Todays Content:

a) Substring Intro

b) Palindrome or Not

c) length of longest Palindromic Substring

d) longest substring with all distinct characters

e) Given a char[] reverse word by word

Substring: Concept of Subarray on a String is called Substring.

a) Continous Part of a String

b) Full String is a Substring

c) Single character is also a substring

Palindrome:

a) madam b) malayalam c) civic d) rada

YES YES YES NO

Q1: Check if a given substring is Palindrome or Not?

	0	1	2	3	4	5	6	7	8	9	10
String str =	a	n	a	m	a	d	a	m	s	p	e

s → s → s ← e ← e

S = 3 e = 7 : Yes

S = 5 e = 8 : No

```
bool ispal(String str, int s, int e){ // str = s  s+1  s+2 ... e-2  e-1  e
    while( s < e){// s<=e is also fine
        if(str.charAt(s) == str.charAt(e)){
            s=s+1, e=e-1;
        }
        else{ return false}
    }
    return true;
}
```

TC: O(N) SC: O(1)

2Q) Given a String, return length of longest Palindromic Substrings

Constraints:

$1 <= N <= 10^3$

Ex1: S = a b a c a b (indices 0 1 2 3 4 5) ans = 5

Ex2: S = a n b c (indices 0 1 2 3) ans = 1

Ex3: S = d a b b a (indices 0 1 2 3 4) ans = 4

Idea1: For every substring, check if it is palindrome or not?

TC: $O(N^2) * O(N) \approx O(N^3)$ SC: $O(1)$

↳ $\approx N^2$ substring

$\approx O(N)$ time to check if a substring is palindrome

```
int longPal(char ch[]){
    TODO:
}
```

Optimize:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Str =	x	b	d	y	z	z	y	d	e	b	d	y	z	y	d	x

$P_1 \leftarrow P_1 \leftarrow P_1$ $\quad$ $P_2 \rightarrow P_2 \rightarrow P_2$

$P_1 \leftarrow P_1 \leftarrow P_1 \leftarrow P_1 = P_2 \rightarrow P_2 \rightarrow P_2 \rightarrow P_2$

len: $P_2 - P_1 - 1 = 8 - 1 - 1 = 6$

obs: Substring $[P_1+1, P_2-1]$

b - a + 1

len: $P_2 - 1 - (P_1 + 1) + 1 =$

$P_2 - 1 - P_1 - \cancel{1} + \cancel{1} = P_2 - P_1 - 1$

Idea:

1. **Odd length Palindromes**

   Take every character as center & calculate, longest palindromic length & consider overall max.

   TC: $O(N) * O(N) = O(N^2)$ SC: $O(1)$

2. **Even length Palindromes**

   Take every adjacent characters as center & calculate longest palindromic length & consider overall max.

   TC: $O(N) * O(N) = O(N^2)$ SC: $O(1)$

3. **Final ans**

   : max of [even length Palindrome, Odd length Palindrome]

   TC: $O(N^2 + N^2) = O(N^2)$ SC: $O(1)$

TODO: Self learning = manacher's = $O(N)$ $O(N)$

Expand on Center?

	0	1	2	3	4	5	6	7	8
S =	a	b	e	f	g	f	e	k	l

$P_1 \leftarrow P_1 \leftarrow P_1 \leftarrow P_1 \; P_2 \rightarrow P_2 \rightarrow P_2$

len $= P_2 - P_1 - 1 = 7 - 1 - 1 = 5$

	-1	0	1	2	3	4	5
S =		a	b	c	b	a	

$P_1 \leftarrow P_1 \leftarrow P_1 \; P_1 \; P_2 \rightarrow P_2 \rightarrow P_2 \rightarrow P_2$

len $= P_2 - P_1 - 1 = 5 - (-1) - 1 =$

$= 6 - 1 = 5$

	0	1	2	3
S =	a	b	c	d

$P_1 \; P_1 \; P_2 \; P_2$

len $= P_2 - P_1 - 1 = 2 - 0 - 1$

$= 1$

```
int longPal(String s){ TC: O(N^2) SC: O(1)
    int N=s.length();
    int even=0, odd=0; // max even length & max odd length
    for(int i=0; i<N; i++){ // Max odd length
        // Take i as center
        int p1=i, p2=i;
        while(p1>=0 && p2<N){
            if(s.charAt(p1)==s.charAt(p2)){
                p1--; p2++
            }
            else{ break}
        }
        int len=p2-p1-1;
        odd = max(odd, len)
    }

    for(int i=0; i<N; i++){ // Max even length
        // Take i & i+1 as centres
        int p1=i, p2=i+1;
        while(p1>=0 && p2<N){
            if(s.charAt(p1)==s.charAt(p2)){
                p1--; p2++
            }
            else{ break}
        }
        int len=p2-p1-1;
        even = max(even, len)
    }

    return max(even, odd);
}
```

0 1 2 3 . . . . N-1 N

p_1 p_2 p_2

10:20 break

Note: Once take a small string & dry run & Check for errors?

Q8 length of longest substring with all distinct characters?

	0	1	2	3	4	5	6	7	
$S_1 =$	a	b	c	a	b	c	d	d	ans = 4

	0	1	2	3	4	5	6	7	
$S_2 =$	s	i	p	p	i	e	r	g	ans = 5

	0	1	2	3	4	
$S_3 =$	a	a	a	a	a	ans = 1

Idea1: For all substrings, check if a substring contains all distinct & get max length

TC: $O(N^2) * O(N) = O(N^3)$ SC: $O(N)$

// Substrings

// If a substring contains all distinct characters

Idea: Insert all characters in hashset & check

Idea2: For every index i:

Calculate length of longest substring with all distinct chars & get overall max

	0	1	2	3	4	5	6	7		
S =	a	b	a	c	e	a	f	e	HS:	c e a f
len =	2	4	3	4	3	3	2	1	Final ans = 4	

```
int longdis(String s){ TC: O(N^2) SC: O(N)
    int N = s.length, ans = 0.
    for(int i = 0; i<N; i++){ // For i: get length of longest substring distinct
        HashSet<Char> hs;
        for(int j = i; j<N; j++){
            if(hs.contains(s.charAt(j)) == false){
                hs.add(s.charAt(j))
            }
        } else { break}
        ans = max(ans, hs.size())
    }
    return ans;
}
```

Ideas:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Str =	~~a~~	~~b~~	~~c~~	~~g~~	~~x~~	~~z~~	~~g~~	~~k~~	l	m	h	a	b	k	
									i						j : reach end : break

Valid:

(0 0)
(0 1)
(0 2)
(0 3)
(0 4)
(0 5)

(1 1)
(1 2)
(1 3)
(1 4)
(1 5)

Skip them

Hashset:

~~a~~ ~~b~~ ~~c~~ ~~g~~ ~~x~~ ~~z~~ ~~g~~ k l m
h a b k

(0 6) $\xrightarrow[S[0]]{hs*}$ (1 6) $\xrightarrow[S[1]]{hs*}$ (2, 6) $\xrightarrow[S[2]]{*}$ (3, 6) $\xrightarrow[S[3]]{*}$ (4, 6) → (4, 7) → (4, 8) → (4, 9)

(4, 9) → (4, 10) → (5, 10) → (5, 11) → (5, 12) → (5, 13) → (6, 13) → (7, 13)

(8, 13) → (8, 14)
Stop

```
int longdis(String s){ TC: O(N) SC: O(N)
    int N = str.length;
    int ans = 0;
    hashset<char> hs;
    int i = 0, j = 0;
    while(j<N){
        if(hs.contains(s.charAt(j)) == false){
            hs.add(s.charAt(j))
            ans = max(ans, hs.size())
            j++
        }
        else{ hs.remove(s.charAt(i))
            i++;
        }
    }
    return ans;
}
```

To check if substring contains all distinct characters

3Q) Given a char ch[] reverse given substring?

Ex:

	0	1	2	3	4	5	6	7
ch[] =	a	n	a	c	o	n	d	a

s = 2 e = 6

s → s → s e ← e ← e

d n o c a

{a n d n o c a a}

```
void reverse(char ch[], int s, int e){
    while(s <= e){
        char t = ch[s]; //swap ch[s] with ch[e]
        ch[s] = ch[e];
        ch[e] = t;
        s++, e--;
    }
}
```

4Q) Given a char[] reverse word by word.

Note: a. No Extra Spaces at start and end

b. Every word Separated by single space

c. No usage of inbuilt library

d. Expected TC: O(N) SC: O(1)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Ex1: ch[] =	l	o	v	e		h	a	t	e		d	a	t	a		c	o	d	e	s

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Step1: Reverse full =	s	e	d	o	c		a	t	a	d		e	t	a	h		e	v	o	l

Step2: Reverse each word =

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Final output:	c	o	d	e	s		d	a	t	a		h	a	t	e		l	o	v	e