & hence exit.

```
n  o  o  n
      ↑↑
      i j
```

s[i] == s[j] ⇒ count = 4 & hence i--, j++ but j gets out of range & hence exit.

Even case

```
n  o  o  n
↑  ↑
i  j
```

s[i] != s[j] ⇒ True & hence go to next case

```
n  o  o  n
   ↑  ↑
   i  j
```

s[i] == s[j] ⇒ True & hence increment count = 4+1 = 5

```
n  o  o  n
↑        ↑
i        j
```

s[i] == s[j] ⇒ True & hence increment count = 5+1 = 6. i-- & j++. But now i & j are out of range

n o o n
↑ ↑
i j

s[i] != s[j] ⇒ move to next case but then j becomes out of range & hence return count now.

Ans = 6

Code

```
int expandAroundIndex (string s, int i, int j){

    int count = 0;
    while (i >= 0 && j < s.length() &&
           s[i] == s[j]) {

        count++;
        i--;
        j++;

    }
    return count;
}

int countPalindromicSubstrings (string s){

    int count = 0;
    int n = s.length();
    for (int i=0; i< s.length(); i++){
      // Odd length substring
      int oddAns = expandAroundIndex (s, i, i
```

```
        count = count + oddAns;
        int evenAns = expandAroundIndex(S, i, i+1);
        count = count + evenAns; //Even length
                                    substring
    }
    return count; //Total substrings
}
```