11/01/2024

DYNAMIC PROGRAMMING
CLASS - 4

1. Longest Common Subsequence (Leetcode-1143)

EXAMPLE 1

Str1 = "ABC"

Str2 = "DEF"

Output = 0

Explanation

	ABC
✓✗✗	A
✗✓✗	B
✗✗✓	C
✓✓✗	AB
✓✗✓	AC
✗✓✓	BC
✓✓✓	ABC
✗✗✗	""

	DEF
✓✗✗	D
✗✓✗	E
✗✗✓	F
✓✓✗	DE
✓✗✓	DF
✗✓✓	EF
✓✓✓	DEF
✗✗✗	""

Longest common subsequence = ""

= length ⇒ 0

EXAMPLE 2

str1 = "ABC"

str2 = "ABCD"

Output = 3

Explanation

A B C	
✓✗✗	A
✗✓✗	B
✗✗✓	C
✓✓✗	AB
✓✗✓	AC
✗✓✓	BC
✓✓✓	$\boxed{ABC}$
✗✗✗	""

A B C D	
✓✗✗✗	A
✗✓✗✗	B
✗✗✓✗	C
✗✗✗✓	D
✓✓✗✗	AB
✓✗✓✗	AC
✓✗✗✓	AD
✗✓✓✗	BC
✗✓✗✓	BD
✗✗✓✓	CD
✓✓✓✗	$\boxed{ABC}$
✓✓✗✓	ABD
✓✗✓✓	ACD
✗✓✓✓	BCD
✓✓✓✓	ABCD
✗✗✗✗	""

Longest common subsequence is
ABC ⇒ 3

Approach 1: Recursion

Str1 A B C

Str2 X Y Z

CASE1 Does match the character of Both strings

एक length Add करदो

$\Rightarrow 1 + REC$

CASE2 DOES NOT MATCH The characters of Both strings

↳ we have two choice

First rejecting the character from str1

~~A~~ B C

X Y Z

Second rejecting the character from str2

ABC

~~X~~YZ

जीरो length Add करदो

$\Rightarrow 0 + \max($First, second$)$

Final output

↳ max(case1 , case2)

Find Base case

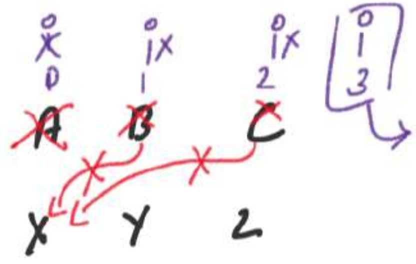
str1 A B C
0 1 2
(i above A)

str2 X Y Z
0 1 2
(j above X)

output $\boxed{0}$

when i=0 and j=0
↳ Not match

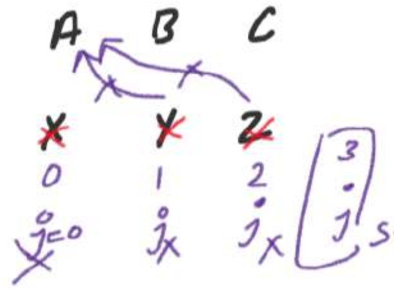
First



→ stop and return 0

Base
if(i >= str1.length)
return 0;

Second



stop and return 0

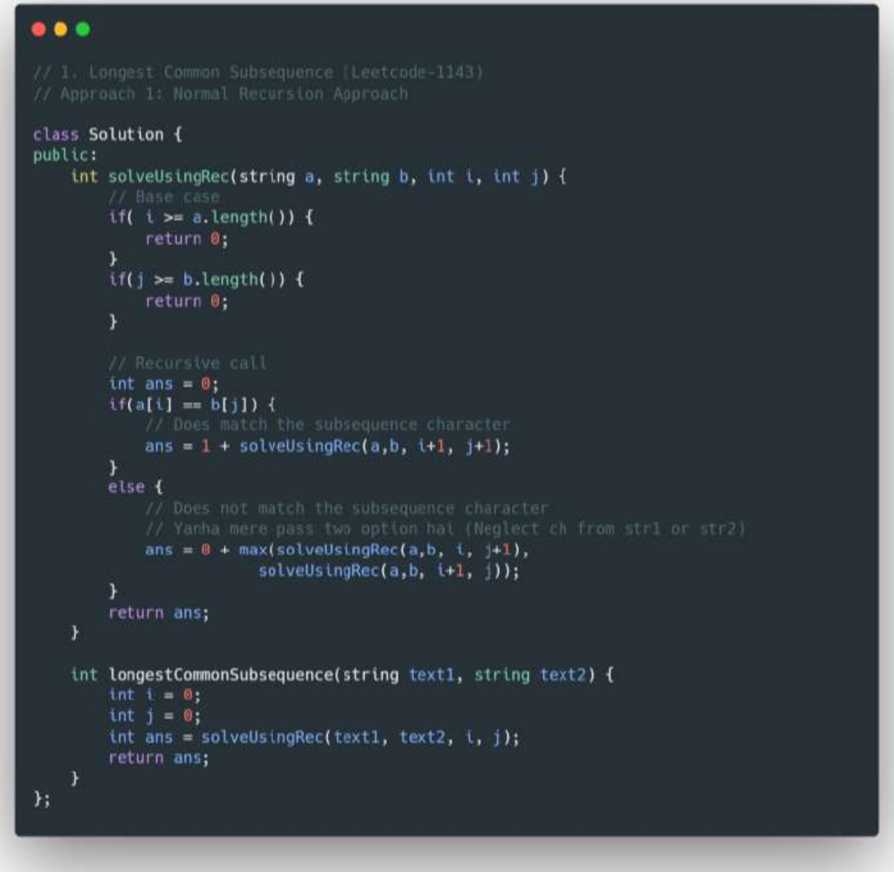
Base
if(j >= str2.length)
return 0;

⇒ 0 + max(0,0)

⇒ 0+0

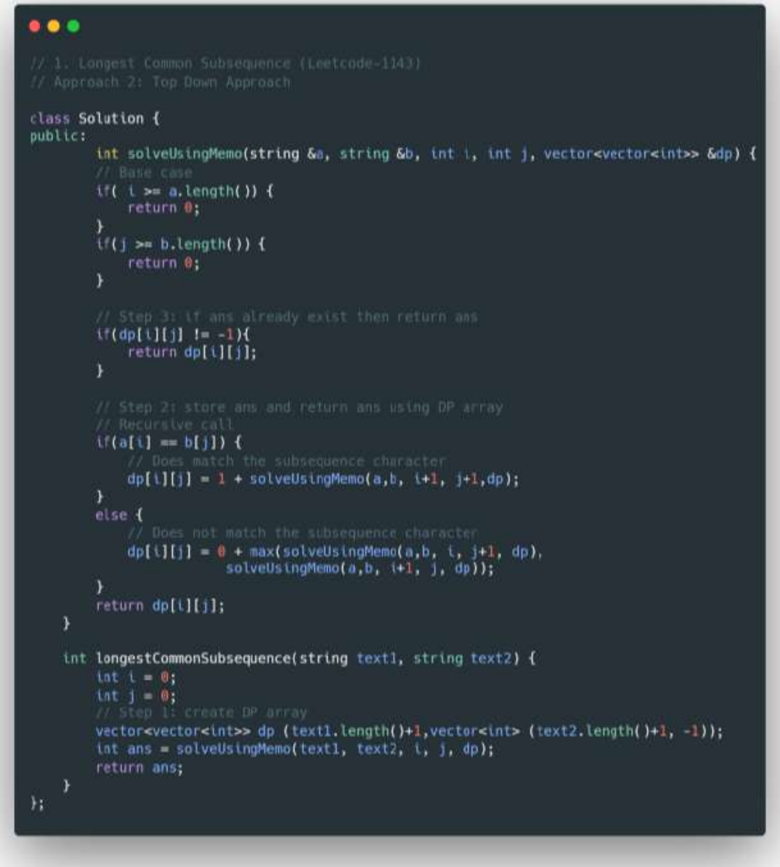
⇒ 0 output

Approach 1: Recursion



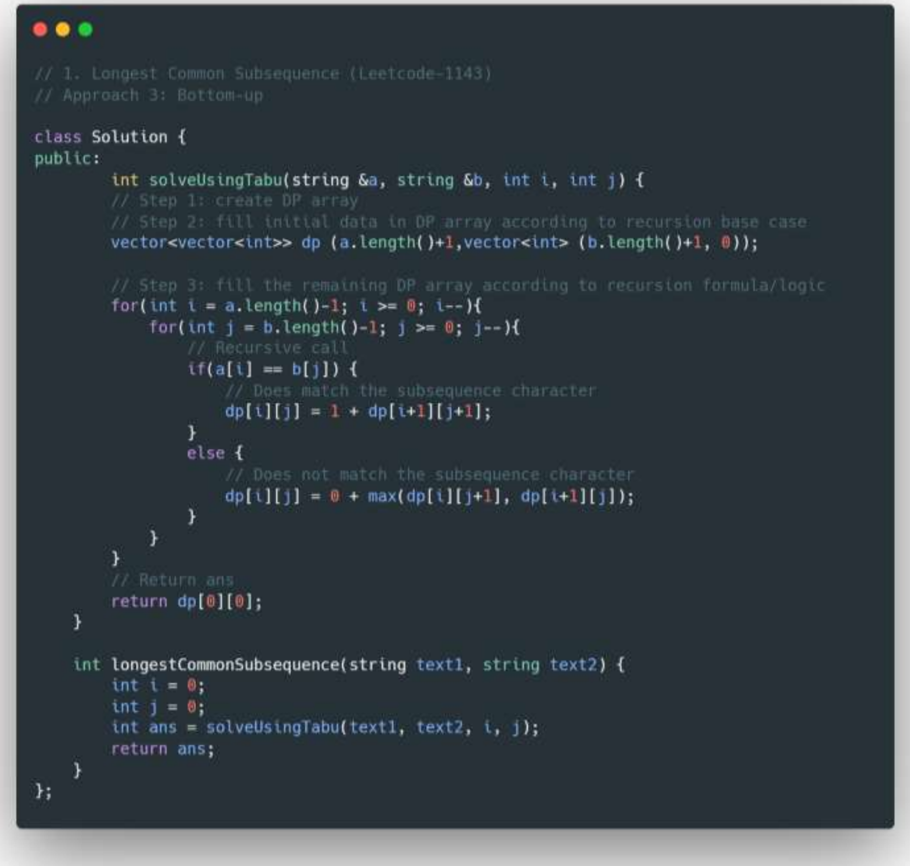
```cpp
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 1: Normal Recursion Approach

class Solution {
public:
    int solveUsingRec(string a, string b, int i, int j) {
        // Base case
        if( i >= a.length()) {
            return 0;
        }
        if(j >= b.length()) {
            return 0;
        }

        // Recursive call
        int ans = 0;
        if(a[i] == b[j]) {
            // Does match the subsequence character
            ans = 1 + solveUsingRec(a,b, i+1, j+1);
        }
        else {
            // Does not match the subsequence character
            // Yanha mere pass two option hai (Neglect ch from str1 or str2)
            ans = 0 + max(solveUsingRec(a,b, i, j+1),
                          solveUsingRec(a,b, i+1, j));
        }
        return ans;
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingRec(text1, text2, i, j);
        return ans;
    }
};
```

Approach 2: Top Down

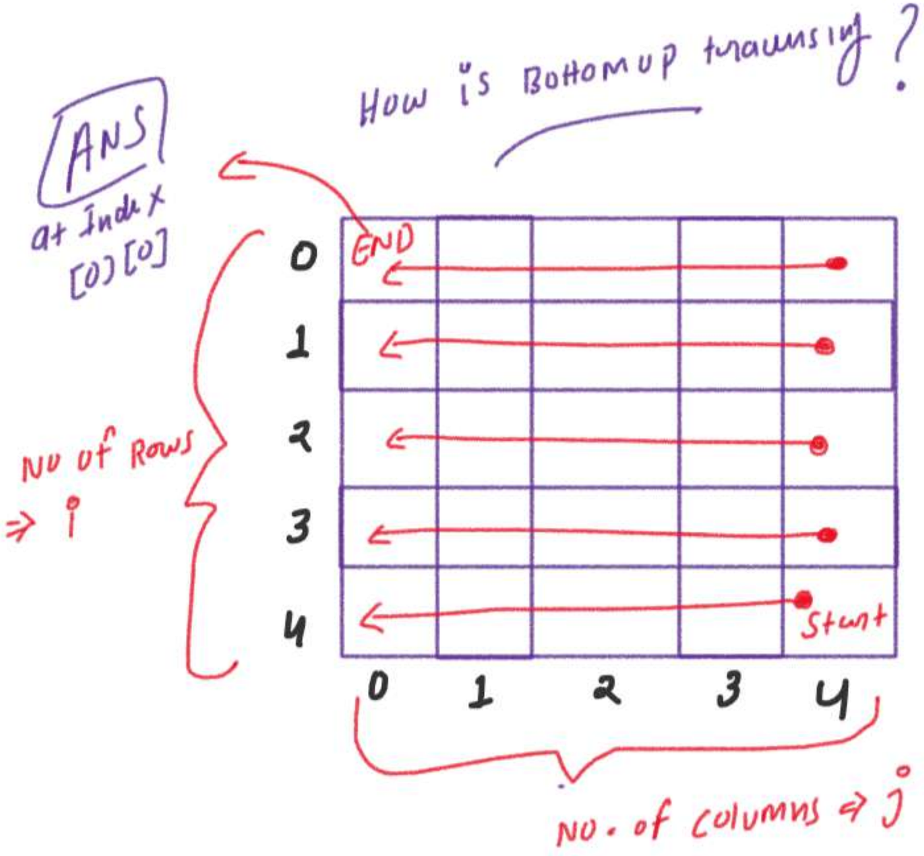


```cpp
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 2: Top Down Approach

class Solution {
public:
        int solveUsingMemo(string &a, string &b, int i, int j, vector<vector<int>> &dp) {
        // Base case
        if( i >= a.length()) {
            return 0;
        }
        if(j >= b.length()) {
            return 0;
        }

        // Step 3: if ans already exist then return ans
        if(dp[i][j] != -1){
            return dp[i][j];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive call
        if(a[i] == b[j]) {
            // Does match the subsequence character
            dp[i][j] = 1 + solveUsingMemo(a,b, i+1, j+1,dp);
        }
        else {
            // Does not match the subsequence character
            dp[i][j] = 0 + max(solveUsingMemo(a,b, i, j+1, dp),
                            solveUsingMemo(a,b, i+1, j, dp));
        }
        return dp[i][j];
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        // Step 1: create DP array
        vector<vector<int>> dp (text1.length()+1,vector<int> (text2.length()+1, -1));
        int ans = solveUsingMemo(text1, text2, i, j, dp);
        return ans;
    }
};
```

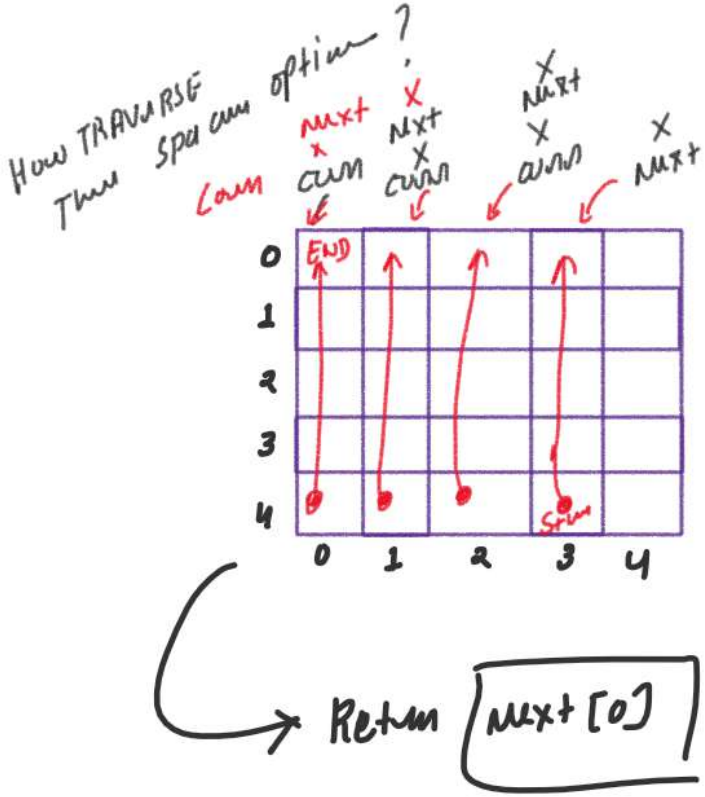
Approach 3: Bottom Up

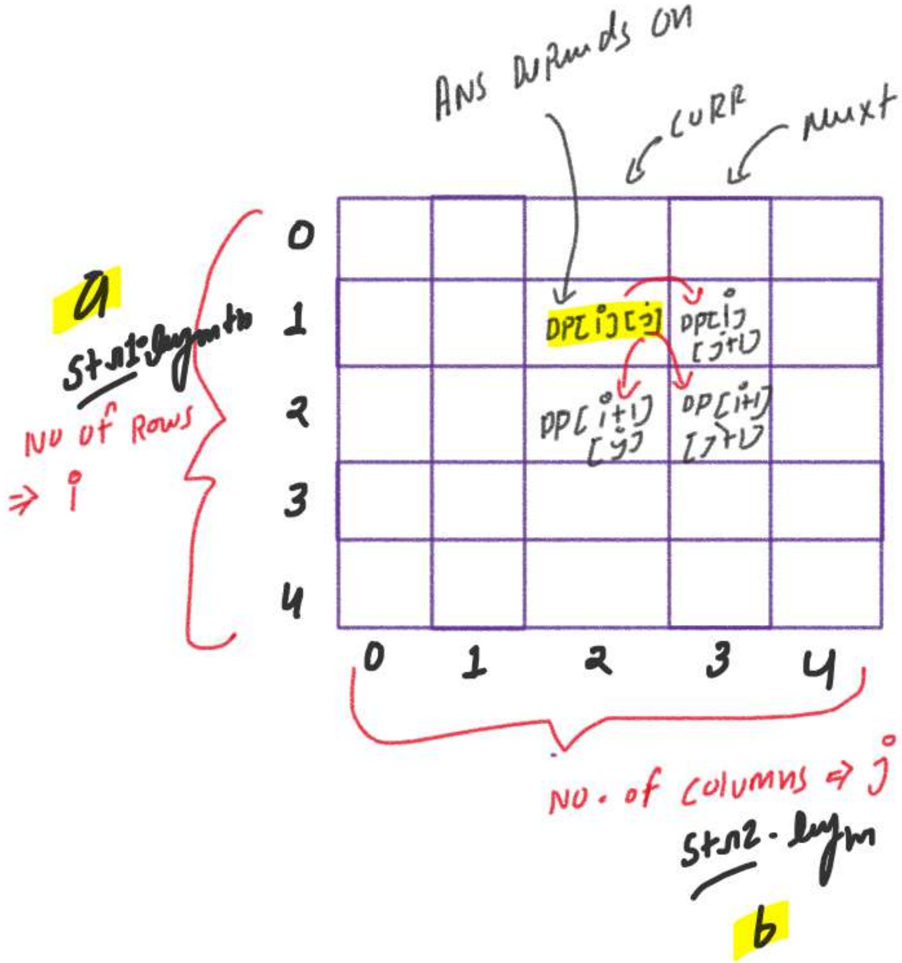


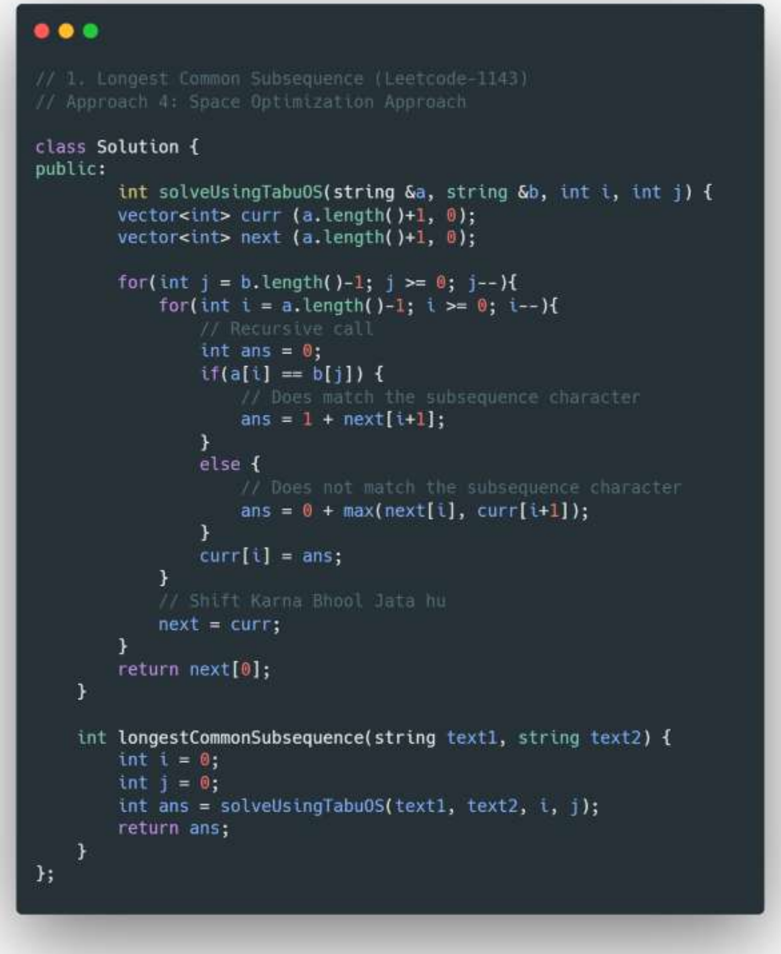
```
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 3: Bottom-up

class Solution {
public:
        int solveUsingTabu(string &a, string &b, int i, int j) {
        // Step 1: create DP array
        // Step 2: fill initial data in DP array according to recursion base case
        vector<vector<int>> dp (a.length()+1,vector<int> (b.length()+1, 0));

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int i = a.length()-1; i >= 0; i--){
            for(int j = b.length()-1; j >= 0; j--){
                // Recursive call
                if(a[i] == b[j]) {
                    // Does match the subsequence character
                    dp[i][j] = 1 + dp[i+1][j+1];
                }
                else {
                    // Does not match the subsequence character
                    dp[i][j] = 0 + max(dp[i][j+1], dp[i+1][j]);
                }
            }
        }
        // Return ans
        return dp[0][0];
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabu(text1, text2, i, j);
        return ans;
    }
};
```



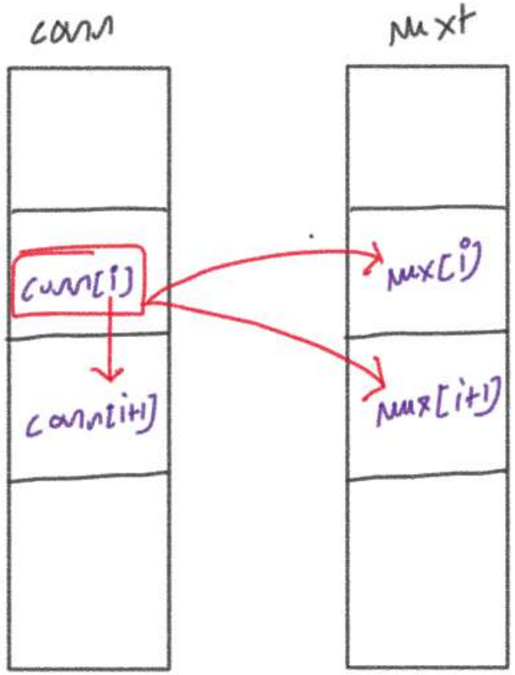
Approach 4: Space Optimization



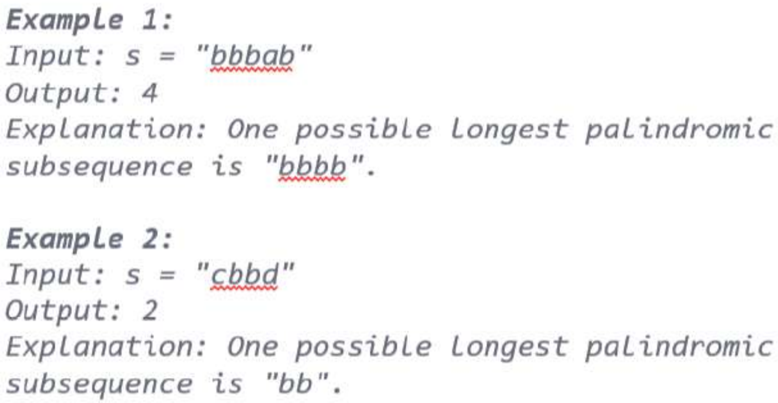




```
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 4: Space Optimization Approach

class Solution {
public:
        int solveUsingTabuOS(string &a, string &b, int i, int j) {
        vector<int> curr (a.length()+1, 0);
        vector<int> next (a.length()+1, 0);

        for(int j = b.length()-1; j >= 0; j--){
            for(int i = a.length()-1; i >= 0; i--){
                // Recursive call
                int ans = 0;
                if(a[i] == b[j]) {
                    // Does match the subsequence character
                    ans = 1 + next[i+1];
                }
                else {
                    // Does not match the subsequence character
                    ans = 0 + max(next[i], curr[i+1]);
                }
                curr[i] = ans;
            }
            // Shift Karna Bhool Jata hu
            next = curr;
        }
        return next[0];
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabuOS(text1, text2, i, j);
        return ans;
    }
};
```



📂 2. Longest Palindrome Subsequence (Leetcode-516)



```
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".
```

Approach

- ↳ store given string in text2
- ↳ reverse given string in text1
- ↳ Apply longest common subsq. Approach

↓

[we will get Ans]

DRY RUN

S

B	B	B	A	B	
✓	✗	✗	✗	✗	B
✗	✓	✗	✗	✗	B
✗	✗	✓	✗	✗	B
✗	✗	✗	✓	✗	A
✗	✗	✗	✗	✓	B
✓	✓	✗	✗	✗	BB
✓	✗	✓	✗	✗	BB
✓	✗	✗	✓	✗	BA
✓	✗	✗	✗	✓	BB
✗	✓	✓	✗	✗	BB
✗	✓	✗	✓	✗	BA
✗	✓	✗	✗	✓	BB
✗	✗	✓	✓	✗	BA
✗	✗	✓	✗	✓	BB
✗	✗	✗	✓	✓	AB
✓	✓	✓	✗	✗	ABB
✓	✓	✗	✓	✗	BBA
✓	✓	✗	✗	✓	BBB
✓	✓	✓	✓	✗	BBBA
✓	✓	✓	✗	✓	BBBB
✓	✓	✗	✓	✓	BBAB
✓	✗	✓	✓	✓	DBAB
✗	✓	✓	✓	✓	BBAB

✓	✓	✓	✓	✓	BBBAB
✗	✗	✗	✗	✗	" "

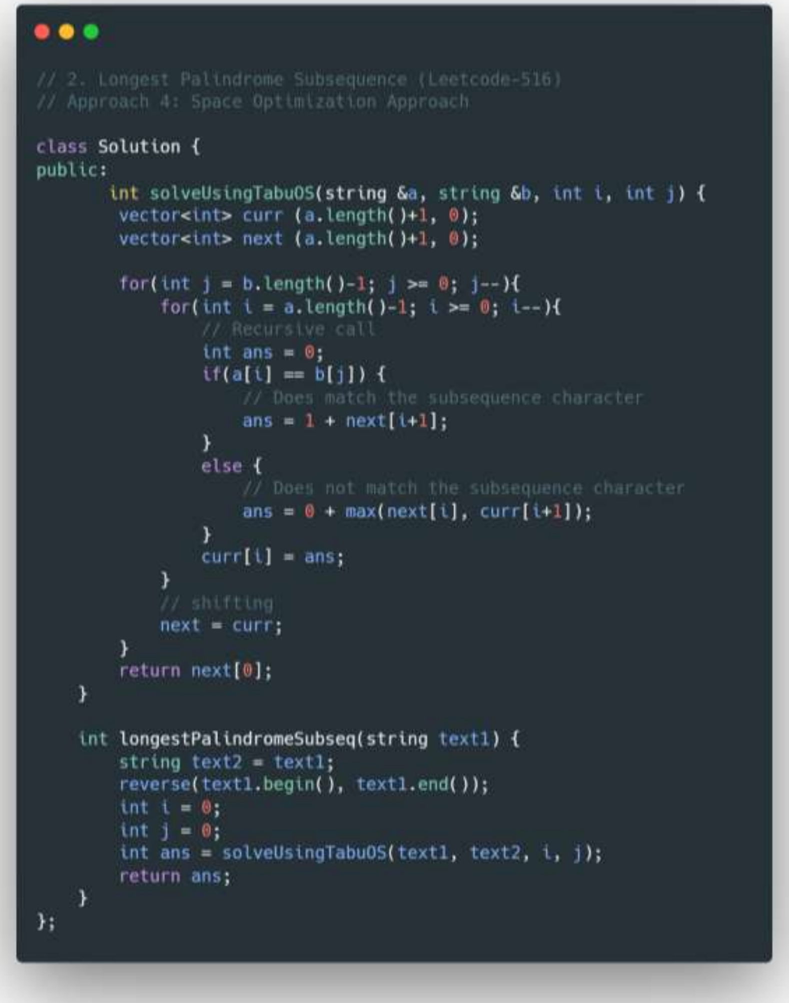
REVERSE

B	A	B	B	B	
✓	✗	✗	✗	✗	B
✗	✓	✗	✗	✗	A
✗	✗	✓	✗	✗	B
✗	✗	✗	✓	✗	B
✗	✗	✗	✗	✓	B
✓	✓	✗	✗	✗	BA
✓	✗	✓	✗	✗	BB
✓	✗	✗	✓	✗	BB
✓	✗	✗	✗	✓	BB
✗	✓	✓	✗	✗	AB
✗	✓	✗	✓	✗	AB
✗	✓	✗	✗	✓	AB
✗	✗	✓	✓	✗	BB
✗	✗	✓	✗	✓	BB
✗	✗	✗	✓	✓	BB
✓	✓	✓	✗	✗	BAB
✓	✓	✗	✓	✗	BAB
✓	✓	✗	✗	✓	BAB
✓	✓	✓	✓	✗	BABB
✓	✓	✓	✗	✓	BABB
✓	✓	✗	✓	✓	BABB
✓	✗	✓	✓	✓	BBBB
✗	✓	✓	✓	✓	ABBB

✓	✓	✓	✓	✓	BABBB
✗	✗	✗	✗	✗	" "

S= BBBB

↳ longest pallindrom subsequence



```
// 2. Longest Palindrome Subsequence (Leetcode-516)
// Approach 4: Space Optimization Approach

class Solution {
public:
       int solveUsingTabuOS(string &a, string &b, int i, int j) {
        vector<int> curr (a.length()+1, 0);
        vector<int> next (a.length()+1, 0);

        for(int j = b.length()-1; j >= 0; j--){
            for(int i = a.length()-1; i >= 0; i--){
                // Recursive call
                int ans = 0;
                if(a[i] == b[j]) {
                    // Does match the subsequence character
                    ans = 1 + next[i+1];
                }
                else {
                    // Does not match the subsequence character
                    ans = 0 + max(next[i], curr[i+1]);
                }
                curr[i] = ans;
            }
            // shifting
            next = curr;
        }
        return next[0];
    }

    int longestPalindromeSubseq(string text1) {
        string text2 = text1;
        reverse(text1.begin(), text1.end());
        int i = 0;
        int j = 0;
        int ans = solveUsingTabuOS(text1, text2, i, j);
        return ans;
    }
};
```

What is palindrome ?

LOVE → EVOL ✗

MOM → MOM ✓

CAR → RAC ✗

RAR → RAR ✓

RR → RR ✓

📂 3. Edit Distance (Leetcode-72)

Problem Statement:
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:

1. Insert a character
2. Delete a character
3. Replace a character

Example

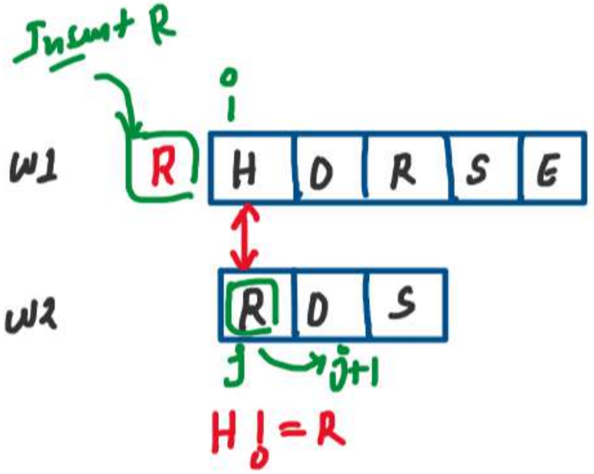
WORD1 = HORSE

WORD2 = ROS

Output 3

$$\begin{tikzcd} \text{WORD1} \arrow[r, bend left] & \text{WORD2} \arrow[l, bend left] \end{tikzcd}$$

Insert

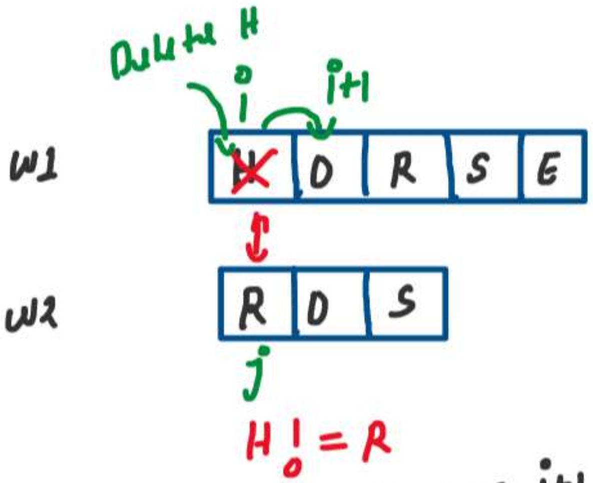


insertOpt = 1 + f(w1, w2, i, j+1);

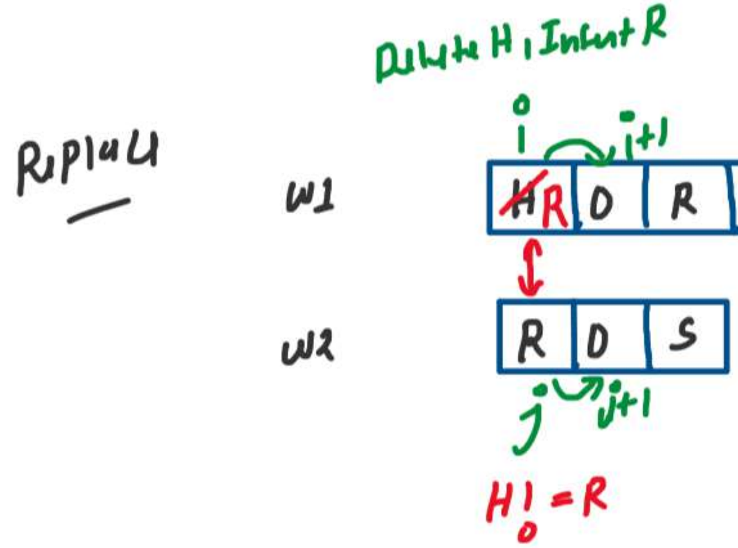
1st step

R.C.

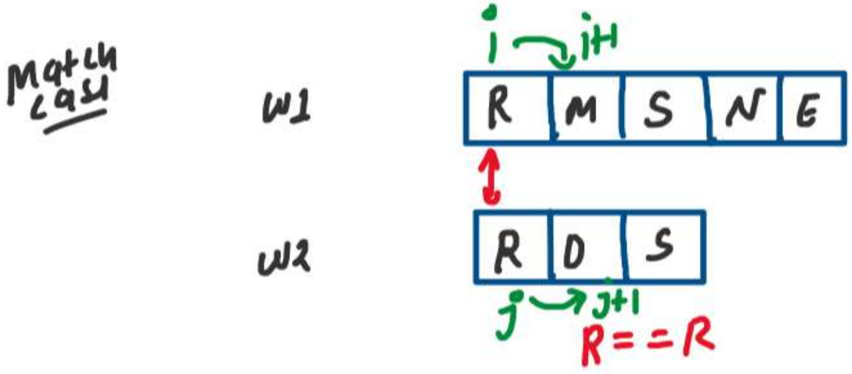
Delete



deleteOpt = 1 + f(w1, w2, i+1, j)



ReplaceOpt = 1 + f(w1,w2, i+1, j+1);

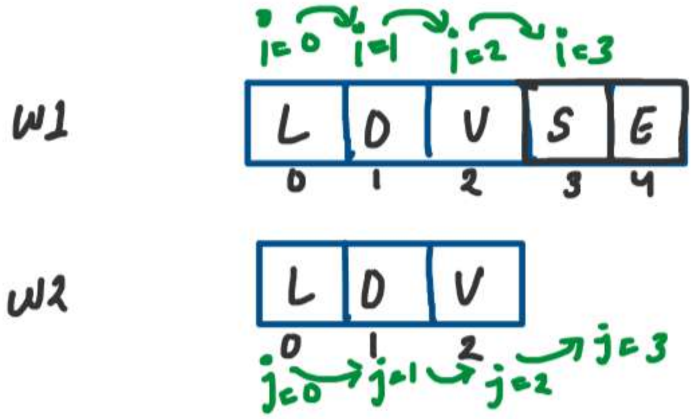


⇒ 0 + f(w1,w2, i+1, j+1)

output

return(min(insert , Replace, Delete))

BASE CASE



size of w1 = 5   size of w2 = 3

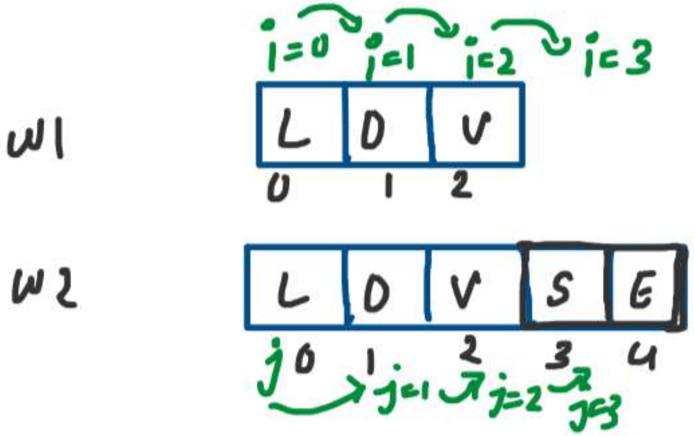
(j >= 3)

return 5-i

= 5-3

= 2

Total operation
2
for S & E



size of w1 = 3   size of w2 = 5

(i >= 3)

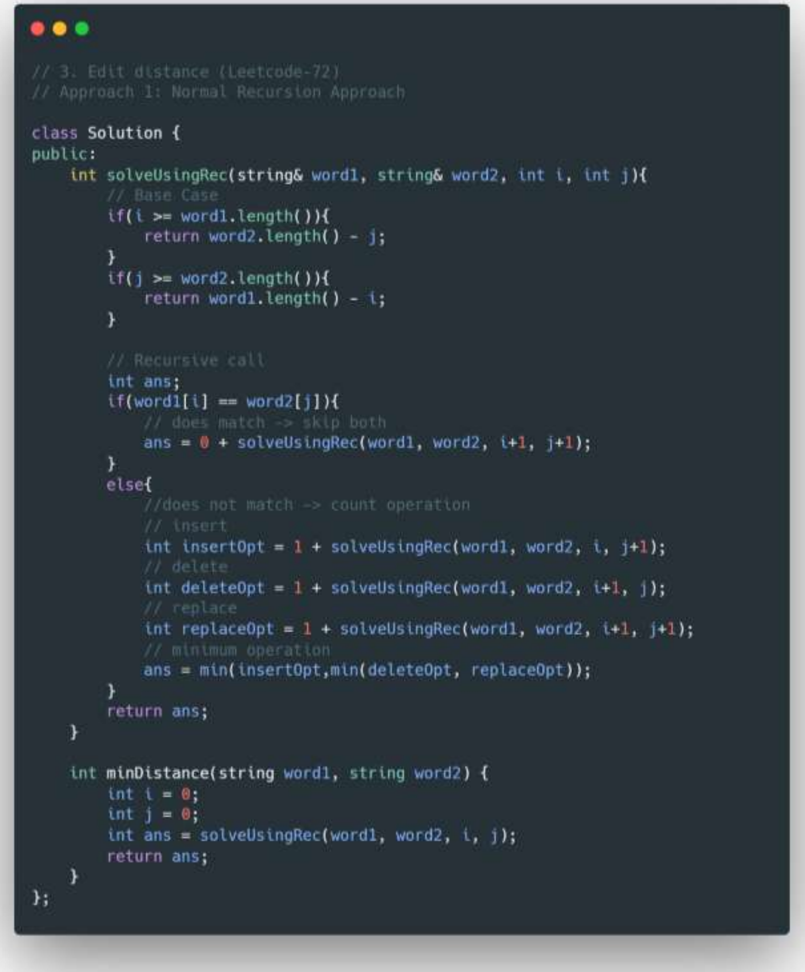
return 5-i

= 5-3

= 2

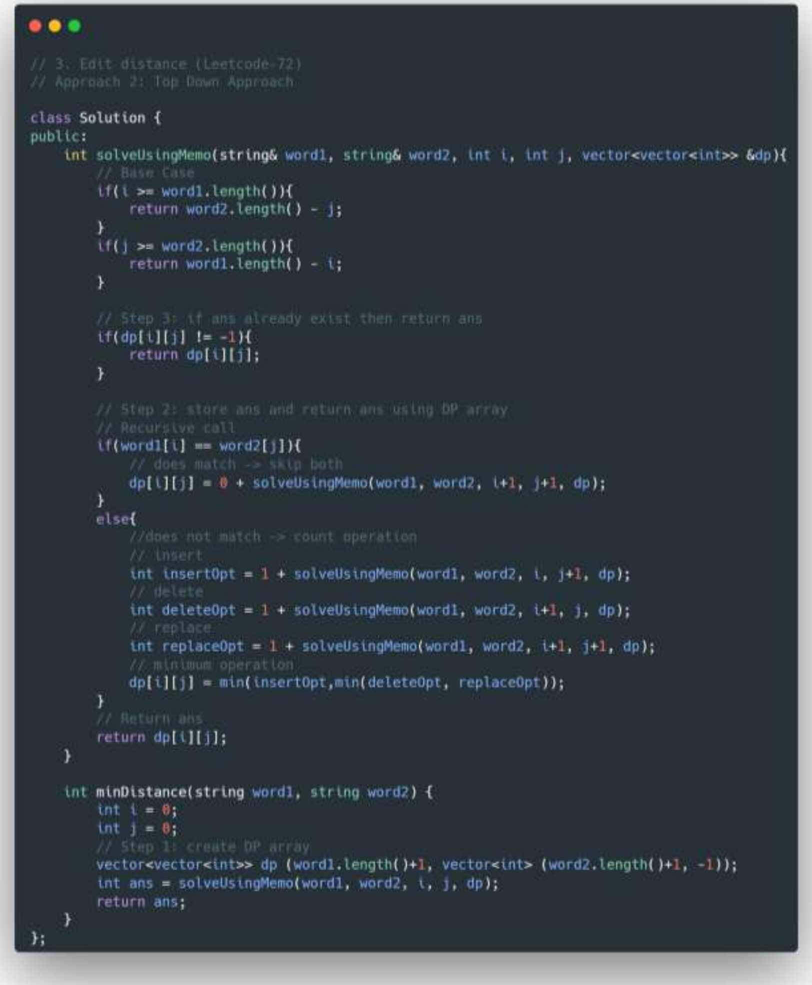
Total operation
2
for S & E

Approach 1: Recursion



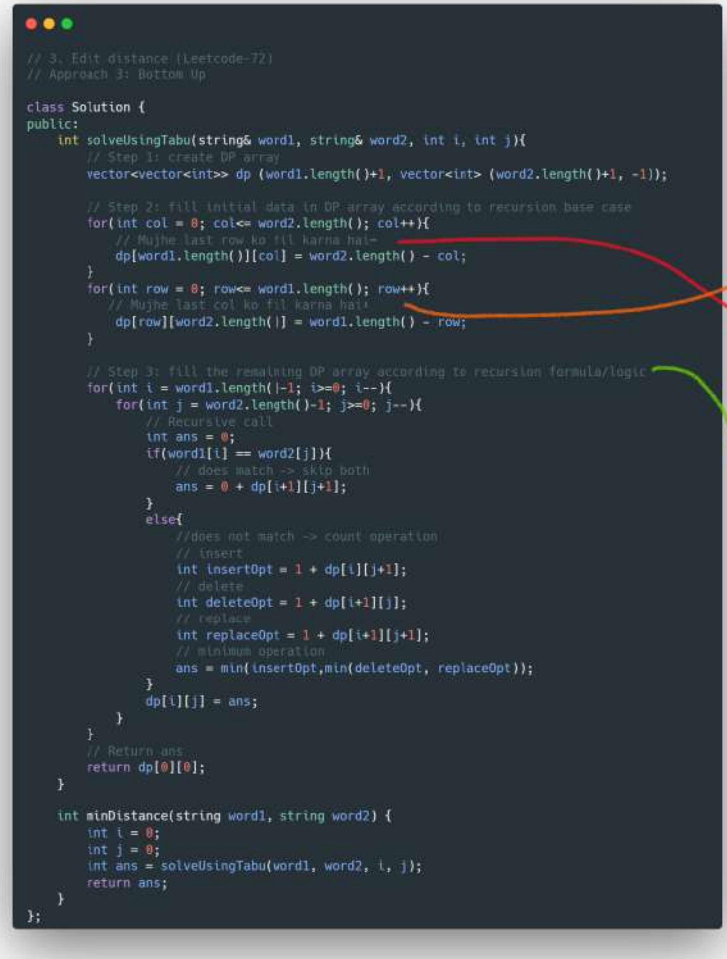
```
// 3. Edit distance (Leetcode-72)
// Approach 1: Normal Recursion Approach

class Solution {
public:
    int solveUsingRec(string& word1, string& word2, int i, int j){
        // Base Case
        if(i >= word1.length()){
            return word2.length() - j;
        }
        if(j >= word2.length()){
            return word1.length() - i;
        }

        // Recursive call
        int ans;
        if(word1[i] == word2[j]){
            // does match -> skip both
            ans = 0 + solveUsingRec(word1, word2, i+1, j+1);
        }
        else{
            //does not match -> count operation
            // insert
            int insertOpt = 1 + solveUsingRec(word1, word2, i, j+1);
            // delete
            int deleteOpt = 1 + solveUsingRec(word1, word2, i+1, j);
            // replace
            int replaceOpt = 1 + solveUsingRec(word1, word2, i+1, j+1);
            // minimum operation
            ans = min(insertOpt,min(deleteOpt, replaceOpt));
        }
        return ans;
    }

    int minDistance(string word1, string word2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingRec(word1, word2, i, j);
        return ans;
    }
};
```

Approach 2: Top Down

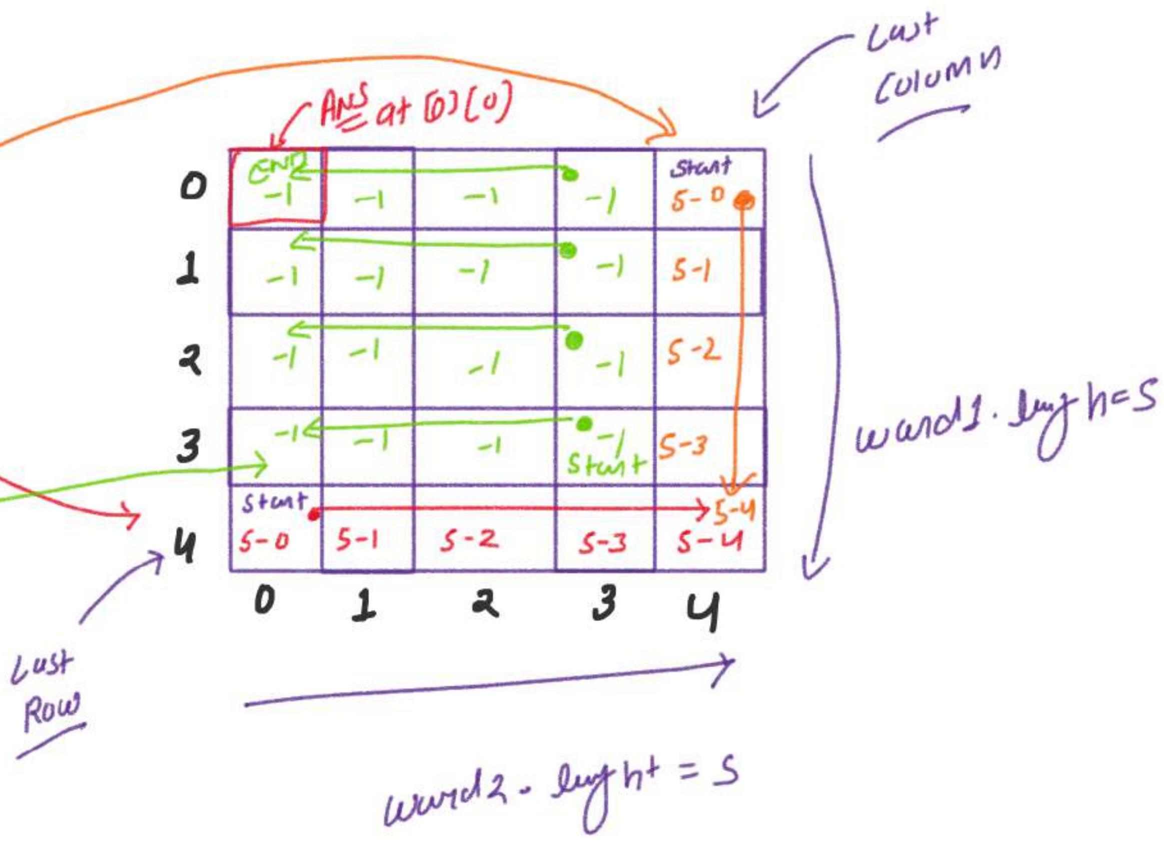


```
// 3. Edit distance (Leetcode-72)
// Approach 2: Top Down Approach

class Solution {
public:
    int solveUsingMemo(string& word1, string& word2, int i, int j, vector<vector<int>> &dp){
        // Base Case
        if(i >= word1.length()){
            return word2.length() - j;
        }
        if(j >= word2.length()){
            return word1.length() - i;
        }

        // Step 3: if ans already exist then return ans
        if(dp[i][j] != -1){
            return dp[i][j];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive call
        if(word1[i] == word2[j]){
            // does match -> skip both
            dp[i][j] = 0 + solveUsingMemo(word1, word2, i+1, j+1, dp);
        }
        else{
            //does not match -> count operation
            // insert
            int insertOpt = 1 + solveUsingMemo(word1, word2, i, j+1, dp);
            // delete
            int deleteOpt = 1 + solveUsingMemo(word1, word2, i+1, j, dp);
            // replace
            int replaceOpt = 1 + solveUsingMemo(word1, word2, i+1, j+1, dp);
            // minimum operation
            dp[i][j] = min(insertOpt,min(deleteOpt, replaceOpt));
        }
        // Return ans
        return dp[i][j];
    }

    int minDistance(string word1, string word2) {
        int i = 0;
        int j = 0;
        // Step 1: create DP array
        vector<vector<int>> dp (word1.length()+1, vector<int> (word2.length()+1, -1));
        int ans = solveUsingMemo(word1, word2, i, j, dp);
        return ans;
    }
};
```

Approach 3: Bottom Up

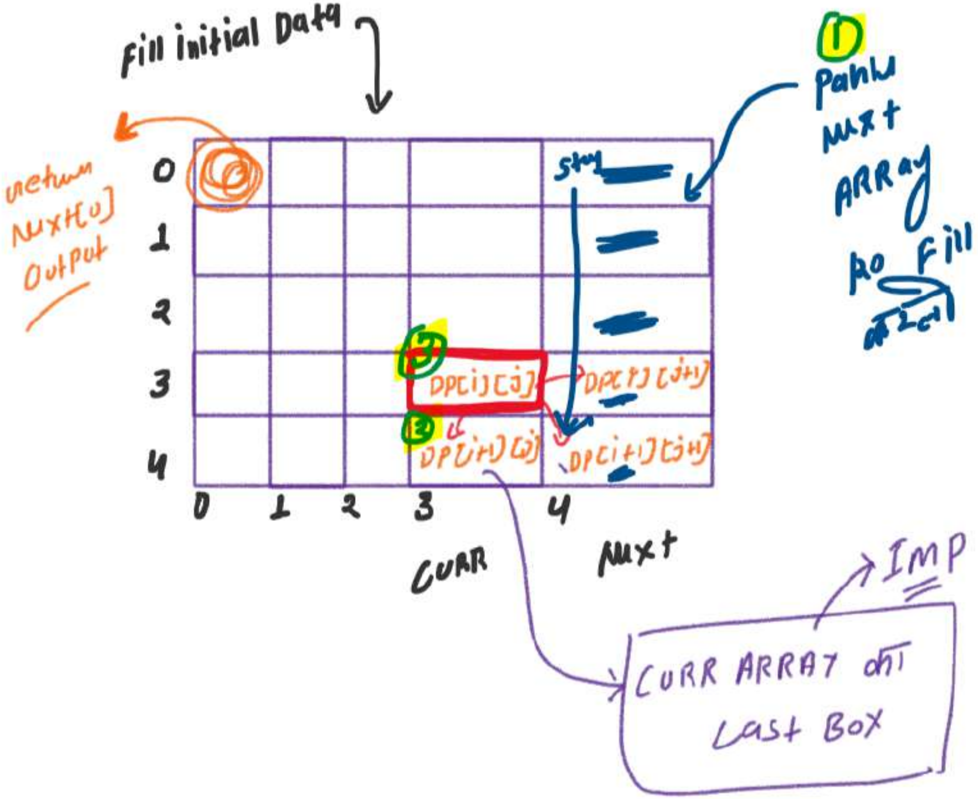


```cpp
// 3. Edit distance (Leetcode-72)
// Approach 3: Bottom Up

class Solution {
public:
    int solveUsingTabu(string& word1, string& word2, int i, int j){
        // Step 1: create DP array
        vector<vector<int>> dp (word1.length()+1, vector<int> (word2.length()+1, -1));

        // Step 2: fill initial data in DP array according to recursion base case
        for(int col = 0; col<= word2.length(); col++){
            // Mujhe last row ko fil karna hai-
            dp[word1.length()][col] = word2.length() - col;
        }
        for(int row = 0; row<= word1.length(); row++){
           // Mujhe last col ko fil karna hai+
            dp[row][word2.length()] = word1.length() - row;
        }

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int i = word1.length()-1; i>=0; i--){
            for(int j = word2.length()-1; j>=0; j--){
                // Recursive call
                int ans = 0;
                if(word1[i] == word2[j]){
                    // does match -> skip both
                    ans = 0 + dp[i+1][j+1];
                }
                else{
                    //does not match -> count operation
                    // insert
                    int insertOpt = 1 + dp[i][j+1];
                    // delete
                    int deleteOpt = 1 + dp[i+1][j];
                    // replace
                    int replaceOpt = 1 + dp[i+1][j+1];
                    // minimum operation
                    ans = min(insertOpt,min(deleteOpt, replaceOpt));
                }
                dp[i][j] = ans;
            }
        }
        // Return ans
        return dp[0][0];
    }

    int minDistance(string word1, string word2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabu(word1, word2, i, j);
        return ans;
    }
};
```

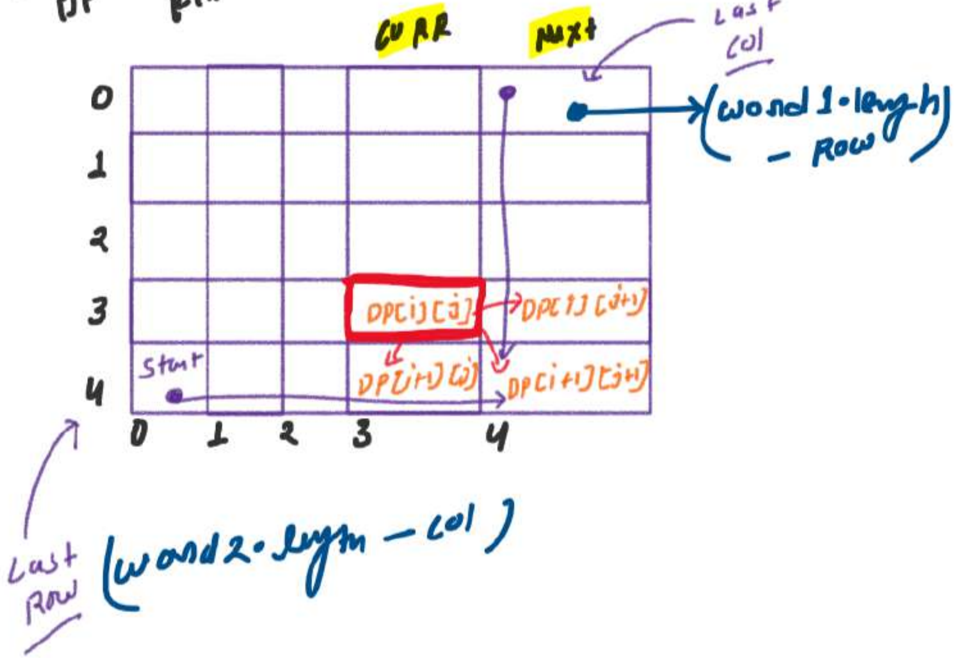


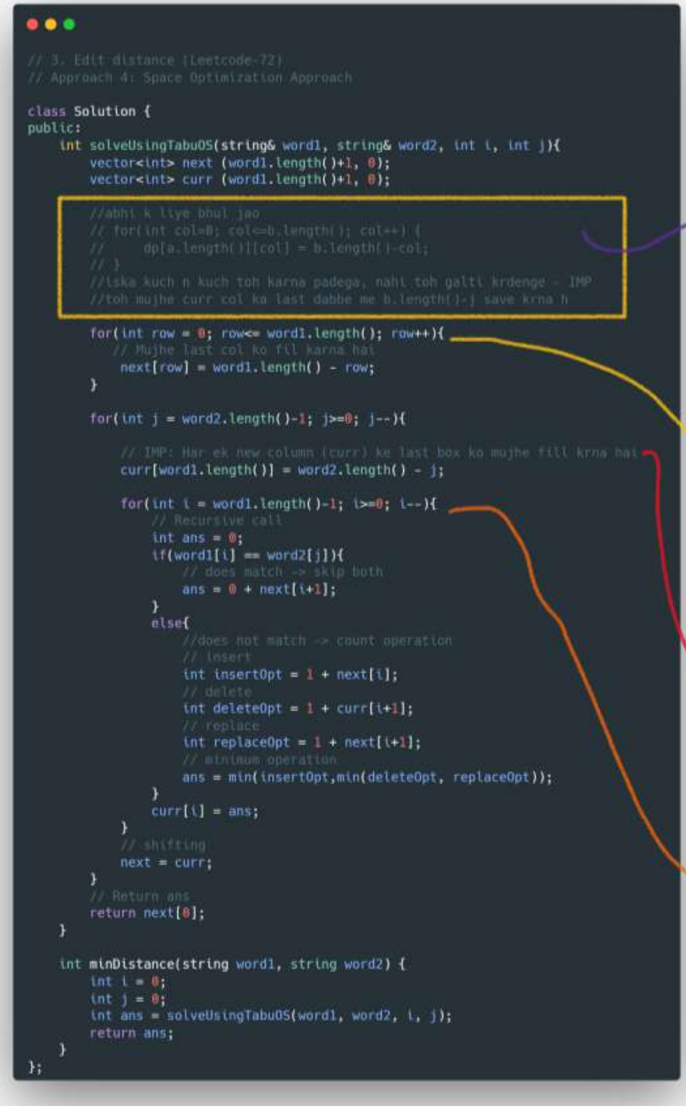
	0	1	2	3	4
0	END -1	-1	-1	-1	Start 5-0
1	-1	-1	-1	-1	5-1
2	-1	-1	-1	-1	5-2
3	-1	-1	-1	-1 Start	5-3
4	Start 5-0	5-1	5-2	5-3	5-4 5-4

Approach 4: Space Optimization



[Tabulation ke time initial DP ka data kaise fill karna tha ?]





```cpp
// 3. Edit distance (Leetcode-72)
// Approach 4: Space Optimization Approach

class Solution {
public:
    int solveUsingTabuOS(string& word1, string& word2, int i, int j){
        vector<int> next (word1.length()+1, 0);
        vector<int> curr (word1.length()+1, 0);

        //abhi k liye bhul jao
        // for(int col=0; col<=b.length(); col++) {
        //     dp[a.length()][col] = b.length()-col;
        // }
        //iska kuch n kuch toh karna padega, nahi toh galti krdenge - IMP
        //toh mujhe curr col ka last dabbe me b.length()-j save krna h
        for(int row = 0; row<= word1.length(); row++){
           // Mujhe last col ko fil karna hai
            next[row] = word1.length() - row;
        }

        for(int j = word2.length()-1; j>=0; j--){

            // IMP: Har ek new column (curr) ke last box ko mujhe fill krna hai
            curr[word1.length()] = word2.length() - j;

            for(int i = word1.length()-1; i>=0; i--){
                // Recursive call
                int ans = 0;
                if(word1[i] == word2[j]){
                    // does match -> skip both
                    ans = 0 + next[i+1];
                }
                else{
                    //does not match -> count operation
                    // insert
                    int insertOpt = 1 + next[i];
                    // delete
                    int deleteOpt = 1 + curr[i+1];
                    // replace
                    int replaceOpt = 1 + next[i+1];
                    // minimum operation
                    ans = min(insertOpt,min(deleteOpt, replaceOpt));
                }
                curr[i] = ans;
            }
            // shifting
            next = curr;
        }
        // Return ans
        return next[0];
    }

    int minDistance(string word1, string word2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabuOS(word1, word2, i, j);
        return ans;
    }
};
```

jab next Array ko First time Fill karungi To उसके Bad me HAME kaise भी करके CURR ARRAY ke last Box ko Fill करना होगा। Kyunki col=0 ko HUM Initially Fill nahi kar sakte hai jaise Tabulation में कर रहे थे।

next ko Fill करो 1

Curr ke last Box को fill करो 2

Dependent Ans ko fill करो 3