String Based Dp

- → longest Common Subsequence
- → Edit distance
- → Regex Matching

IQ) Given 2 Strings, find length of longest Common Subsequence
↳ LCS (S_1 S_2)

S_1 : N & S_2 : M

Eg1:

	0	1	2	3	4	5	6
S_1 :	a	b	b	c	d	g	f
S_2 :	b	a	c	h	e	g	f

Seq1 ⇒ a c g f } ans = 4
Seq2 ⇒ b c g f

Eg2:

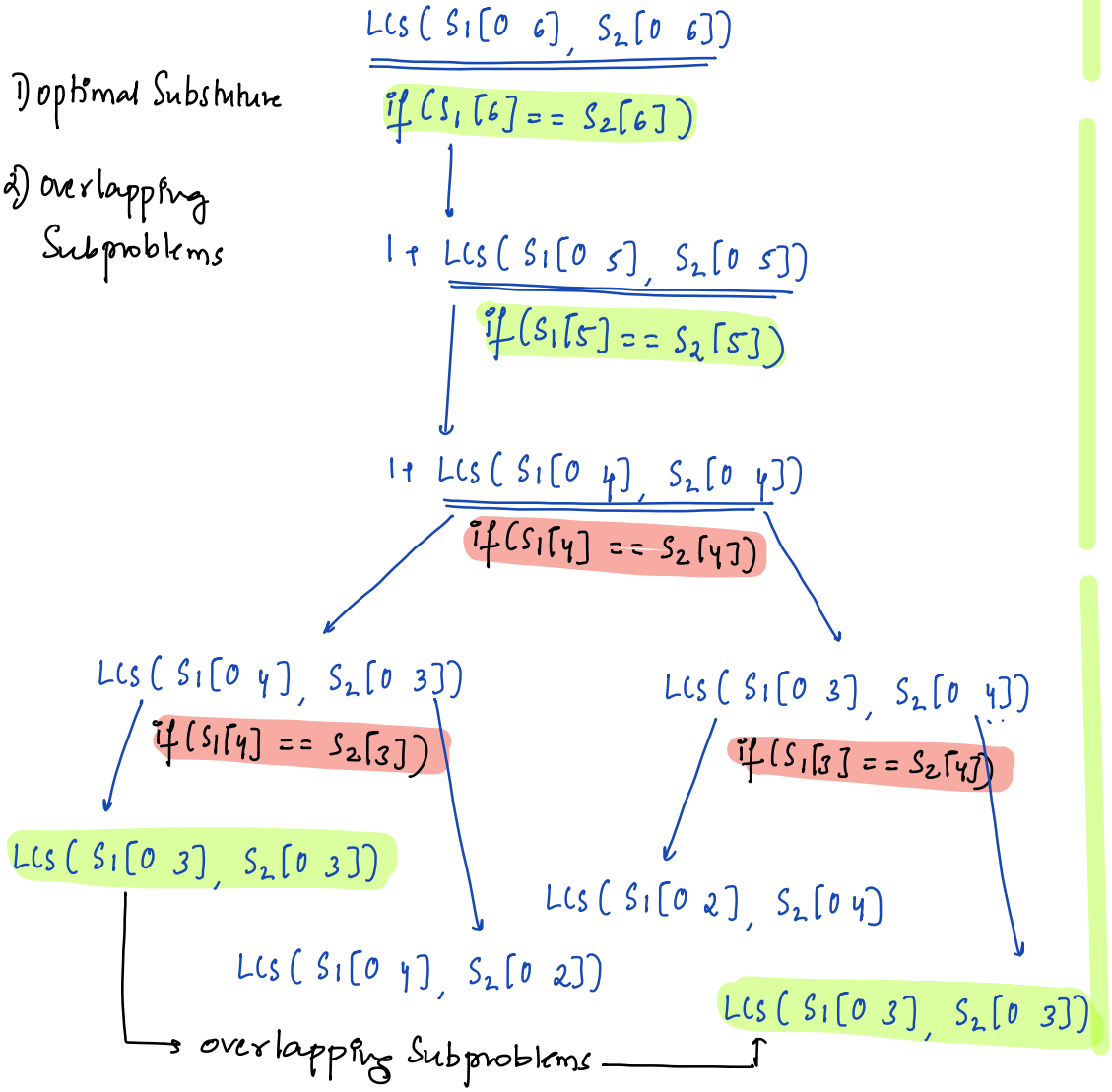
S_1 : k l a g r i p

S_2 : l g i g k m

Seq1 : l g i } ans = 3

1) Optimal Substructure

2) overlapping Subproblems



dp State:

i representing ending index of S_1

j representing ending index of S_2

dp[i,j] = LCS of $S_1[0,i]$ & $S_2[0-j]$

dp Expression:

$$dp[i,j] = \begin{cases} \text{if } (s_1[i] == s_2[j]) \{ \\ \quad 1 + dp[i-1, j-1] \\ \} \\ \text{else} \{ \\ \quad \max(dp[i, j-1], dp[i-1, j]) \\ \} \end{cases}$$

Because we are considering last character in sequence

Recursive Code:

```
int dp[N][M] = -1; // Initialize in main & pass it parameter in function
int Lcs(String s1, String s2, int i, int j, int dp[][]){
    if (i == -1 || j == -1) { return 0; }
    [indicates subproblem called]
    if( dp[i][j] == -1) { first time
        if (s1[i] == s2[j]){
            dp[i][j] = 1 + Lcs(s1, s2, i-1, j-1, dp)
        }
        else{
            dp[i][j] = max{ Lcs(s1, s2, i-1, j, dp)
                            Lcs(s1, s2, i, j-1, dp) }
        }
    return dp[i][j]
}
```

Note: i == -1, indicates S_1 is completed, & there is nothing, with an empty String LCS = 0

TC: $(N*M)*1 \approx O(N*M)$ SC: $O(N*M)$

Tracing:

	0	1	2	3	4
S_1 :	M	A	I	C	A
S_2 :	I	A	I	Y	A	S

$$dp[i,j] = \begin{cases} \text{if } (s_1[i] == s_2[j]) \{ \\ \quad 1 + dp[i-1, j-1] \\ \} \\ \text{else} \{ \\ \quad \max(dp[i, j-1], dp[i-1, j]) \\ \} \end{cases}$$

Fill dp table: dp[5][6]

		0 I	1 A	2 I	3 Y	4 A	5 S
0	M	0	0	0	0	0	0
1	A	0	1	1	1	1	1
2	I	1	1	2	2	2	2
3	C	1	1	2	2	2	2
4	A	1	2	2	2	3	3

A I A

if ($s_1[i] == s_2[j]$)

$dp[i][j] = 1 + dp[i-1][j-1]$

else

$dp[i,j] = \max(dp[i-1,j], dp[i,j-1])$

// To print 1 longest Common Subsequence

i = N-1, j = M-1

String ans = "";

Note: Even if i == -1 || j == -1 we can stop

while (i >= 0 && j >= 0) {

10:32 → 10:40 pm

```
    if (s1[i] == s2[j]) {
        // s1[i] present in ans
        ans = ans + s1[i], i--, j--
    }
    else {
        // dp[i][j] could have dp[i-1][j] or dp[i,j-1]
        if (i > 0 && dp[i][j] == dp[i-1][j]) { i-- }
        else { j-- }
    }
}
return rev(ans)
```

// reverse ans string because we are adding char by char from back

28) Edit distance → Ed

Given 2 strings S_1 & S_2 min operations to be performed on S_1 so that S_1 becomes S_2

In 1 operation of S_1

- We can insert a character in S_1 at any position
- We can replace a charater in S_1 at any position, with any character
- We can delete a charater in S_1 at any position

Ex1:

S_1 : d f a e l
S_2 : f g l

→ min operations

S_1 : ~~d~~ f a ~~e~~ l → 3 operations

S_1 : ~~d~~ f ~~a~~ e l → 3 operations

Ex2:

S_1 : f e h
S_2 : a e k l

S_1 : ~~f~~ e ~~h~~ l → 3 operations

S_1 : d f a e l
S_2 : f g l

Ed (S_1 [0 4] S_2 [0 2]) 1) Optimal Substructure

if (S_1[4] == S_2[2]) 2) overlapp Subproblems

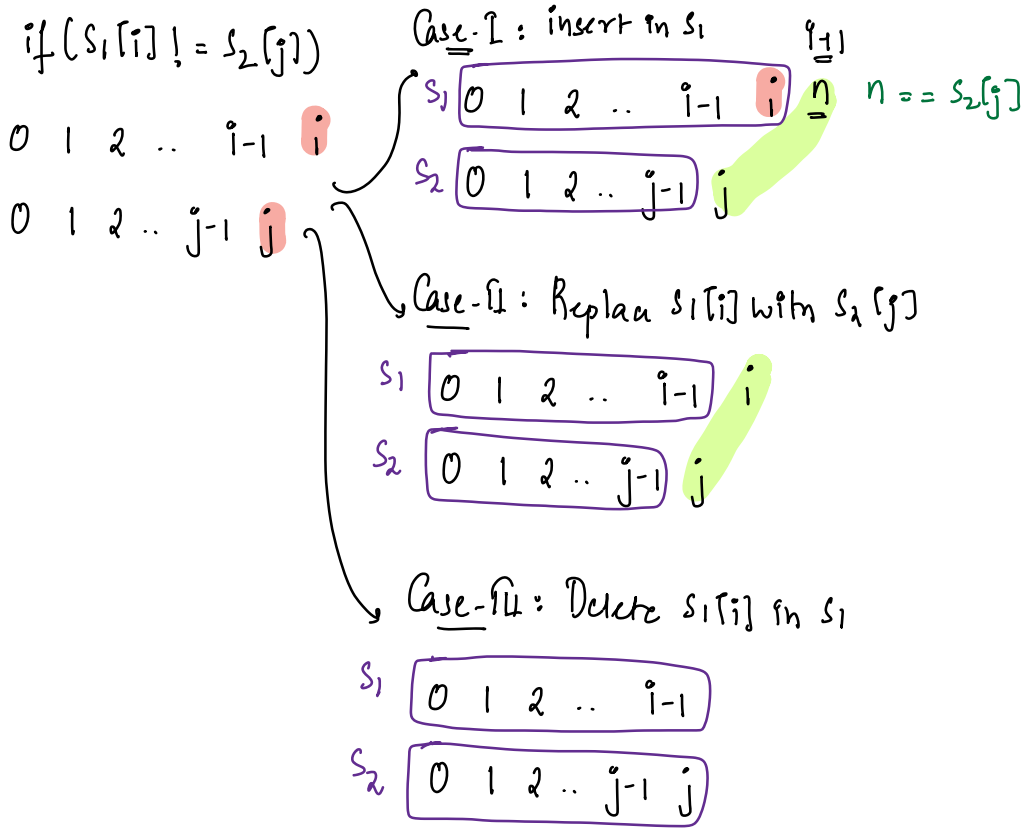
Ed (S_1 [0 3] S_2 [0 1])

if (S_1[3] == S_2[1])

Insert S_1 : Ed (S_1 [0 3] S_2 [0 0])

Replace S_1 : Ed (S_1 [0 2] S_2 [0 0])

Delete S_1 : Ed (S_1 [0 2] S_2 [0 1])



// dp[i,j] = { Min operations required to make S1[0 i] same as S2[0 j] }

// dp[i,j] =

```
if (S1[i] == S2[j]) {
    dp[i,j] = dp[i-1,j-1]
}
else {
    dp[i,j] = 1 + min3 {   // we are doing operation
                  dp[i,j-1]   // insert
                  dp[i-1,j-1] // replace
                  dp[i-1,j]   // delete
              }
}
```

dp table: dp[N][M]

Pseudocode:

```
dp[N][M] = -1 ; // declare in main & pass it parameter

int Ed ( String s1, String s2, int i, int j, int dp[][]){
    if ( i==-1 && j==-1 ) { return 0}
    if ( i==-1){
        // we need to perform j+1 insert        s1 = ""
        return j+1                              s2 = "abcd"
    }
    if (j==-1) {
        //we need to perform i+1 delete         s1 = "abcd"
        return i+1                              s2 = ""
    }
    if(dp[i][j]==-1){
        if(s1[i]==s2[j]){
            dp[i][j] = ed(s1, s2, i-1, j-1, dp)
        }
        else{
            dp[i][j] = 1 + min{ ed(s1, s2, i, j-1, dp) -> insert
                                ed(s1, s2, i-1, j-1, dp) -> replace
                                ed(s1, s2, i-1, j, dp) -> delete }
        }
    }
    return dp[i][j]
}
```

→ Very imp

TC: $O(N*M)$ SC: $O(N*M)$

38) Regex Matching ⇒ Rtm

// Given Text (T) & Patter (P), check if both are same or not

T → In Text it only contains alphabets

P → With alphabets, it contains ?, *

- ? → { it can match any 1 character }
- * → { it can match any number of characters } >=0

Note: P → can have any number of ? & *

T : a p p l e
P : a ? * e
⇒ matching

T : a p p l a e
P : a * a ? e
⇒ not matching

T : a n t
P : a ? * t
⇒ matching

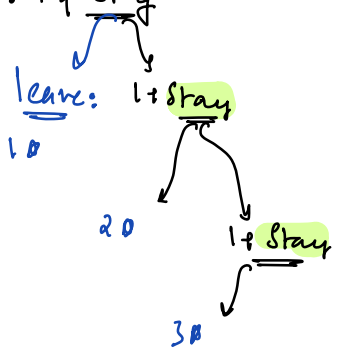
T : ""
P : * * * *
⇒ matching

T : " "
P : ?
⇒ not matching

* : leave it
* : pick 1, leave it
* : pick 2, leave it
* : pick 3, leave it
:
* pick i leave it

or

* : leave it
* : 1 & Stay

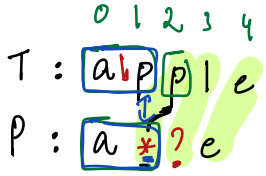


// Start itself : No, Yes → [1, 2, 3, 4, 5]

↳ [No, You play a round] ↳ Play 1 round : Play n Not

↳ Play 1 round : Play n Not

↳ Play 1 rand : Play Not



→ optimal Substructure

→ overlapping Subproblems

RM(T[0 4] P[0 3])

if (T[4] == P[3])

RM(T[0 3] P[0 2])

if (T[3] == P[2]) || P[2] == '?'

RM(T[0 2] P[0 1])

if (T[2] == P[1]) || P[1] == '*'

leave Star, without

RM(T[0 2] P[0 0])

Match a, once, & still let it pick

RM(T[0 1] P[0 1])

// dp State :

$$dp[i,j] = \{ \text{whether we can match } T[0-i] \text{ to } P[0-j] \}$$

// dp Expression

$$dp[i,j] = \begin{cases} \text{if } (T[i] == P[j] \text{ || } P[j] == '?') \{ \\ \quad dp[i,j] = dp[i-1, j-1] \\ \} \\ \text{else if } (P[j] == '*') \{ \\ \quad dp[i,j] = dp[i, j-1] \text{ || } dp[i-1, j] \\ \} \\ \text{else } \{ \\ \quad dp[i,j] = \text{false} \\ \} \end{cases}$$

// dp table : dp[N][M]

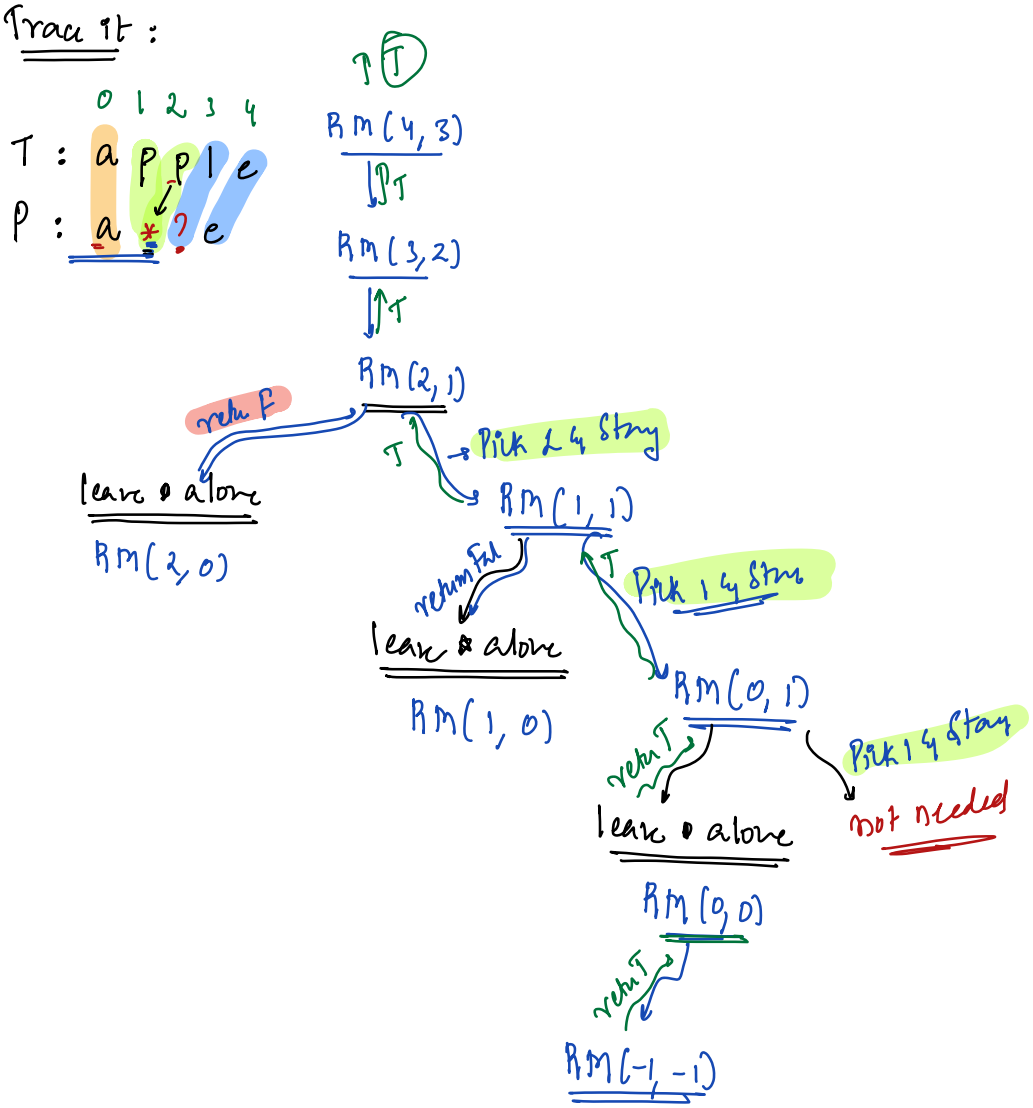
```
int dp[N][M] = -1;   declare in main, pass as parameter

int RM( String T, String P, int i, int j, int dp[][]){
    if( i==-1 && j==-1) { return True }
                          1
    if(j==-1){ // Text left out pattern empty
        return false 0
    }
    if (i==-1) { // Text empty pattern present,
                 // if pattern contains only * return True
        for( k=0; k<=j; k++) {
            if( P[k] != *){
                return false
                       0
            }
        }
        return True
               1
    }

    if ( dp[i][j]==-1){
        if(T[i]==P[j] || P[j]=='?'){
            dp[i,j] = RM(T, P, i-1, j-1, dp)
        }
        else if( P[j]=='*') {
            dp[i,j] = RM(T, P, i-1, j, dp) || RM(T, P, i, j-1, dp)
        }
        else {
            dp[i,j] = false
        }
    }
    return dp[i,j]
}
```

TC: $O(N*M)$ SC: $O(N*M)$

Trace it :


0 1 2 3 4
T : a p p l e
P : a * ? e
RM(4, 3)
RM(3, 2)
RM(2, 1)
return F
T
Pick 2 & Stay
leave alone
RM(2, 0)
RM(1, 1)
return F
T
Pick 1 & Stay
leave alone
RM(1, 0)
RM(0, 1)
return T
Pick 1 & Stay
not needed
leave alone
RM(0, 0)
return T
RM(-1, -1)