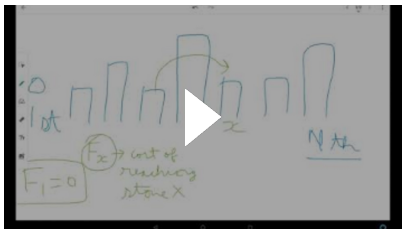
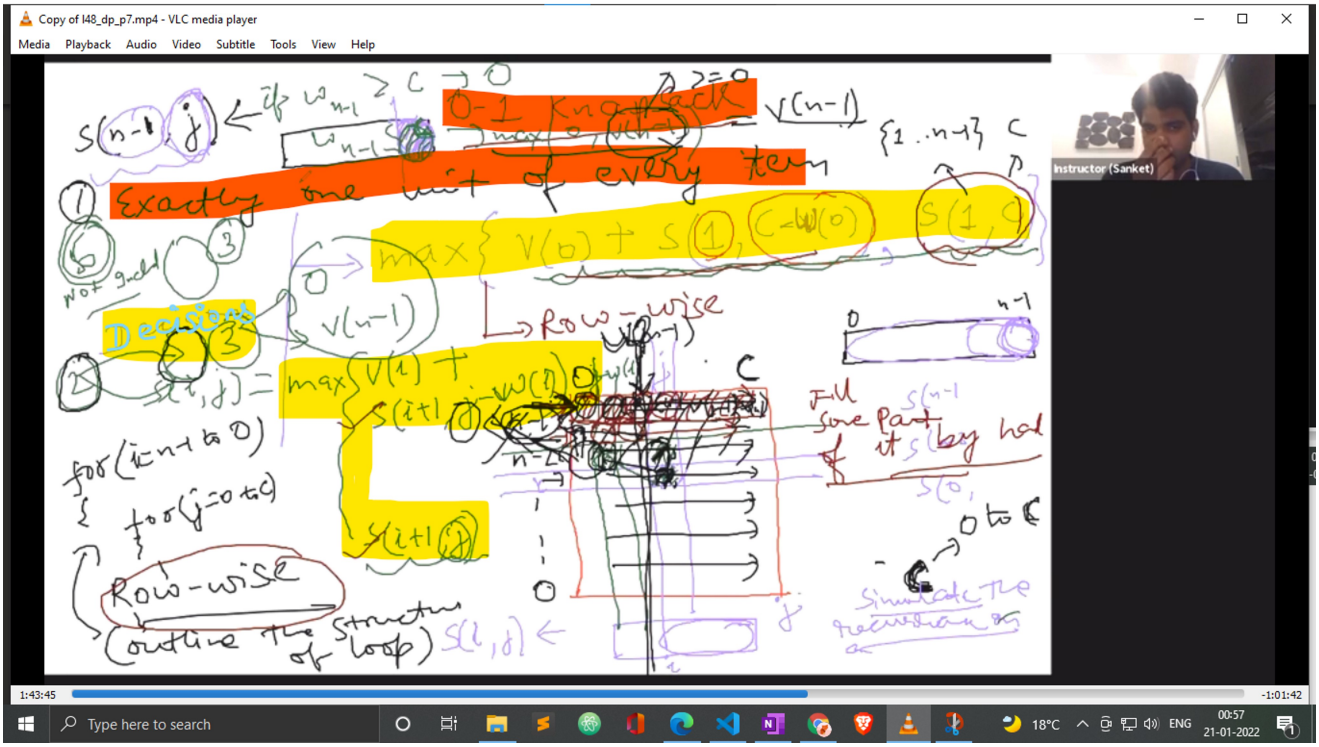
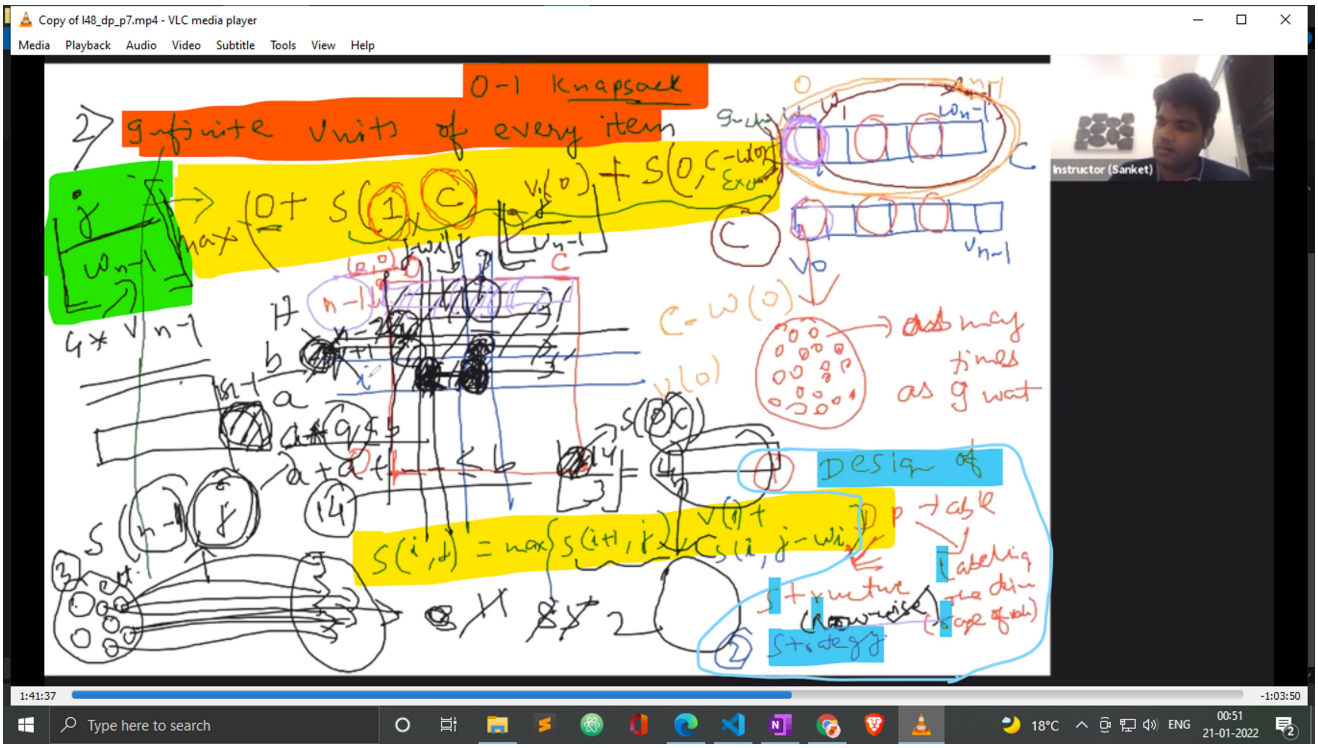
Dynamic Programming

25 October 2021 18:28

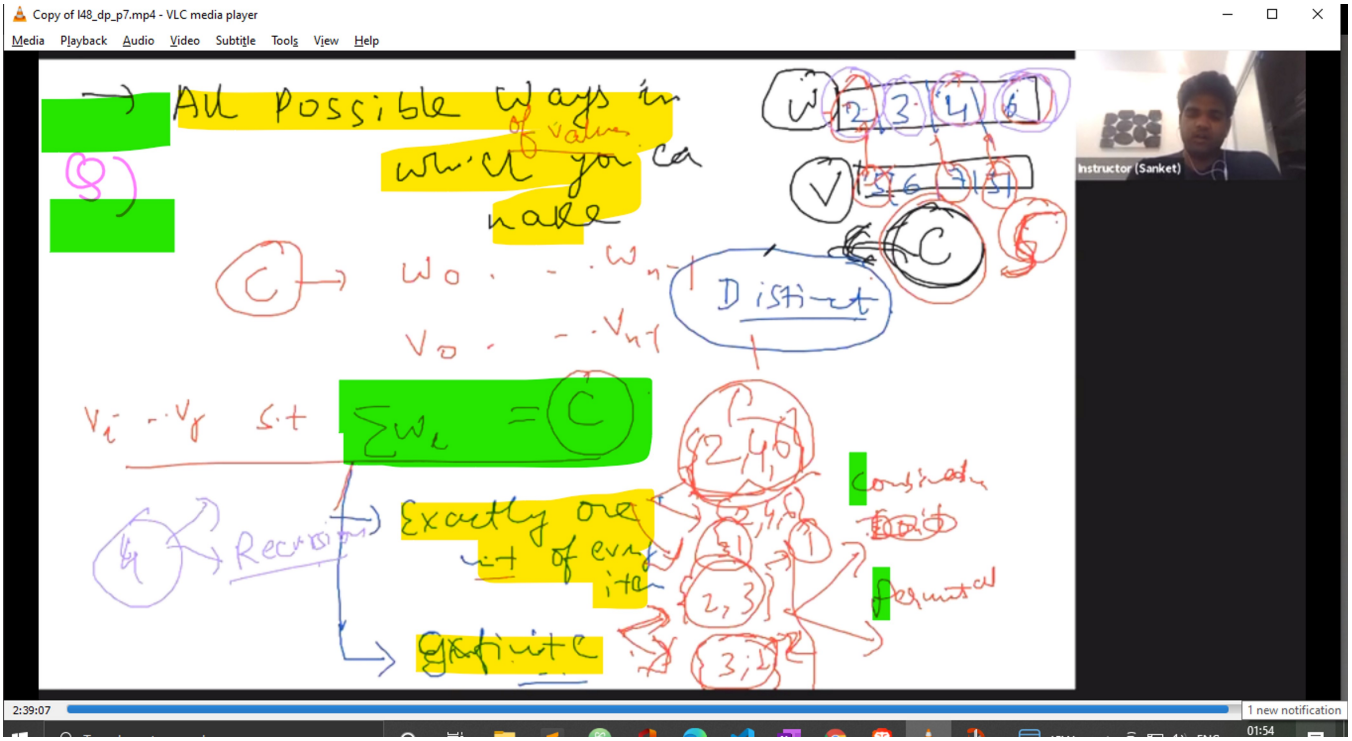
- https://dev.to/nikolaotasevic/dynamic-programming--7-steps-to-solve-any-dp-interview-problem-3870
- https://www.freecodecamp.org/news/demystifying-dynamic-programming-3efafb8d4296/
- Colin galen 2 videos: topic stream
- ★ Errichto (for overview and patterns**): Dynamic Programming lectures
- Utkarsh gupta dp playlist :: Dynamic Programming Hindi







Question::??



==========

Lec 8:

→ DP → 2D in $O(n^2)$

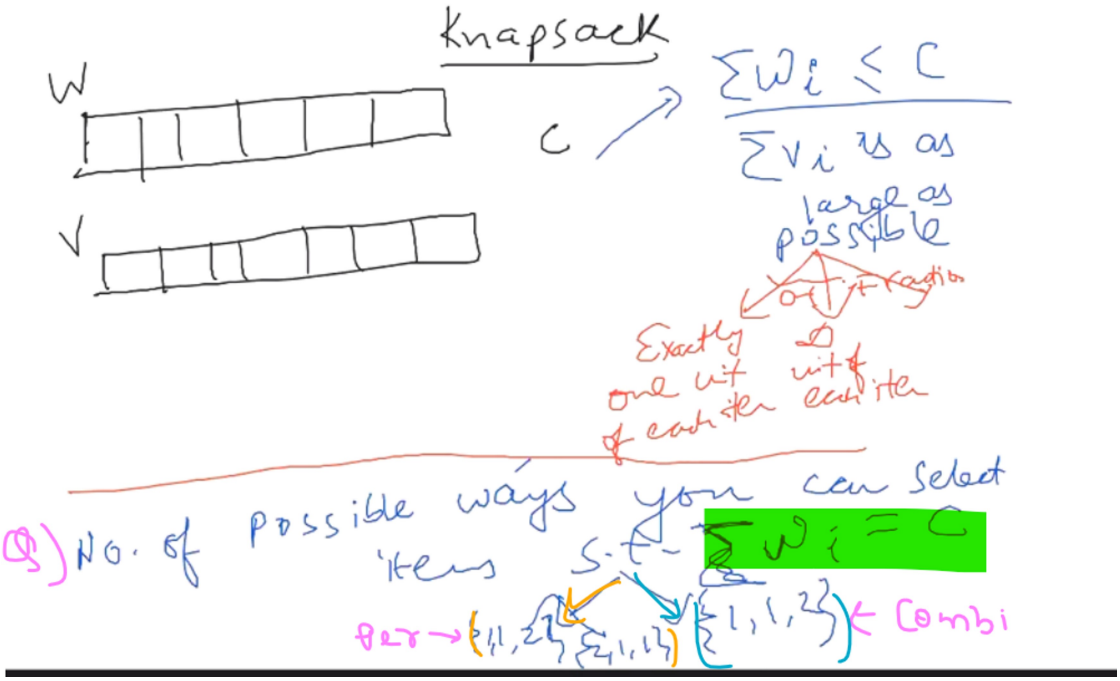
in $O(n^3)$

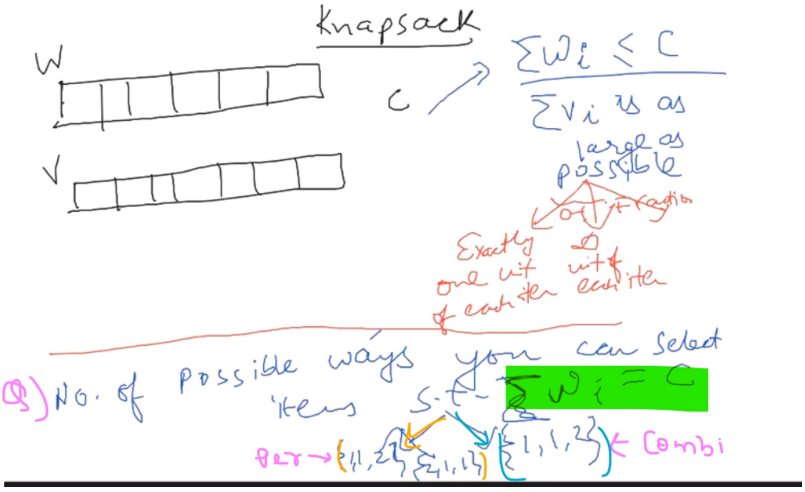
Matrix DP → Recursive Sort?

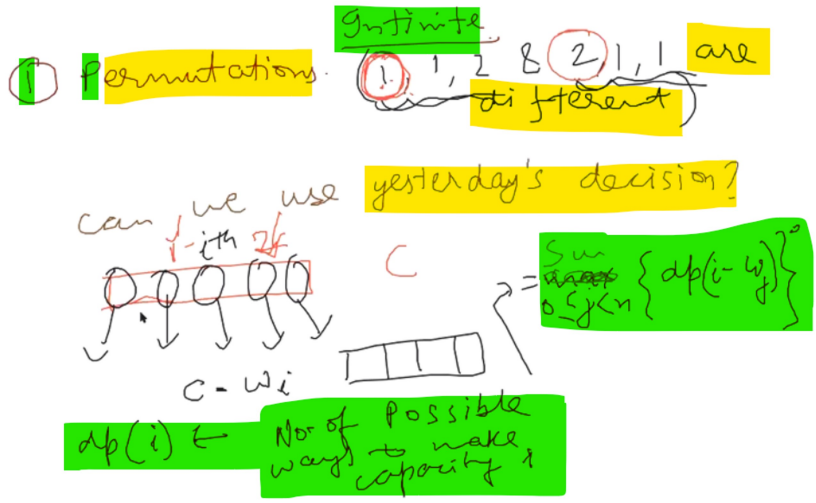
→ Graphs ↔ Traversals / Connectivity

Imp

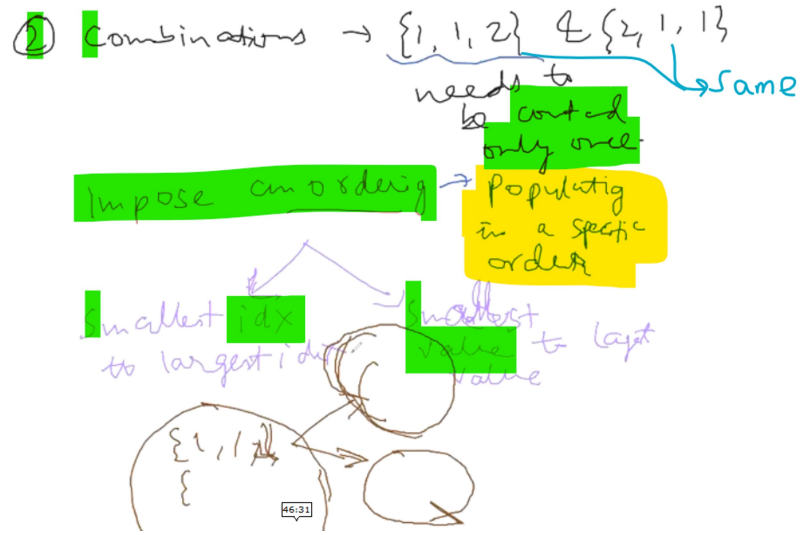
→ Digit DP / DP Bitmasking







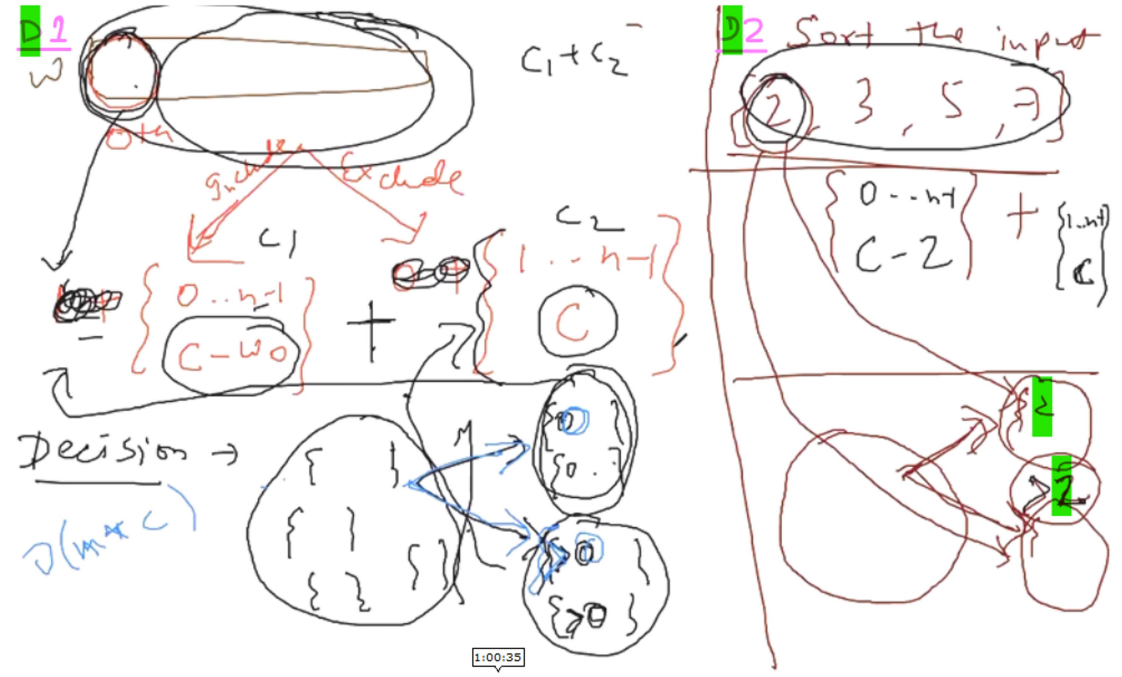
Yesterday's Decision: what's the first item that we could be putting in the bag?/

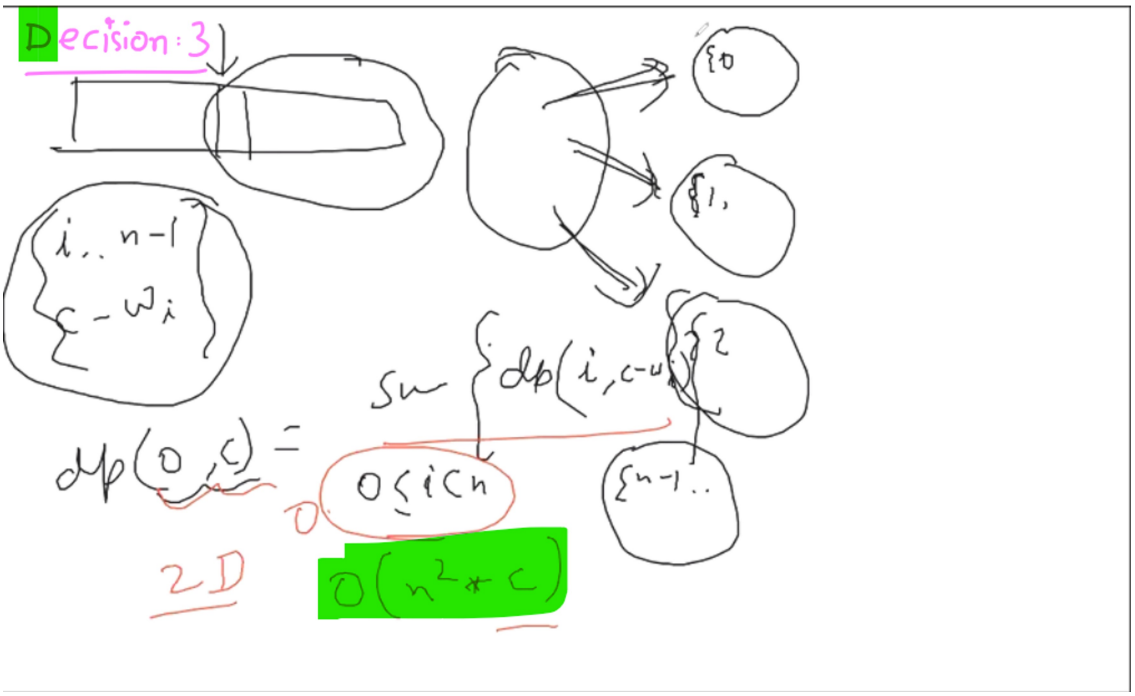


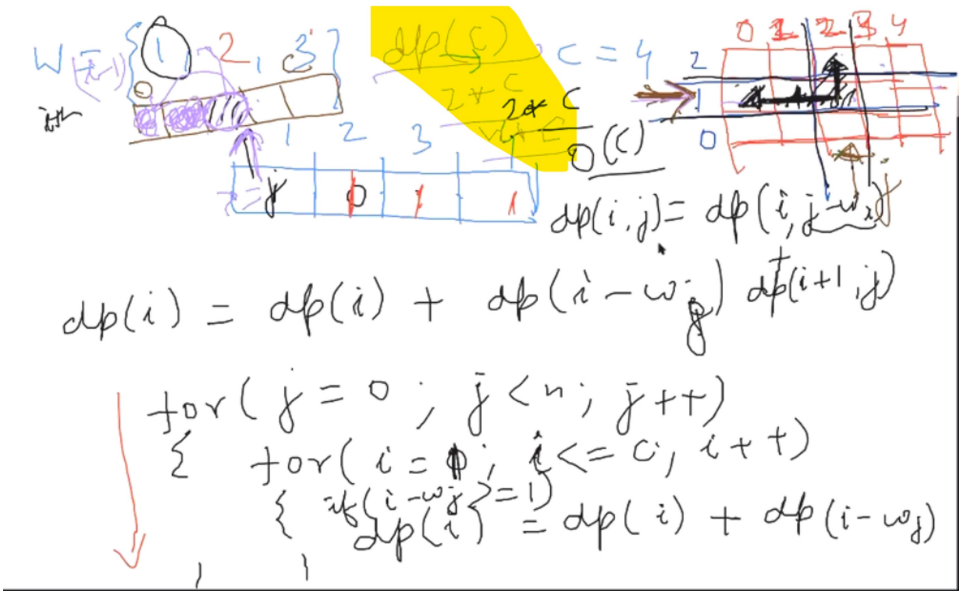
Decision 1: decision on the first element of the solution (*based on index*)
Include- we take the first index element as 1st element in the solution
Exclude- we do not take the first index element as 1st element in the solution

Decision 2: decision on the first element of the solution (*based on value*)
sort the capacity array (some tweak to this decision will make code to work even if there are duplicate elements in the capacity array)
Include- we take the smallest element as first element of the solution
Exclude- we do not take smallest element as the first element in the solution
Branching factor of recursion tree: is 2

Decision 3: decision on the first element of the solution *(based on value)*
Sort the capacity array
We then decide form index 0…n-1 which will be the first unique element to be added in the ans subset, and our subproblem becomes form i+1…n-1, for each 'I' we choose, ans becomes the sum of all possible subproblems ka answer
Branching factor of recursion tree: is n(length of capacity array/no of unique elements in the capacity array)

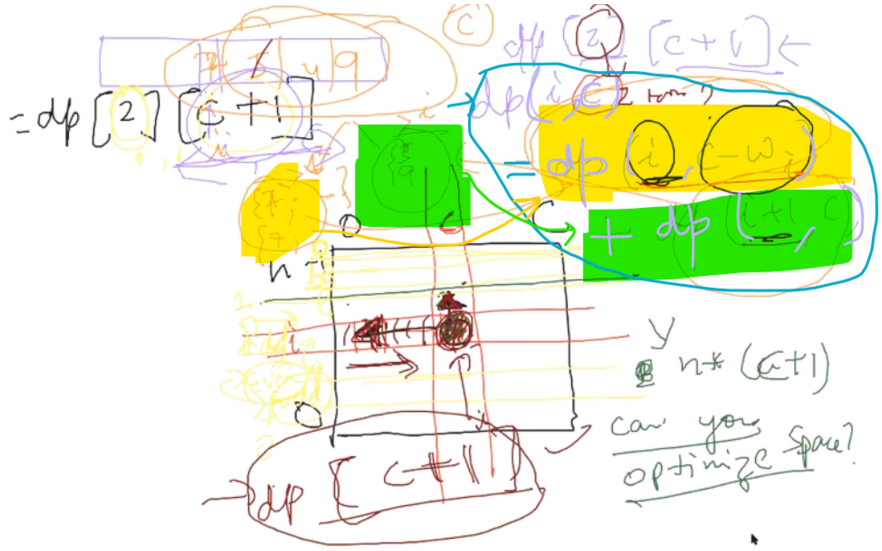

D1
Include
Exclude
C1
C2
C - w0
Decision →
O(n * C)
D2 Sort the input
2, 3, 5, 7
0 .. n-1
C - 2
1 .. n-1
C
1:00:35


Decision: 3
i .. n-1
C - wi
dp(0, C) =
0 ≤ i < n
2D
O(n² * C)



For the above type array filling (upper and left) we can space optimize the dp :
O(C*V) => O(2*C) => O(C)
We won't even be needing 2 rows as we can reuse the existing values to get the new row's dp values by modifying the same row elements.

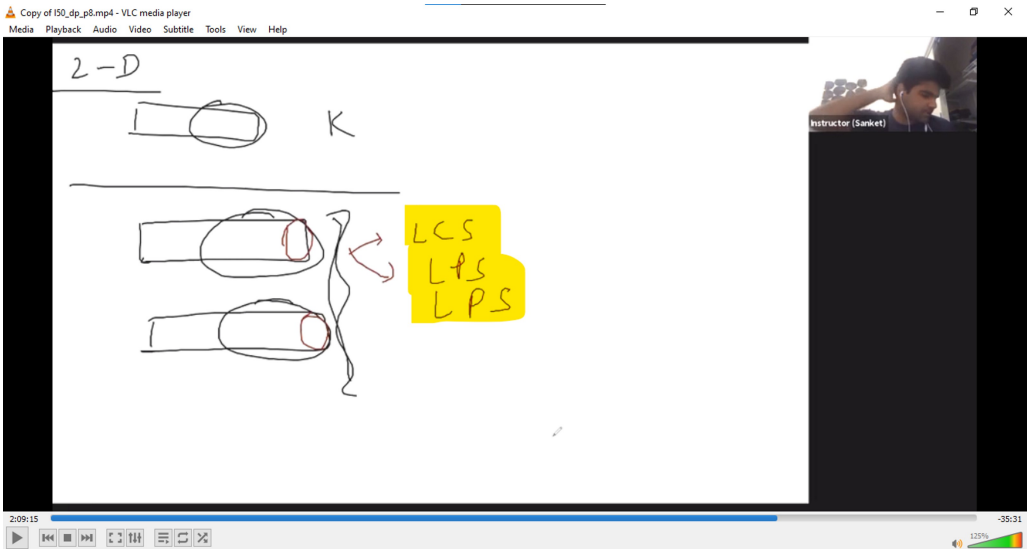
The order of execution of the 2 loops ko opposite krne se we get ans for permutation and combination. Did not get this part??



Story for this decision: dp(I,C)

- One solution will contain all the possible subsets whose 1st element is the element at start index 'i' => subpblm :: dp(i,C-Wi)
- Other solution subset will contain the 1st element as any number after the index 'i' => subpblm :: dp(i+1,C)

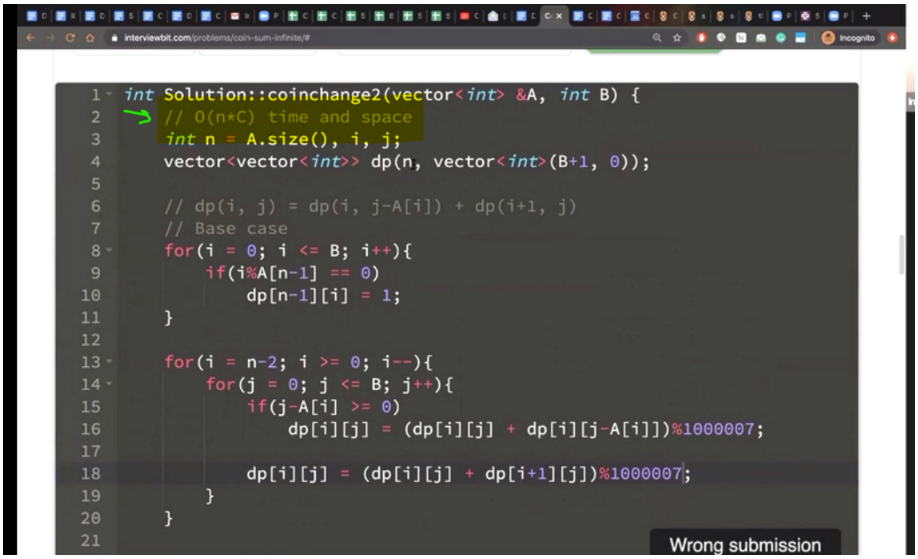
Subproblem is the suffix array for combinations and complete array for permutations!!
Fvf



2d dp:

- 1 suffix array + 1 integer(of certain range) :: knapsack problems and its variants (lec -6 ka mid se, lec7,lec8)
- 2 suffix(/prefix) arrays
    - Longest common subsequence
    - Longest palindromic substring
    - Longest palindromic subsequence

Interviewbit : coin sum infinte



```
    // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
    // Base case
    for(i = 0; i <= B; i++){
        if(i%A[n-1] == 0)
            dp[n-1][i] = 1;
    }

    for(i = n-2; i >= 0; i--){
        for(j = 0; j <= B; j++){
            if(j-A[i] >= 0)
                dp[i][j] = (dp[i][j] + dp[i][j-A[i]])%1000007;

            dp[i][j] = (dp[i][j] + dp[i+1][j])%1000007;
        }
    }

    return dp[0][B];


}
```

Wrong submission has a score penalty. We recommend testing your code before submitting

==========

Same q-> better space complexity

```
    // O(n*C) time and space
    int n = A.size(), i, j;
    vector<vector<int>> dp(2, vector<int>(B+1, 0));

    // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
    // Base case
    for(i = 0; i <= B; i++){
        if(i%A[n-1] == 0)
            dp[(n-1)%2][i] = 1;
    }

    for(i = n-2; i >= 0; i--){
        for(j = 0; j <= B; j++){
            if(j-A[i] >= 0)
                dp[i%2][j] = (dp[(i+1)%2][j] + dp[i%2][j-A[i]])%1000007
            else
                dp[i%2][j] = (dp[(i+1)%2][j])%1000007;
        }
    }

    return dp[0%2][B];
```

o(2*n) space

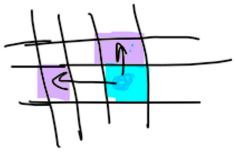
- Here we space optimized the above solution to o(2*n) , since we need only the current row and the row just above...
- Also we use ()%2 to avoid extra time of reassigning/copying the prev row to next row

==========

```
    // O(n*C) time and space
    int n = A.size(), i, j;
    vector<int> dp(B+1, 0);

    // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
    // Base case
    for(i = 0; i <= B; i++){
        if(i%A[n-1] == 0)
            dp[i] = 1;
    }

    for(i = n-2; i >= 0; i--){
        for(j = 0; j <= B; j++){
            if(j-A[i] >= 0)
                dp[j] = (dp[j] + dp[j-A[i]])%1000007;
            else
                dp[j] = (dp[j])%1000007;
        }
    }

    return dp[B];
```

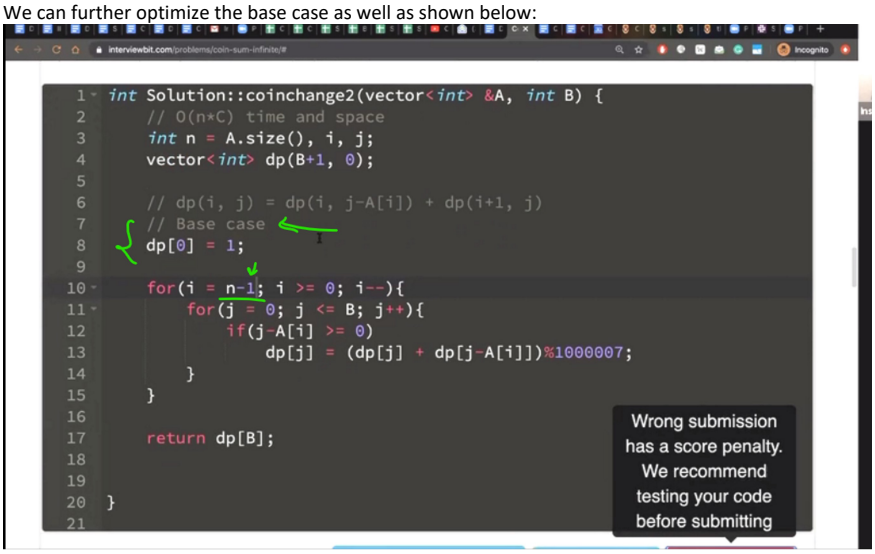
O(n) Space

- O(n) solution since the nature/direction in ehich we fill the dp table is and the previously required values can be obtained from the same array if we fill in left to right ,top to down order!!
- We casn simply remove the first dimension here and the things will work out



==========

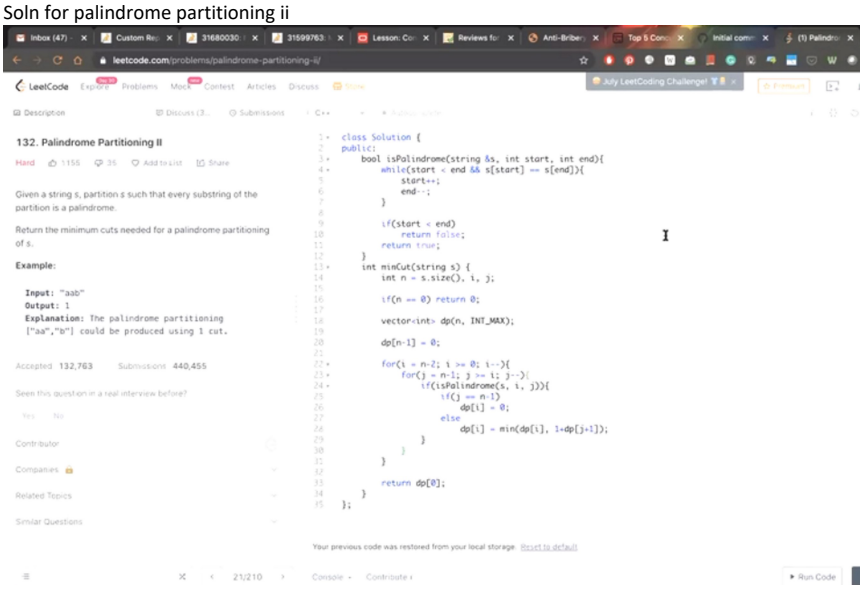
We can further optimize the base case as well as shown below:



```
int Solution::coinchange2(vector<int> &A, int B) {
    // O(n*C) time and space
    int n = A.size(), i, j;
    vector<int> dp(B+1, 0);

    // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
    // Base case
    dp[0] = 1;

    for(i = n-1; i >= 0; i--){
        for(j = 0; j <= B; j++){
            if(j-A[i] >= 0)
                dp[j] = (dp[j] + dp[j-A[i]])%1000007;
        }
    }

    return dp[B];

}
```

Wrong submission has a score penalty. We recommend testing your code before submitting

==========

- **Q) Buy and sell stocks -> version:cool down** -> can be done by o(n) also (approach uising dp):: https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/
- **Q) Palindrome partioning ii**
  - Soln for palindrome partitioning ii
  - 

```
class Solution {
public:
    bool isPalindrome(string &s, int start, int end){
        while(start < end && s[start] == s[end]){
            start++;
            end--;
        }

        if(start < end)
            return false;
        return true;
    }
    int minCut(string s) {
        int n = s.size(), i, j;

        if(n == 0) return 0;

        vector<int> dp(n, INT_MAX);

        dp[n-1] = 0;

        for(i = n-2; i >= 0; i--){
            for(j = n-1; j >= i; j--){
                if(isPalindrome(s, i, j)){
                    if(j == n-1)
                        dp[i] = 0;
                    else
                        dp[i] = min(dp[i], 1+dp[j+1]);
                }
            }
        }

        return dp[0];
    }
};
```

==========

Lec 9:

Not a dp qn:

Goldman Sachs interview question::

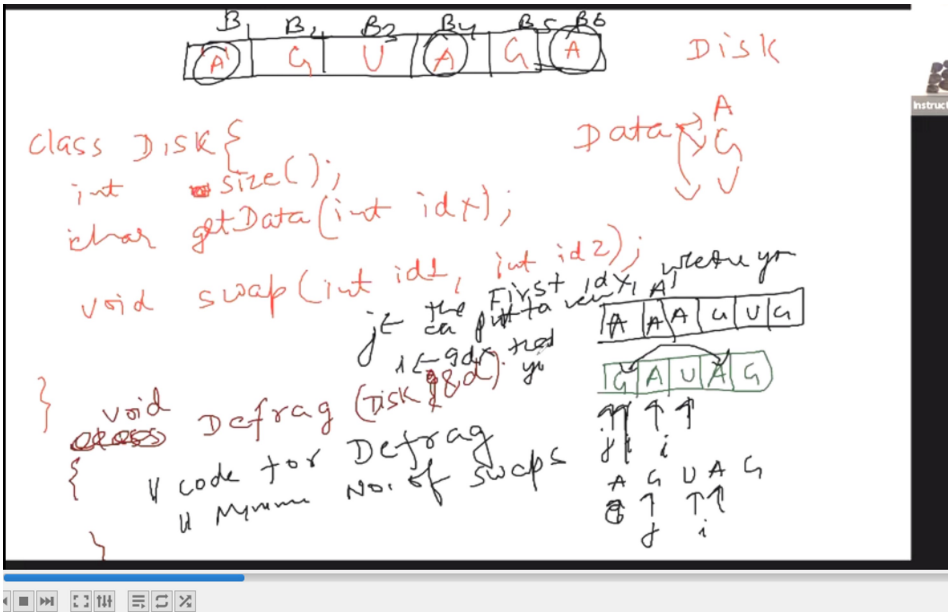
Dynamic Programm...

Audio recording started: 12:18 24 January 2022

Audio recording started: 12:17 24 January 2022

Eg: aguaga
Ans: 2

eg : g a u a g
Ans: 1



Solution:

Dynamic Programm...

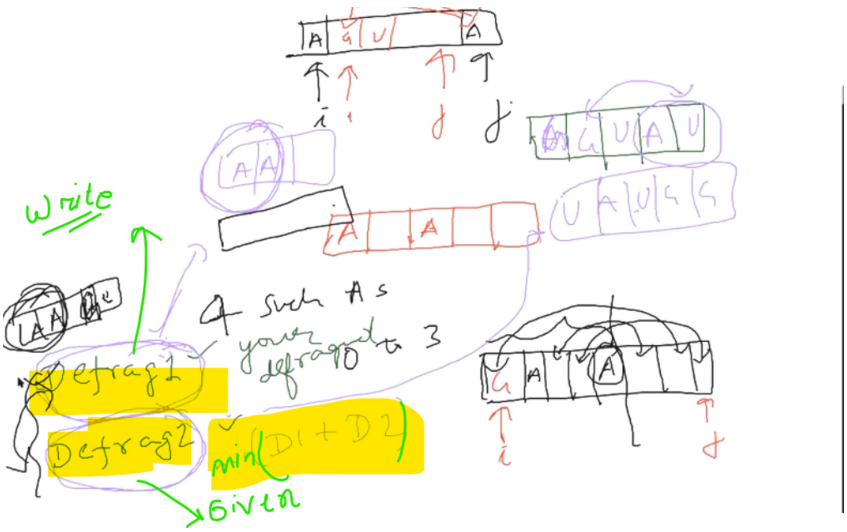
Audio recording started: 12:22 24 January 2022

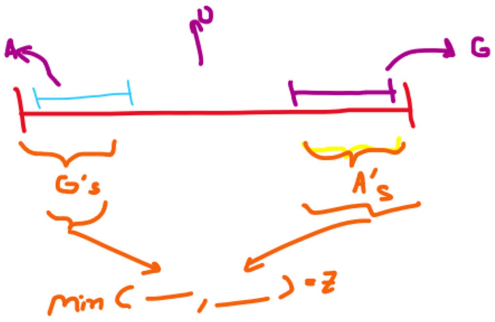
4 metrics that interviewer were looking were:
A) preprocessing(count the no of A's)
B) 2 pointers (they were expecting all would be able to say that)
C) j won't be simply j++ but it will be the next non A character ka index => a loop ofr that also
D) don't need to swap elements that are A and in the index range [0 ... (count_A - 1)]
E) clean code without messing up the variables

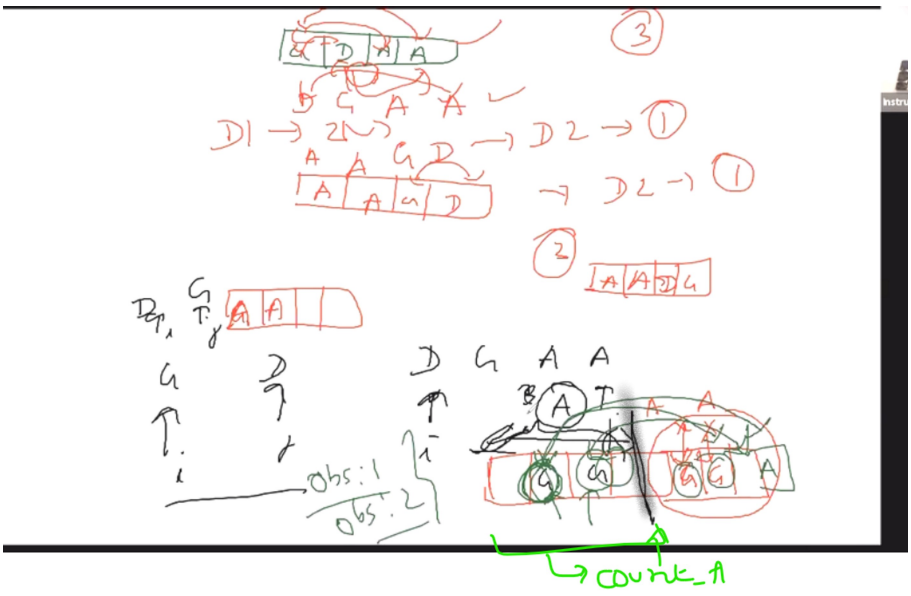
Follow up qn:

Dynamic Programm...

Audio recording started: 13:19 24 January 2022







Soln::
Observtion1: the G's that are in the range [0 … count_A-1] , we can use those to minimize sum
Observation 2:: the G's that appear in the above range can be we swapped greedily with the right most A 's

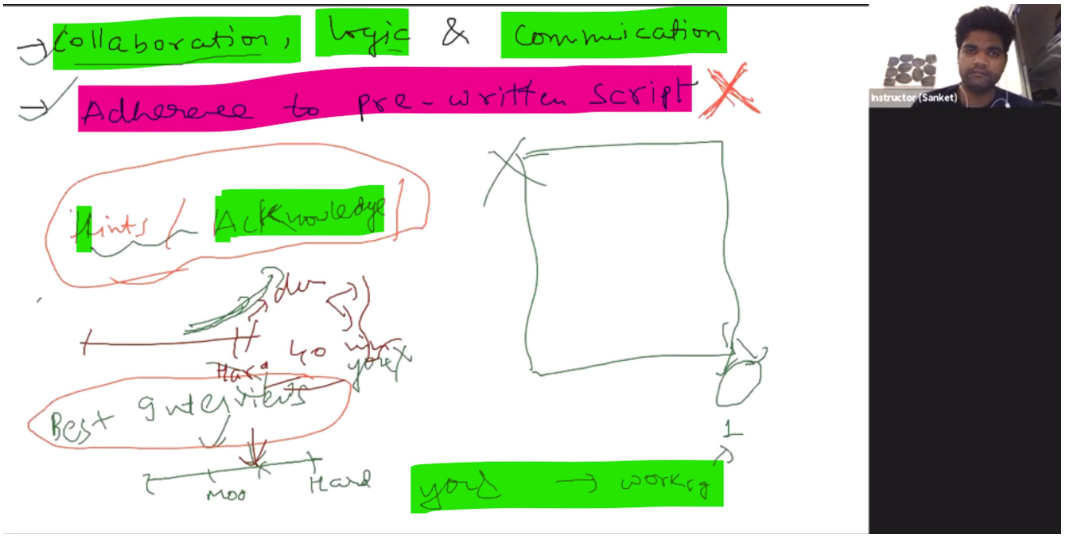
To get an ago for this you can combine the above 2 observations!

- find count of a's in the array
- In first pass swap all g's in the range [0 … count_A-1] , with the rightmost A's in the array
- In next pass swap if there are any D's in the above range then swap them with the a's outside the range [0 … count_A-1]

==========

Interview pointers to keep in mind ::

This is what the interviewer usually looks for , they are given a list by the company usually but this is what it sums up to eventually::



Interview tips:

Whats a good interview::

Dynamic Programm…

Dynamic Programm…

Audio recording started: 14:24 24 January 2022

Kaise prep kar skata hai for these interviews:

Dynamic Programm…

Audio recording started: 14:25 24 January 2022

==========

Q) edit distance: https://leetcode.com/problems/edit-distance/

leetcode.com/problems/edit-distance/

Apps | Building Modern W... | Guide to write an ar... | Workspace | Log in | Innovation... | Problem

LeetCode Explore Problems Interview Contest Discuss Store

Description | Solution | Discuss (999+) | Submissions

72. Edit Distance

Hard 7481 87 Add to List Share

Given two strings `word1` and `word2`, return *the minimum number of operations required to convert* `word1` *to* `word2`.

You have the following three operations permitted on a word:

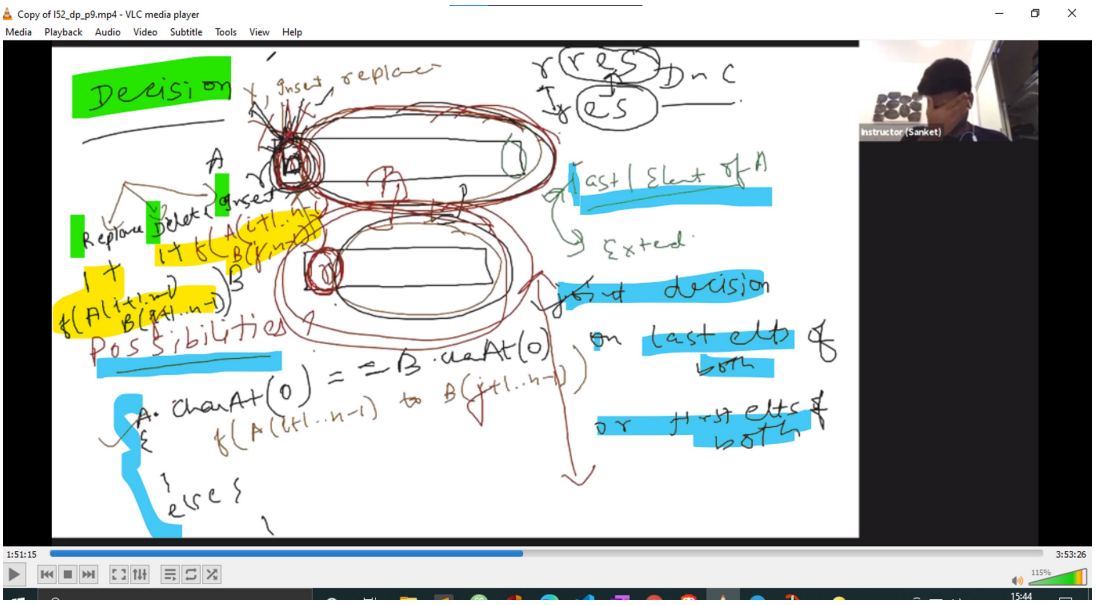
- Insert a character
- Delete a character
- Replace a character

Example 1:

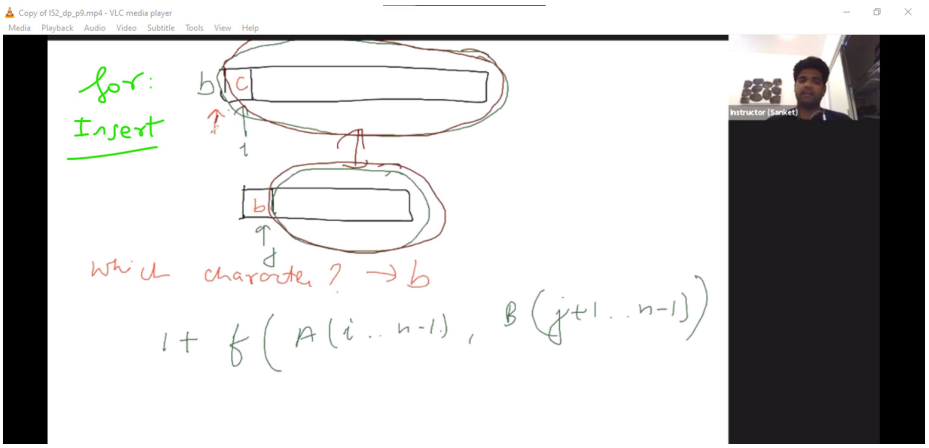
```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

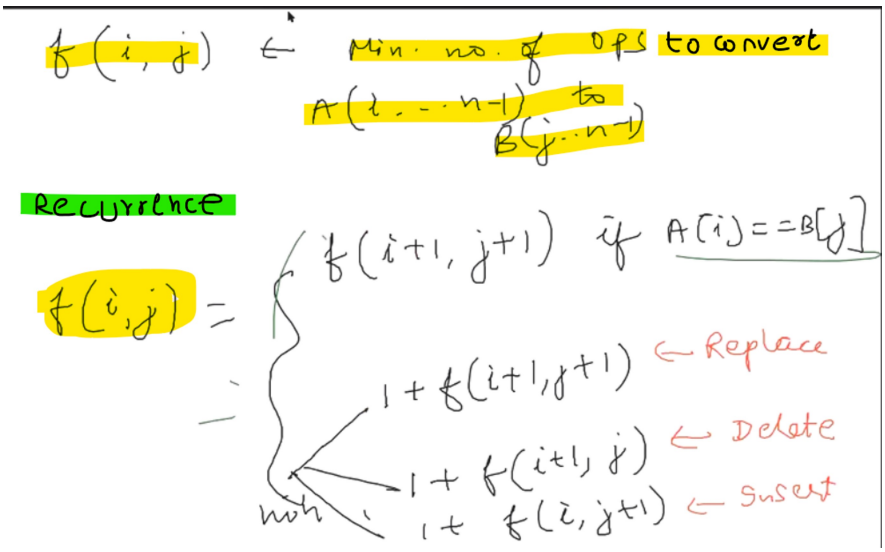
Example 2:

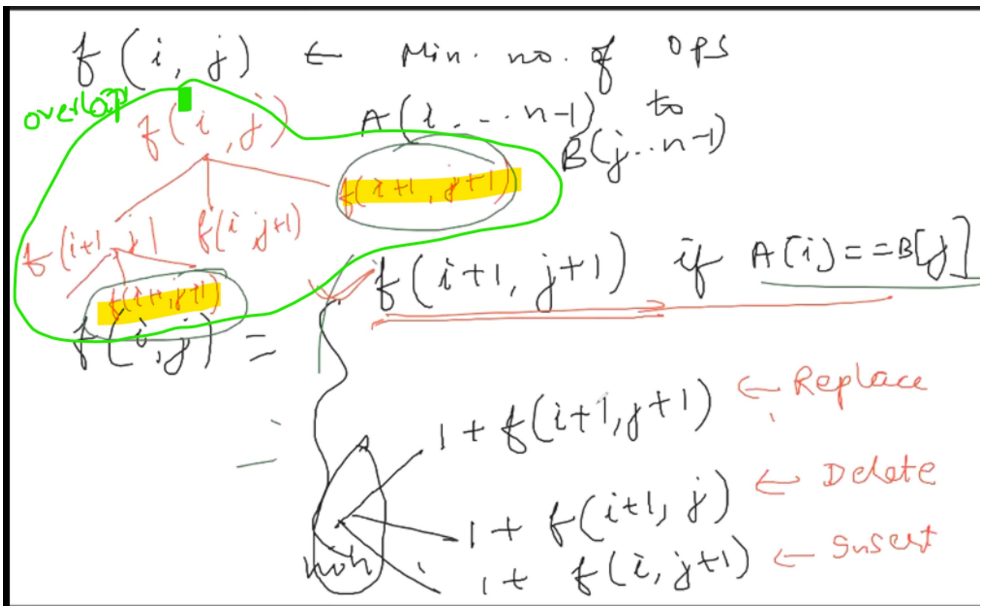
```
Input: word1 = "intention", word2 = "execution"
Output: 5
```



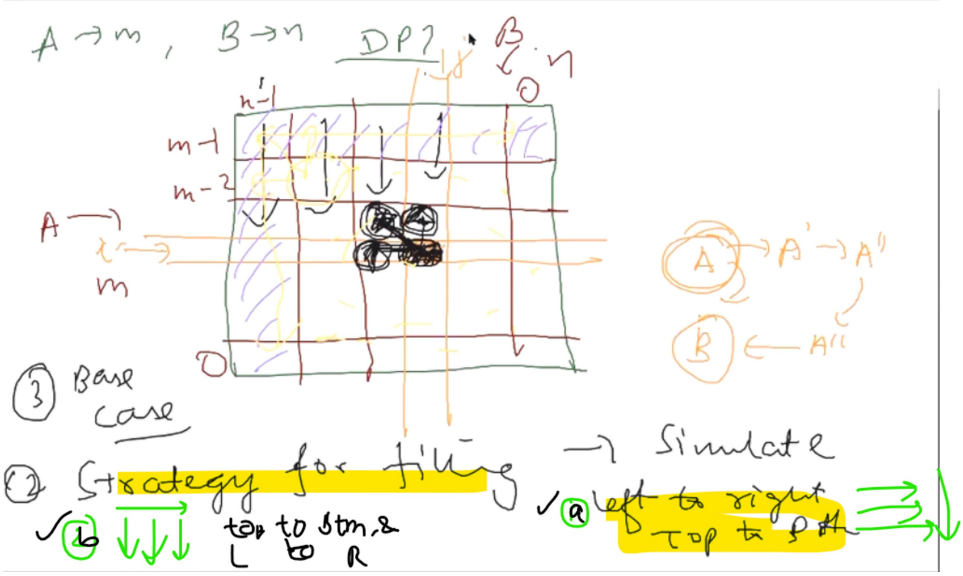
We take **a joint decision** on the last element of word1 and word2







We have overlapping subproblems -> dp as well



Base cases:

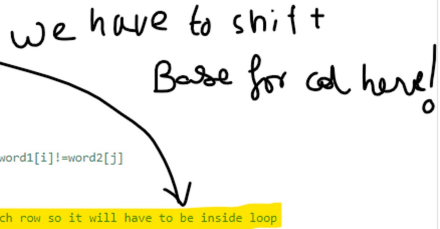
- Most safe way is to fill the left most column and the topmost row (for above dp table design)

Code: prblm: edit distance :: https://leetcode.com/problems/edit-distance/

```cpp
class Solution {
public:
    int minDistance(string word1, string word2) {
        //1->2
        int n=word1.size(),m=word2.size(),i,j;
        //dp(i,j) <- no of ways to convert word1[i ... n-1] to word2[j ... m-1]
        vector<vector<int>> dp(n+1,vector<int>(m+1,0)); //dp(0,0) <- ans

        //Base Case
        //fill the buffer/padded row and column
        for(j=0;j<=m;j++){// for i=n => word1 is empty
            dp[n][j]=m-j; //only insertion operations
        }
        for(i=0;i<n;i++){ // for j=m => word2 is empty
            dp[i][m]=n-i; //only deletion operations
        }

        //Recurrence Relation
        //dp(i,j) = dp(i+1,j+1) ; if word1[i]==word2[j]
        //        =  min(1+dp(i,j+1),1+dp(i+1,j),1+dp(i+1,j+1) )
        //           for insert,delete & replace operations resepectively, if word1[i]!=word2[j]

        //fill from right  to left and bottom to up
        for(i=n-1;i>=0;i--){
            for(j=m-1;j>=0;j--){
                if(word1[i]==word2[j]){
                    dp[i][j]=dp[i+1][j+1];
                }
                else{
                    dp[i][j]= 1 + min(dp[i][j+1],min(dp[i+1][j],dp[i+1][j+1]));
                }
            }
        }
        return dp[0][0];

        // for(i=0;i<=n;i++){
        //     for(j=0;j<=m;j++)
```

Space optimized:

```cpp
class Solution {
public:
    int minDistance(string word1, string word2) {
        //1->2
        int n=word1.size(),m=word2.size(),i,j;
        //dp(i,j) <- no of ways to convert word1[i ... n-1] to word2[j ... m-1]
        vector<vector<int>> dp(2,vector<int>(m+1,0)); //dp(0,0) <- ans

        //Base Case
        //fill the buffer/padded row and column
        for(j=0;j<=m;j++){// for i=n => word1 is empty
            dp[n%2][j]=m-j; //only insertion operations
        }
        // for(i=0;i<2;i++){ // for j=m => word2 is empty
        //     dp[i%2][m]=n-i; //only deletion operations
        // }

        //Recurrence Relation
        //dp(i,j) = dp(i+1,j+1) ; if word1[i]==word2[j]
        //        =  min(1+dp(i,j+1),1+dp(i+1,j),1+dp(i+1,j+1) )
        //           for insert,delete & replace operations resepectively, if word1[i]!=word2[j]

        //fill from right  to left and bottom to up
        for(i=n-1;i>=0;i--){
            dp[i%2][m]=n-i; // **NOTE: we have to update the 1st column for each row so it will have to be inside loop
            for(j=m-1;j>=0;j--){
                if(word1[i]==word2[j]){
                    dp[i%2][j]=dp[(i+1)%2][j+1];
                }
                else{
                    dp[i%2][j]= 1 + min(dp[i%2][j+1],min(dp[(i+1)%2][j],dp[(i+1)%2][j+1]));
                }
            }
        }
        return dp[0][0];//NOT: ans will always be at 0,0 only
        // n3,1 -> n2,0 -> n1,1 -> n0,0

        // for(i=0;i<=n;i++){
        //     for(j=0;j<=m;j++)
        //         cout<<dp[i][j]<<" ";
```



Note: Edit distance (above problem) has a lot of applications!
Edit distance is intuitively giving us how far is word1 from word2 ? Metric used is the no of operations to convert word1 to word2
Applications of Edit Distance:

- ***Auto correct*** (map a word1 to nearest word (word2) in dictionary)
- ***NLP / IR*** (information retrieval)
    - Eg its used in google search ka : Did you mean ____ ?
    - Searching algorithms ka similar words
- ***Genome sequencing*** (**DNA matching algorithm** uses **edit distance** to compare 2 dna's (/dna strands, which are nothing but string representation of genes))
- Plagiarism checks could also be done using this edit distance

==========

Longest common subsequence : https://leetcode.com/problems/longest-common-subsequence/

1143. Longest Common Subsequence

Medium 5401 62 Add to List Share

Given two strings `text1` and `text2`, return *the length of their longest* ***common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, `"ace"` is a subsequence of `"abcde"`.

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its
length is 3.
```

Example 2:

```cpp
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        // Space Optimized :: O(2*n)
        int n=text1.length(),m=text2.length(),i,j;
        //combinatorical => DnC
        // Recursive Reln
        // f(i,j) <- lcs of text1(i ... n-1) & text2(j .. m-1)

        // f(i,j) = 1 + f(i+1,j+1) ;if(text1[i]==text2[j])
        //        = max(f(i,j+1),f(i+1,j)) ; otherwise
        // repeating subpblms => dp
        vector<vector<int>> dp(2,vector<int> (m+1,0));
        //Base Case
        // Padding mein : last row and last col will be 0
        //(since,if either text1 or text2 is empty => length of lcs=0)

        // for(i=0;i<n;i++){
        //     //j=m => text2 is empty
        //     dp[i][m]=0;
        // }
        // for(j=0;j<m;j++) dp[n][j]=0;

        for(i=n-1;i>=0;i--){
            for(j=m-1;j>=0;j--){
                if(text1[i]==text2[j]) dp[i%2][j]=1+dp[(i+1)%2][j+1];
                else dp[i%2][j]=max(dp[(i+1)%2][j],dp[i%2][j+1]);
            }
        }
        return dp[0][0];

    }
};
```

Your previous code was restored from your local storage. Reset to default

Intuition

LCS is a well-known problem, and there are similar problems here:

- 1092. Shortest Common Supersequence
- 1062. Longest Repeating Substring
- 516. Longest Palindromic Subsequence

From <https://leetcode.com/problems/longest-common-subsequence/discuss/348884/C%2B%2B-with-picture-O(nm)>

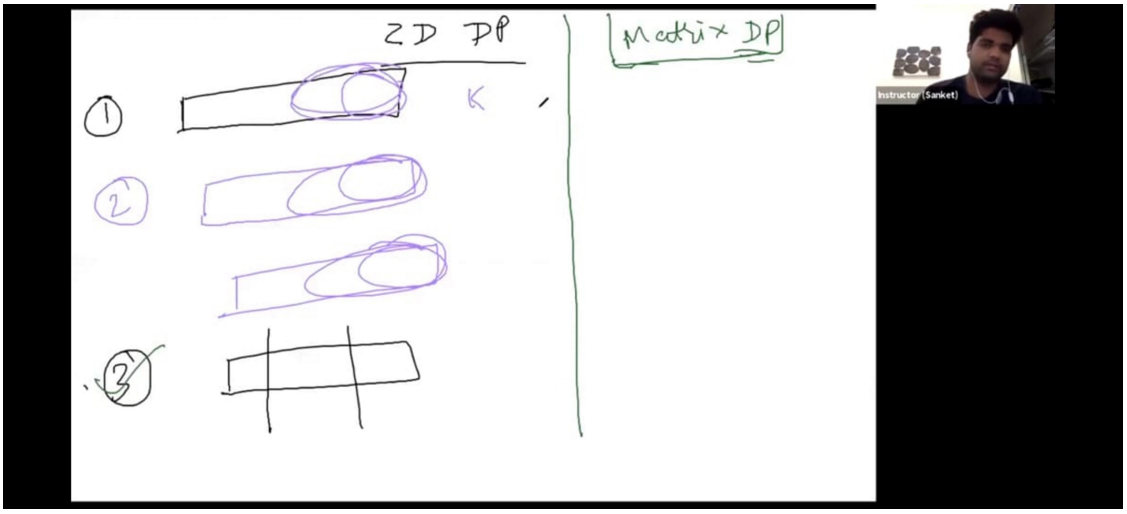
Similar Questions

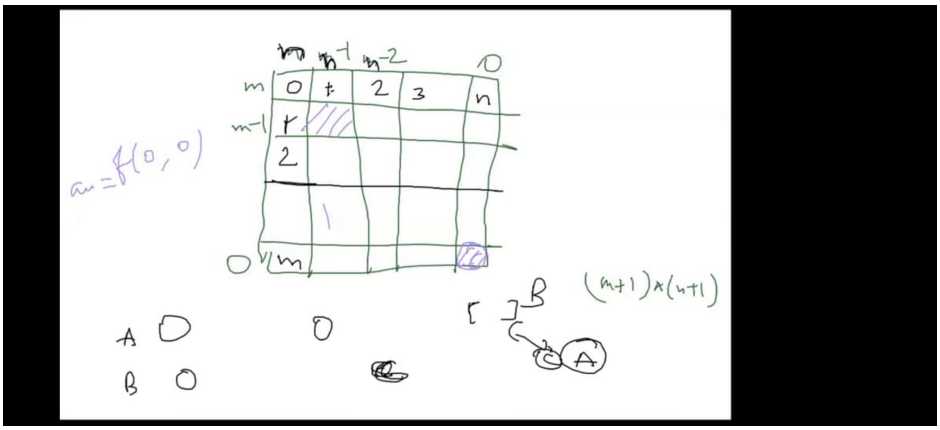
Longest Palindromic Subsequence	Medium
Delete Operation for Two Strings	Medium
Shortest Common Supersequence	Hard

- Distinct subsequences problm (sanket sir)

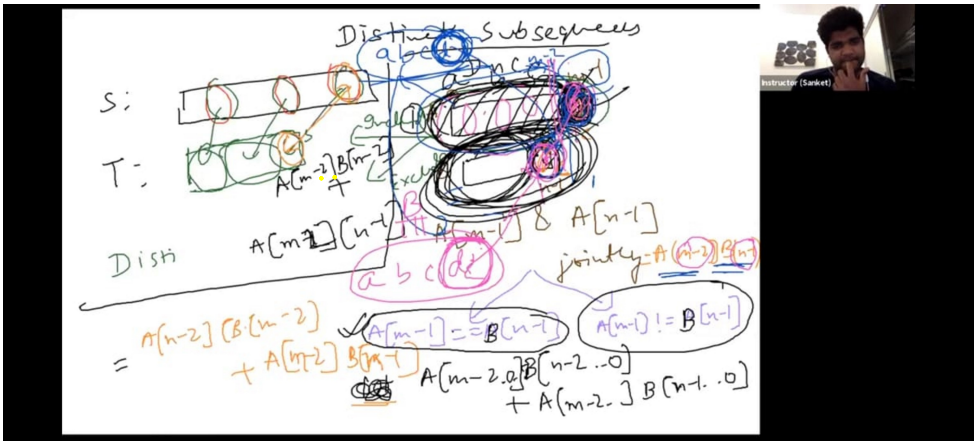
==========

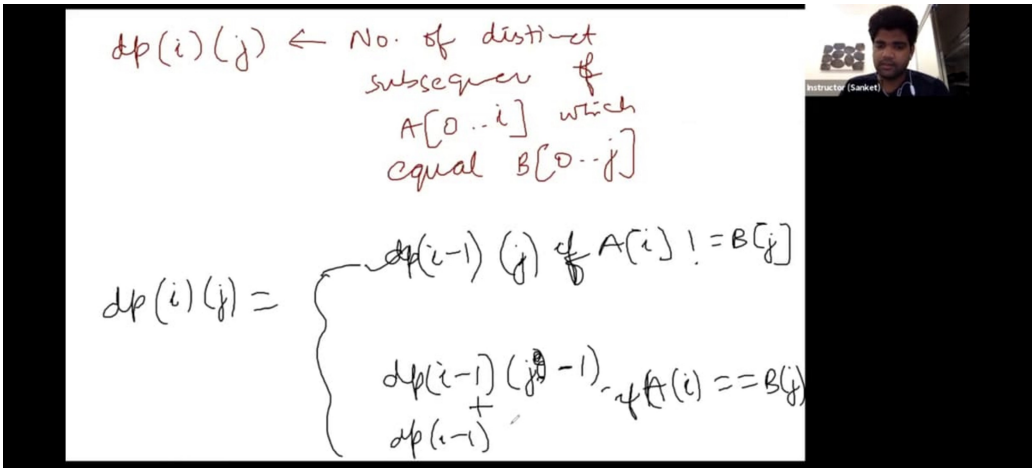
Lec 10

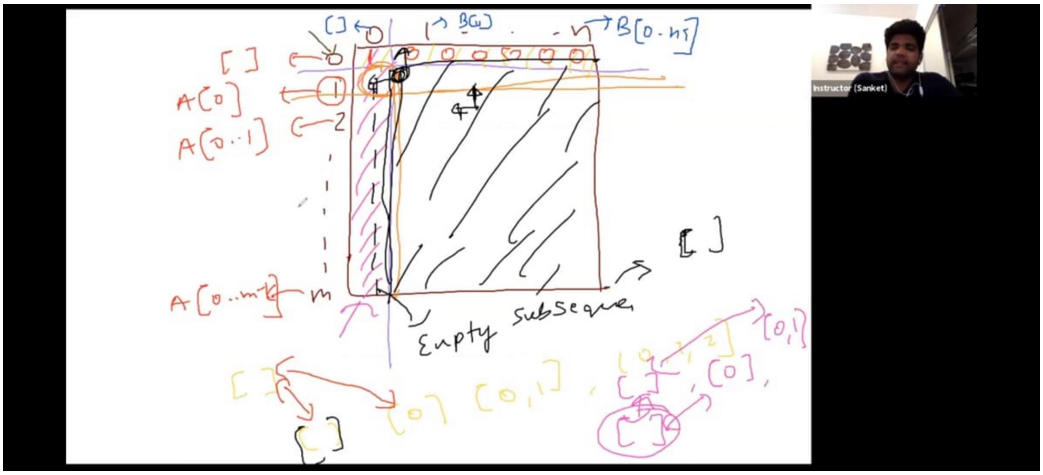


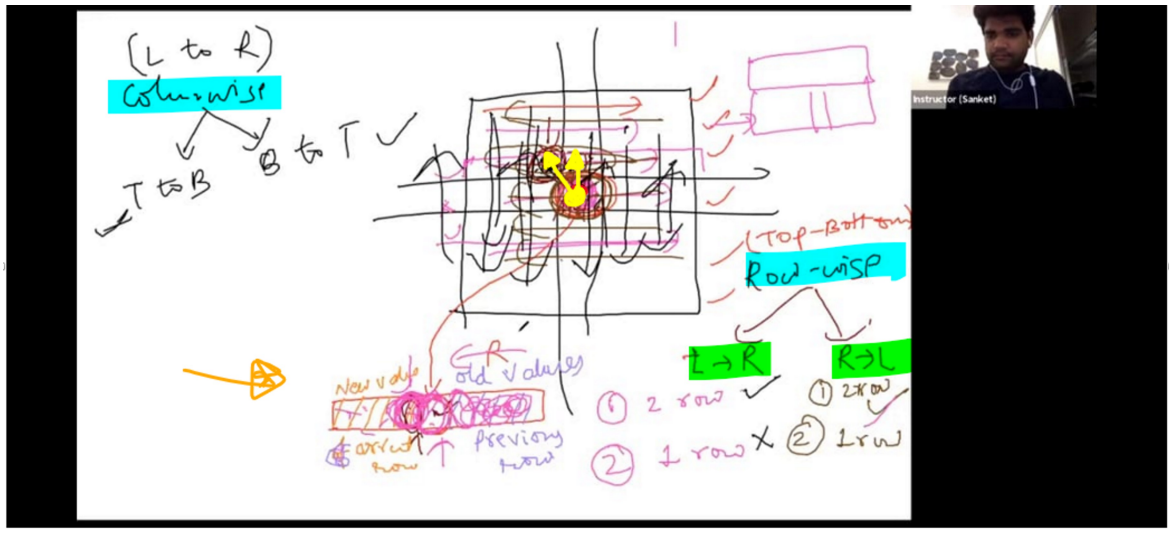


Distinct Subsequences: https://leetcode.com/problems/distinct-subsequences/
Solution :https://leetcode.com/problems/distinct-subsequences/submissions/
(add this to github & in DP notes!!)





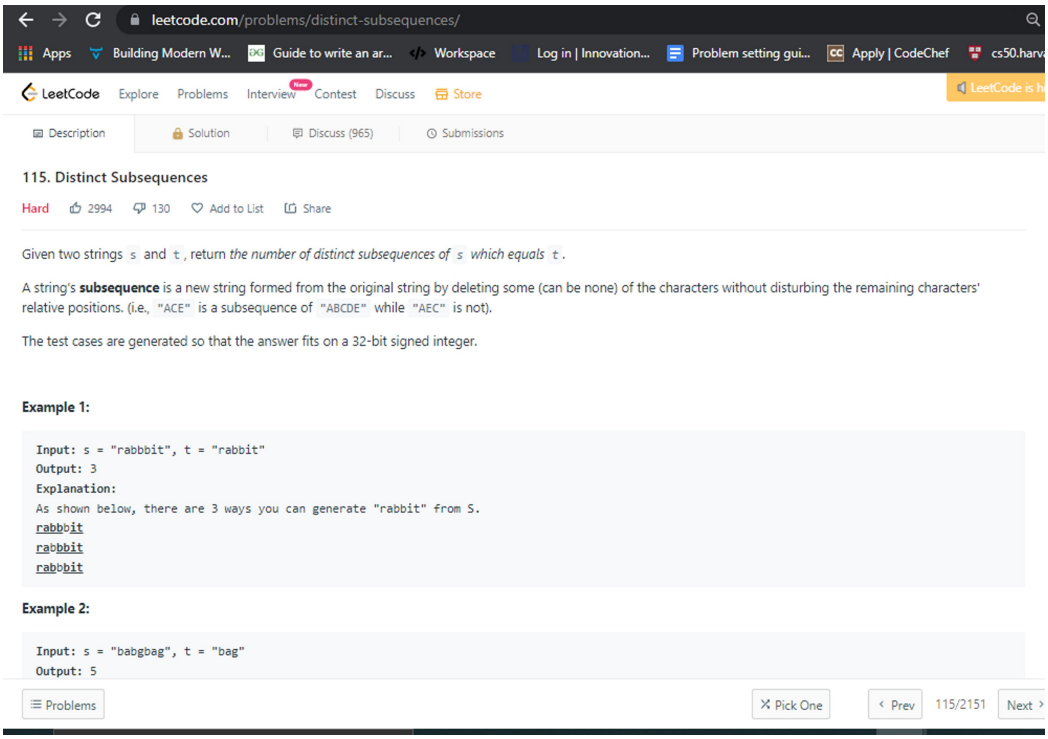


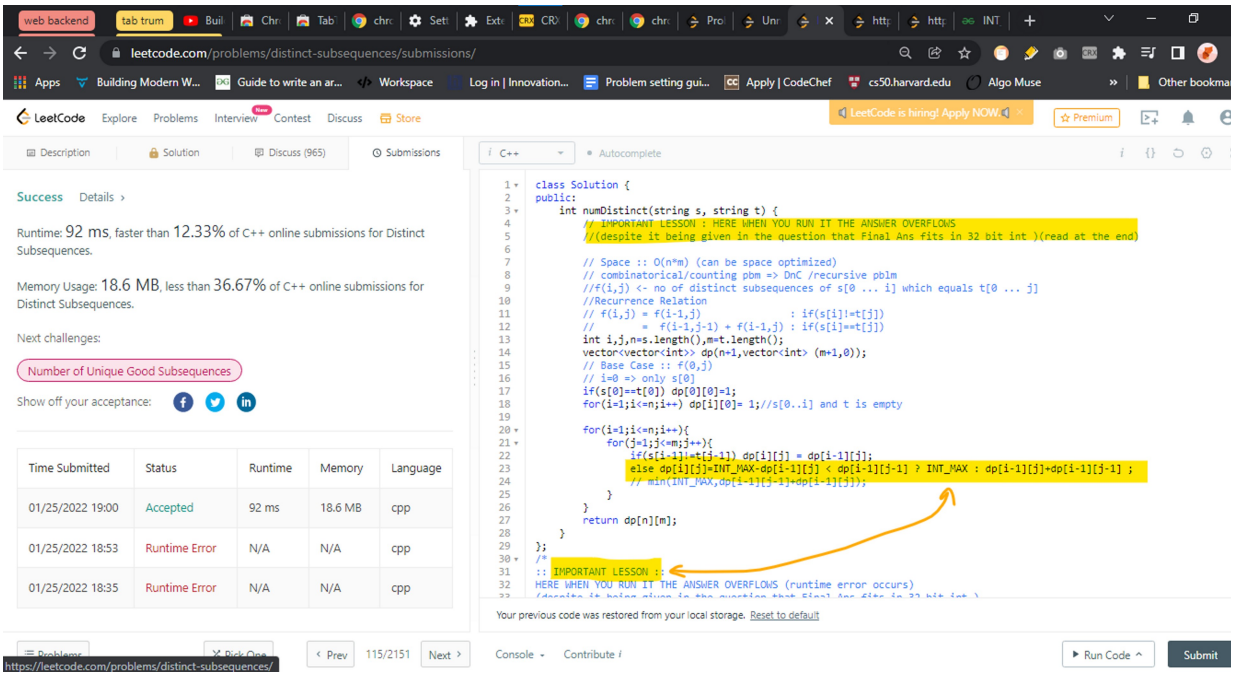


Ways to fill the table for above subproblem::
If :: f(i,j) depends upon : f(i-1,j-1) & f(i-1,j)

Strategy to fill the table	Direction		Possible ?		
Row wise	Left to Right (L->R)	Using 2 rows	Yes		
		Using 1 row	No	Bcaz the new values are replacing the old values which we need to fill the next elements in the same rows	But ,We can do it with 1row and 1 variable
	Right to Left (R->L)	Using 2 rows			
		Using 1 row		Old values are not disturbed	
Column wise	Top to Bottom	Using 2 rows			
		Using 1 row		Old values which we need further are disturbed	
	Bottom to Top	Using 2 rows			
		Using 1 row		Old values are not disturbed	

2D DP :
Next Subproblem type:: any general subarray (and not suffix / prefix arrays)
Q) longest palindromic subsequence :: https://leetcode.com/problems/distinct-subsequences/


leetcode.com/problems/distinct-subsequences/
LeetCode Explore Problems Interview Contest Discuss Store
Description Solution Discuss (965) Submissions
115. Distinct Subsequences
Hard 2994 130 Add to List Share
Given two strings s and t, return the number of distinct subsequences of s which equals t.
A string's subsequence is a new string formed from the original string by deleting some (can be none) of the characters without disturbing the remaining characters' relative positions. (i.e., "ACE" is a subsequence of "ABCDE" while "AEC" is not).
The test cases are generated so that the answer fits on a 32-bit signed integer.
Example 1:
Input: s = "rabbbit", t = "rabbit"
Output: 3
Explanation:
As shown below, there are 3 ways you can generate "rabbit" from S.
rabbbit
rabbbit
rabbbit
Example 2:
Input: s = "babgbag", t = "bag"
Output: 5
Problems
Pick One
Prev 115/2151 Next


leetcode.com/problems/distinct-subsequences/submissions/
LeetCode Explore Problems Interview Contest Discuss Store
Description Solution Discuss (965) Submissions
Success Details
Runtime: 92 ms, faster than 12.33% of C++ online submissions for Distinct Subsequences.
Memory Usage: 18.6 MB, less than 36.67% of C++ online submissions for Distinct Subsequences.
Next challenges:
Number of Unique Good Subsequences
Show off your acceptance:
Time Submitted Status Runtime Memory Language
01/25/2022 19:00 Accepted 92 ms 18.6 MB cpp
01/25/2022 18:53 Runtime Error N/A N/A cpp
01/25/2022 18:35 Runtime Error N/A N/A cpp
class Solution {
public:
int numDistinct(string s, string t) {
// IMPORTANT LESSON : HERE WHEN YOU RUN IT THE ANSWER OVERFLOWS
//(despite it being given in the question that Final Ans fits in 32 bit int)(read at the end)
// Space :: O(n*m) (can be space optimized)
// combinatorical/counting pbm => DnC /recursive pblm
//f(i,j) <- no of distinct subsequences of s[0 ... i] which equals t[0 ... j]
//Recurrence Relation
// f(i,j) = f(i-1,j) : if(s[i]!=t[j])
// = f(i-1,j-1) + f(i-1,j) : if(s[i]==t[j])
int i,j,n=s.length(),m=t.length();
vector<vector<int>> dp(n+1,vector<int> (m+1,0));
// Base Case :: f(0,j)
// i=0 => only s[0]
if(s[0]==t[0]) dp[0][0]=1;
for(i=1;i<=n;i++) dp[i][0]= 1;//s[0..i] and t is empty
for(i=1;i<=n;i++){
for(j=1;j<=m;j++){
if(s[i-1]!=t[j-1]) dp[i][j] = dp[i-1][j];
else dp[i][j]=INT_MAX-dp[i-1][j] < dp[i-1][j-1] ? INT_MAX : dp[i-1][j]+dp[i-1][j-1] ;
// min(INT_MAX,dp[i-1][j-1]+dp[i-1][j]);
}
}
return dp[n][m];
}
};
/*
:: IMPORTANT LESSON ::
HERE WHEN YOU RUN IT THE ANSWER OVERFLOWS (runtime error occurs)
Your previous code was restored from your local storage. Reset to default
Prev 115/2151 Next
Console Contribute
Run Code
Submit

C++ · Autocomplete

```cpp
                // min(INT_MAX,dp[i-1][j-1]+dp[i-1][j]);
            }
        }
        return dp[n][m];
    }
};
/*
:: IMPORTANT LESSON ::
HERE WHEN YOU RUN IT THE ANSWER OVERFLOWS (runtime error occurs)
(despite it being given in the question that Final Ans fits in 32 bit int )

Reason/Explaination: The above problem happens if the solution for a subproblem is greater than the solution of the final problem.
Same thing is happening here!

Eg: s="aaa" , t="ab"
here soln for subproblem dp(3,1) = 3 (since t="aaa" & s="a")
but soln for final ans: dp(3,2) = 0 (since t="aaa" & s="a" & t has no commmon subsequence equal to s);
here the subproblem ans is greater than actual/final ans

*/
/*
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Longest Palindromic subsequence:

leetcode.com/problems/longest-palindromic-subsequence/

Apps · Building Modern W... · Guide to write an ar... · Workspace · Log in | Innovation... · Proble

Description · Solution · Discuss (999+) · Submissions

516. Longest Palindromic Subsequence

Medium · 4576 · 239 · Add to List · Share

Given a string `s`, find *the longest palindromic* ***subsequence****'s length in* `s`.

A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

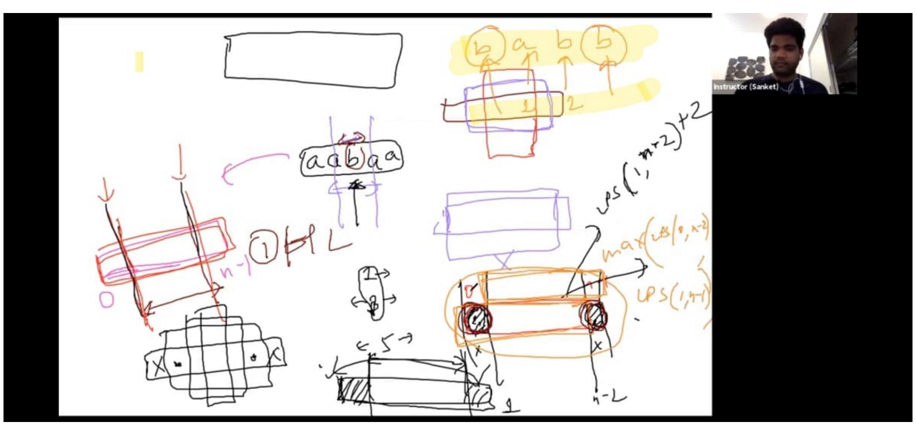
```
Input: s = "bbbab"
Output: 4
Explanation: One possible longest palindromic subsequence is "bbbb".
```

Example 2:

```
Input: s = "cbbd"
Output: 2
Explanation: One possible longest palindromic subsequence is "bb".
```

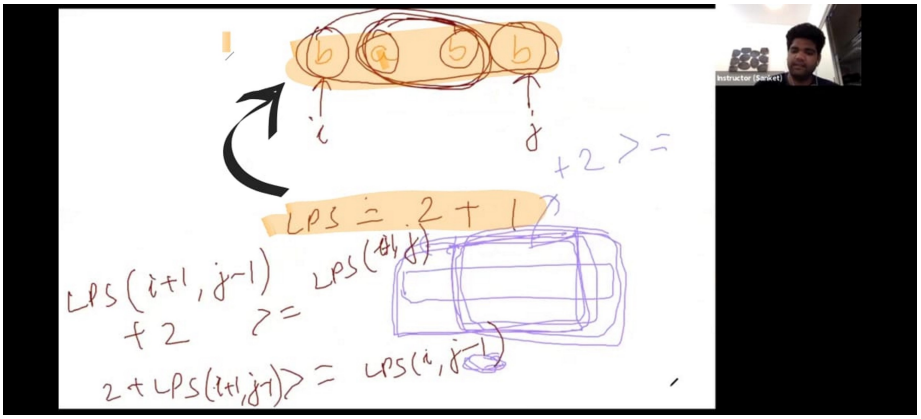
Constraints:

- `1 <= s.length <= 1000`
- `s` consists only of lowercase English letters.



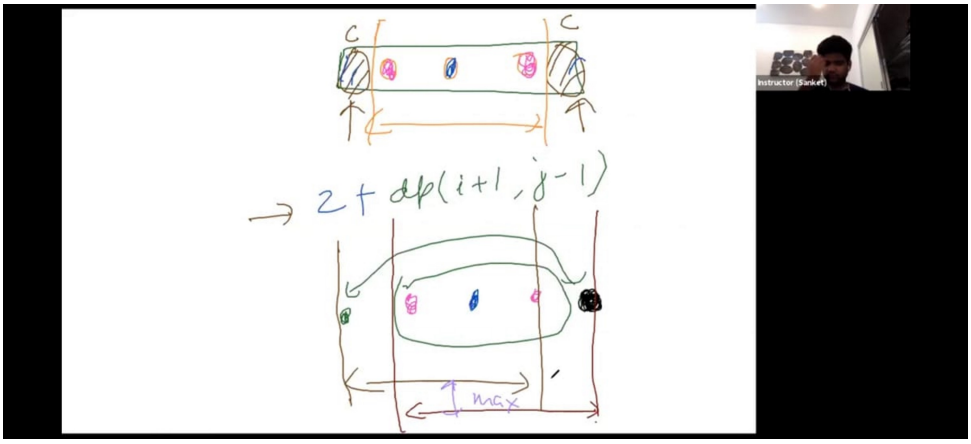
.

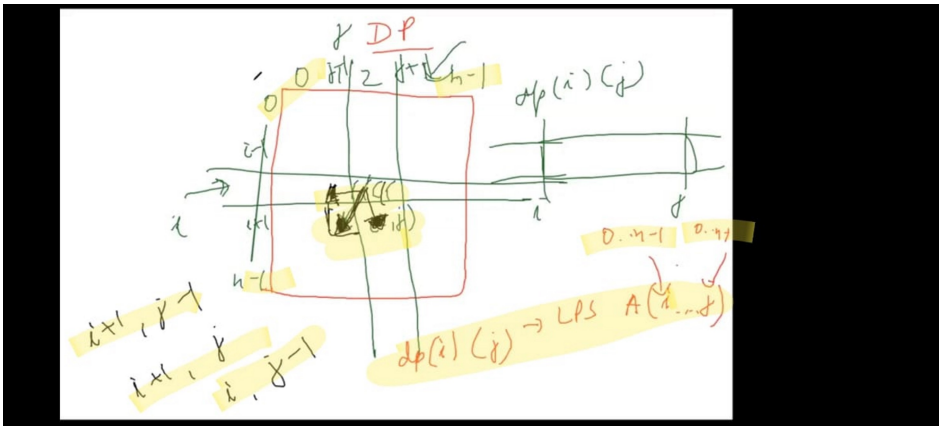
.

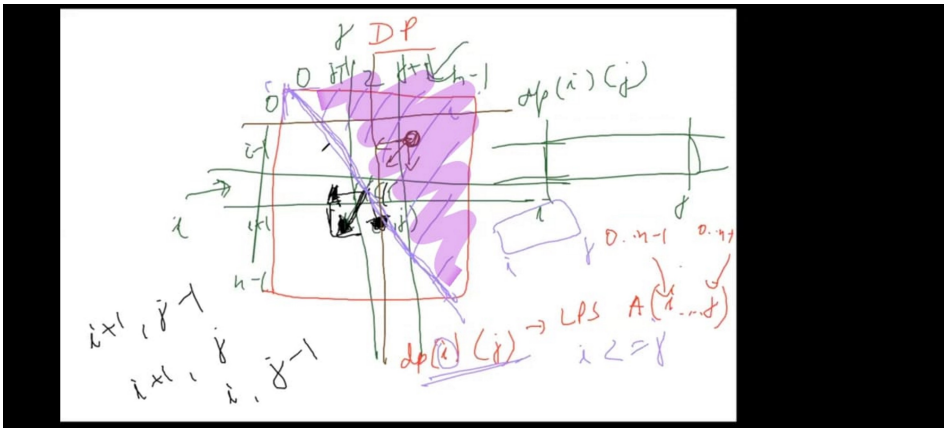

b a a b
i
j
+2 >=
LPS = 2 + 1
LPS(i+1, j-1)
+2 >=
LPS(i, j)
2+LPS(i+1,j-1) >= LPS(i, j)

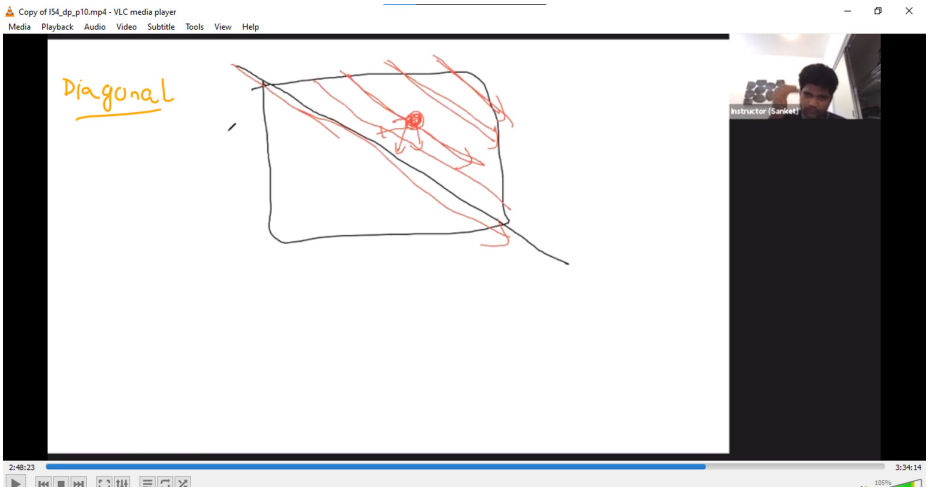
.

.


c
c
a
a
a
2 + dp(i+1, j-1)
max


DP
0 1 2 i i+1 n-1
dp(i) (j)
i
j
i+1, j-1
i+1, j
i, j-1
dp(i) (j) → LPS A(i...j)


DP
0 1 2 i i+1 n-1
dp(i) (j)
i
j
i+1, j-1
i+1, j
i, j-1
dp(i) (j) → LPS A(i...j)
i <= j



Strategies to fill the DP table:

-> **row wise**(left to right & bottom to top) (base case principal diagonal => fill that by hand)

-> One strategy to **fill** could also be ***diagonally*** ,since at each step we need values that were in the previous diagonal

leetcode.com/problems/longest-palindromic-subsequence/submissions/

Apps | Building Modern W... | Guide to write an ar... | Workspace | Log in | Innovation... | Problem setting gui... | Apply | CodeChef | cs50.harvard.edu | Algo Muse | Other bookmarks

Description | Solution | Discuss (999+) | Submissions

Success Details ›

Runtime: 211 ms, faster than 16.87% of C++ online submissions for Longest Palindromic Subsequence.

Memory Usage: 72.9 MB, less than 50.43% of C++ online submissions for Longest Palindromic Subsequence.

Next challenges:

Longest Palindromic Substring | Palindromic Substrings | Count Different Palindromic Subsequences | Longest Palindromic Subsequence II | Maximize Palindrome Length From Subsequences | Maximum Product of the Length of Two Palindromic Subsequences

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
01/27/2022 00:04	Accepted	211 ms	72.9 MB	cpp
01/26/2022 15:28	Wrong Answer	N/A	N/A	cpp

C++ | Autocomplete

```cpp
class Solution {
public:
    /* Approach 1: s = original & s' = reverse of s, find the LCS(s,s') ; LCS=longest common subsequence
       Approach 2: expand palindrome around the center */
    int longestPalindromeSubseq(string s) {
        // Solution for Approach 2
        // Optimization pblm (since longest) => can think of DnC (Divide and Conquer) approach
        // Here our subproblem can be thought as this: for LPS(s[0 .... n-1])
        // if(s[0]==s[n-1]) then LPS(s[0 .... n-1]) = 2+LPS(s[1 ... n-2])
        // else LPS(s[0 ... n-1]) = max(LPS(s[1 ... n-1]), LPS(s[0 ... n-2]) )

        // form above its clear that our subproblem will be any general subarray of string s  => s[i ... j]
        // => we have 2 variables here i & j (start & end indexes of any subarray)
        int i,j,n=s.length();
        vector<vector<int>> dp(n,vector<int>(n,0));

        //Recurrence Relation
        // dp(i,j) <- length of LPS in s[i ... j]
        // dp(i,j) = dp[i+1][j-1] : if(s[i]==s[j])
        //         = max(dp[i][j-1],dp[i+1,j]) : if(s[i]!=s[j])

        //Base Case
        for(i=0;i<n;i++) dp[i][i]=1;

        for(i=n-2;i>=0;i--){//bottom to up
            for(j=i+1;j<n;j++){//left to right
                if(s[i]==s[j]) dp[i][j]=2+dp[i+1][j-1];
                else{
                    dp[i][j]=max(dp[i+1][j],dp[i][j-1]);
                }
            }
        }
        //dp[0][n-1] :: final ans
        return dp[0][n-1];
    }
```

Your previous code was restored from your local storage. Reset to default

- Space optimized soln is also possible here(O(2*n) instead of O(n*n))

Solution 3: Bottom up DP (Space Optimized)

- Since we need to access up to 2 dp states: current dp state `dp` and previous dp state `dpPrev`.
- So we can optimize to achieve `O(N)` in Space Complexity.

C++ | Python3

```cpp
class Solution {
public:
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<int> dp(n, 0), dpPrev(n, 0);
        for (int i = n - 1; i >= 0; --i) {
            dp[i] = 1;
            for (int j = i+1; j < n; ++j) {
                if (s[i] == s[j]) {
                    dp[j] = dpPrev[j-1] + 2;
                } else {
                    dp[j] = max(dpPrev[j], dp[j-1]);
                }
            }
            dp.swap(dpPrev);
        }
        return dpPrev[n-1];
    }
};
```

-

- Top down approach code:

Solution 3: Bottom up DP (Space Optimized)

- Since we need to access up to 2 dp states: current dp state `dp` and previous dp state `dpPrev`.
- So we can optimize to achieve `O(N)` in Space Complexity.

C++ | Python3

```cpp
class Solution {
public:
    int longestPalindromeSubseq(string s) {
        int n = s.size();
        vector<int> dp(n, 0), dpPrev(n, 0);
        for (int i = n - 1; i >= 0; --i) {
            dp[i] = 1;
            for (int j = i+1; j < n; ++j) {
                if (s[i] == s[j]) {
                    dp[j] = dpPrev[j-1] + 2;
                } else {
                    dp[j] = max(dpPrev[j], dp[j-1]);
                }
            }
            dp.swap(dpPrev);
        }
        return dpPrev[n-1];
    }
};
```

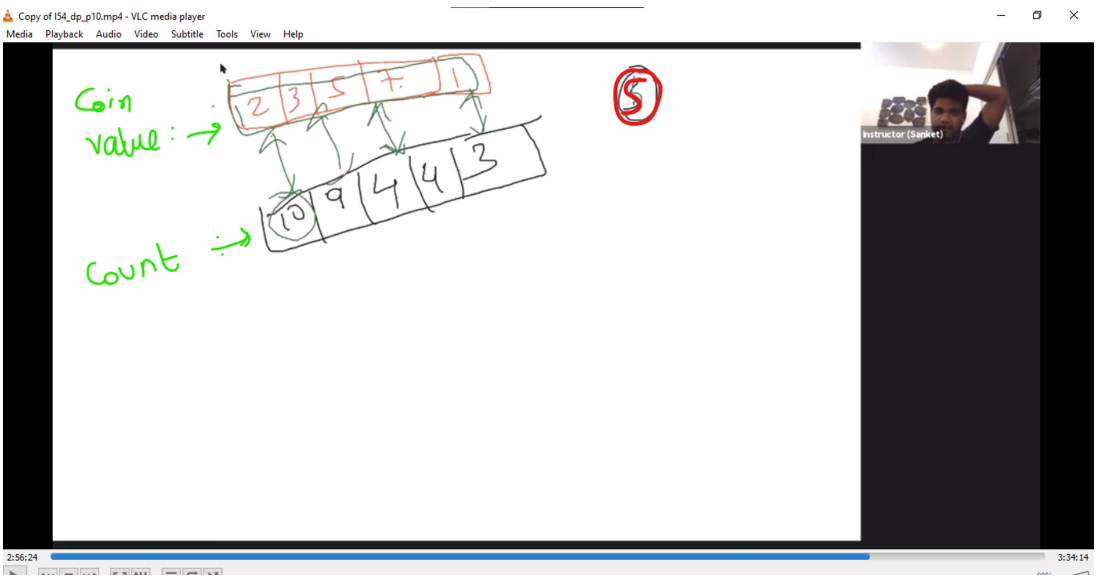
Q) Coin Change problem (at most k_i coins for each coin type):
Given value of different coins and frequency of each coin is given) . Find the no of ways to get sum = S

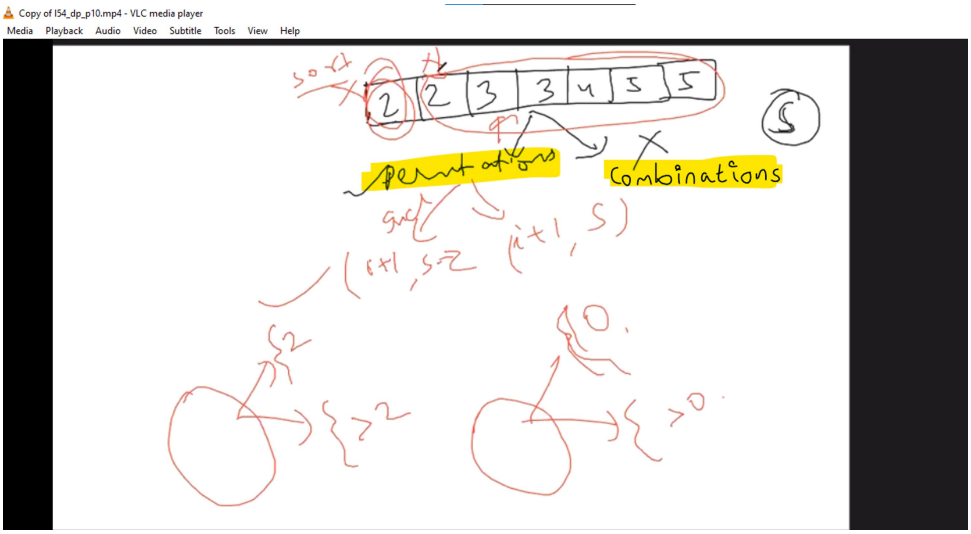
Dynamic Programm...

Dynamic Programm...

Audio recording started: 18:49 26 January 2022

Audio recording started: 18:47 26 January 2022





Slack questions:

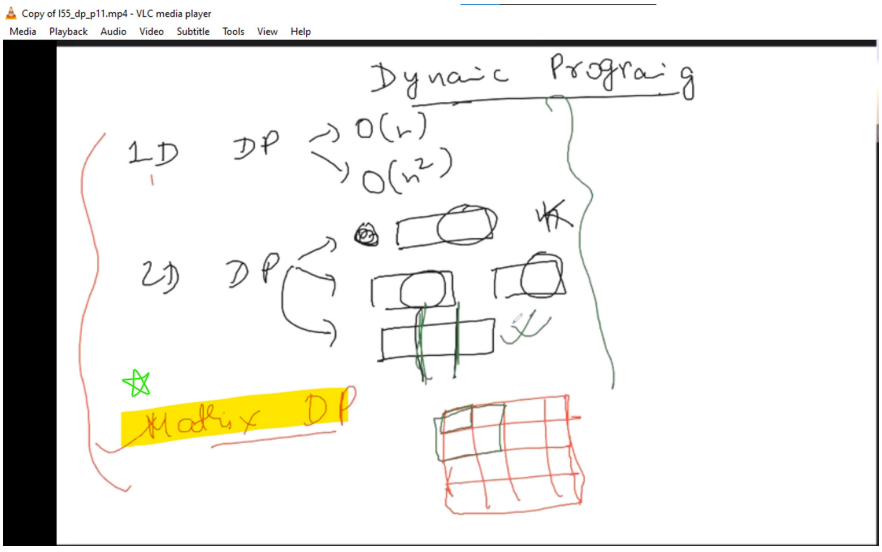
- Wildcart matching :: https://leetcode.com/problems/wildcard-matching/
- Buy & stock sell
- Regular expression matching :: https://leetcode.com/problems/regular-expression-matching/
- https://leetcode.com/problems/number-of-matching-subsequences/

==========

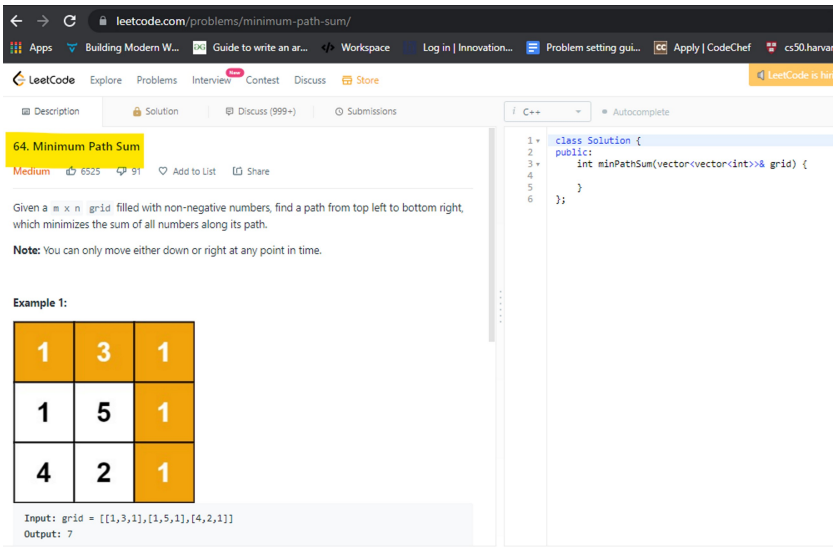
Lec 11

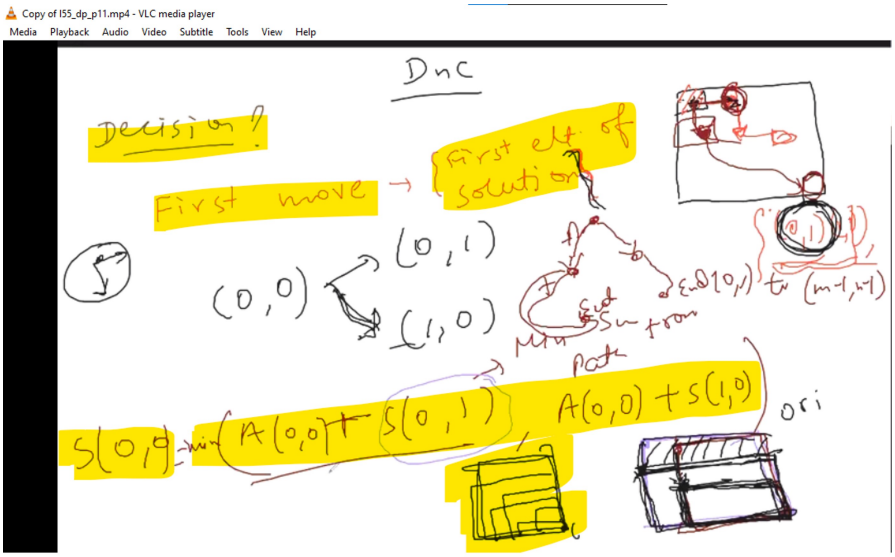
Amazon : nitk

- OA: merge 2 sorted linked list
- Deep copy linked list
- *** Critical connections in a network
- Bridges in graph :
  - Bridges in Graph | Cut Edge
  - https://cp-algorithms.com/graph/bridge-searching.html

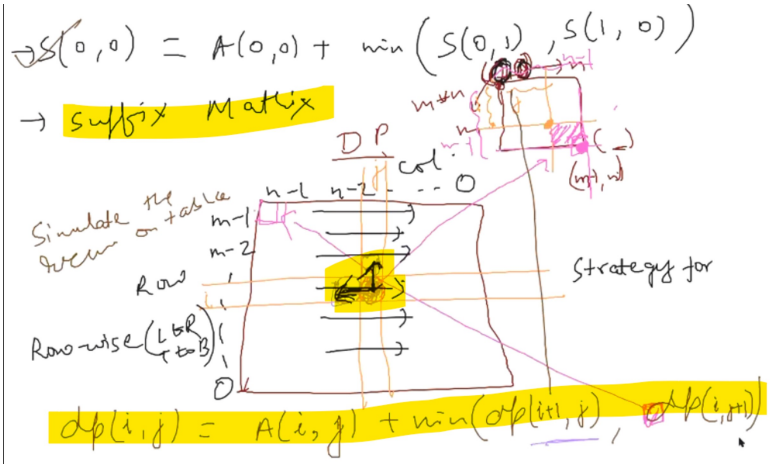


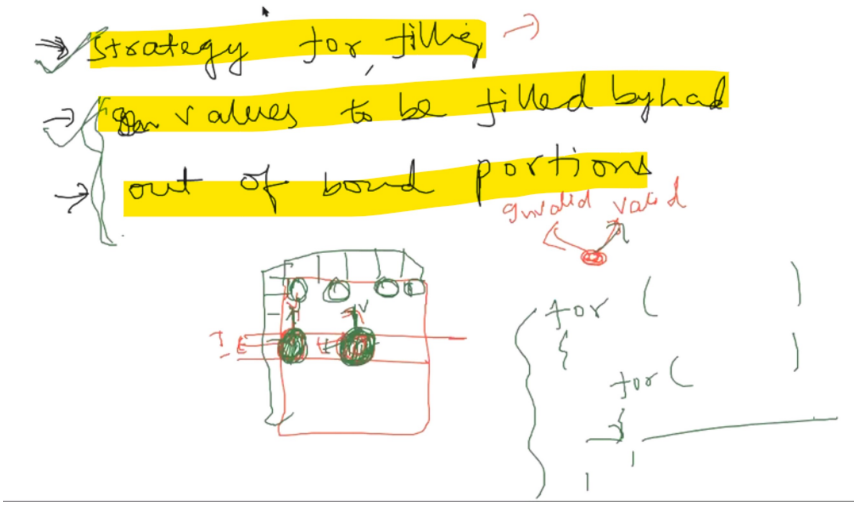
Q)

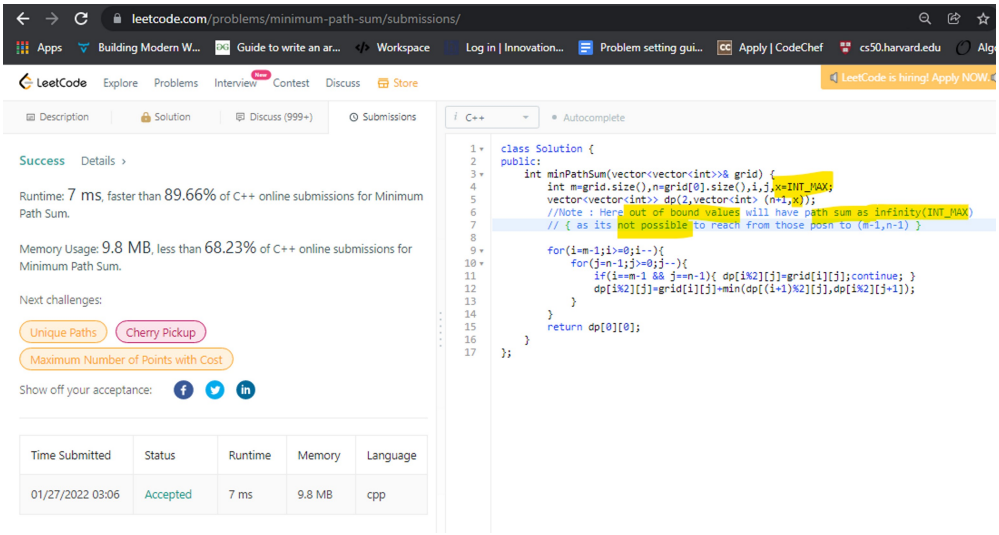




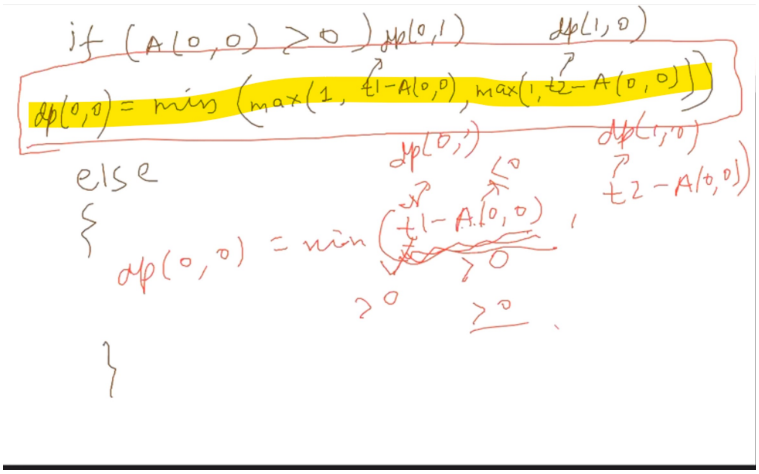
Here subproblem is suffix matrices



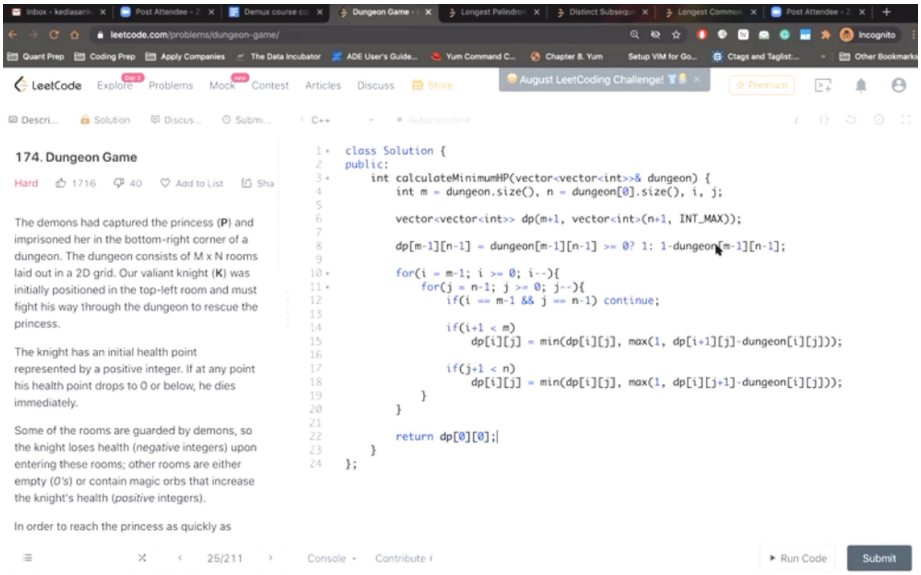




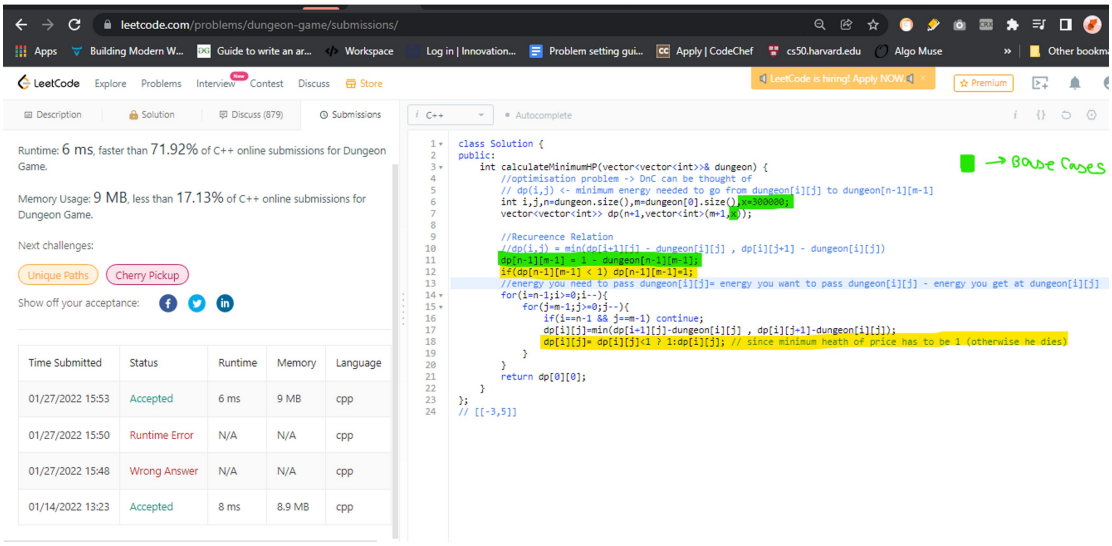
Dungeon game:



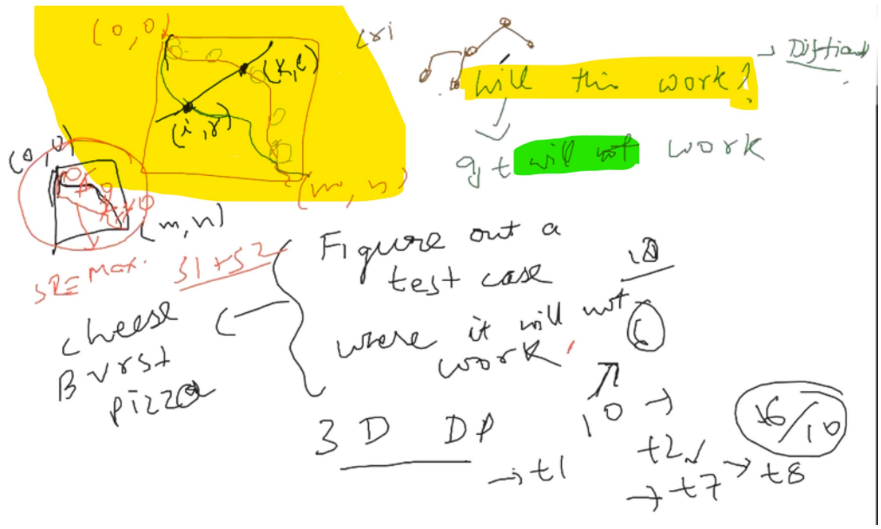
Sir's soln:



My solution :



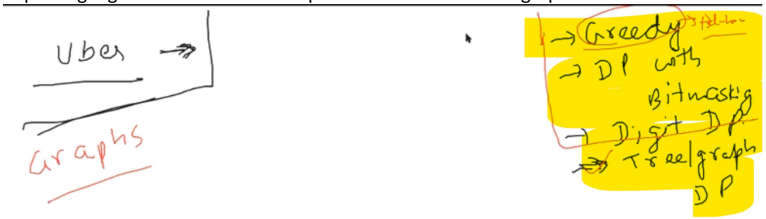
<->Q) **Cherry pickup** (hard) (Uber) **TO DO**



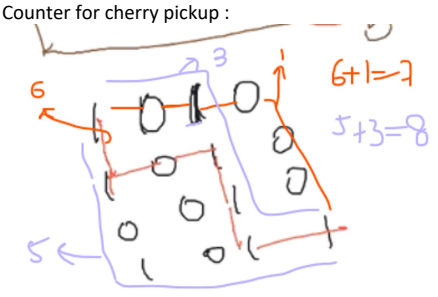
Dynamic Programm...

Audio recording started: 17:08 27 January 2022

Topics highlighted below are less imp & will be covered after graphs



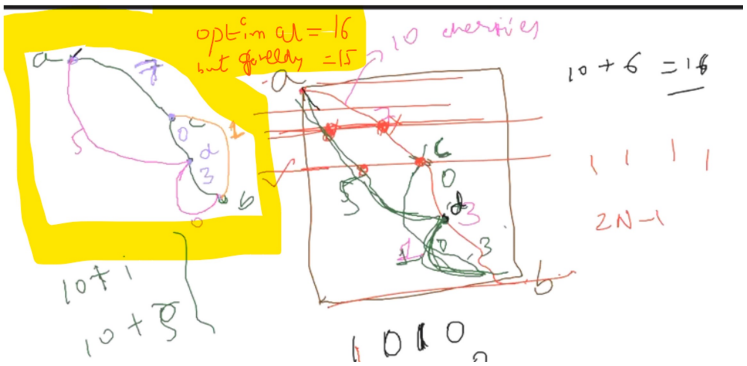
Counter for cherry pickup :



Dynamic Programm...

Audio recording started: 18:51 27 January 2022

Another counter example for why approach suggested in voice recording won't work:



Optimal : 8+8=16
But voice recording approach gives: 10+5 =15

We have to use 3d dp to solve this cherry pickup problem

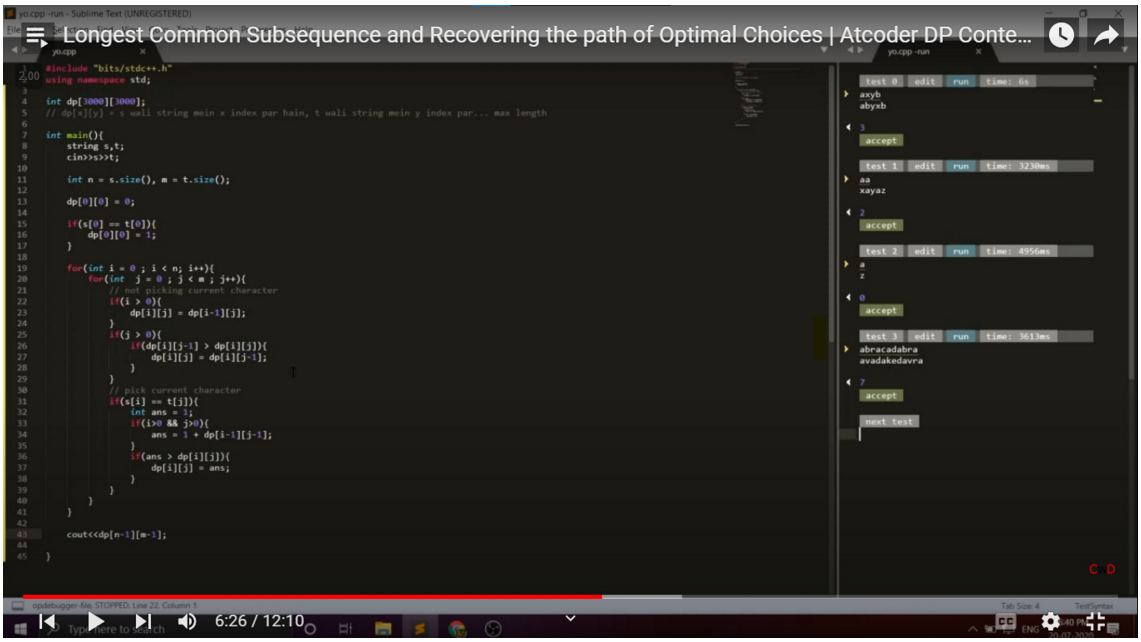
Just Pointers to solve these problems are discussed at the end of this lecture

- **940 .** Distinct Subsequences ii
  - This is enumeration questions in these we enumerate in order
- ** **1235.** Maximum profit in job scheduling (google)
- **714.** best time to buy and sell stocks with transaction fee

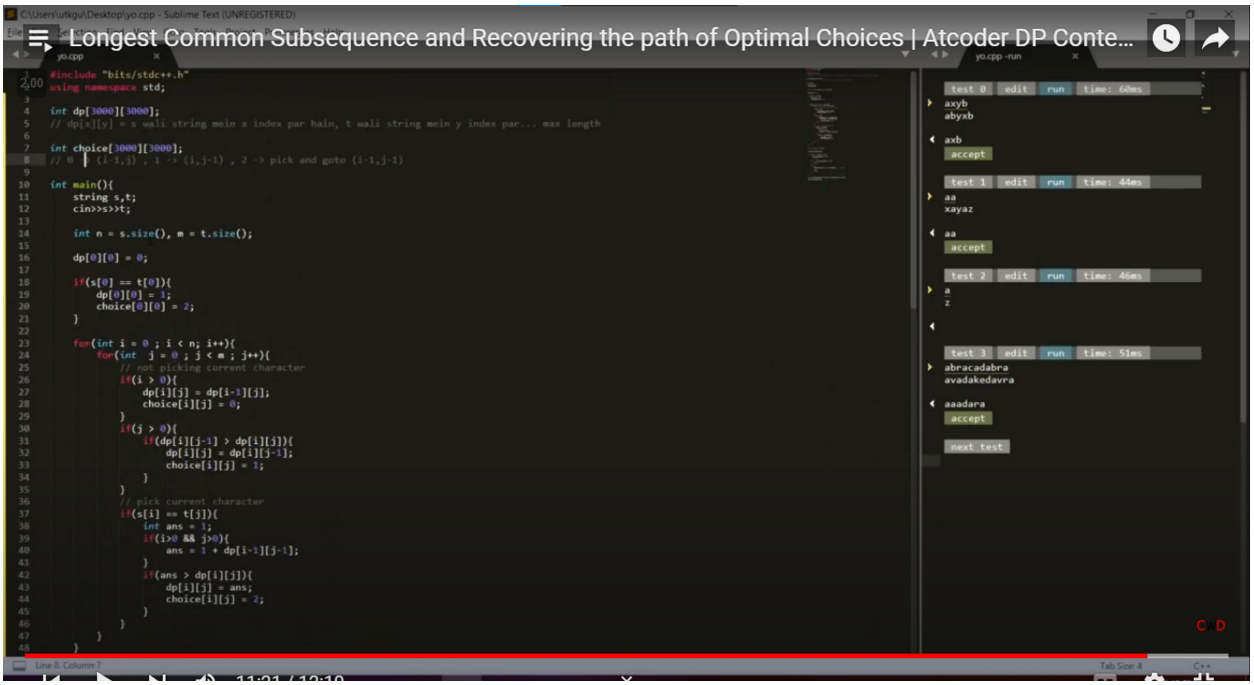
==========

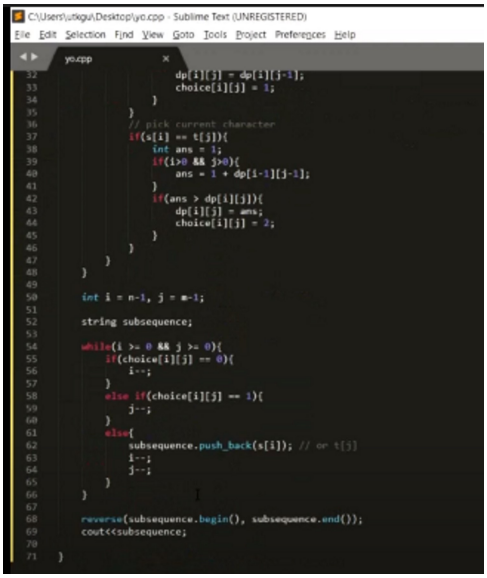
- https://practice.geeksforgeeks.org/problems/painting-the-fence3727/1/
  - Soln: https://www.geeksforgeeks.org/painting-fence-algorithm/

- Length of longest common subsequence :: Longest Common Subsequence and Recovering the path of Optimal Choices | Atcoder DP Contest Problem F



- Lcs with the LCS string also :


Longest Common Subsequence and Recovering the path of Optimal Choices | Atcoder DP Conte...
yo.cpp
yo.cpp -run
test 0 edit run time: 60ms
axyb
abyxb
axb
accept
test 1 edit run time: 44ms
aa
xayaz
aa
accept
test 2 edit run time: 46ms
a
z
test 3 edit run time: 51ms
abracadabra
avadakedavra
aaadara
accept
next test


C:\Users\utkgu\Desktop\yo.cpp - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
yo.cpp