647. Palindromic Substrings

Approch-1 Brute force Approch

→ Generate all substrings

↳ T.C. = $O(n^2)$

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

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

↳ T.C. = $O(n)$

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

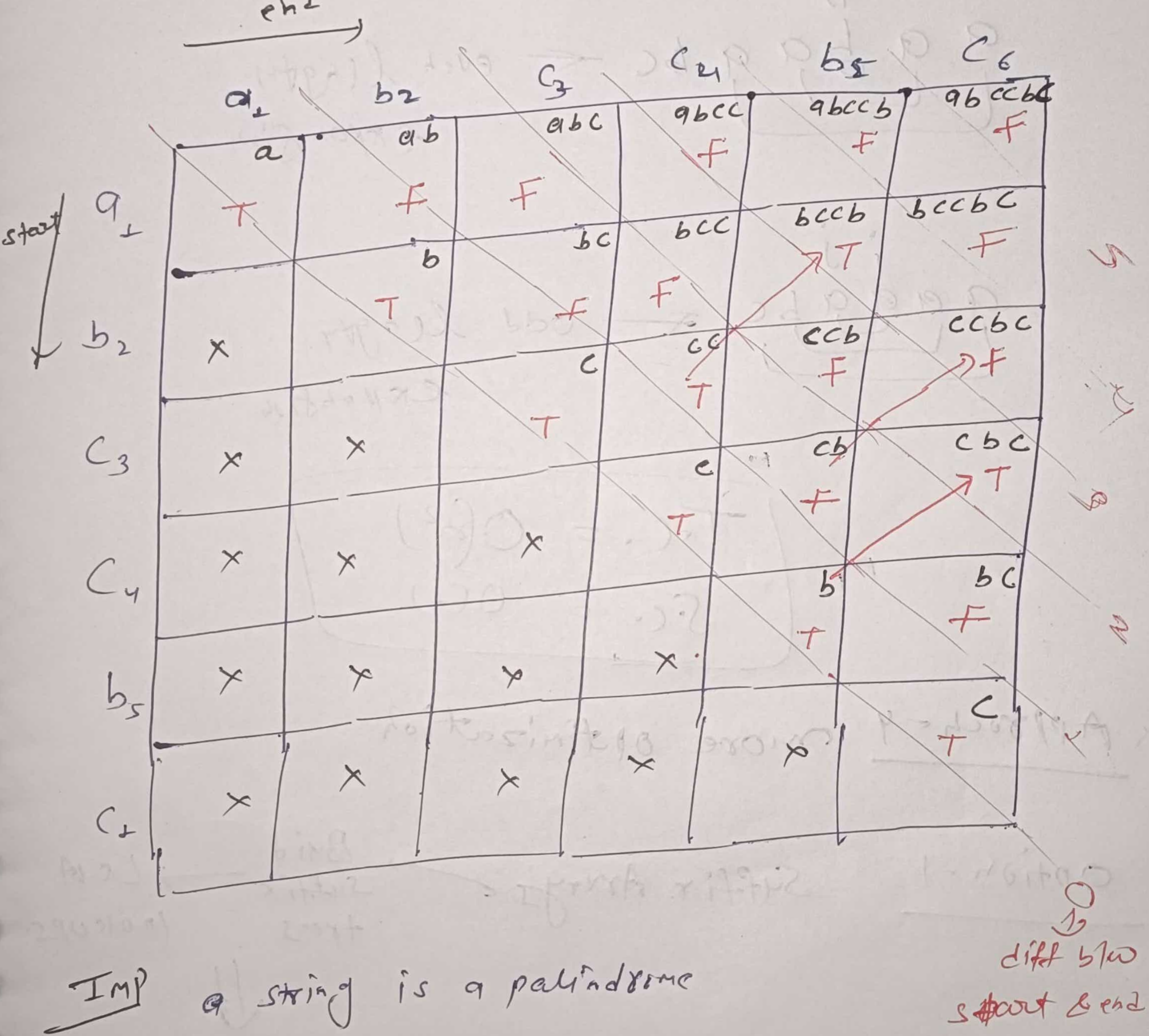
Approch-2

DP approch.

IMP:- This is a new metod for DP approch.

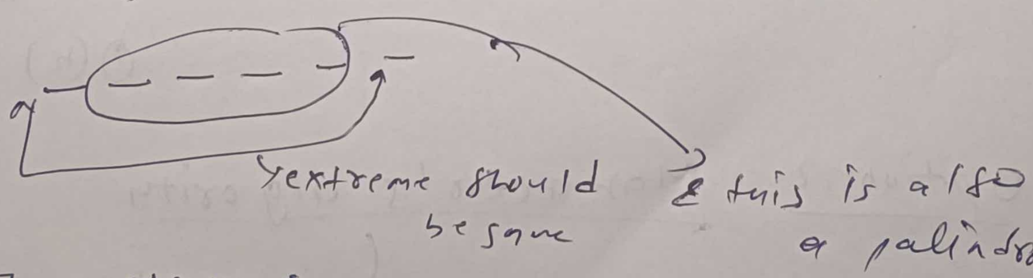
→ Diagonal traversal.

ex "abccbc"


end →
start ↓
5
4
3
2
1
0
diff b/w start & end

	a_1	b_2	c_3	c_4	b_5	c_6
a_1	a T	ab F	abc F	abcc F	abccb F	abccbc F
b_2	×	b T	bc F	bcc F	bccb T	bccbc F
c_3	×	×	c T	cc T	ccb F	ccbc F
c_4	×	×	×	c T	cb F	cbc T
b_5	×	×	×	×	b T	bc F
c_1	×	×	×	×	×	c T

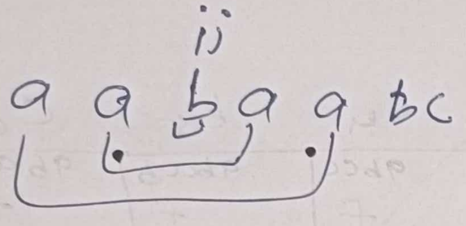
Imp a string is a palindrome



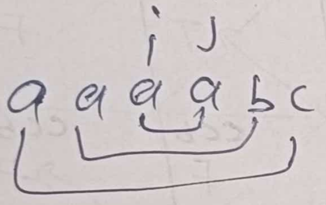
cbc → extreme same

cbc → This result ↳ [row+1, col-1]

Approch-3 Two pointer approch.



← even length. expansion.



← odd length expansion

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

Approch-4 more optimization

Option-1 Suffix Arrays ← Buid suffix trees → LCA lookups

⇓

O(n) time

Option-2 Manacer's algorith

↳ 2 pointer approcy on steroids ⇒ O(n)