

647. Palindromic Substrings

Approach-1 Brute Force Approach

→ generate all substrings

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

$$\hookrightarrow \text{no. of substrings} = \frac{n(n+1)}{2}$$

→ check for every substring that is this substring is a palindrome or not.

$$\hookrightarrow T.C. = O(n)$$

$$\boxed{T.C. = O(n^2 \times n) \Rightarrow O(n^3)}$$

Approach-2

~~DP approach~~

DP approach.

IMP:- This is a new method for DP approach.

→ Diagonal traversal.

ex

"a b c c b c"

end →

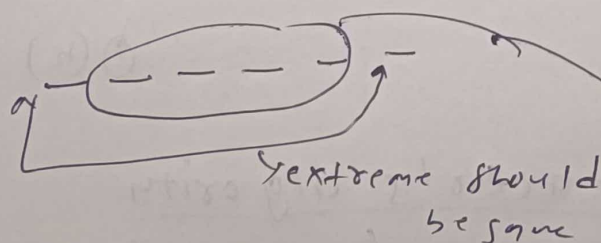
start ↓

	a ₁	b ₂	c ₃	c ₄	b ₅	c ₆
a ₁	a	ab	abc	abcc	abccb	abccbc
b ₂	x	b	bc	bcc	bccb	bccbc
c ₃	x	x	c	cc	ccb	ccbc
c ₄	x	x	x	c	cb	cbc
b ₅	x	x	x	x	b	bc
c ₆	x	x	x	x	x	c

(Note: The table contains handwritten 'T' for True and 'F' for False. Red arrows trace a path from (a₁, c₆) to (b₂, c₅) to (c₃, c₄) to (c₄, c₃) to (b₅, c₂) to (c₆, c₁), indicating a valid palindrome sequence.)

Imp a string is a palindrome

diff b/w start & end



extreme should be same

& this is also a palindrome

[c₆, c₁] → extreme same
 [c₆, c₁] → this result [row+1, col-1]

Approach-3 Two pointer approach.

ii
a a b a a b c ← even length expansion

iii
a a a a b c ← odd length expansion

$$\begin{aligned} T.C. &= O(n^2) \\ S.C. &= O(1) \end{aligned}$$

Approach-4 more optimization

Option-1

Suffix Array

Build suffix trees

→ LCA lookups



$O(n)$ time

Option-2

Manacher's algorithm

(2 pointer approach

on steroids

$O(n)$