Today's Agenda:- Starting 7:05

① Longest Increasing Subsequence

② Russian Doll Envelopes.

③ Count of All Palindromic Substrings.

④ Palindromic Partitioning.

① Longest Increasing Subsequence (LIS)

Ex: 1 :- {6, 9, 10, 13, 20}. ans = 5.

Ex: 2 :- {13, 6, 2, 1} ans = 1.

Ex: 3: {10, 3, 12, 7, 9, 11, 20, 13, 6, 8}.

ans = 5.

BF:- Consider all the subsequences. Find if they are increasing, store its length & find max length.

$O(2^N * N)$.

0	1	2	3 (i)	4	5	6	7	8	9
10	3	12	7	9	11	20	13	6	8

dp → { 1 1 2 2 . }

Store LIS ending at each index may help.

dp(i) → LIS ending at index i.

Code

```
int dp[N];
dp[0] = 1;        ans = 1;
for (i=1;  i<N;  i++) {
        max = 0;
    for (j=0; j<i; j++) {
        if (arr[j] < arr[i]) {
            max = max(max, dp[j]);
        }
    }
    dp[i] = max + 1;
    ans = max(ans, dp[i]);
}
return ans;
```

T.C → $O(N^2)$

S.C → $O(N)$

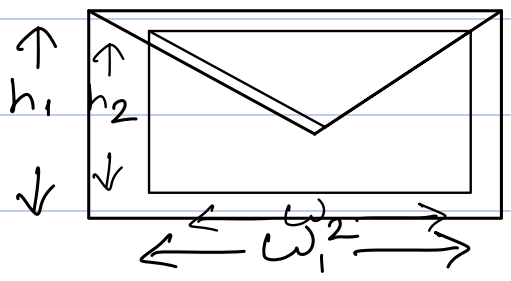
Russian Doll Envelopes

N - different envelopes.

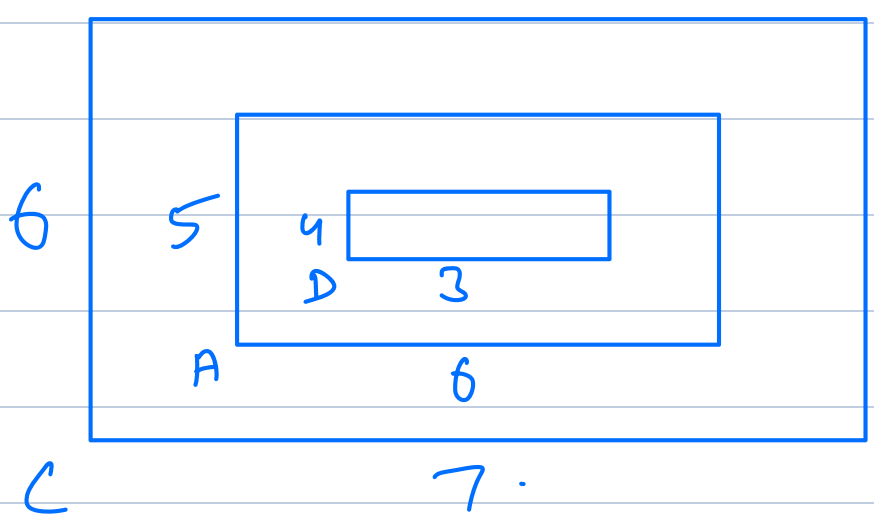
Find max count of envelopes that can be put in a style envelope.

* Rotation is not allowed

	h	w	
A	5	6 - (2)	
B	6	4	
C	6	7 - (1)	$h_1 > h_2$
D	4	3 - (3)	$w_1 > w_2$



ans = 3.



Brain- X

A → 30 ; B = 24 ; C = 42 ; D = 12.

	h	w	
A	2	15.	X
B.	10	1	

How about if I had circles?

↳ Sort the above array & remove all the duplicate.

1 2 2 3 4 4 5

ans = 5.

Ex:2; h → {9, 5, 10, 3, 4, 2}
w → {3, 4, 8, 2, 3, 7}.

↓ sort it based on ht

h → {2, 3, 4, 5, 9, 10}
w → {7, 2, 3, 4, 3, 8}.

Algorithm.

(1) Sort based on the height

(2) Apply LIS on width.

0 1 2 3 4

h = {1, 2, 2, 3, 4}.
w = {3, 5, 6, 2, 9}

dp → {1 2 2 1 3}.

```
int dp[N], ans;
dp[0] = 1;
for (i = 1; i < N; i++) {
    max = 0;
    for (j = 0; j < i; j++) {
        if (wt[j] < wt[i] && ht[j] < ht[i]) {
            max = max(max, dp[j]);
        }
    }
    dp[i] = max + 1;
    ans = max(ans, dp[i]);
}
return ans;
```

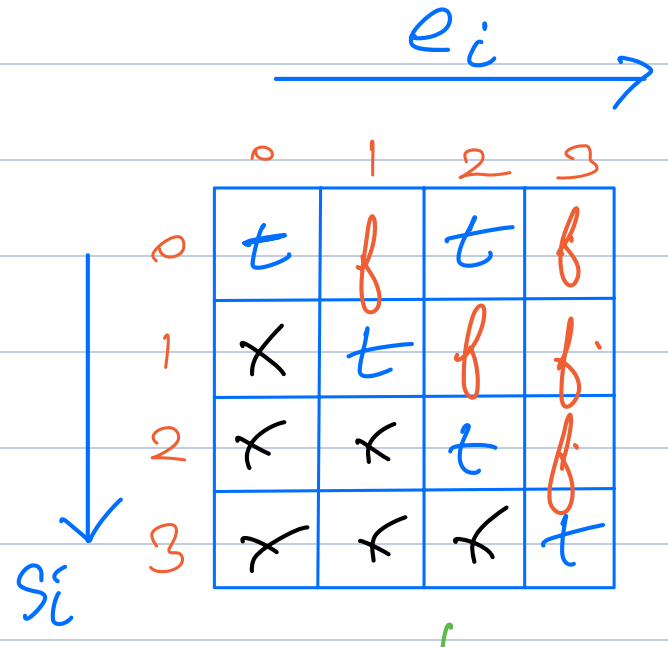
T.C → $O(N^2 + N \log N)$

S.C → $O(N)$.

3. Given a string. For every substring check if that is a palindrome or not.

$S =$ "abac" (0 1 2 3)

a ab aba abac
b ba bac
a ac
c



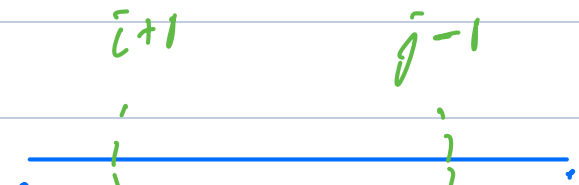
B.F:- Consider all the substrings & iterate on them to check if palindrome or not.

$O(N^2 * N) \simeq O(N^3)$

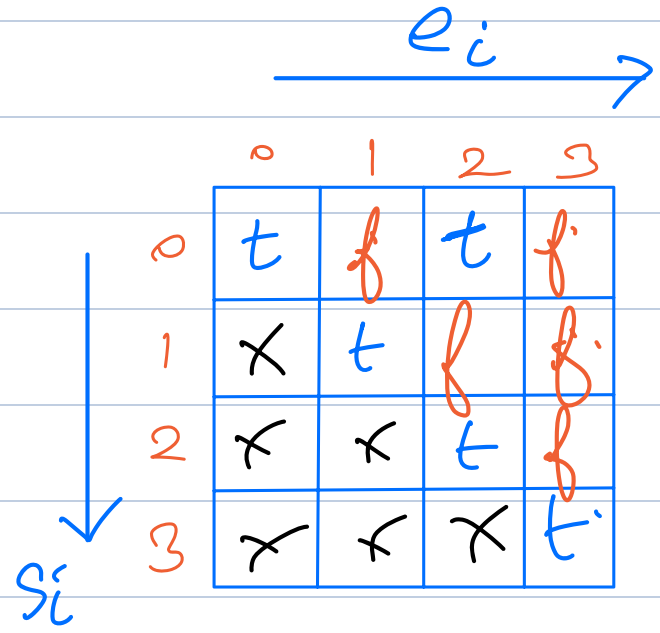
Obsr-

(1) All S.S. of length 1 are palindrome.

(2)




```
if (s[i] == s[j])
  { ans[i][j] = ans[i+1][j-1]; }
else
  ans[i][j] = false;
```



0 1 2 3

a b a c

```
gap = 0;
dp[i][j] = true;
```

	gap=1.	gap=2.
a		
b	ab	aba
c	ba	bac
d.	ac	

gap=3.

abac

[i, j]

[i+1, j-1]

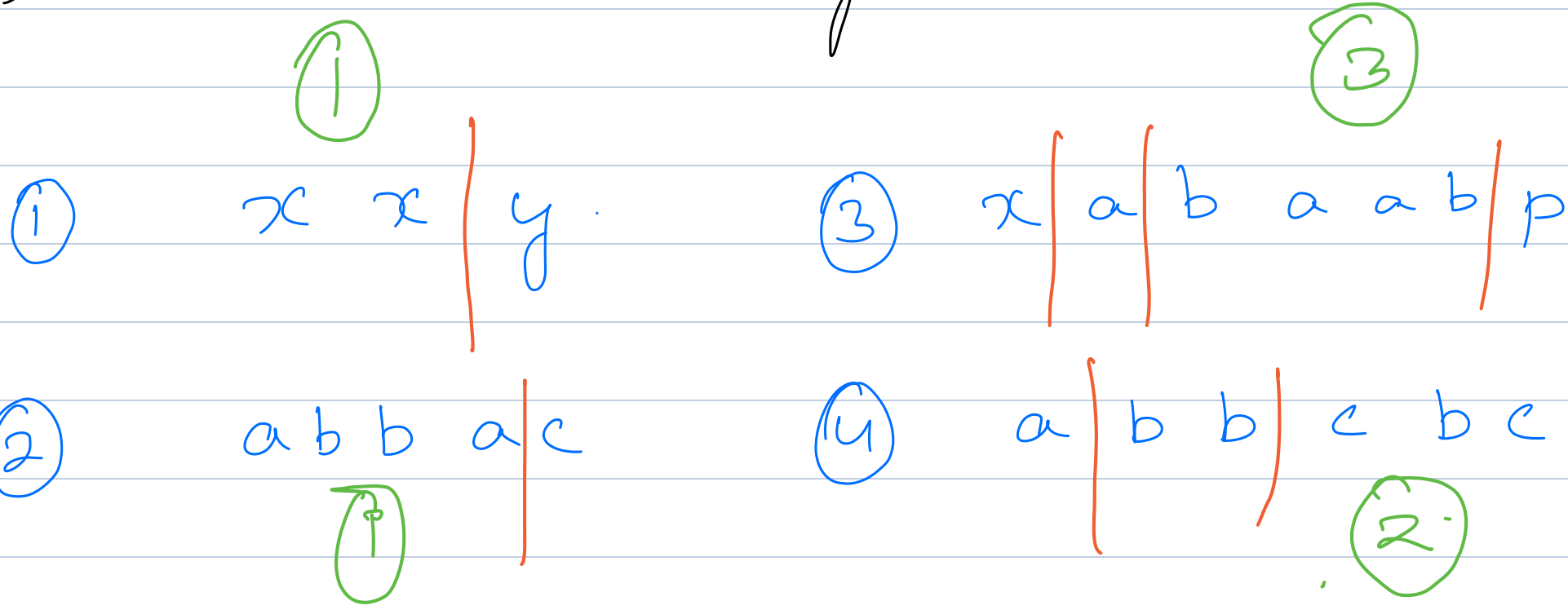
Code

```
boolean dp[N][N];
for (gap = 0; gap < N; gap++) {
    for (i=0, j=gap ; j < N ; i++, j++) {
        if (gap == 0)
            { dp[i][j] = true; }
        else if (gap == 1) {
            dp[i][j] = (str[i] == str[j]);
        }
        else {
            if (str[i] == str[j])
                { dp[i][j] = dp[i+1][j-1]; }
            else
                { dp[i][j] = false; }.
        }
    }
}
return dp;
```

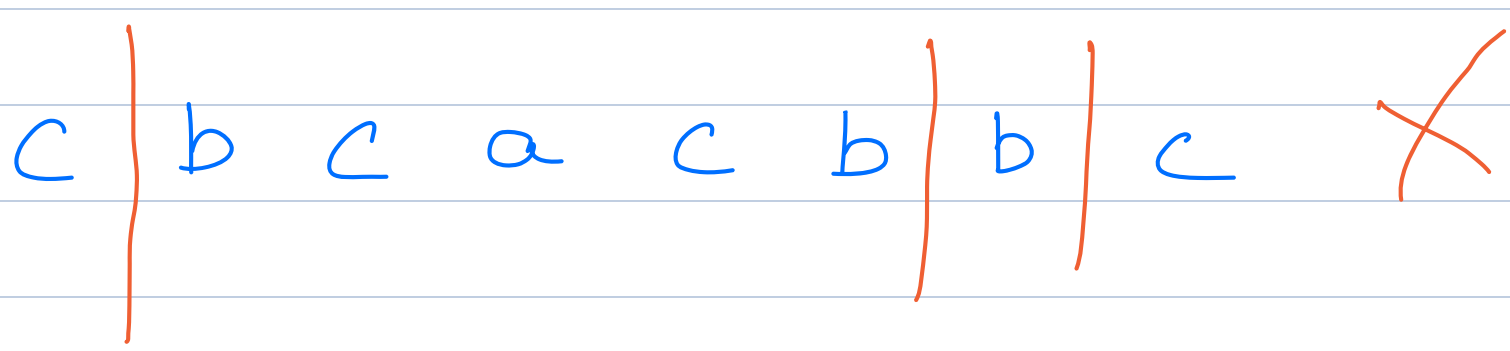
T.C → $O(N^2)$..

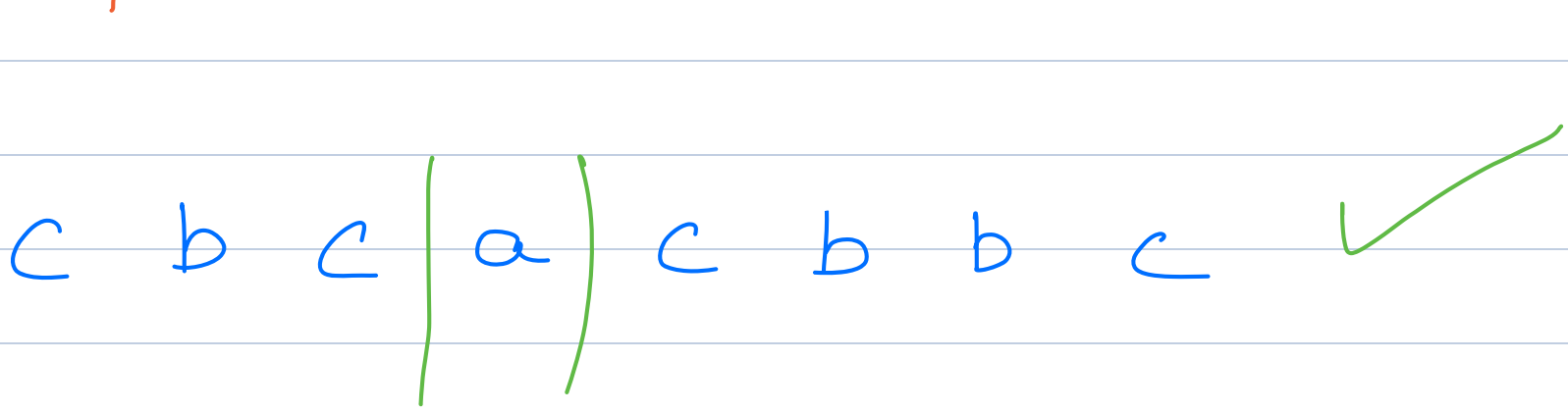
S.C → $O(1)$

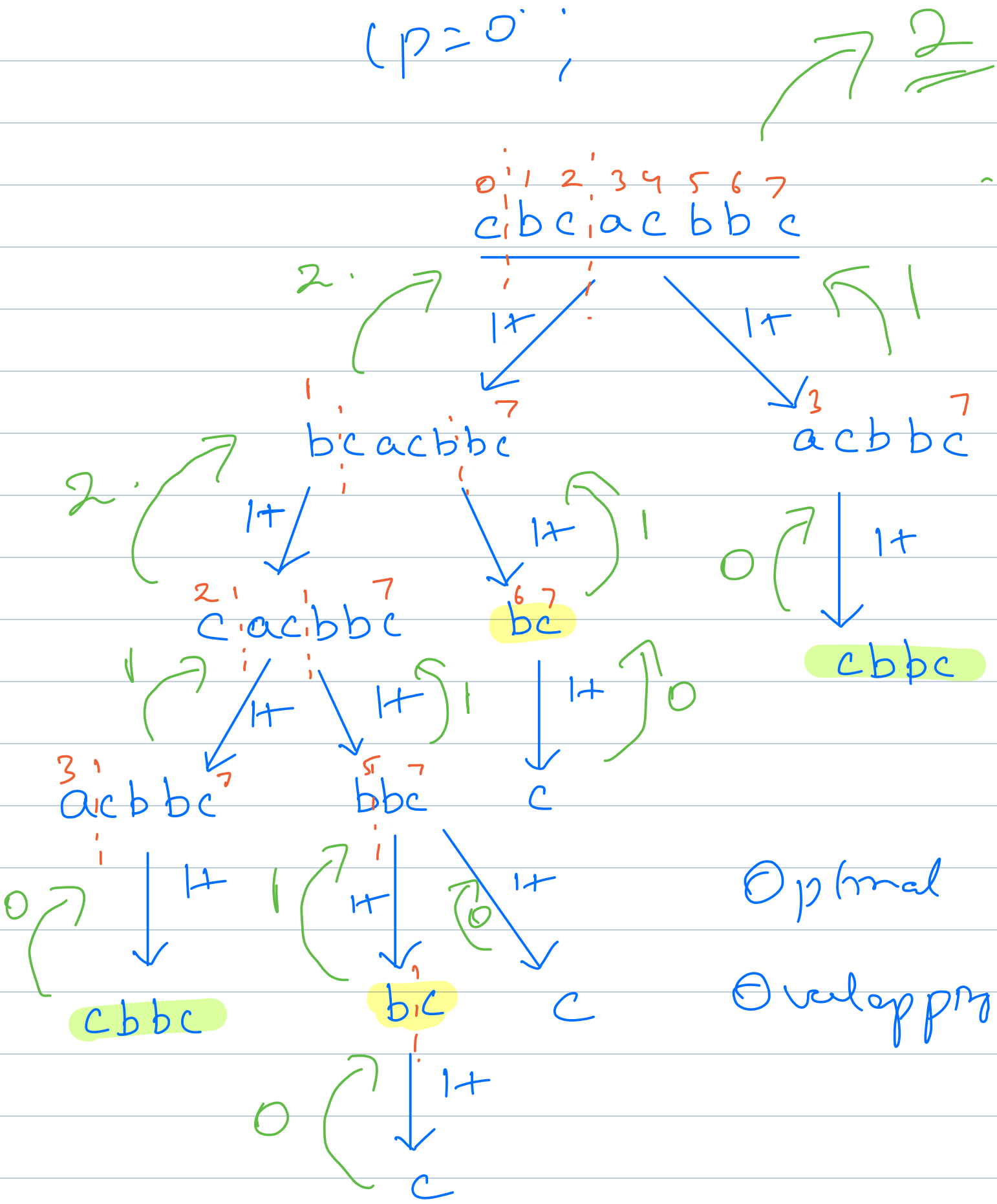
Find min. no of cuts to partition the string such that all partitions are palindrome.



Find the longest palindromic substring & make a cut.







Subproblem → Min cuts to create all palindromic partitions of a substring starting at i & ending at N-1.

dp[i] → Min cuts to create all palindromic partitions in string i to N-1.

For checkPalindrom, refer Q-3.

Code:

```
                                          0    N-1
                                          |    |
int minCuts(str, i, j, dp[N]) {
    if (checkPalindrome(str, i, j)
                        == true) {
        return 0;
    }
    if (dp[i] != -1) {return dp[i];}
        min = INT_MAX;                    N-1
                                          |
    for (cp = i ; cp < j ; cp++) {        |
                                  0   2   |
        if (checkPalindrome(str, i, cp)   |
                        == true) {        |
            min = min(min,                |
                  minCuts(str, cp+1, j)   |
                );                        |
        }                                 |
    }                                     |
    dp[i] = min + 1;                      |
    return dp[i];                         |
}
```

T.C → $O(N^2 + N)$.

S.C → $O(N^2 + N)$.

bottom-up → H.W.