DSA

23/8/25

Strings

★ **String:** Sequence or array of character, its represented using double quotes or ''
"Hello"

"Hello" → All elements characters of strings must be inside the quotes

★ **Character:** Single symbol that represents a letter, number or other symbol on the keyword
Represented using its ASCII value.
' ' single quotes.

Computer stores everything in binary, but how do we store strings?

ch = A
A → ASCII → Binary

'A' → 65	+32 →	'a' → 97		'0' → 48	
'B' → 66	← -32	'b' → 98		'1' → 49	
'C' → 67		'c' → 99		'2' → 50	
'D' → 68		'd' → 100		'3' → 51	
'E' → 69		'e' → 101		'4' → 52	
⋮		⋮		⋮	
'Z' → 90		'z' → 122		'9' → 57	

☆ Switch Case:

Given a char[] consisting of only alphabets [either lowercase or uppercase]. Print all the characters of string in such a way that for all lowercase characters, print its uppercase character & vice versa.

Hello → hELLO

- Lowercase → Uppercase subtract 32
- Upper → Lower add 32

```
def solve(A):
    N = len(A)
    result = []
    for i in range(N):
        if 'a' <= A[i] <= 'z':
            result.append(chr(ord(A[i]) - 32))
        else:
            result.append(chr(ord(A[i]) + 32))
    return "".join(result) → becomes a string
```

chr(ord(A[i]) - 32)

if A[i] = 'a'

- → ord(A[i]) = 97
- → 97 - 32 = 65
- → chr(65) = A

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

✱ Check palindrome :

Check whether substring is palindrome or not.

Approach : abcdadedefg

A[s] == A[e]
A[s] != A[e]

```
def palindrome(s, start, end):
    while start < end:
        if s[start] == s[end]:
            start += 1
            end -= 1
        else:
            return False
    return False
```

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

Longest Palindrome :

Given a string s, calculate the length of longest palindrome substring in s.

"anamadamm"

madam → (5)

Idea 1 → 1. Generate all substrings # O(N²)
2. Check all substrings for being a palindrome #O(N)
3. If the substring is palindrome, maximize the length # O(1)

Total TC : O(N²) * O(N)
= O(N³)

Idea 2 → Optimized str[] = a b c b a k d

1> Assume $p_1, p_2 = 2$

$p_1 = p_2 = c$

$p_1 -= 1, p_2 += 1, p_1 = 1, p_2 = 3 \rightarrow b$

$p_1 -= 1, p_2 += 1, p_1 = 0, p_2 = 4 \rightarrow a$

$p_1 -= 1, p_2 += 1, p_1 = -1, p_2 = 5 \rightarrow$ OB (stop)

$p_1 = -1, p_2 = 5$

[a b] = b - a + 1

(a b) = b - a + 1 - 2 = b - a - 1

$(p_1, p_2) = p_2 - p_1 - 1$

$= 5 - (-1) - 1 = (5)$

2> Assume $p_1, p_2 = 2$

$p_1 = p_2 = c$

$p_1 -= 1, p_2 += 1, p_1 = 1, p_2 = 3 \rightarrow b$

$p_1 -= 1, p_2 += 1, p_1 = 0, p_2 = 4 \rightarrow a, d$ stop

$p_1 = 0, p_2 = 4 \Rightarrow p_2 - p_1 - 1 \Rightarrow 4 - 0 - 1 = 3$

3> Assume $p_1 = 5, p_2 = 6$

$p_1 = k, p_2 = k$

$p_1 -= 1, p_2 += 1, p_1 = 4, p_2 = 7 \rightarrow c$

$p_1 -= 1, p_2 += 1, p_1 = 3, p_2 = 8 \rightarrow d$

$p_1 -= 1, p_2 += 1, p_1 = 2, p_2 = 9 \rightarrow$ stop

OR

$p_1 = 2, p_2 = 9 \Rightarrow p_2 - p_1 - 1$

$9 - 2 - 1$

$= 6$

4> Assume $p_1 = 2, p_2 = 3$,

$p_1 -= 1, p_2 += 1, p_1 = 1, p_2 = 4 \rightarrow a$

$p_1 -= 1, p_2 += 1, p_1 = 0, p_2 = 5 \rightarrow c, d$

stop

$p_1 = 0, p_2 = 5$

$p_2 - p_1 - 1 = 5 - 0 - 1$

$= 4$

☆ Optimized Idea :

- Every single character can act as a center
- Take single character as a center and expand & get palindrome length
  - Odd Length Palindrome
- Take adjacent characters as centers & expand & get palindrome length.
  - Even Length Palindrome.

☆ Code :

```
def longestPalindrome(s):
    max_len = 0
    n = len(s)
    # Take every char as centes
    for c in range(n):
        p1 = p2 = c    # odd length palindrome
        while p1 >= 0 and p2 < n :  # O(N)
            if s[p1] != s[p2]:
                break

            p1 -= 1
            p2 += 1

        new_len = p2 - p1 - 1
        max_len = max(max_len, new_len)
```

Even length palindrome

$p_1 = c$

$p_2 = c+1$

while $p_1 >= 0$ and $p_2 < n$: # O(N)

if $s[p_1] != s[p_2]$:

break

$p_1 -= 1$

$p_2 += 1$

new_len = $p_2 - p_1 - 1$

max_len = max(max_len, new_len)

return max_len

TC : O(N) (O(N) + O(N)) $\Rightarrow$ O(N²) + O(N²)

$= 2N^2$

$\Rightarrow O(N^2)$

SC : O(1)