Todays Content:

a) longest common Subsequence
b) Edit distance
c) Regex Matching

1Q) Given 2 Strings $S_1 : N$ & $S_2 : M$

: find length of longest Common Subsequence ~ { Data ordered based on index }

Ex1:

	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

S_1: a b c g f
S_2: b a c g f
} not matching

CS : b c g f, a c g f ans = 4

Ex2:

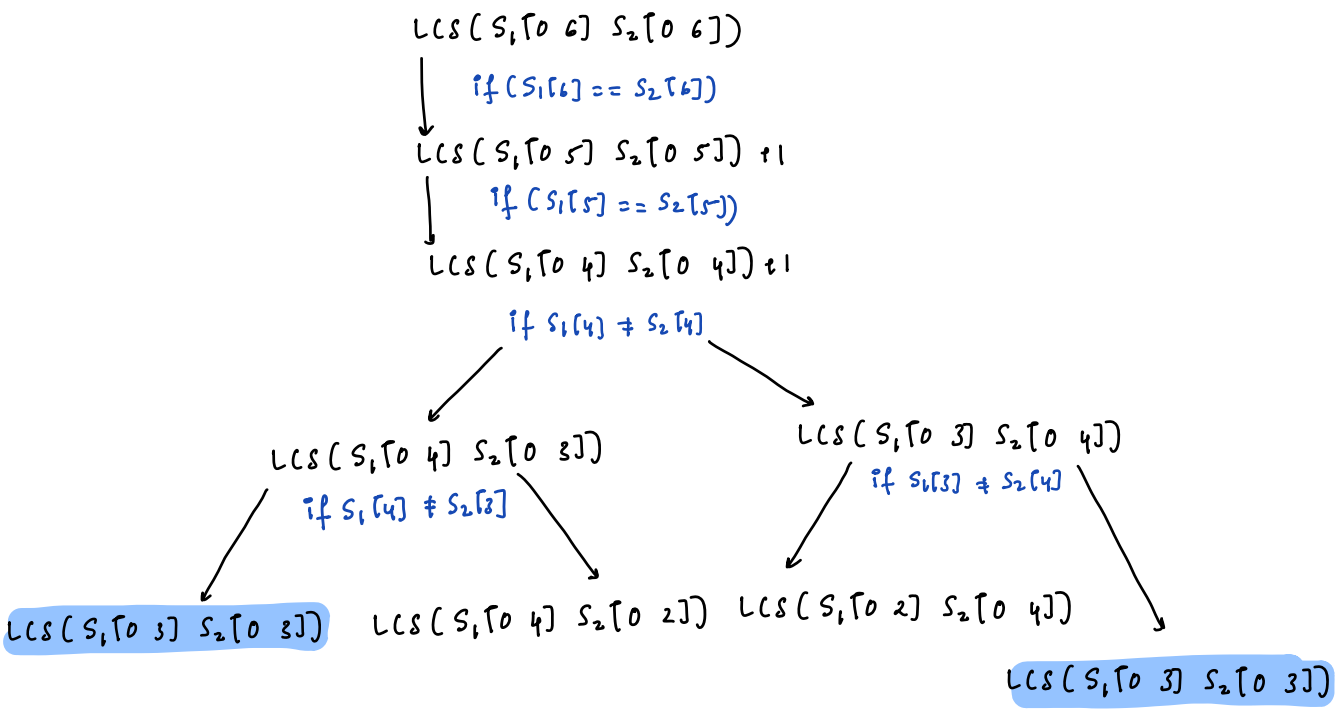
	0	1	2	3	4	5	6
S_1 :	k	l	a	g	r	i	p
S_2 :	l	g	i	g	k	m	

ans = lgi, ans = 3

Idea: # Get LCS between $S_1[0-6]$ & $S_2[0-6]$

	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

1. Subproblems ✓
2. Overlapping ✓



Dp Steps:

dpState: dp(i, j) = length of longest common sub $S_1[0..i]$ & $S_2[0..j]$

dpExpression:

S_1: 0 1 2 3 .. i-1 i

S_2: 0 1 2 3 .... j-1 j

```
if( S1[i] == S2[j] ){
    dp(i, j) = dp(i-1, j-1) + 1
}
else{
    dp(i, j) = max( dp(i, j-1) , dp(i-1, j) )
}
```

Final ans: Given S_1 N & S_2 M, lcs($S_1[0, N-1]$, $S_2[0, M-1]$)

Final ans = dp(n-1, m-1)

Table: int dp[N][M] TC: # States * TC for each state $N*M * 1 \rightarrow O(N*M)$

```
int dp[N][M] = INVALID / -1 / ve..

int lcs( String s1, String s2, int i, int j){
    if (i<0 || j<0) { return 0; } // if one string empty nothing in common
    if( dp[i][j] == -1) { // 1st time calling
        if( s1[i] == s2[j] ) { // match
            dp[i][j] = lcs(s1, s2, i-1, j-1) + 1
        }
        else { // mis match
            dp[i][j] = max { lcs(s1, s2, i, j-1)
                             lcs(s1, s2, i-1, j) }
        }
    }
    return dp[i][j]
}
```

Tracing:

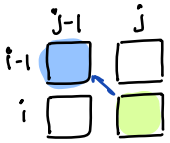
	0	1	2	3	4	5
S_1:	M	A	I	C	A	
S_2:	I	A	I	Y	A	S

dp[5][6]:

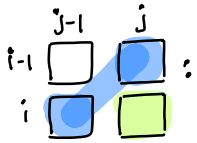
S_1:		S_2:	I	A	I	Y	A	S
			0	1	2	3	4	5
M	0		0	0	0	0	0	0
A	1		0	1	1	1	1	1
I	2		1	1	2	2	2	2
C	3		1	1	2	2	2	2
A	4		1	2	2	2	3	3

dp(i, j):

```
if(S1[i] == S2[j]) {
    dp(i,j) = dp(i-1, j-1) + 1
}
else {
    dp(i,j) = max(dp(i-1,j), dp(i, j-1))
}
```



obs: If matching depends on Top-left



obs: If not matching depends on max of Top & left cell.

Tracing: Final ans: dp[4][5] = 3

$S_1[4] != S_2[5]$

dp[3][5] = 2

dp[4][4] = 3

$S_1[4] == S_2[4]$: Pick $S_1[4]$

dp[3][3]

TODO Code:

Doubts Session

10:7 → 10:45pm

Edit distance

Given 2 Strings S_1 & S_2 min operations to perform on S_1 so that $S_1 \to S_2$

In 1 operation of S_1

i → We can insert any character in S_1 at any pos

d → We can delete a character in S_1 at any pos,

r → We can replace a character in S_1 at any pos, with any char

Eg1:

	0	1	2	3	4
S_1 :	d	f	a	e	l
S_2 :	f	g	l		

S_1 : ~~d~~ f ~~a~~ e→g l ans = 3 operations

S_2 : f g l

	0	1	2	3
S_1 :	f	e	h	
S_2 :	a	e	k	l

S_1 : f→a e h→k l ans = 3 operations

S_2 : a e k l

min ope req to transform $S_1[0\ 4] \to S_2[0\ 3]$

Idea:

	0	1	2	3	4
S_1 :	d	f	a	e	l
S_2 :	f	g	k	l	

1. Subproblems
2. overlapping ?

Edit($S_1[0-4]$ $S_2[0\ 3]$)

if ($S_1[4] == S_2[3]$)

Edit($S_1[0-3]$ $S_2[0\ 2]$)

	0	1	2	3
S_1 :	d	f	a	e→k
S_2 :	f	g	k	

replace

min.

Edit($S_1[0-2]$ $S_2[0\ 1]$)

1

delete

	0	1	2	3
S_1 :	d	f	a	~~e~~
S_2 :	f	g	k	

Edit($S_1[0-2]$, $S_2[0\ 2]$)

1

insert

	0	1	2	3	4
S_1 :	d	f	a	e	k
S_2 :	f	g	k		

Edit($S_1[0-3]$ $S_2[0\ 1]$)

1

dp Steps:

dp state: dp[i,j]: min Operations to transform $S_1[0\ i] \to S_2[0\ j]$

dp exp: if($S_1[i] == S_2[j]$) {

  dp[i,j] = dp[i-1, j-1]

  S_1: 0 1 2 3 .. i-1 i

  S_2: 0 1 2 3 ... j-1 j

} else {

  replace: dp[i-1, j-1]

  S_1: 0 1 2 3 .. i-1 i → $S_1[i] == S_2[j]$

  S_2: 0 1 2 3 ... j-1 j

  delete: dp[i-1, j]

  S_1: 0 1 2 3 .. i-1 i ✗ delete $S_1[i]$

  S_2: 0 1 2 3 ... j-1 j

  insert: dp[i, j-1]

  S_1: 0 1 2 3 .. i-1 i $S_2[j]$ insert $S_2[j]$

  S_2: 0 1 2 3 ... j-1 j

  dp[i,j] = 1 + min(dp[i-1, j], dp[i, j-1], dp[i-1, j-1])

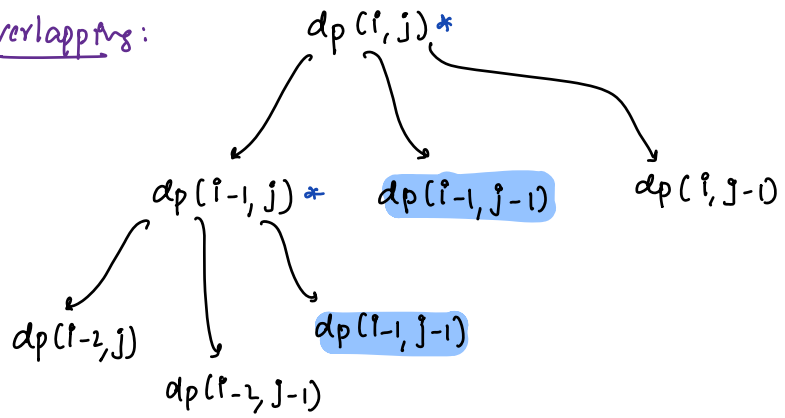
}

final ans: S_1 N & S_2 M, min operation $S_1[0, N-1] \to S_2[0\ M-1]$

dp[N-1, M-1]

Dp table: dp[N][M] TC: O(N*M)

Overlapping:



```
int dp[N][M] = INVALID/-1/-ve

int edit(String s1, String s2, int i, int j){
    if(i<0 && j<0){
        // s1:[0 -i] && s2 [0 -j], Both are empty strings
        return 0;
    }
    if(i<0){
        // s1:[0-i]  s2:[0 j]
        len: 0    len: j+1  s1 → s2 perform j+1 insert operation on s1
        return j+1
    }
    if(j<0){
        // s1:[0 i]  s2[0-i]
        len: i+1    s2: 0   s1 → s2 perform i+1 delete operations on s1
        return i+1
    }
    if(dp[i][j]==-1){
        if(s1[i]==s2[j]){
            dp[i][j] = edit(s1, s2, i-1, j-1);
        }
        else{
            dp[i][j] = 1+min{ edit(s1, s2, i, j-1);
                              edit(s1, s2, i-1, j-1);
                              edit(s1, s2, i-1, j); }
        }
    }
    return dp[i][j]
}
```

Hint:

Regex Matching:

Given Text (T) & Pattern (P) check if both are same or not

T → In Text it contains only alphabets

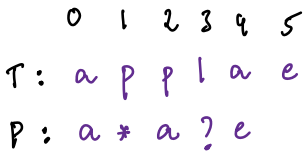
P → With alphabets, it contains ?, *

? → it can match any 1 character

* → it can match any number of characters >= 0

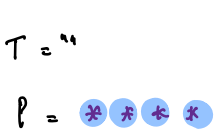
Ex1: matching

Ex2: not matching

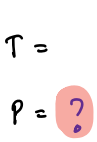


Ex3: matching

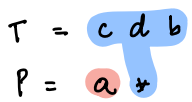
Ex4: matching



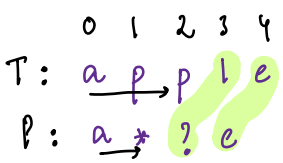
Ex5: not matching



Ex6: not matching



→ # check if text T[0-4] is matching with P[0-3]



Reg(T[0-4], P[0-3])

T[4] == P[3]

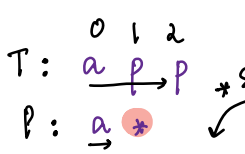
Reg(T[0-3], P[0-2])

T[3] != P[2] P[2] = ?

Reg(T[0-2], P[0-1])

T[2] != P[1] P[1] = *

* skip



Reg(T[0-2], P[0-0])

* Pick & Skip

Reg(T[0-1], P[0-1])

Dp Steps:

dpState: dp(i,j) = Check if T[0 i] matching P[0 j]

dp Expression:

```
if ( T[i] == P[j] || P[j] == '?' ){
|   dp[i][j] = dp(i-1, j-1)
}
else if ( P[j] == '*' ){
|   dp(i,j) = { dp(i, j-1)
|              or
|              dp(i-1, j) }
}
else{
|   dp(i,j) = False
|   // not matching
}
```

T: 0 1 2 3 .. i-1 i

P: 0 1 2 3 ... .. j-1 j, T[j] = ?

T: 0 1 2 3 .. i-1 i leave *

P: 0 1 2 3 ... .. j-1 j*

T: 0 1 2 3 .. i-1 i match & stay

P: 0 1 2 3 ... .. j-1 j*

T: 0 1 2 3 .. i-1 i

P: 0 1 2 3 ... .. j-1 j,

T[i] != P[j] && P[j] != ? && P[j] != '*'

Final ans: Given T_N & P_M check if

↓ T[0, N-1] matching P[0, M-1]

dp(N-1, M-1):

Dp table: int dp[n][M] TC: O(N*M)

```
int dp[N][M] = INVALID/-1/...

int Regex(String T, String P, int i, int j){
    if(i<0 && j<0){
        // T:[0 -i] && P[0 -i], Both are empty strings
        return 1;
    }
    else if(i<0){
        // T:[0-i] && P[0 j]
        len: 0          len: j+1
        k=0; k<=j; k++){
            if(P[j] != '*'){
                return 0
            }
        }
        return 1
    }
    else if(j<0){
        // T:[0 i] & P[0 -i] can never match
        return 0
    }

    if(dp[i][j] == -1){
        if(T[i] == P[j] || P[j] == '?'){
            dp[i][j] = Regex(T, P, i-1, j-1)
        }
        else if(P[j] == '*')
            dp[i][j] = {Regex(T, P, i, j-1) | Regex(T, P, i-1, j)}
        else
            dp[i][j] = 0
        }
    }
    return dp[i][j]
}
```

obs: Can empty string match with non empty pattern. If all characters from [0-j] should be *

→ optimize:

p: * * a b c ⟷ p: * a b c

p: * * * a * * b * c → p: * a * b * c

obs: In our pattern, if we have more than 1 * consecutively replace it * make change at start & call dp.