Lecture 20: Longest Common Subsequence.

The states that we would be given 2 ~~dp~~ strings and we count no. of characters which are ~~comm~~ common and in a subsequence
Ex.

a b c d e f a b e f

Ans ⇒ 4 ("abef")

Now try to understand that like problems in DP on subsequences we performed pick/not pick. Similiary in DP on strings we perform match/not match. So we compare each character and if it is a match, +1 to ans and reduce both string pointers by -1. (Since we are starting from end of string).
Now, if it doesn't match, try to under. by the following example

c a | a c
↑ ↑

§ Since currently they are not matching and if reduce ~~any~~ both pointers then we may loose ~~on~~ future ans.
Thus, we make '2' recursive calls, where we reduce one pointer in both string in different recursive calls. So as to cover all possibilities.

Base Case :- Since we are reducing and comparing strings, till ind == 0, the base case would be after that case i.e. ind < 0, this signifies string is over and we return 0; because there is no string left to compare.

Create a dp[][] of size that is equal to size of string1 & string2

Code

```
int f(int i, int j, string &s, string &t,
                    vector<vector<int>> &dp) {

    if(i<0 || j<0)
        return 0;
    if(dp[i][j] != -1) return dp[i][j];

    if(s[i] == t[j])
        return 1 + f(i-1, j-1, s, t, dp);

    return 0 + max(f(i, j-1, s, t, dp), f(i-1, j, s, t, dp));
}
```

make these dp[i][j] =

main recurrence when s[i] != t[j]

Time — O(N x M).

Space — O(N x M) + O(N+M)

A.S.S - Auxilary Stack Space

Try to make recursion and notice why (N+M).

Tabulation

Now, Since we know hoco make tabulation, Copy Base case. But in this case, base case is at (-1) which in tabulation form is not possible thus we make a right shift at every index. And base case change and become

if (i==0 || j==0) return 0;

Simply, intialize the dp[][] as equal to 0. Copy recurrence and proceed as ~~follows~~ always.

Code

```
int f(string &s, string &t) {

    int n = s.size(), m = t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            int notsame = 0 + max(dp[i-1][j], dp[i][j-1]);
            int same = 0
            if(s[i-1] == t[j-1])
                same = 1 + dp[i-1][j-1];
            dp[i][j] = max(notsame, same);
        }
    }

    return dp[n][m];
}
```

Time → O(NM)
Space → O(NM)

Lecture 21 Longest Common Substring.

Now, in the case of substring, it has to be contiguous
i.e ABCD | ABZD.
Ans 2 ('AB')

for this we can use the concept of longest common subs... but the part of not matching.

$$\max(f(i-1, j), f(i, j-1))$$

cannot be used because this function creates all possibilities i.e subsequences too!
Thus for not matching cases make everything = 0.
Lets Do a Dry Run.

ABCD | ABZD

i ↓ \ j →	0	1	2	3	4	
		A	B	C	D	
0	0	0	0	0	0	Base Case
1	A	1	0	0	0	
2	B	0	2	0	0	
3	Z	0	0	0	0	
4	D	0	0	0	1	
	Base Case ↑					

Take the maximum of all nos.

Code

```
int f(string &s, string &t){

    int n = s.size();
    int m = t.size();

    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));
    int ans = 0;
    for(int i=0; i<=n; i++){
        for(int j=1; j<=m; j++){
            if(dp[s[i-1]==t[j-1]){
            dp[i][j] ans = 1+ dp[i-1][j-1];
              ans = max(ans, dp[i][j]);
            }
        }
    }
    return ans;
}
```

Lecture 22: Longest Palindromic Subsequences

~~Trees only on~~

The brute force method is to take all Subsequences store them, & check which are palindrome and print the longest one.

~~The op~~ The optimized way is to ~~tell~~ notice that string S = 'b b b a b b a b'

LPS = b b b a b b b.

Notice if we take another string t = b a b b a b b b i.e the reverse of 'S'. And now try to ~~make~~ find the longest Common Subsequence.

Notice when you find Common Subsequence of 2 strings which are reverse of each other. It will eventually be a palindrome.

Lecture 23: Minimum ~~Steps~~ Insertion Steps to make a String Palindrome.

Now, first try to understand that we can make any ~~str~~ string into a palindrome. How?

Suppose we have,

a a b c d.

The most naive Soln is to take another string same as this, reverse and add it to original.

i.e,

a a b c d d c b a a. it is now a palindrome.

But this means the no. of insertions is 'n' (length of Str) which is not optimal.

So how approach such question. Simple Think. of it as, we first calculate the longest palindrome subsequence, take them out and then the rest left we will just club them together and reverse it and add wherever necessary. Eg.

'aabcd'

here 'aa' is a palindrom.

'bcd', take another dcb and add to beginning i.e

dcbaabcd.

Since Quest^{n} doesn't want string, it only requires no. of char's added. Simply calculate longest palindrome and ~~it~~ subtract from total length.

Shortest Common Supersequence : Lecture 24

The problem states that we are given 2 strings and we need to create another string which would consist of both in it. But we have to tell the SHORTEST.

A = "BRUTE" , B = "GROOT"

Ans ⇒ C = "BGRUOOTE"
This word has both 'BRUTE' & 'GROOT' with minimum length i.e 8

Now first try to understand the logic, observe if calculate Longest Common Subsequence in both A & B. i.e 'RT', then the minimum length would always be.

$$n + m - LCS$$

- n : length of string 1
- m : length of string 2
- LCS : Common char.

Now, as we did in print LCS, we take the dp Table. Start from the bottom most cell make our way up we check for the following condition.

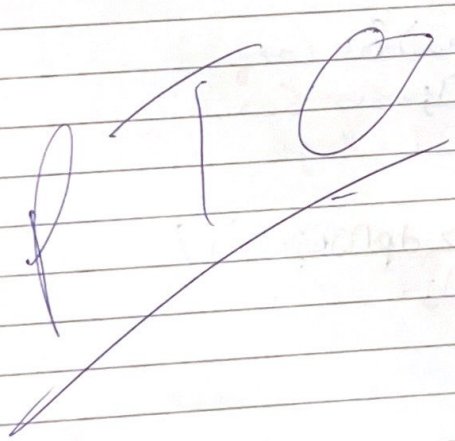
```
if (S[i-1] = t[j-1]).  // That means it's a common
    ans += S[i-1];      char; add to the ans and
    i--, j--;           move diagnoally up
```

Now, we check if the element up or Left one larger., if up is greater we go up and add 'S' char. to the ans., and if we go left then we add 't'

```
if (dp[i-1][j] > dp[i][j-1])
        ans += s[i-1];
        i--;
else
        ans += t[j-1];
        j--;
```

We do this process till either one of them reaches '0' and then break and if one reach '0', there has to be some characters left in other, we simply add those characters to ~~left~~ ans.

~~int~~ ~~string~~ f(~~string~~ s, ~~string~~ t) {.



```
string f(string s, string t){
    int n=s.size(), m=t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            int notmatch= max(dp[i-1][j], dp[i][j-1]);
            int match = 0;
            if(s[i-1] == t[j-1])
                match = 1+dp[i-1][j-1];
            dp[i][j]= max(match, notmatch);
        }
    }
    string ans;
    int i=n, j=m;
    while(i>0 && j>0){
        if(s[i-1] == t[j-1]){
            ans += s[i-1];
            i--, j--;
        }
        else if(dp[i-1][j] > dp[i][j-1]){
            ans += s[i-1];
            i--;
        }
        else{
            ans += t[j-1];
            j--;
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```

Distinct Subsequences. : Lecture 25

It is another standard pick/not pick problem. (or) as DP in strings we say match/notmatch....

All we have to do is count ~~str~~ how many times string 't' occures in string S. For Recurrence our parameter will be (i, j) that are indexes of string S & t, starting at end.

Since it is a counting problem, we return the ~~subse~~ summation of pick/notpick.

#Memoization

```
int f(int i, int j, string &s, string &t,
                    vector<vector<int>>&dp){
if(dp[i][j]!=-1) return dp[i][j]
                if (j<0) return 1;
                if (i<0) return 0;
                int notpick = f(i-1, j, s, t, dp);
                int pick = 0
                if (s[i]==t[j])
                    pick = f(i-1, j-1, s, t, dp);

                return dp[i][j] = pick + notpick;
}
```

For base case try to understand,
S = bagbbag , t = bag.

if i<0, then that means we traversed whole 'S' and couldn't find a match.

Whereas if j<0, then that means we found whole of 't', because we decrease 'j' only when it's a match

Tabulation.

Same Methodology, convert base cases → Copy recurrence

```
int Solve (String s, String t) {
    int n = s.size();
    int m = t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for(int i=0; i<=n; i++)
        dp[i][0] = 1;

    for (int i=1; i<=n; i++) {
        for(int j=1; j<=m; j++) {
            int notpick = dp[i-1][j];
            int pick = 0;
            if (s[i-1] == t[j-1])
                pick = dp[i-1][j-1];
            dp[i][j] = pick + notpick;
        }
    }
    return dp[n][m];
}
```

Understanding base case, like we did in LCS problem, shifting everything to the right, because we can't write case for (i<0) & (j<0) in tabulation format.

Lecture 26 : Edit Distance.

In this Question we are given 3 ways to edit a 'S' to make it into string 't'. That are →

① Insert
② Remove
③ Replace.

We have solved similiar questions before, the underlying principle is the same. i.e match / notmatch.

S = "horse" , t = "ros"

S-1) → replace 'h' → 'r'
S-2) → remove 'r' at S[2]
S-3) → remove 'e' at S[4]

and we got our string equal in minimum steps possible

We know the recurrence will f(i, j) where i & j are indexes of S & t resp., Starting from end we compare each character and have the following choices

Match	Not Match
If it is a match, we don't need to perform any editing and simply decrease the size of both strings and move on.	① Insert → we do an imaginary insertion, at a place ahead of curr char, which is obviously equal to t[j], and Thus, we stay at same place in String 'S' and decrease 't' and continue comparision. ② Remove → Simply Decrease string 'S' and keep 't' same.

③ Replace → If we are gonna replace a char in S, then it will obv be same as t, thus we can decrease both strings by 1, since we made them equal.

* Base Case

Try Understand, the Base Case would arise when either one of them is exhausted, and try to understand when either one of them is exhausted we just have to return the leftover of ~~this~~ other string length for complete transformation.

And Since we want minimum steps required for transformation, we simply take minimum of all 4 action.

- → Match
- → Insert
- → Remove
- → Replace

Time Complexity → O(N × M).

Space Complexity → O(N × M) + O(N + M)

O(N × M) → dp arr[][]

O(N + M) → Auxiliary stack.

Code (Memoization).

```
int f(string &s, string &t, int i, int j, vector<vector<int>> &dp){
    if(j<0)
        return i+1;
    if(i<0)
        return j+1;
    if(dp[i][j]!=-1)
        return dp[i][j];

    int match = 1e4;
    if(s[i]==t[j]);
        match = f(s,t,i-1,j-1,dp);
    int insert = 1+ f(s,t,i,j-1,dp);
    int remove = 1+ f(s,t,i-1,j,dp);
    int replace = 1+ f(s,t,i-1,j-1,dp);

    dp[i][j] = min(min(match,insert), min(replace, remove));
    return dp[i][j];
}
```

Tabulation.

Same Rules + Shifting Right by one., try understanding bases cases that when i==0 → we have values of j from 0 → m & similiarly for j==0.

```
int Solve (String S, String t){
    int n = S.size(), m = t.size();
    vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

    for(int i=0 ; i<=m; i++)
        dp[0][i] = i;
    for(int i=0; i<=n; i++)
        dp[i][0] = i;

    for(int i=1 ; i<=n ; i++) {
        for(int j=1 ; j<=m; j++) {
            int match = 1e4;
            if(S[i-1] == t[j-1])
                match = dp[i-1][j-1];
            int insert = 1+ dp[i][j-1];
            int remove = 1+ dp[i-1][j];
            int replace = 1+ dp[i-1][j-1];

            dp[i][j]= min(min(match, insert),
                          min(replace, remove));
        }
    }
    return dp[n][m];
}
```

Time Comp → O(N×M)

Space Comp → O(N×M)

#Lecture 27: Wildcard Matching

This is yet another match / not match problem with modification, here we have two Special Char.

① "?" → this means that we can match it with any other char of other string.

② "*" → This means we can match any Subsequence with other string. Even NULL char.

We are given a string 'S' and after checking acc to ques we have to tell if it is equal to string 't'. Return T/F.

S = "AB?D" , t = "ABCD"

Then we follow the same pattern as follows, if (S[i] == t[j]) call the function & decrease the value of i & j, where i = n-1 & j = m-1, But with an extra condition as, if S[i] == '?' still it's a match and we do the same function call.

Now if we step on a '*', things get a bit tricky.

S = "AB*EF" t = "ABCDEF"

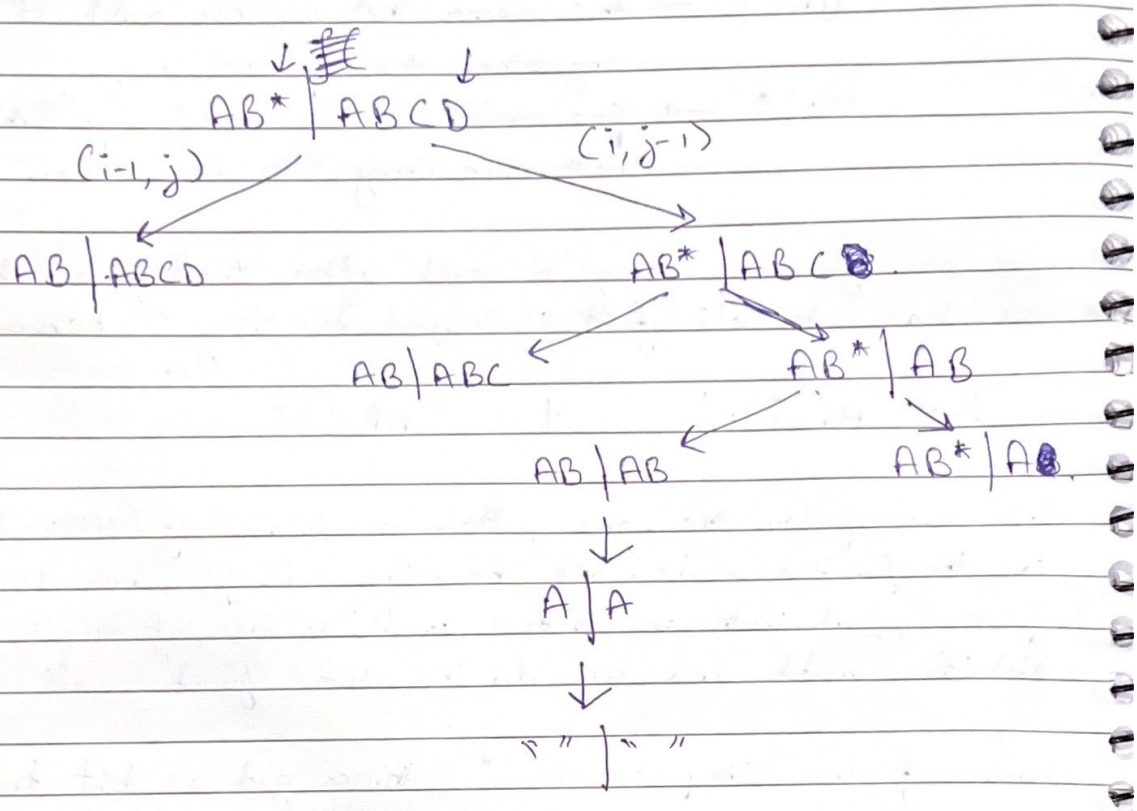
Since we start from end, we'll have a clear match on E & F without any problem.

But after that we don't know how much should * Contain So as to make S == t. for that try out all possible Subsequences. Try to understand, the recurrence f(i-1, j) & f(i, j-1); This simply means take and not take with stoppage at '*'.

We can have all possible states with these 2 recurrences. And if either of them returns True. We return true.

Let's ~~Do~~ Dry Run it

S = "AB*"
t = "ABCD"



This Recursive calls, helps us to calculate all possibilities.

Now, Let's try to understand Base Cases.

The base case logic, would be same as previous question, we see which and when a string exhausts.

* if both Strings exhaust → Match found return True.
* if string 'S' exhaust → Their is still 't' remaining to be match with NULL string 'S'. Return false.

PTO

* If String 't' exhausts → if only string S remains, try to understand observe that the remaining characters have to be '*' to ~~xxx~~ treat them as NULL. If any char exists Return False, Else Return true.

Convert the code into Memoization dp[n][m];

```
bool f(int i, int j, string &s, string &t, vector<vector<int>> &dp){
    if(i<0 && j<0)
        return true;
    if(i<0 && j>=0)
        return false;
    if(i>=0 && j<0){
        for(int x=0; x<=i; x++){
            if(s[x] != '*')
                return false;
        }
        return true;
    }
    if(dp[i][j] != -1)
        return dp[i][j];
    if(s[i]==t[j] || s[i]=='?')
        return dp[i][j]= f(i-1,j-1,s,t,dp);
    if(s[i]=='*')
        return dp[i][j]= f(i-1,j,s,t,dp) || f(i,j-1,s,t,dp);
    return dp[i][j]= false;
}
```

Time Comp — O(N x M)
Space Comp — O(N x M) + O(N+M)

Tabulation

Same logic of shifting towards Right, Observe and think carefully about base cases

```
bool Solve (String S, String t){
    int n= s.size(), m= t.size();
    vector<vector<bool>> dp(n+1, vector<bool>(m+1, false));
    dp[0][0] = true
    for(int i=1; i<=n; i++){
        for(int x=1; x<=i, x++){
            if (s[x-1] != '*'){
                flag= false;
                break;
            }
        }
        dp[i][0] = flag == false ? false : true;
    }
    for (int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if (s[i-1] == t[j-1] || s[i-1] == '?')
                dp[i][j] = dp[i-1][j-1];
            else if (s[i-1] == '*')
                dp[i][j] = dp[i-1][j] || dp[i][j-1];
            else
                dp[i][j] = false;
        }
    }
    return dp[n][m]
}
```

Time — O(N x M)
Space — O(N x M)