**LC#5 LONGEST PALINDROMIC SUBSTRING:**

**APPROACH 1:USING DP**

class Solution {

public String longestPalindrome(String s) {

int n = s.length();

if(n==1)

return s;

boolean dp[][] = new boolean[n][n];

int maxlen = 1 , start = 0;

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

dp[i][i] = true;

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

{

if(s.charAt(i)==s.charAt(i+1))

{

dp[i][i+1] = true;

maxlen = 2;

start = i;

}

}

for(int len = 3 ; len<=n;len++)

{

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

{

int j = i +len-1;

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

{

maxlen = len;

start = i;

dp[i][j] = true;

}

}

}return s.substring(start,start+maxlen);

}

}

**2)USING TWO POINTER (EXPAND AROUND CENTER)**

class Solution {

public String longestPalindrome(String s) {

if (s == null || s.length() < 2) return s;

int start = 0, end = 0;

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

int len1 = expand(s, i, i); // odd length

int len2 = expand(s, i, i + 1); // even length

int len = Math.max(len1, len2);

if (len > end - start) {

start = i - (len - 1) / 2;

end = i + len / 2;

}

}

return s.substring(start, end + 1);

}

private int expand(String s, int l, int r) {

while (l >= 0 && r < s.length() && s.charAt(l) == s.charAt(r)) {

l--;

r++;

}

return r - l - 1;

}

}

**LC#647:PALINDROMIC SUBSTRINGS**

**APPROACH:EXPAND AROUND CENTER(OPTIMAL FOR INTERVIEWS)**

class Solution {

    public int countSubstrings(String s) {

        int n = s.length();

        int count = 0;

        for(int i = 0 ;i < n ; i++)

        {

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

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

        }

        return count;

    }

    public int isPalindrome(int start , int end , String s)

    {int count = 0;

        while(start>=0 && end < s.length() && s.charAt(start)==s.charAt(end) )

        {

            count++;

            start--;

            end++;

        }

        return count;

    }

}

**APPROACH 2:USING DP**

class Solution {

public int countSubstrings(String s) {

int n = s.length();

int count = 0;

boolean dp[][] = new boolean[n][n];

for(int i = 0 ; i<n ;i++)

{

dp[i][i] = true;

count++;

}

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

{

if(s.charAt(i)==s.charAt(i+1))

{

dp[i][i+1] = true;

count++;

}}

for(int l = 3 ; l<=n ;l++)

{

for(int i = 0 ; i<=n-l ; i++)

{

int j = i+l-1;

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

{

dp[i][j] = true;

count++; } } }

return count;} }