

Dynamic Programming

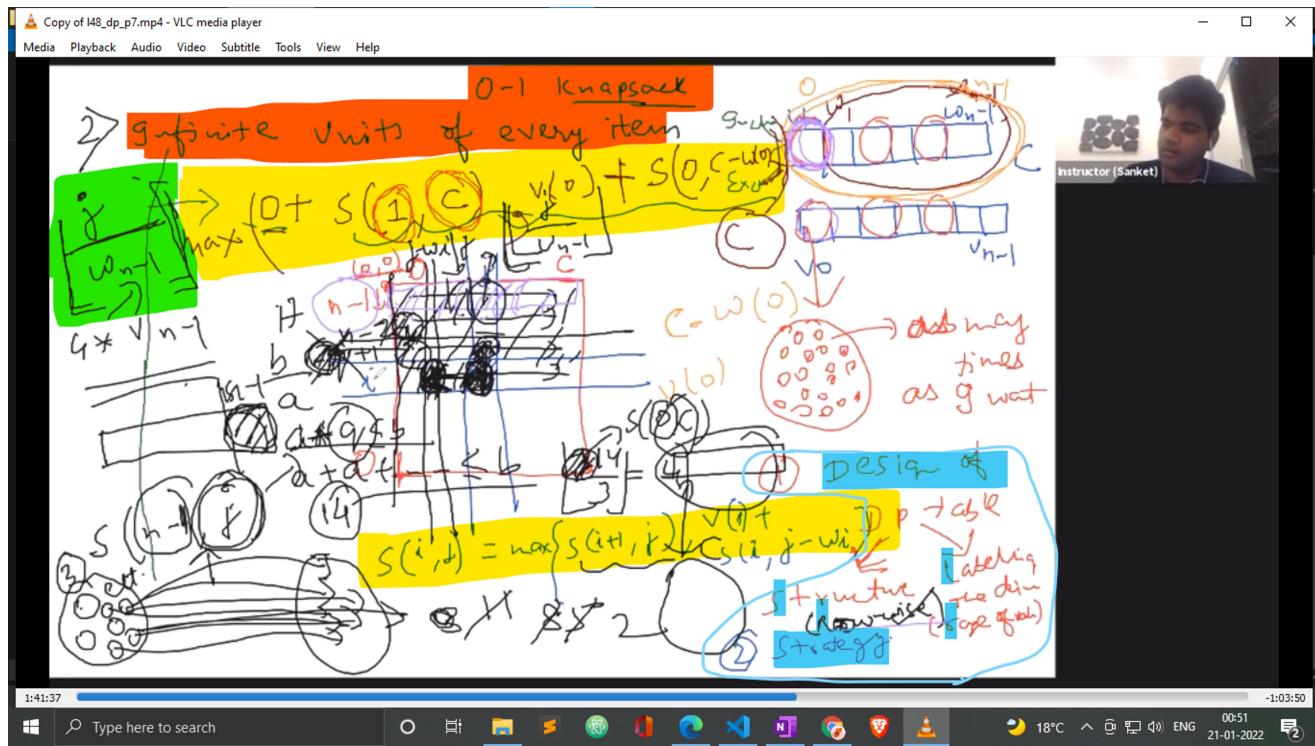
25 October 2021 18:28

- <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](#)

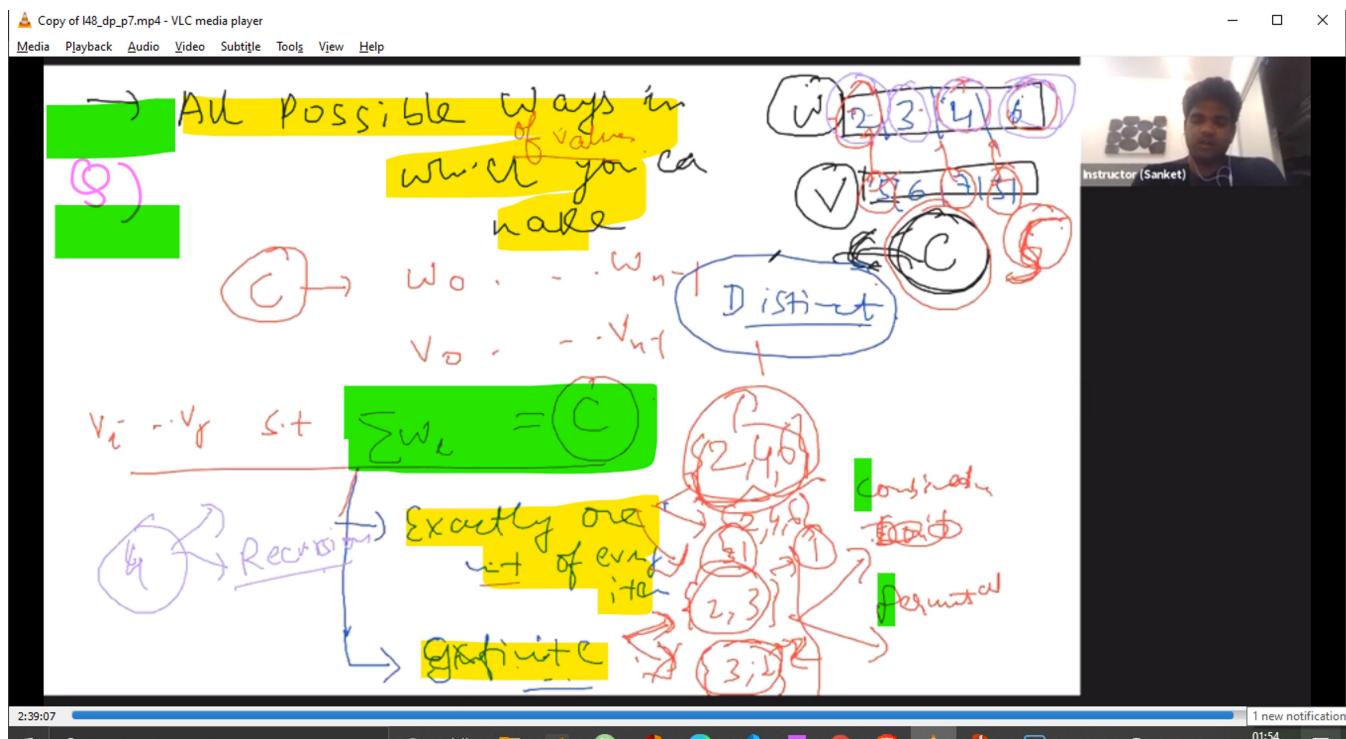


Copy of l48_dp_p7.mp4 - VLC media player

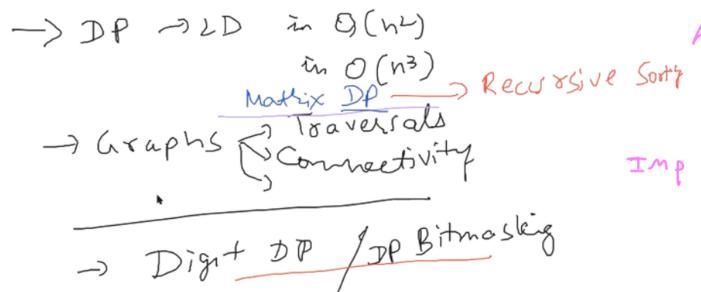
Media Playback Audio Video Subtitle Tools View Help



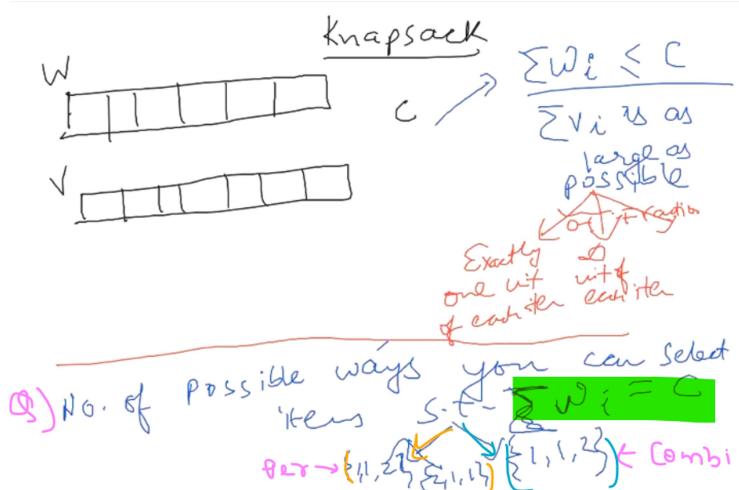
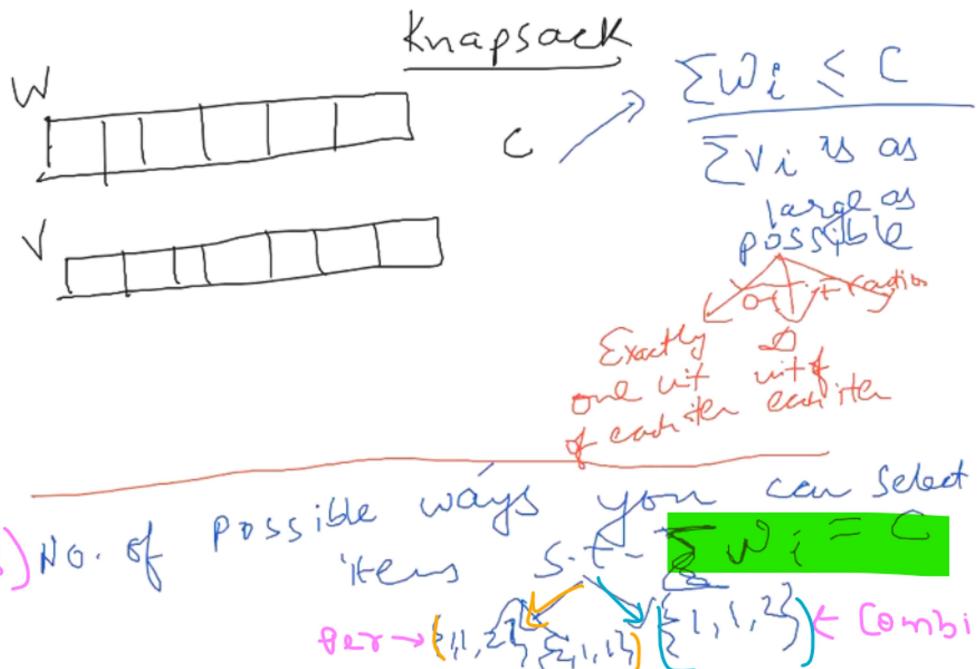
Question::??



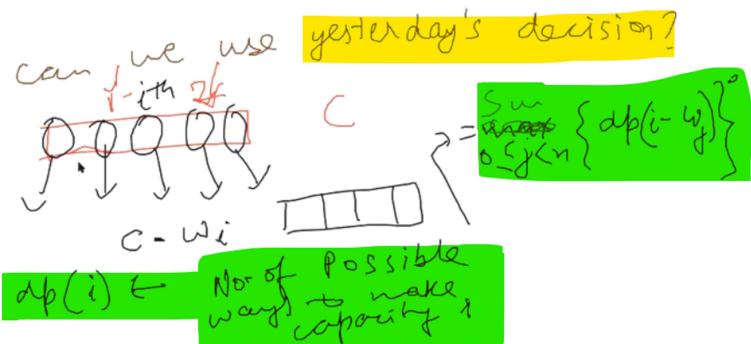
Lec 8:



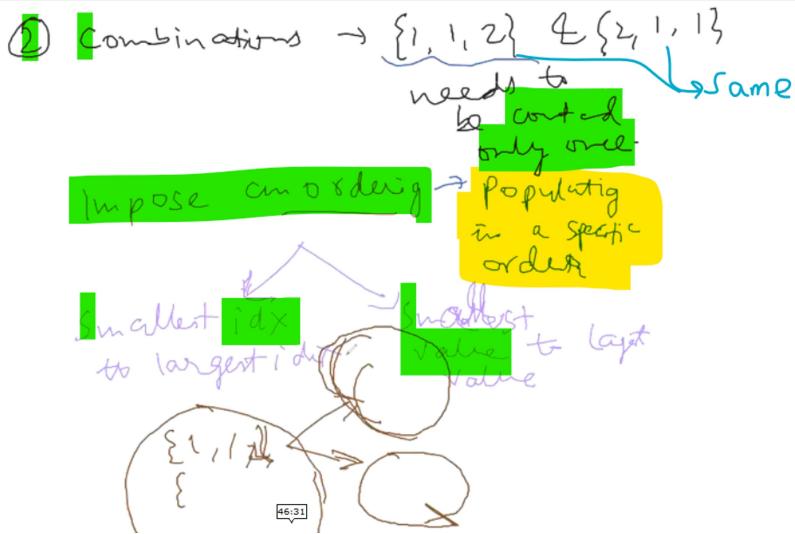
Imp



① Permutations. Intuitively, $\{1, 1, 2\}$ & $\{2, 1, 1\}$ are different)



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

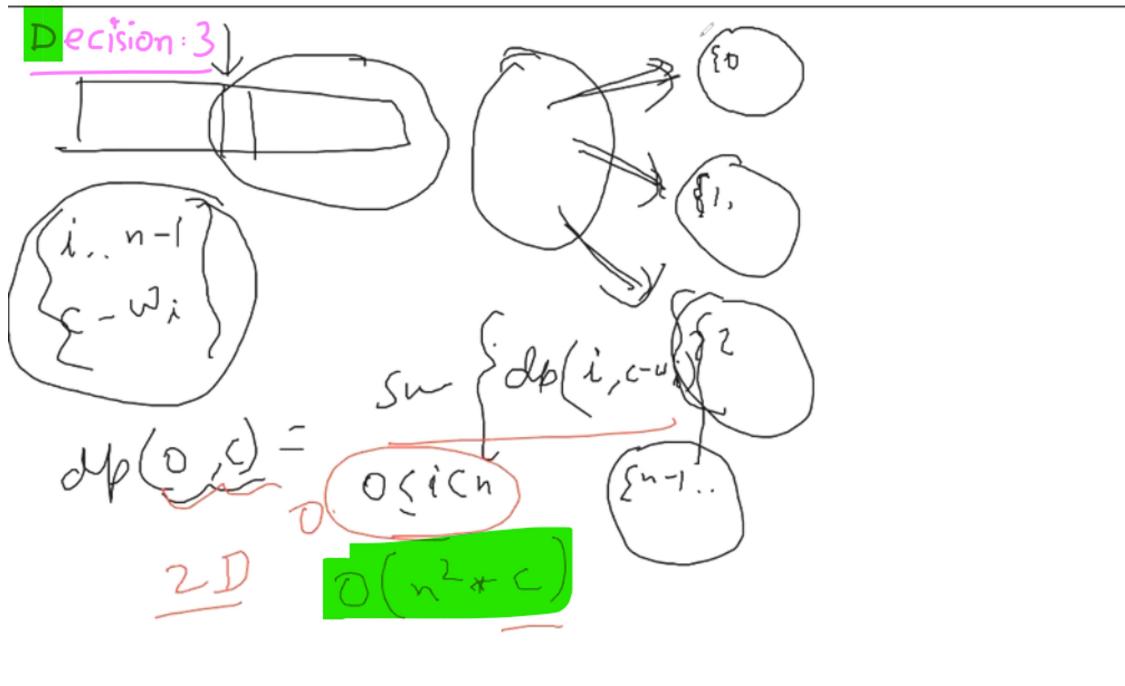
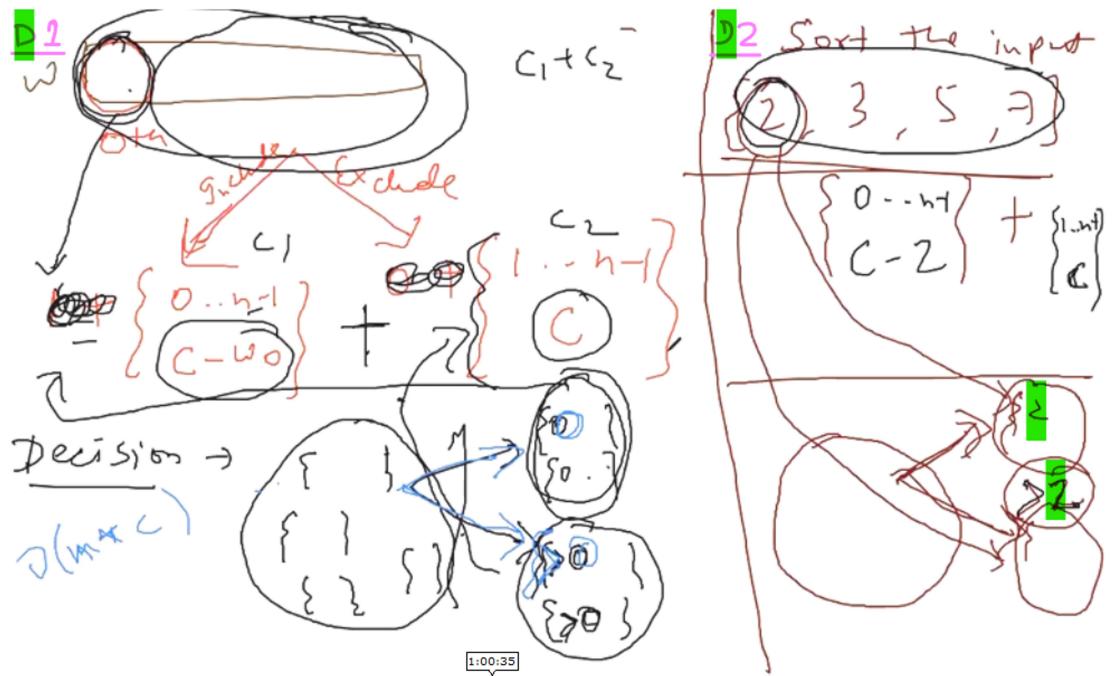
Branching factor of recursion tree: is 2

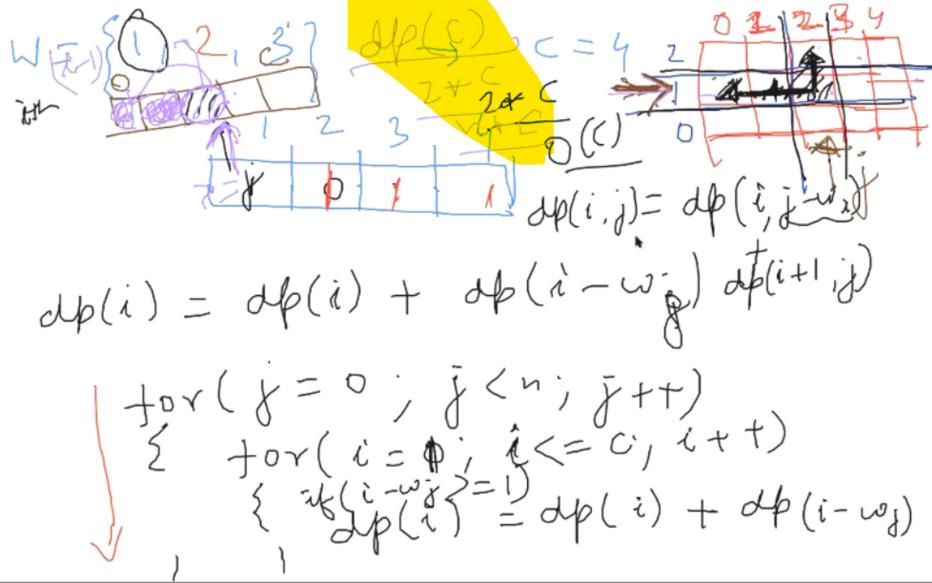
Decision 3: decision on the first element of the solution (*based on value*)

Sort the capacity array

We then decide from index $0 \dots n-1$ which will be the first unique element to be added in the ans subset, and our subproblem becomes from $i+1 \dots n-1$, 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)



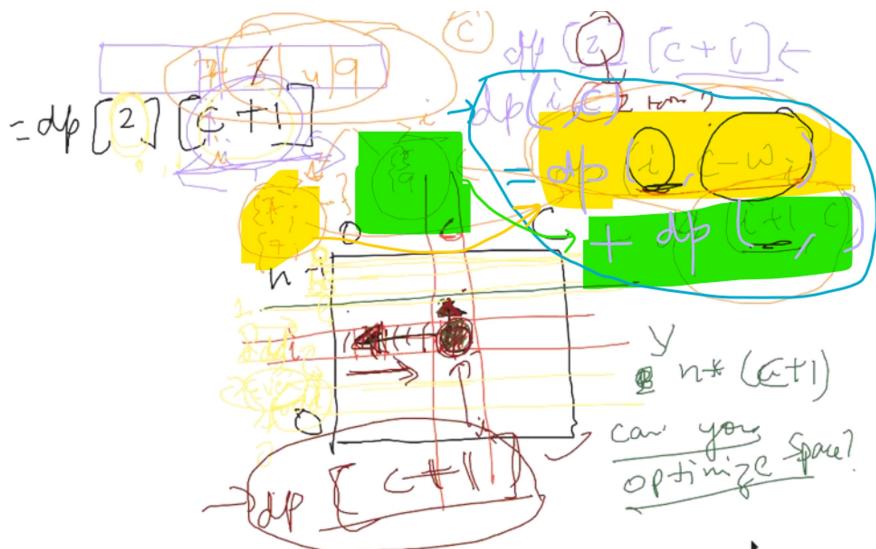


For the above type array filling (upper and left) we can space optimize the dp :

$$O(C^*V) \Rightarrow O(2^*C) \Rightarrow O(C)$$

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??

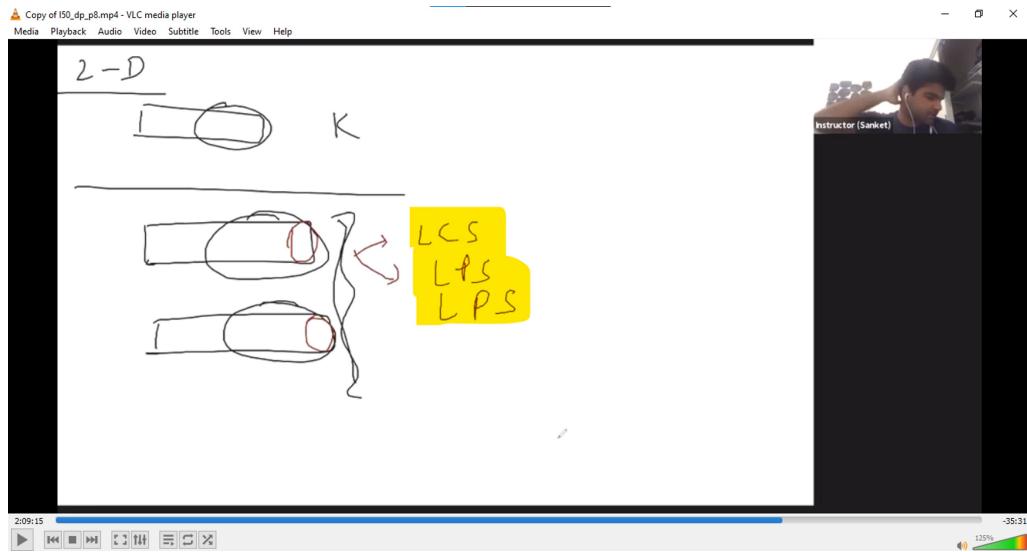


Story for this decision: $dp(i, C)$

- One solution will contain all the possible subsets whose 1st element is the element at start index 'i' \Rightarrow subpbm :: $dp(i, C - W_i)$
- Other solution subset will contain the 1st element as any number after the index 'i' \Rightarrow subpbm :: $dp(i+1, C)$

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
 - o Longest common subsequence
 - o Longest palindromic substring
 - o Longest palindromic subsequence

Interviewbit : coin sum infinite

```

1 int Solution::coinchange2(vector<int> &A, int B) {
2     // O(n*C) time and space
3     int n = A.size(), i, j;
4     vector<vector<int>> dp(n, vector<int>(B+1, 0));
5
6     // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
7     // Base case
8     for(i = 0; i <= B; i++){
9         if(i%A[n-1] == 0)
10             dp[n-1][i] = 1;
11     }
12
13     for(i = n-2; i >= 0; i--){
14         for(j = 0; j <= B; j++){
15             if(j-A[i] >= 0)
16                 dp[i][j] = (dp[i][j] + dp[i][j-A[i]])%1000007;
17
18             dp[i][j] = (dp[i][j] + dp[i+1][j])%1000007;
19         }
20     }
21 }
```

```

5
6      // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
7      // Base case
8      for(i = 0; i <= B; i++){
9          if(i%A[n-1] == 0)
10             dp[n-1][i] = 1;
11     }
12
13     for(i = n-2; i >= 0; i--){
14         for(j = 0; j <= B; j++){
15             if(j-A[i] >= 0)
16                 dp[i][j] = (dp[i][j] + dp[i][j-A[i]])%1000007;
17
18             dp[i][j] = (dp[i][j] + dp[i+1][j])%1000007;
19         }
20     }
21
22     return dp[0][B];
23
24
25 }
26

```

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

=====

Same q-> better space complexity

```

2      // O(n*C) time and space
3      int n = A.size(), i, j;
4      vector<vector<int>> dp(2, vector<int>(B+1, 0));
5
6      // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
7      // Base case
8      for(i = 0; i <= B; i++){
9          if(i%A[n-1] == 0)
10             dp[(n-1)%2][i] = 1;
11     }
12
13     for(i = n-2; i >= 0; i--){
14         for(j = 0; j <= B; j++){
15             if(j-A[i] >= 0)
16                 dp[i%2][j] = (dp[(i+1)%2][j] + dp[i%2][j-A[i]])%1000007;
17             else
18                 dp[i%2][j] = (dp[(i+1)%2][j])%1000007;
19         }
20     }
21
22     return dp[0%2][B];
23
24

```

$O(2 \times n)$
space

- Here we space optimized the above solution to $O(2 \times n)$, 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

=====

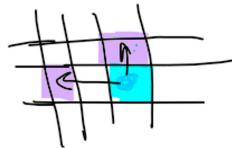
```

2      // O(n*C) time and space
3      int n = A.size(), i, j;
4      vector<int> dp(B+1, 0);
5
6      // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
7      // Base case
8      for(i = 0; i <= B; i++){
9          if(i%A[n-1] == 0)
10             dp[i] = 1;
11     }
12
13     for(i = n-2; i >= 0; i--){
14         for(j = 0; j <= B; j++){
15             if(j-A[i] >= 0)
16                 dp[j] = (dp[j] + dp[j-A[i]])%1000007;
17             else
18                 dp[j] = (dp[j])%1000007;
19         }
20     }
21
22     return dp[B];
23
24

```

$O(n)$
Space

- $O(n)$ solution since the nature/direction in which 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 can simply remove the first dimension here and the things will work out



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

```

1 int Solution::coinchange2(vector<int> &A, int B) {
2     // O(n*C) time and space
3     int n = A.size(), i, j;
4     vector<int> dp(B+1, 0);
5
6     // dp(i, j) = dp(i, j-A[i]) + dp(i+1, j)
7     // Base case
8     dp[0] = 1;
9
10    for(i = n-1; i >= 0; i--){
11        for(j = 0; j <= B; j++){
12            if(j-A[i] >= 0)
13                dp[j] = (dp[j] + dp[j-A[i]])%10000007;
14        }
15    }
16
17    return dp[B];
18
19
20 }
21

```

=====

- Q) Buy and sell stocks -> version:cool down -> can be done by $O(n)$ also (approach using dp):: <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/>

- Q) Palindrome partitioning ii

- o Soln for palindrome partitioning ii

```

132. Palindrome Partitioning II
Hard 1155 3% Add to List Share
Given a string s, partition s such that every substring of the partition is a palindrome.
Return the minimum cuts needed for a palindrome partitioning of s.
Example:
Input: "aab"
Output: 1
Explanation: The palindrome partitioning ["aa", "b"] could be produced using 1 cut.

Accepted 132,763 Submissions 440,455
Seen this question in a real interview before?
Yes No
Contributor
Companies
Related Topics
Similar Questions
Your previous code was restored from your local storage. Restore to default

```

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

```

class DISK{
    int size();
    char getData(int id);
    void swap(int id1, int id2);
    void defrag(DISK& d);
}
    
```

Solution:



Dynamic
Programm...

Audio recording started: 12:22 24 January 2022

4 metrics that interviewer were looking were:

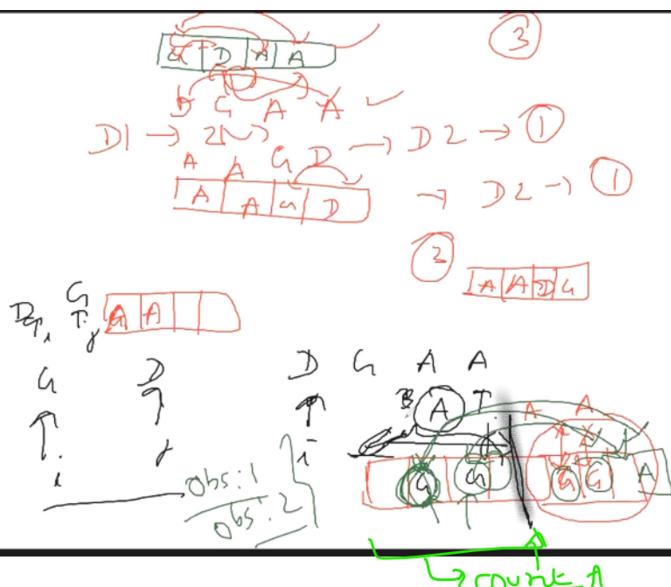
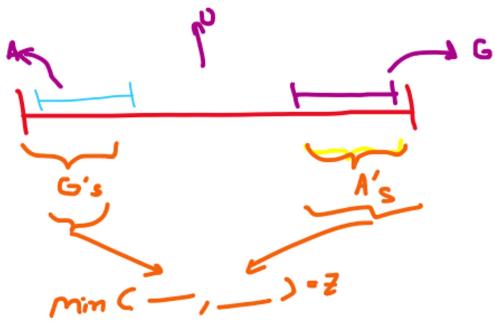
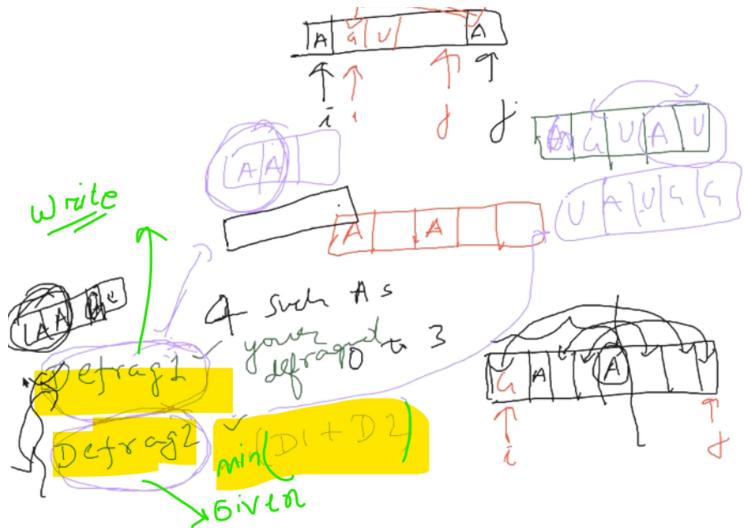
- A) preprocessing(count the no of A's)
- B) 2 pointers (they were expecting all would be able to say that)
- C) j won't be simply j++ but it will be the next non A character ka index => a loop ofr that also
- D) don't need to swap elements that are A and in the index range [0 ... (count_A - 1)]
- E)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

Follow up qn:



Dynamic
Programm...

Audio recording started: 13:19 24 January 2022



Soln::

Observation1: the G's that are in the range [0 ... count_A-1], we can use those to minimize sum

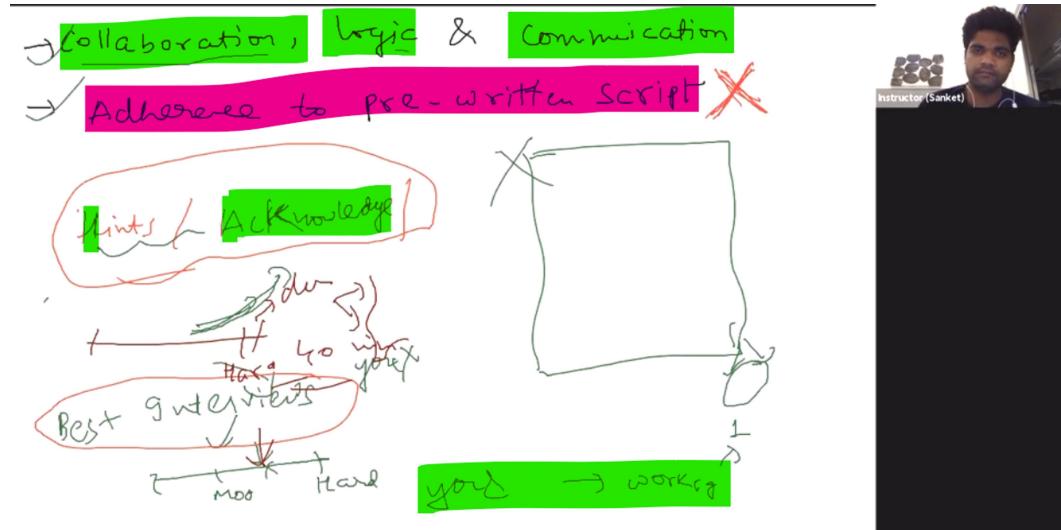
Observation 2:: the G's that appear in the above range can be swapped greedily with the right most A's

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]
- =====

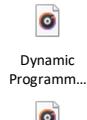
Interview pointers to keep in mind ::

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/>

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

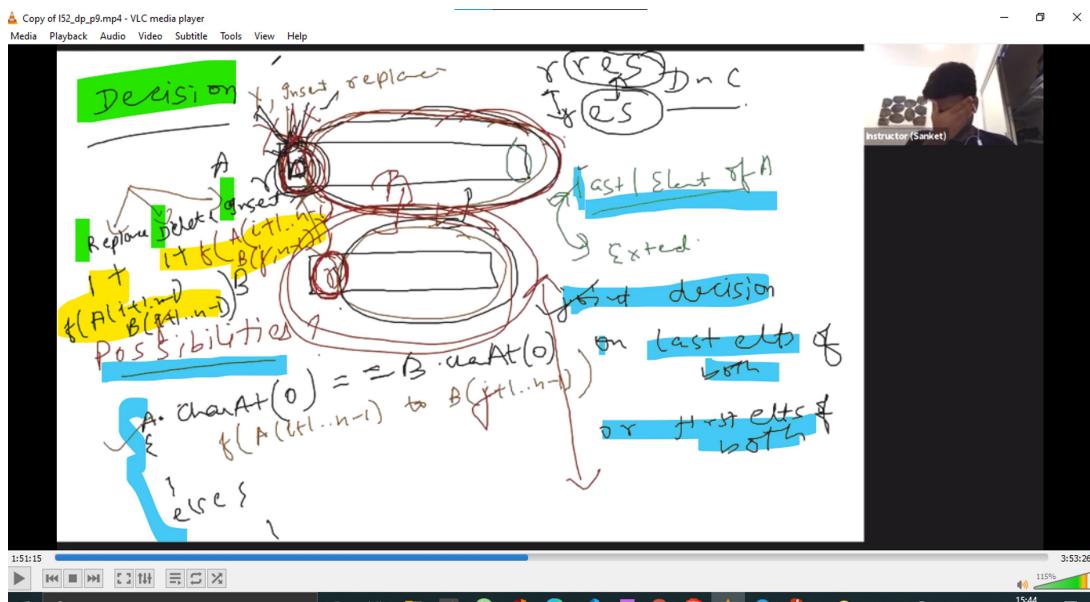
Explanation:

`horse` \rightarrow `orse` (replace 'h' with 'r')
`orse` \rightarrow `ore` (remove 's')
`ore` \rightarrow `os` (remove 'e')

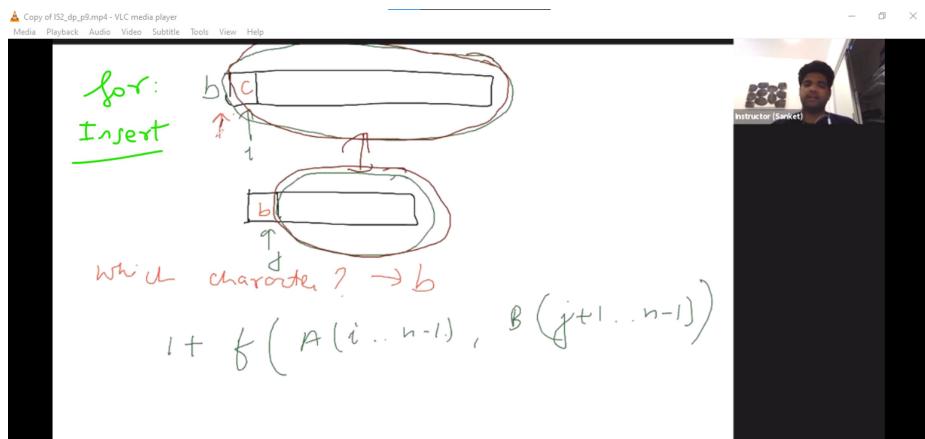
Example 2:

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

Output: 5



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



$f(i, j) \leftarrow$ min. no. of ops to convert
 $A[i \dots n-1] \text{ to } B[j \dots n-1]$

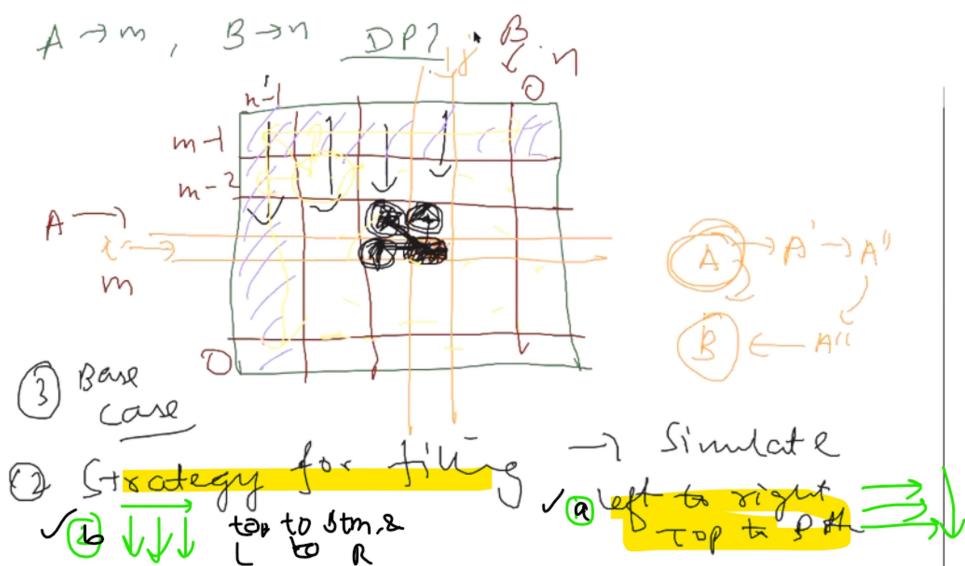
Recurrence

$$f(i, j) = \begin{cases} f(i+1, j+1) & \text{if } A[i] == B[j] \\ 1 + f(i+1, j+1) & \leftarrow \text{Replace} \\ 1 + f(i+1, j) & \leftarrow \text{Delete} \\ 1 + f(i, j+1) & \leftarrow \text{Insert} \end{cases}$$

$f(i, j) \leftarrow$ min. no. of ops

$$f(i, j) = \begin{cases} f(i+1, j+1) & \text{if } A[i] == B[j] \\ 1 + f(i+1, j+1) & \leftarrow \text{Replace} \\ 1 + f(i+1, j) & \leftarrow \text{Delete} \\ 1 + f(i, j+1) & \leftarrow \text{Insert} \end{cases}$$

We have overlapping subproblems \rightarrow dp as well



Base cases:

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

```

1 class Solution {
2 public:
3     int minDistance(string word1, string word2) {
4         //1->2
5         int n=word1.size(),m=word2.size(),i,j;
6         //dp(i,j) <- no of ways to convert word1[i ... n-1] to word2[j ... m-1]
7         vector<vector<int>> dp(n+1,vector<int>(m+1,0)); //dp(0,0) <- ans
8
9         //Base Case
10        //fill the buffer/padded row and column
11        for(j=0;j<=m;j++){// for i=n => word1 is empty
12            dp[n][j]=m-j; //only insertion operations
13        }
14        for(i=0;i<n;i++){ // for j=m => word2 is empty
15            dp[i][m]=n-i; //only deletion operations
16        }
17
18        //Recurrence Relation
19        //dp(i,j) = dp(i+1,j+1) ; if word1[i]==word2[j]
20        //      = min(1+dp(i,j+1),1+dp(i+1,j),1+dp(i+1,j+1) )
21        //          for insert,delete & replace operations respectively, if word1[i]!=word2[j]
22
23        //fill from right to left and bottom to up
24        for(i=n-1;i>=0;i--){
25            for(j=m-1;j>=0;j--){
26                if(word1[i]==word2[j]){
27                    dp[i][j]=dp[i+1][j+1];
28                }
29                else{
30                    dp[i][j]= 1 + min(dp[i][j+1],min(dp[i+1][j],dp[i+1][j+1]));
31                }
32            }
33        }
34        return dp[0][0];
35
36    // for(i=0;i<n;i++){
37    //     for(j=0;j<=m;j++)

```

Space optimized:

```

1 class Solution {
2 public:
3     int minDistance(string word1, string word2) {
4         //1->2
5         int n=word1.size(),m=word2.size(),i,j;
6         //dp(i,j) <- no of ways to convert word1[i ... n-1] to word2[j ... m-1]
7         vector<vector<int>> dp(2,vector<int>(m+1,0)); //dp(0,0) <- ans
8
9         //Base Case
10        //fill the buffer/padded row and column
11        for(j=0;j<=m;j++){// for i=n => word1 is empty
12            dp[1][j]=m-j; //only insertion operations
13        }
14        // for(i=0;i<2;i++) { // for j=m => word2 is empty
15        //     dp[1%2][m]=n-i; //only deletion operations
16        //}
17
18        //Recurrence Relation
19        //dp(i,j) = dp(i+1,j+1) ; if word1[i]==word2[j]
20        //      = min(1+dp(i,j+1),1+dp(i+1,j),1+dp(i+1,j+1) )
21        //          for insert,delete & replace operations respectively, if word1[i]!=word2[j]
22
23        //fill from right to left and bottom to up
24        for(i=n-1;i>=0;i--){
25            dp[1%2][m]=n-i; // **NOTE: we have to update the 1st column for each row so it will have to be inside loop
26            for(j=m-1;j>=0;j--){
27                if(word1[i]==word2[j]){
28                    dp[1%2][j]=dp[(i+1)%2][j+1];
29                }
30                else{
31                    dp[1%2][j]= 1 + min(dp[1%2][j+1],min(dp[(i+1)%2][j],dp[(i+1)%2][j+1]));
32                }
33            }
34        }
35        return dp[0][0];//NOT: ans will always be at 0,0 only
36        // n3,1 -> n2,0 -> n1,1 -> n0,0
37
38    // for(i=0;i<n;i++){
39    //     for(j=0;j<=m;j++)
40    //         cout<<dnf1111l<< " ";

```

we have to shift
Base for col here

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)
 - o Eg its used in google search ka : Did you mean ____ ?
 - o 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

=====

Largest 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:

```
1 * class Solution {
2 *     public:
3 *         int longestCommonSubsequence(string text1, string text2) {
4 *             // Space Optimized :: O(2^n)
5 *             int n=text1.length(),m=text2.length(),i,j;
6 *             // combinatorical => DnC
7 *             // Recursive Reln
8 *             // f(i,j) <- lcs of text1(i ... n-1) & text2(j .. m-1)
9 *
10 *             // f(i,j) = 1 + f(i+1,j+1) ;if(text1[i]==text2[j])
11 *             //      = max(f(i,j+1),f(i+1,j)) ; otherwise
12 *             // repeating subpblm => dp
13 *             vector<vector<int>> dp(2,vector<int> (m+1,0));
14 *             //Base Case
15 *             // Padding min : last row and last col will be 0
16 *             // (since,if either text1 or text2 is empty => length of lcs=0)
17 *
18 *             // for(i=0;i<n;i++){
19 *             //     //j=m => text2 is empty
20 *             //     dp[1][m]=0;
21 *             // }
22 *             // for(j=0;j<m;j++) dp[n][j]=0;
23 *
24 *             for(i=n-1;i>=0;i--){
25 *                 for(j=m-1;j>=0;j--){
26 *                     if(text1[i]==text2[j]) dp[i%2][j]=1+dp[(i+1)%2][j+1];
27 *                     else dp[i%2][j]=max(dp[(i+1)%2][j],dp[i%2][j+1]);
28 *                 }
29 *             }
30 *             return dp[0][0];
31 *
32 *         }
33 *     };
}
```

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](#)

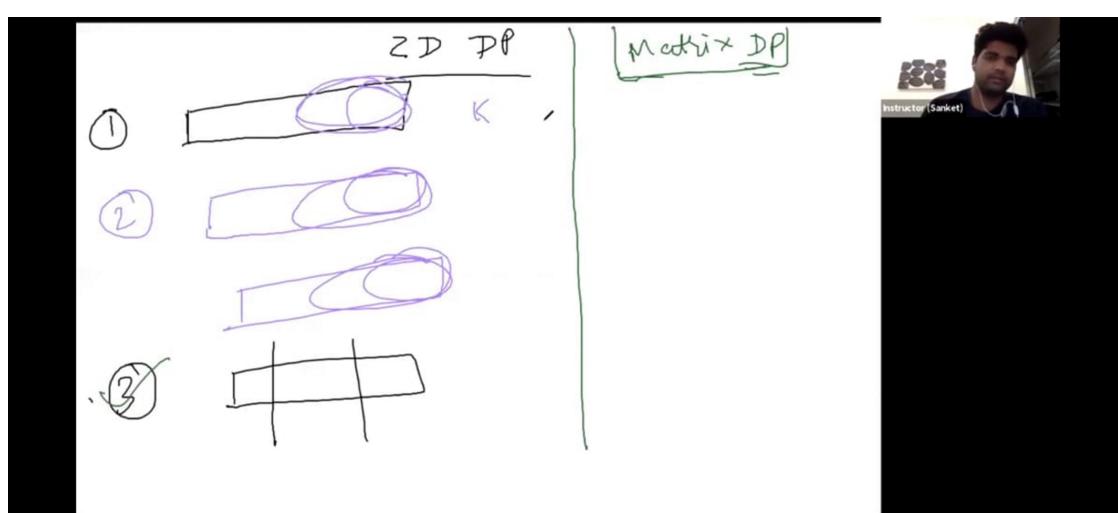
From [https://leetcode.com/problems/longest-common-subsequence/discuss/348884/C%2B%2B-with-picture-O\(nm\)>](https://leetcode.com/problems/longest-common-subsequence/discuss/348884/C%2B%2B-with-picture-O(nm)>)

Similar Questions

LONGEST PALINDROMIC SUBSEQUENCE	Medium
DELETE OPERATION FOR TWO STRINGS	Medium
SHORTEST COMMON SUPERSEQUENCE	Hard

- Distinct subsequences problem (sanket sir)

Lec 10



m	$m-1$	$m-2$	0	
0	$+$	2	3	n
$m-1$	$\cancel{1}$	$\cancel{2}$	$\cancel{3}$	\cancel{n}
1				
m				

$a_{m,n} = f(0,0)$

$A \quad D$

$B \quad O$

$D \quad O$

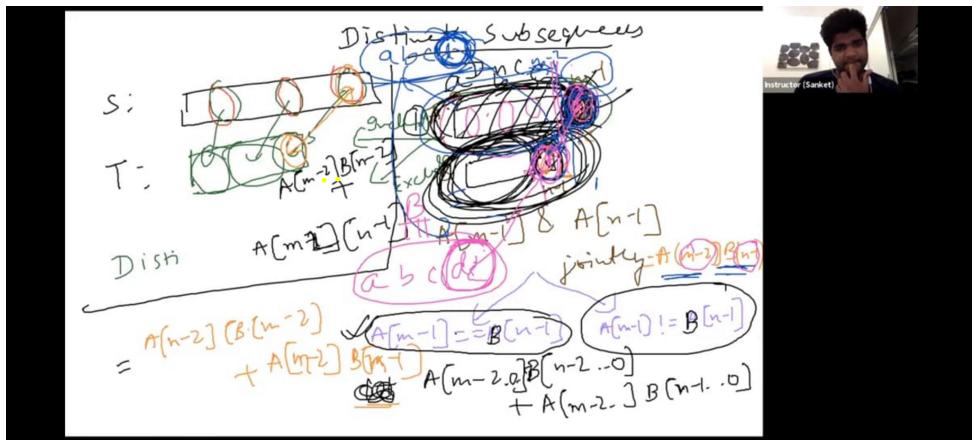
$B \quad A$

$(m+1) \times (n+1)$

Distinct Subsequences: <https://leetcode.com/problems/distinct-subsequences/>

Solution: <https://leetcode.com/problems/distinct-subsequences/submissions/>

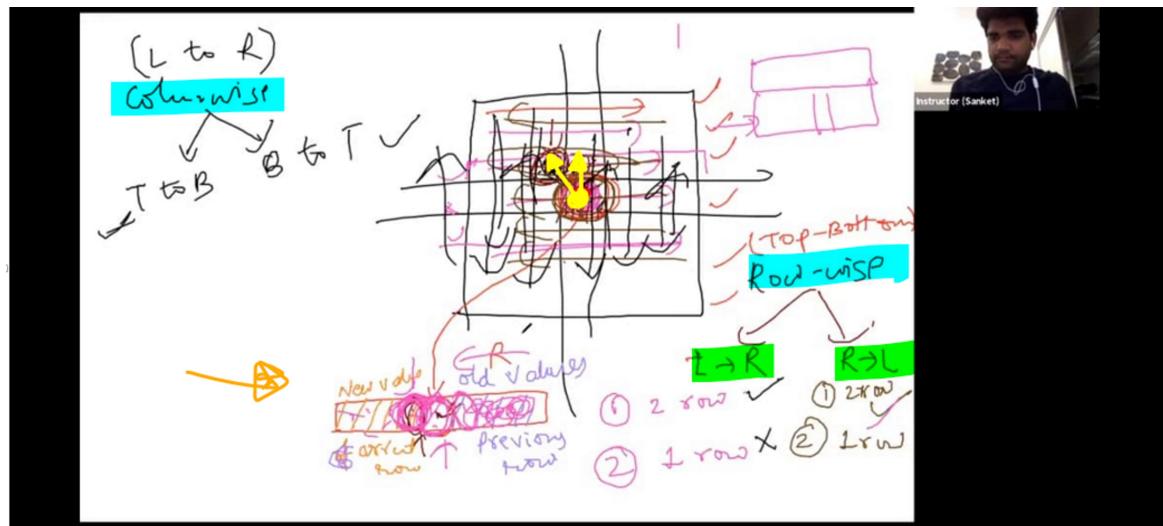
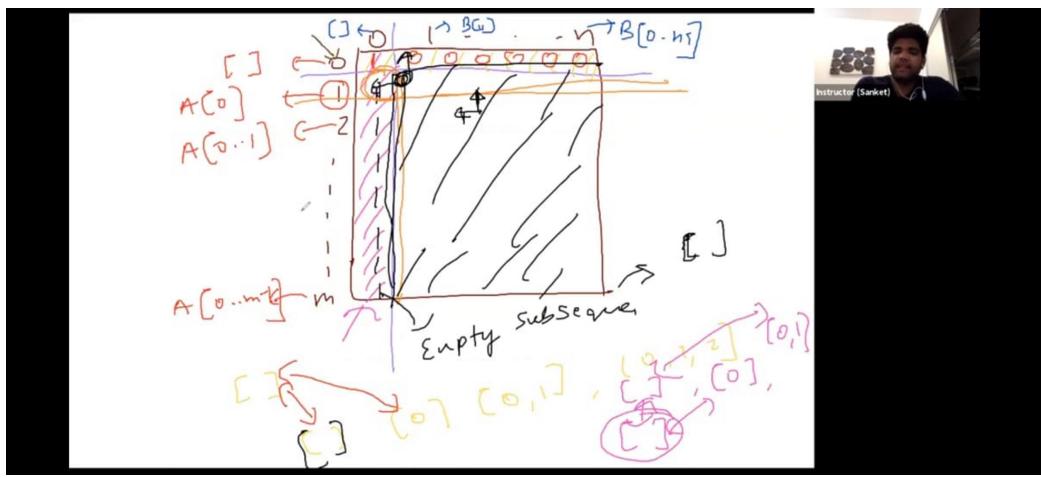
(add this to github & in DP notes!!)



$dp(i)(j) \leftarrow \text{No. of distinct subsequences of } A[0..i] \text{ which are equal to } B[0..j]$

$dp(i)(j) = \begin{cases} dp(i-1)(j) & \text{if } A(i) \neq B(j) \\ dp(i-1)(j-1) + dp(i-1)(j) & \text{if } A(i) == B(j) \end{cases}$

Instructor (Sanket)



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	Bcz 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/

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

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
```

Pick One < Prev 115/2151 Next >

web backend tab trum Built Chr Tab chr CRX chrome chrome Pro Unr http:// http:// go INT +

leetcode.com/problems/distinct-subsequences/submissions/

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

LeetCode Explore Problems Interview Contest Discuss Store

Description Solution Discuss (965) Submissions

C++ Autocomplete

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

```

1+ class Solution {
2+ public:
3+     int numDistinct(string s, string t) {
4+         // IMPORTANT LESSON : HERE WHEN YOU RUN IT THE ANSWER OVERFLOWS
5+         // despite it being given in the question that Final Ans fits in 32 bit int )(read at the end)
6+
7+         // Space :: O(n*m) (can be space optimized)
8+         // combinatorial/counting pbm => DnC /recursive pbm
9+         // f(i,j) <- no of distinct subsequences of s[0... i] which equals t[0... j]
10+        // Recurrence Relation
11+        // f(i,j) = f(i-1,j) : if(s[i]==t[j])
12+        // f(i,j) = f(i-1,j-1) + f(i-1,j) : if(s[i]==t[j])
13+        int i,j=n-s.length(),m=t.length();
14+        vector<vector<int>> dp(n+1,vector<int> (m+1,0));
15+        // Base Case :: f(0,0)
16+        // i=0 => only s[0]
17+        if(s[0]==t[0]) dp[0][0]=1;
18+        for(i=1;i<n;i++){
19+            for(j=1;j<m;j++){
20+                for(j1=j+1;j1<=m;j1+=1){
21+                    if(s[i]==t[j1]) dp[i][j] = dp[i-1][j];
22+                    else dp[i][j] = INT_MAX;
23+                    if(dp[i][j]<INT_MAX-dp[i-1][j-1] ? dp[i-1][j]-dp[i-1][j-1] : INT_MAX);
24+                    else min(INT_MAX,dp[i-1][j-1]+dp[i-1][j]);
25+                }
26+            }
27+        }
28+        return dp[n][m];
29+    }
30+ }
31+ /**
32+ * IMPORTANT LESSON :
33+ * HERE WHEN YOU RUN IT THE ANSWER OVERFLOWS (runtime error occurs)
34+ * (due to it being given in the question that Final Ans fits in 32 bit int )
35+ */

```

Your previous code was restored from your local storage. [Reset to default](#)

https://leetcode.com/problems/distinct-subsequences/

< Pick One < Prev 115/2151 Next > Console Contribute i Run Code ^ Submit

Longest Palindromic subsequence:

← → ↻ leetcode.com/problems/longest-palindromic-subsequence/

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:

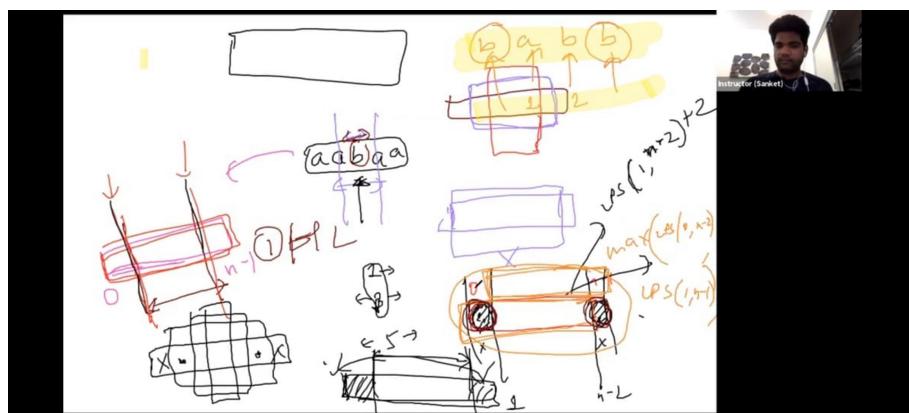
```
Input: s = "bbbab"
Output: 4
Explanation: One possible longest palindromic subsequence is "bbbb".
```

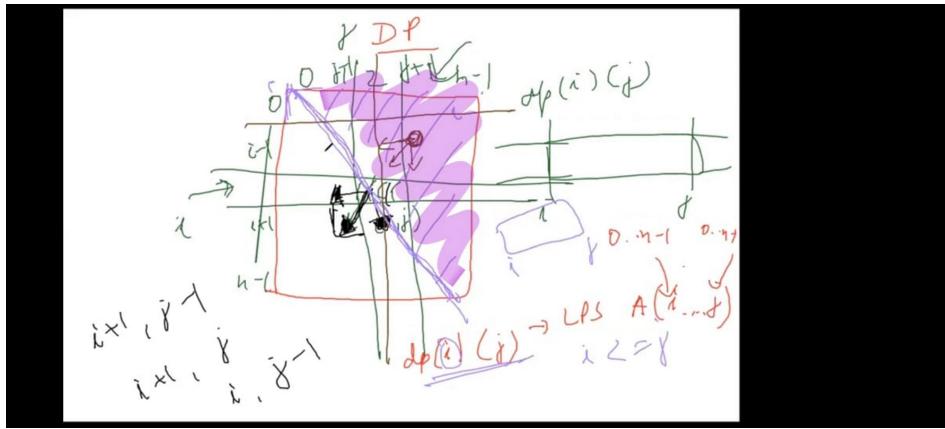
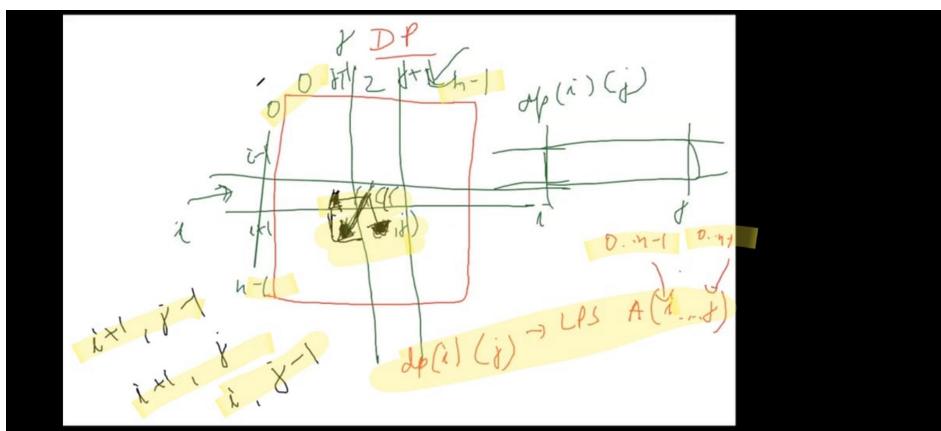
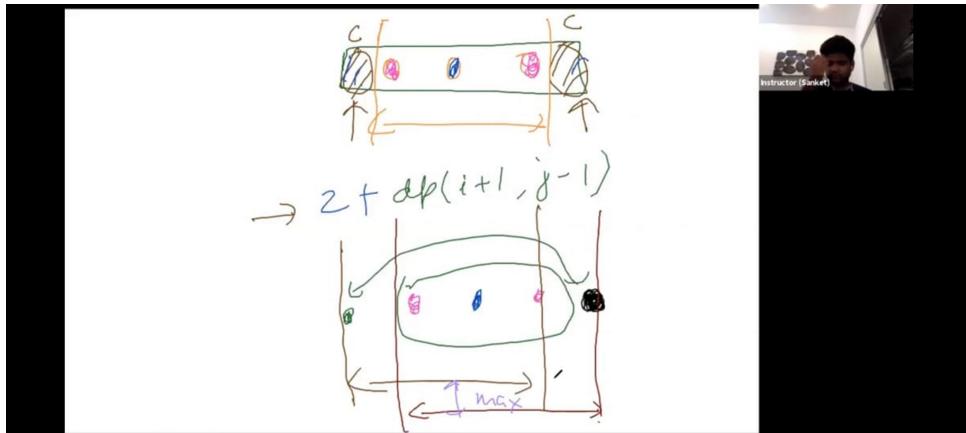
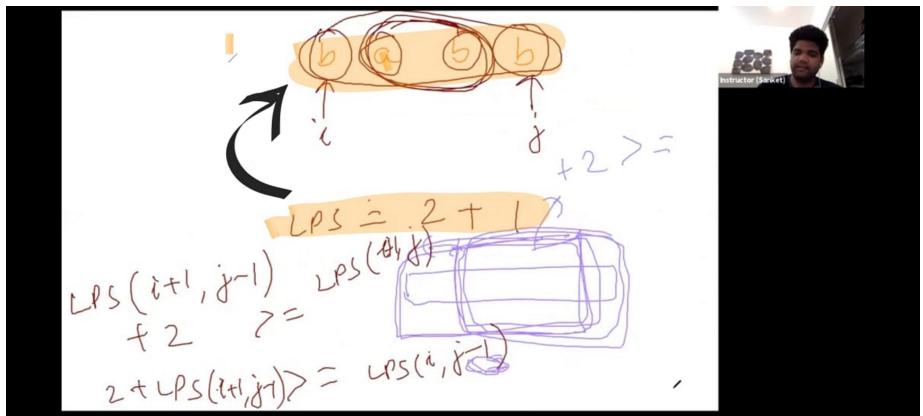
Example 2:

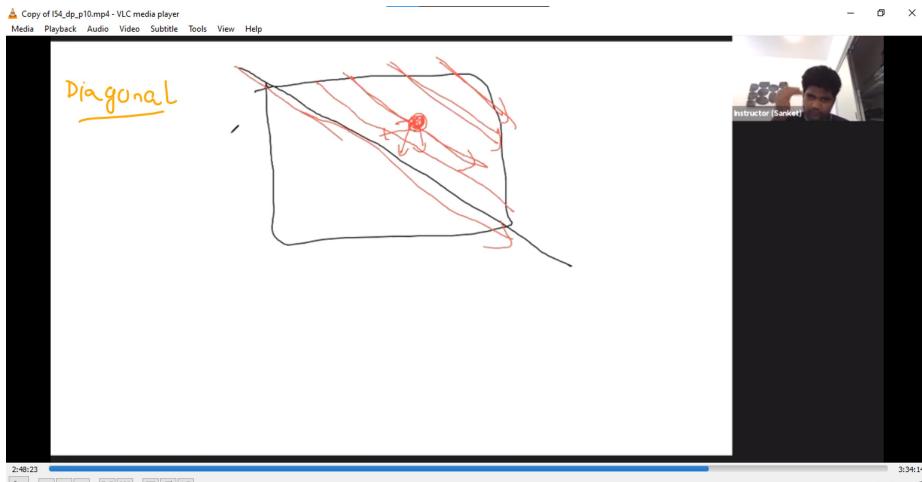
```
Input: s = "cbbd"
Output: 2
Explanation: One possible longest palindromic subsequence is "bb".
```

Constraints:

- $1 \leq s.length \leq 1000$
 - s consists only of lowercase English letters.







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/

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

i C++ Autocomplete

```

1 class Solution {
2 public:
3     /* Approach 1: s = original & s' = reverse of s, find the LCS(s,s') ; LCS=longest common subsequence
4      * Approach 2: expand palindrome around the center */
5     int longestPalindromeSubseq(string s) {
6         // Solution for Approach 2
7         // Optimization probm (since longest) => can think of DnC (Divide and Conquer) approach
8         // Here our subproblem can be thought as this: for LPS(s[0 ... n-1])
9         // if(s[0]==s[n-1]) then LPS(s[0 ... n-1]) = 2+LPS(s[1 ... n-2])
10        // else LPS(s[0 ... n-1]) = max(LPS(s[1 ... n-1]), LPS(s[0 ... n-2]))
11
12        // form above its clear that our subproblem will be any general subarray of string s => s[i ... j]
13        // => we have 2 variables here i & j (start & end indexes of any subarray)
14        int i,j,n=s.length();
15        vector<vector<int>> dp(n,vector<int>(n,0));
16
17        //Recurrence Relation
18        // dp(i,j) <- length of LPS in s[i ... j]
19        // dp(i,j) = dp(i+1)[j-1] : if(s[i]==s[j])
20        //                   = max(dp[i][j-1],dp[i+1][j]) : if(s[i]!=s[j])
21
22        //Base Case
23        for(i=0;i<n;i++) dp[i][i]=1;
24
25        for(i=n-2;i>=0;i--) //bottom to up
26            for(j=i+1;j<n;j++){//left to right
27                if(s[i]==s[j]) dp[i][j]=2+dp[i+1][j-1];
28                else{
29                    dp[i][j]=max(dp[i+1][j],dp[i][j-1]);
30                }
31            }
32        //dp[0][n-1] :: final ans
33        return dp[0][n-1];
34    }
35}
```

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

```

1 class Solution {
2 public:
3     int longestPalindromeSubseq(string s) {
4         int n = s.size();
5         vector<int> dp(n, 0), dpPrev(n, 0);
6         for (int i = n - 1; i >= 0; --i) {
7             dp[i] = 1;
8             for (int j = i+1; j < n; ++j) {
9                 if (s[i] == s[j]) {
10                     dp[j] = dpPrev[j-1] + 2;
11                 } else {
12                     dp[j] = max(dpPrev[j], dp[j-1]);
13                 }
14             }
15             dp.swap(dpPrev);
16         }
17         return dpPrev[n-1];
18     }
19 }
```

- 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

```
1 class Solution {
2 public:
3     int longestPalindromeSubseq(string s) {
4         int n = s.size();
5         vector<int> dp(n, 0), dpPrev(n, 0);
6         for (int i = n - 1; i >= 0; --i) {
7             dp[i] = 1;
8             for (int j = i+1; j < n; ++j) {
9                 if (s[i] == s[j]) {
10                     dp[j] = dpPrev[i+1] + 2;
11                 } else {
12                     dp[j] = max(dpPrev[j], dp[j-1]);
13                 }
14             }
15             dp.swap(dpPrev);
16         }
17         return dpPrev[0];
18     }
19 }
```

O

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...



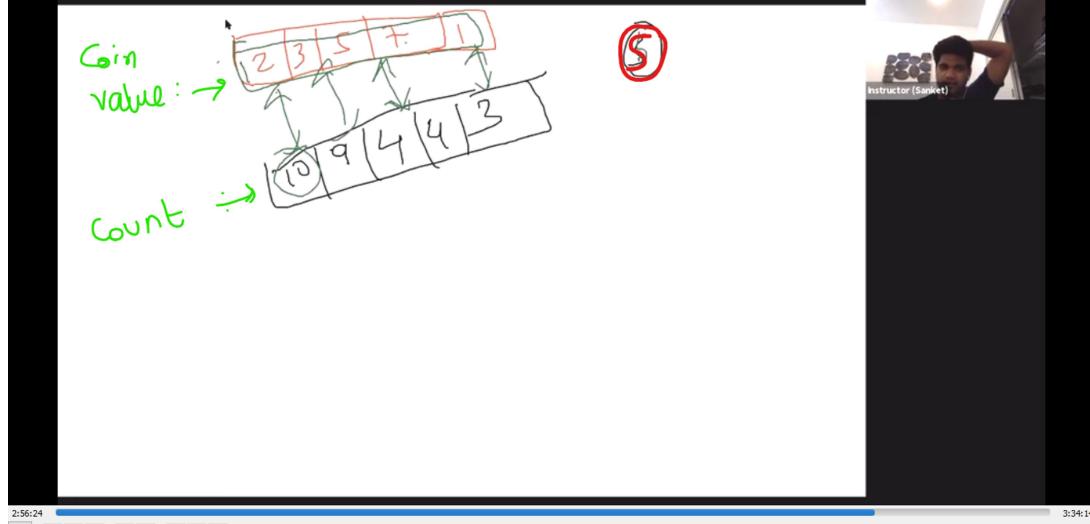
Dynamic
Programm...

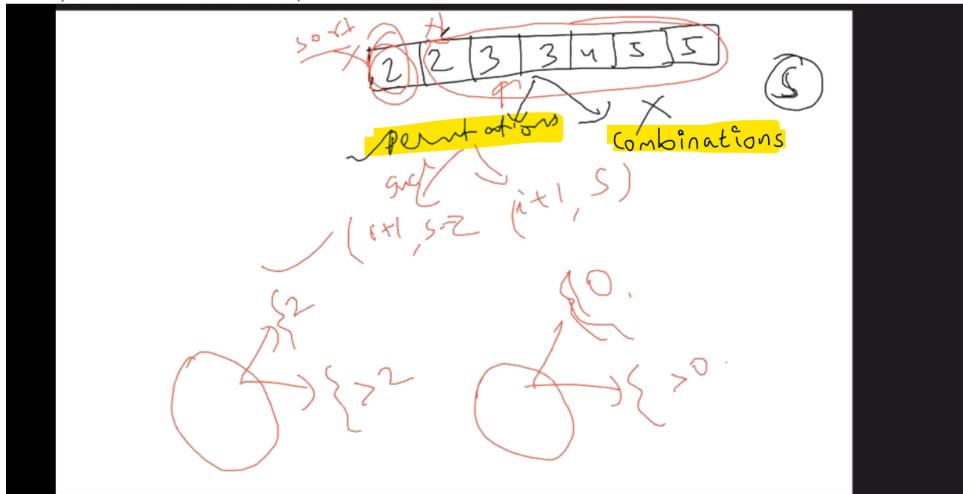
Audio recording started: 18:49 26 January 2022

Audio recording started: 18:47 26 January 2022

⚠ Copy of l54_dp_p10.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help





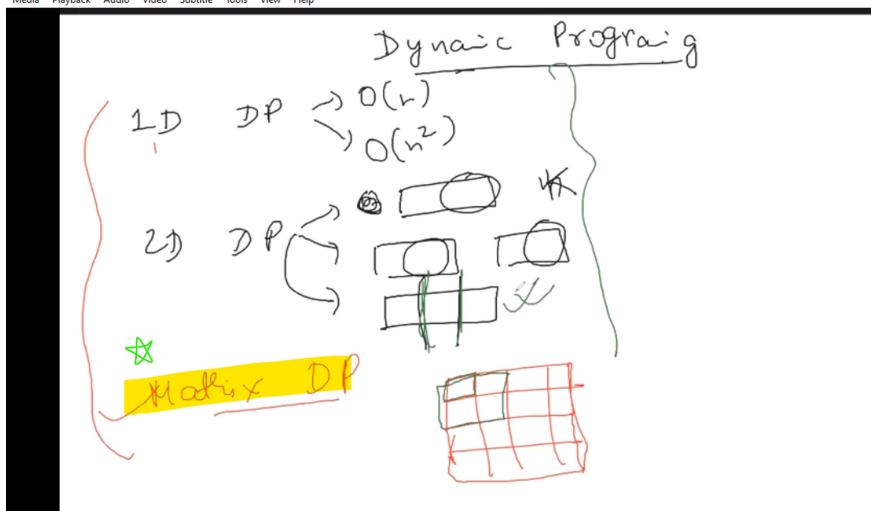
Slack questions:

- Wildcard 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 :
 - o [Bridges in Graph | Cut Edge](#)
 - o <https://cp-algorithms.com/graph/bridge-searching.html>



leetcode.com/problems/minimum-path-sum/

Apps Building Modern Web Guide to write an ar... Workspace Log in | Innovation... Problem setting gui... Apply | CodeChef cs50.harvard

LeetCode Explore Problems Interview Contest Discuss Store

Description Solution Discuss (999+) Submissions Autocomplete

64. Minimum Path Sum

Medium 6525 91 Add to List Share

Given a $m \times n$ grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:

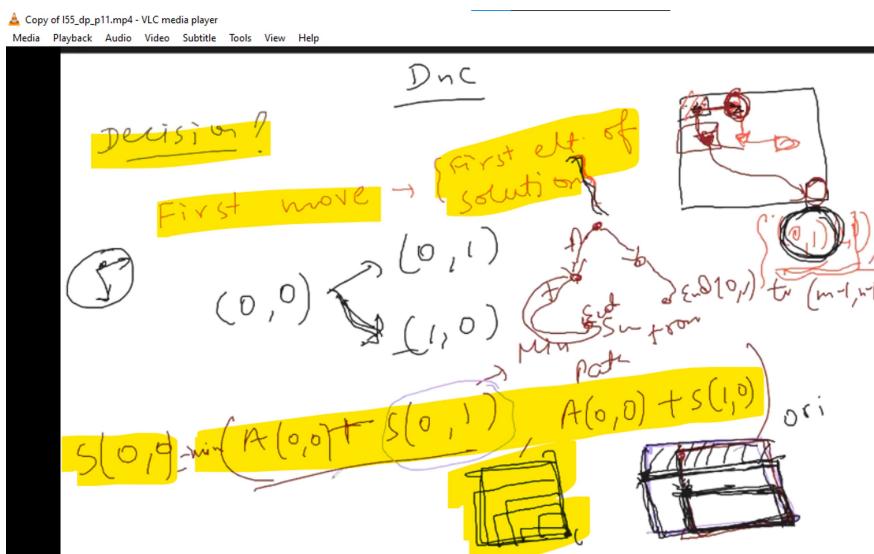
1	3	1
1	5	1
4	2	1

Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7

```

1* class Solution {
2*     public:
3*         int minPathSum(vector<vector<int>>& grid) {
4*             ...
5*         }
6* };

```



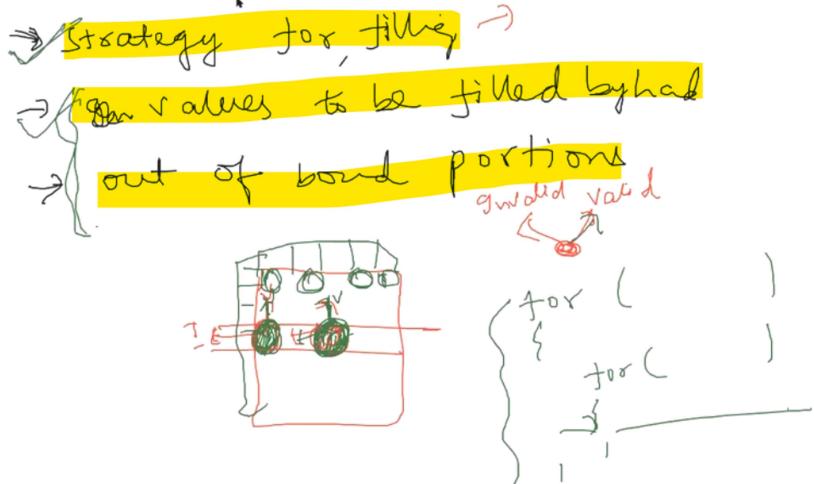
Here subproblem is suffix matrices

$$S(0,0) = A(0,0) + \min(S(0,1), S(1,0))$$

→ suffix Matrix

Simulate table with Row-wise (L to R) and Col-wise (n to m)

Strategy for $dpl(i,j) = A(i,j) + \min(\text{split}(i,k), \text{dp}(i,m))$



leetcode.com/problems/minimum-path-sum/submissions/

Success Details >

Runtime: 7 ms, faster than 89.66% of C++ online submissions for Minimum Path Sum.

Memory Usage: 9.8 MB, less than 68.23% of C++ online submissions for Minimum Path Sum.

Next challenges:

- Unique Paths
- Cherry Pickup
- Maximum Number of Points with Cost

Show off your acceptance: [f](#) [t](#) [in](#)

Time Submitted	Status	Runtime	Memory	Language
01/27/2022 03:06	Accepted	7 ms	9.8 MB	cpp

```

1+ class Solution {
2+ public:
3+     int minPathSum(vector<vector<int>>& grid) {
4+         int m=grid.size(),n=grid[0].size(),i,j,x=-INT_MAX;
5+         vector<vector<int>> dp(2,vector<int>(n+1));
6+         //Note : Here out of bound values will have path sum as infinity(INT_MAX)
7+         // { as its not possible to reach from those from to (m-1,n-1) }
8+
9+         for(i=m-1;i>=0;i--){
10+             for(j=n-1;j>=0;j--){
11+                 if(i==m-1 & j==n-1){ dp[i][j]=grid[i][j];continue; }
12+                 dp[i][j]=grid[i][j]+min(dp[(i+1)%2][j],dp[i%2][j+1]);
13+             }
14+         }
15+         return dp[0][0];
16+     }
17+ };

```

Dungeon game:

if $(A(0,0) \geq 0) \quad d\mu(0,1) \quad d\mu(1,0)$
 $d\mu(0,0) = \min(\max(1, t_1 - A(0,0)), \max(1, t_2 - A(0,0)))$
 else
 $\{$
 $d\mu(0,0) = \min(t_1 - A(0,0), t_2 - A(0,0))$
 $\}$

Sir's soln:

174. Dungeon Game

The demons had captured the princess (P) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of $M \times N$ rooms laid out in a 2D grid. Our valiant knight (K) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (negative integers) upon entering these rooms; other rooms are either empty (0's) or contain magic orbs that increase the knight's health (positive integers).

In order to reach the princess as quickly as

```
1+ class Solution {
2+ public:
3+     int calculateMinimumHP(vector<vector<int>>& dungeon) {
4+         int m = dungeon.size(), n = dungeon[0].size(), i, j;
5+
6+         vector<vector<int>> dp(m+1, vector<int>(n+1, INT_MAX));
7+
8+         dp[m-1][n-1] = dungeon[m-1][n-1] >= 0 ? 1 - dungeon[m-1][n-1];
9+
10+        for(i = m-1; i > 0; i--){
11+            for(j = n-1; j >= 0; j--){
12+                if(i == m-1 && j == n-1) continue;
13+
14+                if(i+1 < m)
15+                    dp[i][j] = min(dp[i][j], max(1, dp[i+1][j]-dungeon[i][j]));
16+
17+                if(j+1 < n)
18+                    dp[i][j] = min(dp[i][j], max(1, dp[i][j+1]-dungeon[i][j]));
19+
20+            }
21+
22+        }
23+
24+        return dp[0][0];
25+    }
26+}
```

My solution :

Runtime: 6 ms, faster than 71.92% of C++ online submissions for Dungeon Game.

Memory Usage: 9 MB, less than 17.13% of C++ online submissions for Dungeon Game.

Next challenges:

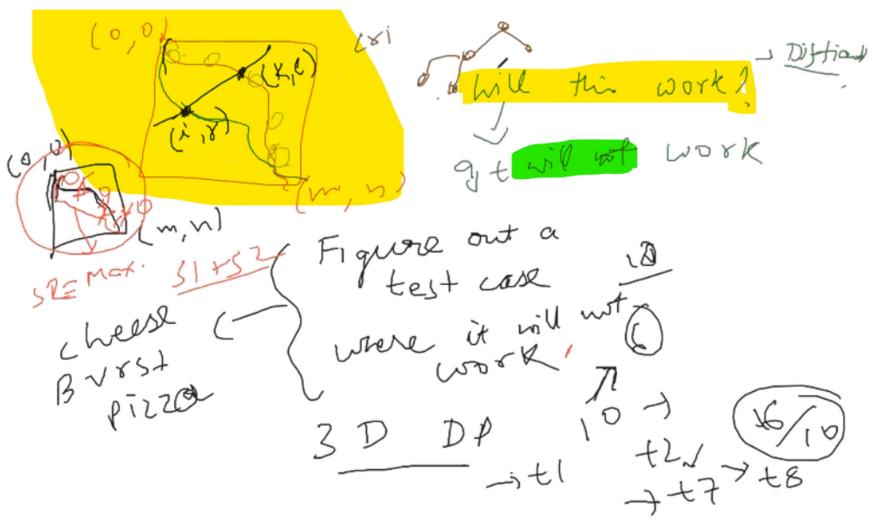
[Unique Paths](#) [Cherry Pickup](#)

Show off your acceptance:   

Time Submitted	Status	Runtime	Memory	Language
01/27/2022 15:53	Accepted	6 ms	9 MB	cpp
01/27/2022 15:50	Runtime Error	N/A	N/A	cpp
01/27/2022 15:48	Wrong Answer	N/A	N/A	cpp
01/14/2022 13:23	Accepted	8 ms	8.9 MB	cpp

```
class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {
        //optimisation problem - DnC can be thought of
        // dp[i,j] <- minimum energy needed to go from dungeon[i][j] to dungeon[n-1][m-1]
        int i,j,n=dungeon.size(),m=dungeon[0].size(); //n=3000000
        vector<vector<int>> dp(n+1,vector<int>(m+1,0));
        
        //Reurrence Relation
        //dp[i][j] = min(dp[i+1][j], dungeon[i][j] - dp[i][j+1] - dungeon[i][j])
        dp[0][0]=1; //dungeon[0][0]-1;
        if(dp[0][0]<1){dp[0][0]=1;} //energy you need to pass dungeon[0][0]=energy you want to pass dungeon[0][0] - energy you get at dungeon[0][0]
        for(i=1;i<n;i++){
            for(j=0;j<m;j++){
                if(i==n-1 && j==m-1){continue;
                    dp[i][j]=min(dp[i+1][j],dungeon[i][j] , dp[i][j+1]-dungeon[i][j]);
                    dp[i][j]= dp[i][j]>1?dp[i][j]:1; // since minimum health of price has to be 1 (otherwise he dies)
                }
            }
        }
        return dp[0][0];
    }
}; // [-3,5]
```

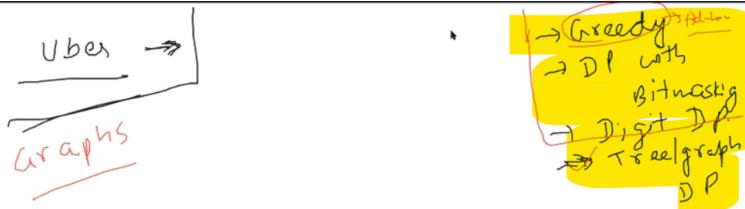
<->Q) Cherry pickup (hard) (Uber) TO DO



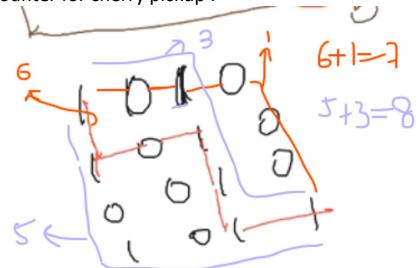
Dynamic Programm...

Audio recording started: 17:08 27 January 2022

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



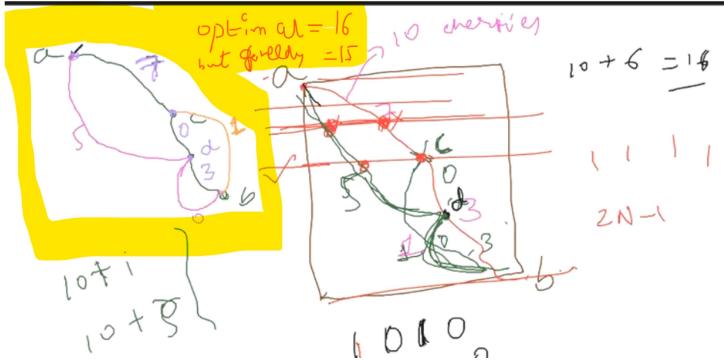
Counter for cherry pickup :



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**
 - o 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/>
 - o 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](#)

```
#include <bits/stdc++.h>
using namespace std;
int dp[3000][3000];
// dp[x][y] -> wall string main x index par hain, t wall string main y index par... max length
int main(){
    string s;
    cin>>s;
    int n = s.size(), m = t.size();
    dp[0][0] = 0;
    for(int i = 0 ; i < n; i++){
        for(int j = 0 ; j < m; j++){
            // not picking current character
            if(i > 0){
                dp[i][j] = dp[i-1][j];
            }
            if(j > 0){
                if(dp[i][j-1] > dp[i][j]){
                    dp[i][j] = dp[i][j-1];
                }
            }
            // pick current character
            if(s[i] == t[j]){
                int ans = dp[i-1][j-1];
                if((i>0)&&(j>0)){
                    ans = 1 + dp[i-1][j-1];
                }
                if(ans > dp[i][j]){
                    dp[i][j] = ans;
                }
            }
        }
    }
    cout<<dp[n-1][m-1];
}
```

- Lcs with the LCS string also :

C:\Users\kingui\Desktop\yo.cpp - Sublime Text (UNREGISTERED)

Longest Common Subsequence and Recovering the path of Optimal Choices | Atcoder DP Conte...

```
#include <bits/stdc++.h>
using namespace std;
int dp[3000][3000];
// dp[i][j] = wali string main x index par hain, t wali string mein y index par... max length
int choice[3000][3000];
// 0 | {i-1,j} , i->{i,j-1} , 2 -> pick and goto {i-1,j-1}
int main(){
    string s,t;
    cin>>s;
    cin>>t;
    int n = s.size(), m = t.size();
    dp[0][0] = 0;
    if(s[0] == t[0]){
        dp[0][0] = 1;
        choice[0][0] = 2;
    }
    for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < m ; j++){
            // not picking current character
            if(i > 0){
                if(dp[i-1][j] == dp[i-1][j]){
                    choice[i][j] = 0;
                }
                if(j > 0){
                    if(dp[i][j-1] > dp[i][j]){
                        dp[i][j] = dp[i][j-1];
                        choice[i][j] = 1;
                    }
                }
                // pick current character
                if(s[i] == t[j]){
                    int ans = 1;
                    if(i>0 && j>0){
                        ans = 1 + dp[i-1][j-1];
                    }
                    if(ans > dp[i][j]){
                        dp[i][j] = ans;
                        choice[i][j] = 2;
                    }
                }
            }
        }
    }
    int i = n-1, j = m-1;
    string subsequence;
    while(i >= 0 && j >= 0){
        if(choice[i][j] == 0){
            i--;
        }
        else if(choice[i][j] == 1){
            j--;
        }
        else{
            subsequence.push_back(s[i]); // or t[j]
            i--;
            j--;
        }
    }
    reverse(subsequence.begin(), subsequence.end());
    cout<<subsequence;
}
```

C:\Users\kingui\Desktop\yo.cpp - Sublime Text (UNREGISTERED)

```
File Edit Selection Find View Goto Tools Project Preferences Help
```

```
#include <bits/stdc++.h>
using namespace std;
int dp[3000][3000];
choice[3000][3000];
} // pick current character
if(s[i] == t[j]){
    int ans = 1;
    if(i>0 && j>0){
        ans = 1 + dp[i-1][j-1];
    }
    if(ans > dp[i][j]){
        dp[i][j] = ans;
        choice[i][j] = 2;
    }
}
}
int i = n-1, j = m-1;
string subsequence;
while(i >= 0 && j >= 0){
    if(choice[i][j] == 0){
        i--;
    }
    else if(choice[i][j] == 1){
        j--;
    }
    else{
        subsequence.push_back(s[i]); // or t[j]
        i--;
        j--;
    }
}
reverse(subsequence.begin(), subsequence.end());
cout<<subsequence;
```