Today's Content:

→ 2D Matrix Problems → 45 mins

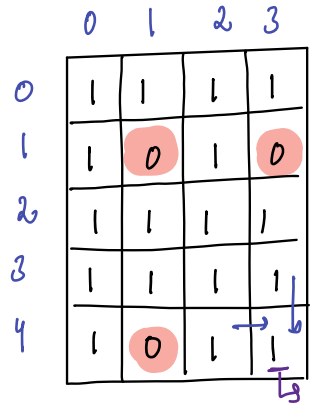
→ Given S_1 & S_2 find longest palindromic Subsequence?

a) print any 1 longest common Sequence

b) longest Palindromic Subsequence

c) Min characters to be deleted to make entire string palindrome

20) Number of ways to go from (0, 0) → (BR cell)



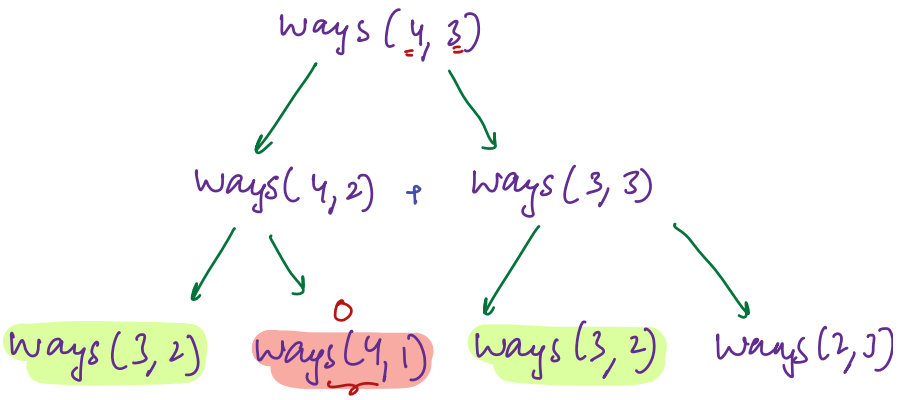
a) From cell → right
↓ Bottom

b) 0 indicates blocked cell

We cannot go from Blocked cell

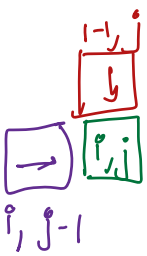
1) optimal substructure ✓

2) overlapping ✓



3) dp[i][j] = { Number of ways to reach 0,0 → i, j }

4) dp Expression




```
dp[i][j] = if(mat[i][j] == 0) {
               dp[i][j] = 0
           }
           else
               if(i >= 1)   if(j >= 1)
               dp[i][j] = dp[i-1][j] + dp[i][j-1]
```

Edge Cases: i == 0 || j == 0

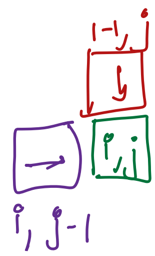
5) dp Table : dp[N][M]

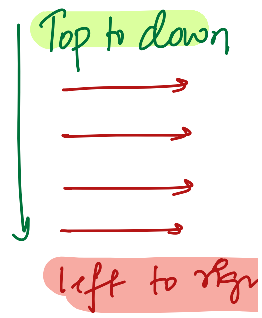
6) **Base Conditions:** → Take care of 0^{th} row & 0^{th} col

	0	1	2	3
0	1	1	0	1
1	1	1	1	0
2	1	1	1	1
3	1	1	1	1
4	1	0	1	1

dp

	0	1	2	3
0	1	1	0	~~1~~ 0
1	1	1	1	0
2	1	1	1	1
3	1	1	1	1
4	1	0	1	1





7) Psudo code:

```
if(mat[0][0] == 0) {return 0}        ← start is Blocked
int dp[N][M] = {0}                   Is {Initializing to 0}
dp[0][0] = 1;

j=1; j<M; j=j+1){
    if( mat[0][j] == 0) { break }
    else  dp[0][j] = 1
}

i=1; i<N; i=i+1){
    if( mat[i][0] == 0) { break }
    else  dp[i][0] = 1
}

i=1; i<N; i++){
    j=1; j<M; j++){
        if(mat[i][j] != 0){
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
        }
    }
}
return dp[N-1][M-1]
```

{TODO: update given matrix to solve problem}

TC: $(N*M)*1$. SC: $(N*M)$ → optimization try {O(2M)} discuss Saturday

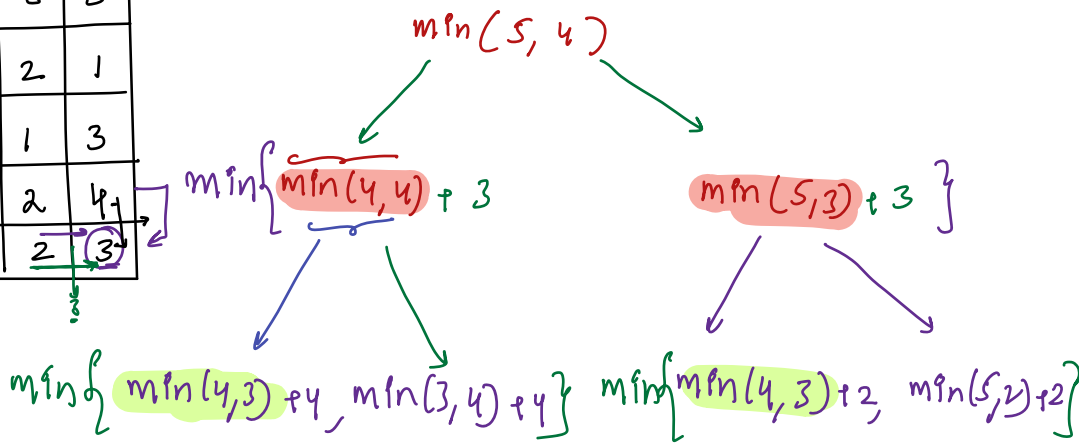
28)

1) mat[i][j] indicates cost to enter cell

2) min cost to travel from [0,0] → [5,4]

3) ↓ →

	0	1	2	3	4
0 →	2	1	3	7	3
1	3	3	4	8	3
2	0	5	1	2	1
3	0	2	3	1	3
4	7	1	3	2	4
5	4	1	5	2	3



// dp[i][j] = { min cost to go from (0,0) → i,j }

// dp[i][j] = min { dp[i-1][j] + mat[i][j], dp[i][j-1] + mat[i][j] }

↳ min cost from (0,0) → (i-1, j)

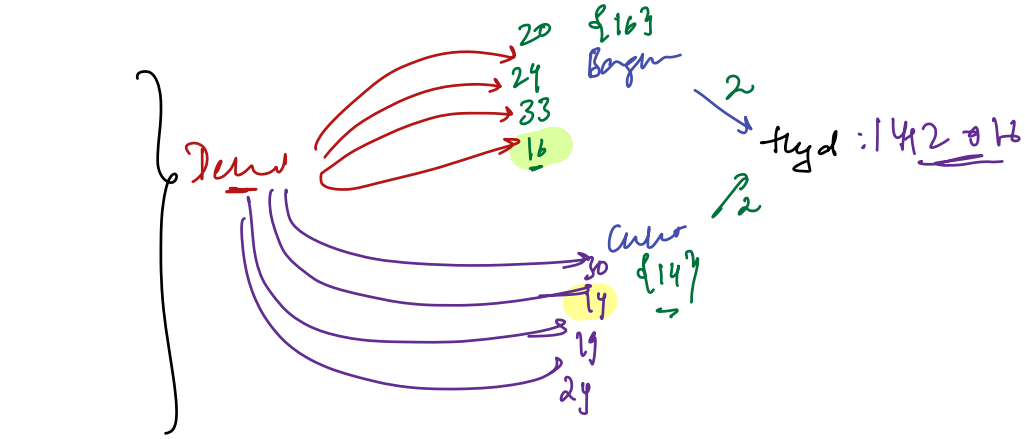
↳ min cost (0,0) → (i, j-1)

dp[i][j] = min { dp[i-1][j], dp[i][j-1] } + mat[i][j]

// dp table:

dp[N][M]

// TO DO



308) Given 2 strings, find the length of longest Common Subsequence

$S_1 : N$, $S_2 : M$

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

ans1: a c g f ans2: b c g f

len = 4

Eg2:

	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: l g i

len = 3

1) optimal

2) overlapping

LCS {S1[0 6], S2[0 6]}

→ if (S1[6] == S2[6]) : { match last character}

1 + LCS {S1[0 5], S2[0 5]}

→ if (S1[5] == S2[5]) { match last character}

1 + LCS {S1[0 4], S2[0 4]}

max{ LCS {S1[0 4], S2[0 3]} LCS {S1[0 3], S2[0 4]} }

LCS {S1[0 3], S2[0 3]}

LCS {S1[0 4], S2[0 2]}

LCS {S1[0 2], S2[0 4]}

LCS {S1[0 3], S2[0 3]}

→ indicate end index of S_1

→ indicate end index of S_2

// dp[i][j] = LCS (S_1[0 i] , S_2[0 j])

// dp[i][j] =

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])
}

```
int dp[N][M] = -1
int LCS ( String S1 , int i, String S2, int j) {
    if( i == -1 || j == -1) return 0
    if( dp[i][j] == -1) { 1st time visiting state
        if( S1[i] == S2[j]) {
            dp[i][j] = 1 + LCS (S1, i-1, S2, j-1)
        }
        else {
            dp[i][j] = max { LCS (S1, i-1, S2 j)
                             LCS (S1, i, S2, j-1) }
        }
    }
    return dp[i][j]
}
```

(0, 3) → {-1, 3} ; (2, 0) → {2, -1}

{empty = } { } { } empty

TC : O($N*M$) + O(1)

SC : O($N*M$)

→ { We cannot optimized dp table size in recursion }

{ Recursion + Extra space } = Memoization

Tracing:

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

Note: S_1 : { } S_2 : {a b d} } lcs $(S_1, S_2) = 0$

Recursively: { i==0 || j==0 } Edge case

Fill dp Table:

S_2:

S_1:		0 I	1 A	2 I	3 Y	4 A	5 S
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 → final ans

```
if (s1[i] == s2[j]) {
    dp[i][j] = 1 + dp[i-1][j-1]
}
else {
    dp[i][j] = max(dp[i-1][j], dp[i][j-1])
}
```

// print 1 longest common subsequence ?

i = N-1, j = M-1 → fill the table

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

```
if (s1[i] == s2[j]) { print(s1[i]); i = i-1, j = j-1 }
    { printing in reverse order }
else {
    if (i != 1 && dp[i][j] == dp[i-1][j]) { i = i-1 }
    else { j = j-1 }
}
```

3(8) Given string find length longest palindrome subsequence.

google

S_1 : c d g a b c a h d

rev S_1 : d h a c b a g d c

LCS(S, rev(S))

palindrome {Longest Comm. Seq}