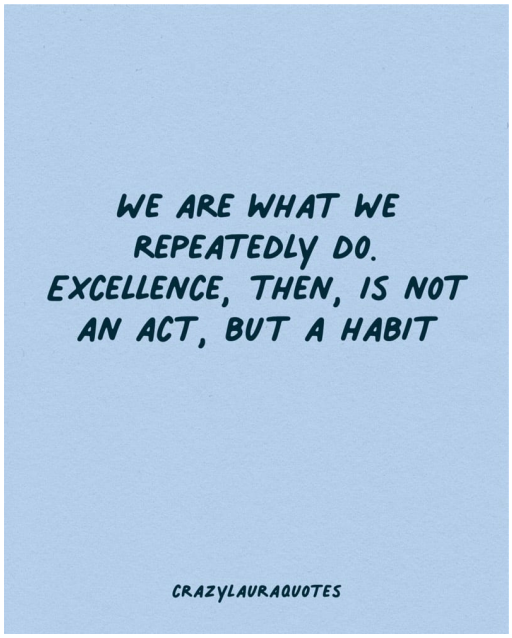
Dp → 6



Good Evening

Content

01. Longest Increasing Subseq
02. Russian Envelopes
03. Check if string is palindrome
04. Min palindromic cut

Q Given ar[N], find length of longest increasing subsequence

ar[] = {6, 9, 10, 13, 20} ans=5

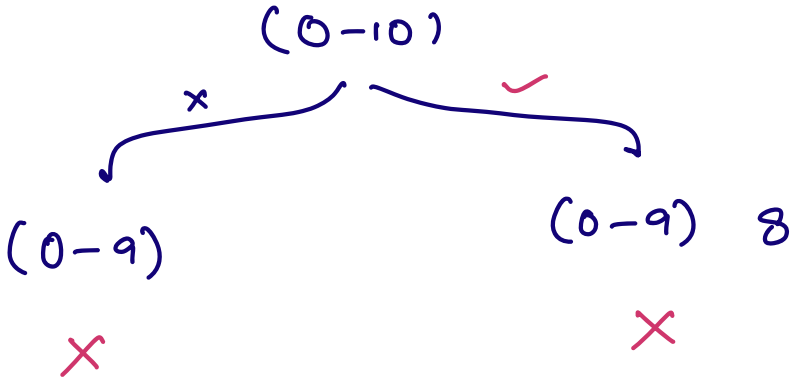
ar[] = {13, 6, 2, 1} ans=1

ar[] = {0, 8, 4, 2, 10, 6, 11, 15}

len	LIS
3	→ 6, 11, 15
4	→ 8, 10, 11, 15
5	→ 0, 4, 6, 11, 15

ar[] = {10, 3, 12, 7, 9, 11, 20, 11, 13, 6, 8}
0 1 2 3 4 5 6 7 8 9 10

5 → {3, 7, 9, 11, 13}



$$lis(arr, i) \Rightarrow \max\left[\underbrace{0[0\text{-}9] \quad lis(arr, j)}_{arr[j] < arr[i]}\right] + 1$$

```
main ( ) {
    ans = 0;
    for (i = 0; i < n; i++) {
        ans = max(ans, lis(arr, i));
    }
}

int lis(arr, i) {
    max = 0
    for (j = 0; j < i; j++) {
        if (arr[j] < arr[i]) {
            max = Math.max(max, lis(arr, j));
        }
    }
    return max + 1
}
```

dp[i] → length of LIS ending at idx i

arr[ ] =

10	3	12	7	9	11	20	11	13	6	8

dp[ ] =

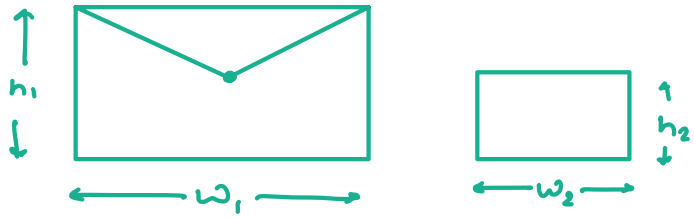
0	1	2	3	4	5	6	7	8	9	10
1	1	2	2	3	4	5	4	5	2	3

```
ans = 0

int dp[N];

dp[0] = 1;

for(i=1; i<n; i++){
    max = 0;
    for(j=0; j<i; j++){
        if(ar[j] < ar[i]){
            max = maximum(max, dp[j]);
        }
    }
    dp[i] = max+1;
    ans = max(ans, dp[i]);
}

return ans;
```

TC: $O(n^2)$

SC: $O(n)$

Russian Doll Envelopes



N - different envelopes

Find max count of envelopes that can be put in a single envelope.

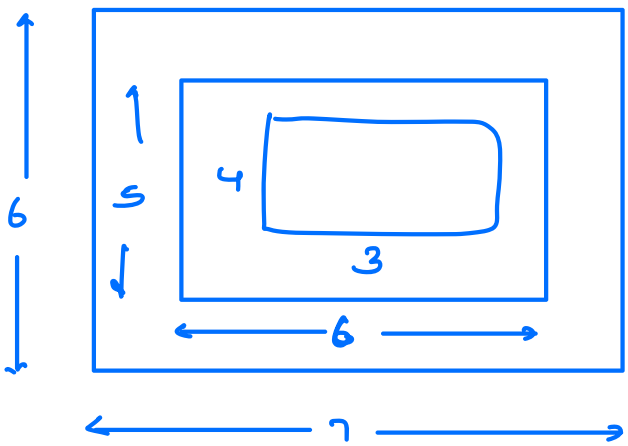
* Rotation of envelope is not allowed

Condn to put an envelope → $h[i] < h[j]$

$w[i] < w[j]$

	ht	width
A →	5	6
B →	6	4
C →	6	7
D →	4	3

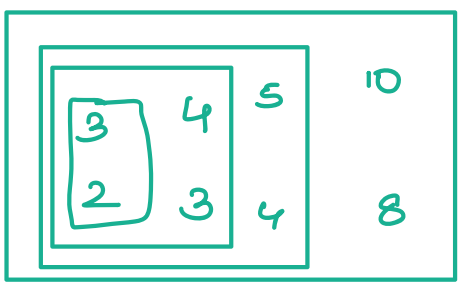
Ans = 3



h[] → {9 5 10 3 4 2}

w[] → {3 4 8 2 3 7}

Ans = 4



Sort arr on the basis of ht

h[small] < h[larger]

w[small] < w[larger]

h[] → { 2 3 4 5 9 10 }

w[] → { 7 2 3 4 3 8 }

Idea → 01. Sort the arr on the basis of ht ✓

02. LIS on the basis of width &
make sure ht is not equal

h[] → { 1 2 3 4 4 5 7 10 10 }

w[] = { 10 3 7 9 11 20 11 6 13 }

Ans = 5

lis → { 1 1 2 3

TC → $O(n^2 + n\log n)$

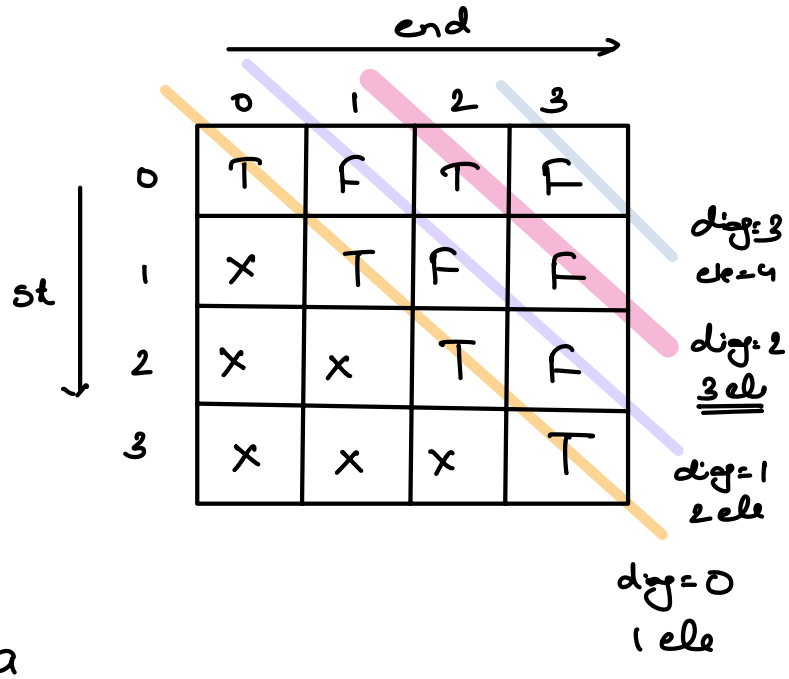
SC → $O(n)$

* Overlapping Bridges → Practice

10 pm → 10:10 pm

9 Given a string, for every substring, check if that is palindrome or not.

S = "abac"
0 1 2 3



str[i] == str[j] → dp[i][j] ⇒ dp[i+1][j-1]

str[i] != str[j] → dp[i][j] → false

⇒

dif	0	1	2	3
	0 0	0 1	0 2	0 3
	1 1	1 2	1 3	
	2 2	2 3		
	3 3			

```
for ( d = 0 ; d < n ; d++ ) {
    r = 0 , c = d ;
    while ( c < n ) {
        if ( d == 0 ) { dp[r][c] = True ; }
        else if ( d == 1 ) { dp[r][c] = ( str[r] == str[c] ) ;
        else {
            if ( str[r] != str[c] ) dp[r][c] = False
            else dp[r][c] = dp[r+1][c-1] ;
        }
        r++, c++ ;
    }
}
return dp[][] ;
```

TC : $O(n^2)$

SC : $O(1)$

* Count of palindromic substring → No. of times we have true in dp

* Longest palindromic substring

↳ max diagonal that has true on it

ans = diag + 1 ;

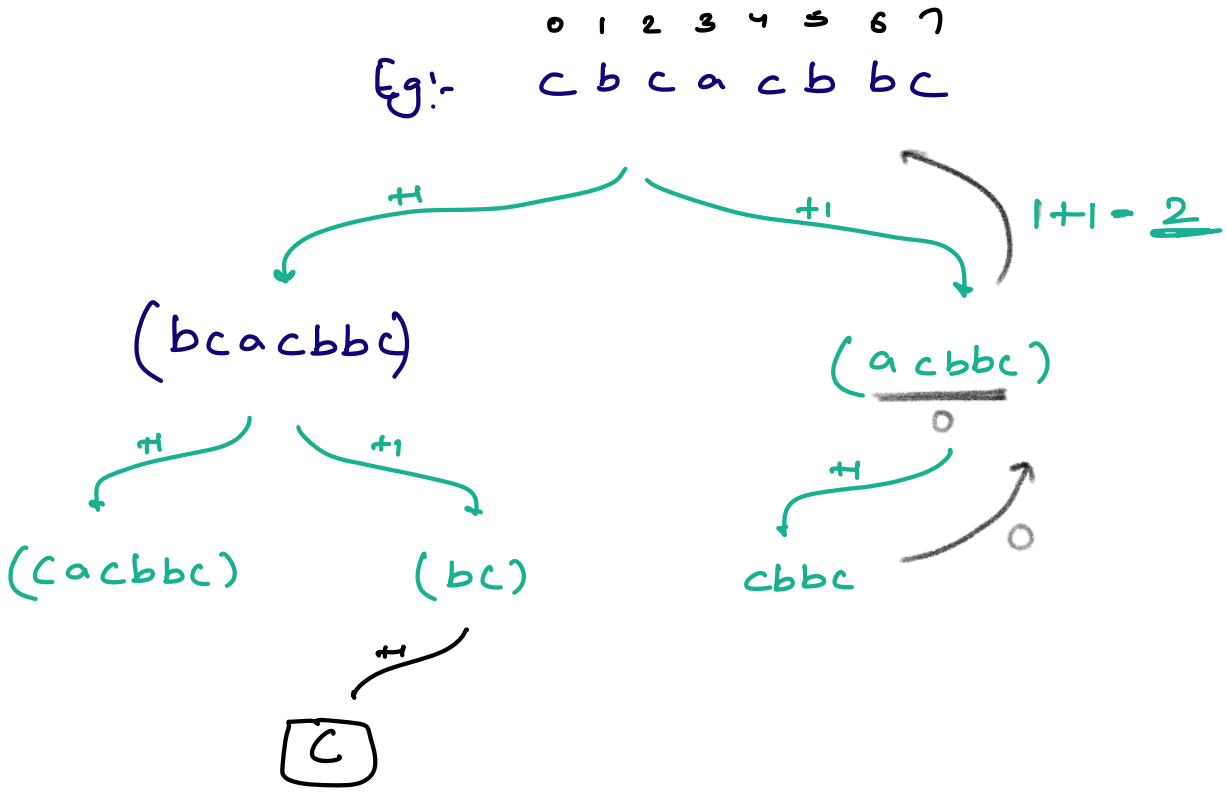
Q Find min no. of cuts to partition the string such that all partitions are palindrome

Eg :- x x | y Ans = 1

Eg :- x | a | b a a b | p Ans : 3

Eg :- a b b a | c Ans = 1

a | b b | c b c Ans = 2



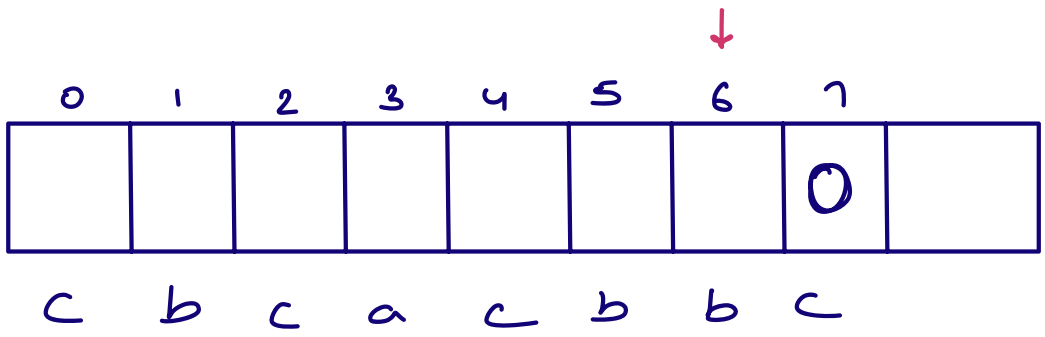
01. Create a palindromic dp

```
int minCuts ( str, si, ei )
{
    if ( isPalindrome(str, si, ei) == T){
        return 0;
    }

    for (cut = si ; cut < ei ; cut++){
        if ( isPalindrome(str, si, cut) == true){
            min = min(min, minCut (str, cut+1 , ei ) );
        }
    }

    return min+1;
}
```

* Doubts

str = cbcacbbc

dp[i] = min cut from st to e




```
for ( i = n-1 ; i ≥ 0 ; i - - ){
    if ( isPalin ( str, i, n ) == true) dp[i] = 0
    for ( cut = i ; cut < n ; cut ++) {
        if ( isPalindrome ( i, cut){
            min = min ( min, dp[cut+1]);
        }
    }
    dp[i] = 1 + min
}
```