Today's Agenda :-

1) No of ways to reach from (0,0) to (N-1),(M-1)

2) No of ways to reach from (0,0) to (N-1),(M-1) with blocked cells

3) Dungeons & Princess

4) Maximum Sum Subsequence without adjacent elements

Steps of Dynamic Programming :

1) Optimal Substructure
2) Overlapping Subproblems

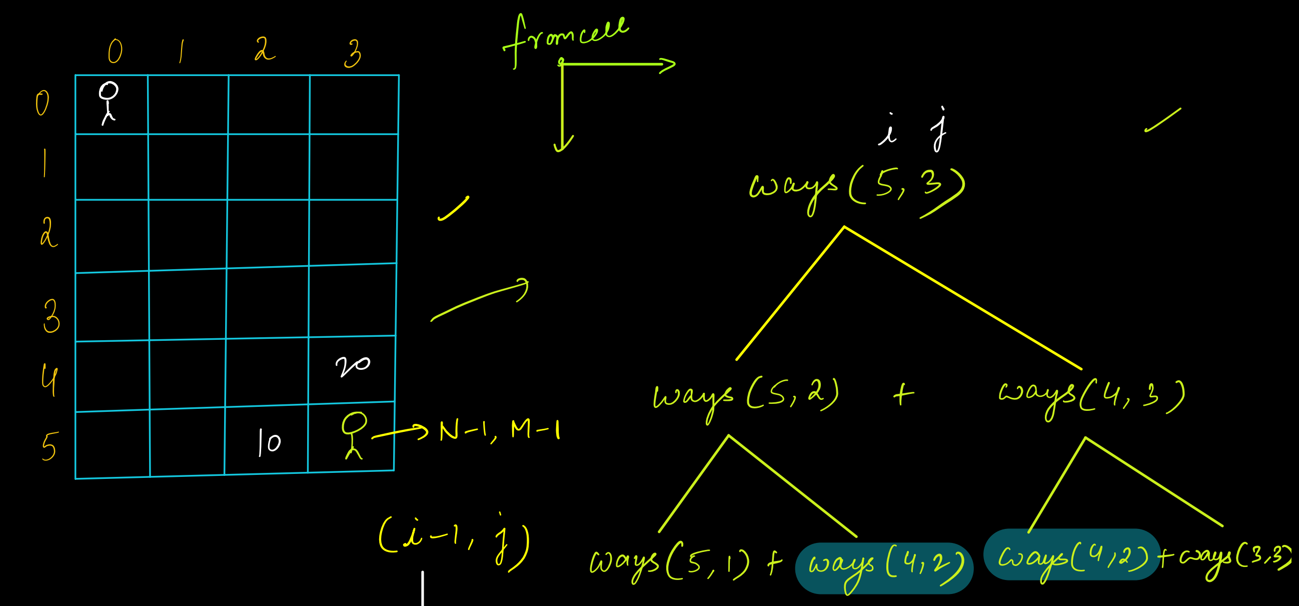
3) dP state   dP[i] = ? (Assumption)

4) dp expression   (Main Logic)

5) dp intialization (Base Condition)

N * M

1Q) Number of ways to go from (0,0) → (BR case)



$$\begin{tikzcd}
 & (i-1, j) \arrow[d] \\
(i, j-1) \arrow[r] & \boxed{(i, j)}
\end{tikzcd}$$

dP state :-

$$\boxed{dP[i][j] = \text{No of ways to reach } (i, j)}$$

dP expression :-

$$\boxed{dP[i][j] = dP[i][j-1] + dP[i-1][j]}$$

$i = 0$ or $j = 0$

dP initialization :-

N * M

int dP[N][M] ;

for $i = 0$ or $j = 0$, formula fails.

$\forall_{i=0}^{N-1} \; dP[i][0] = 1$ and $\forall_{j=0}^{M-1} \; dP[0][j] = 1$

Pseudo Code :-

```
int dP[N][M]
for( int j = 0; j < M; j ++) { dP[0][j] = 1 }
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++) {
        dP[i][j] = dP[i][j-1] + dP[i-1][j]
    }
}
return dP[N-1][M-1]
```

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

Recursive
Top Down

```
int dP[N][M] = {-1}
int   ways( int i, int j) {
      if(i == 0 or j == 0) return 1
      if (dP[i][j] != -1) return dP[i][j]
      dP[i][j] = ways(i-1, j) + ways(i, j-1)
      return dP[i][j];
}
```

2Q) Number of ways to go from (0,0) → (BR case)

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

a) From cell → Right
↓
Bottom

b) '0' indicates blocked cells

We cannot go from blocked cell.

dP expression :-

$$dP[i][j] = \begin{cases} \text{if}(mat[i][j]==0) \\ \quad dP[i][j] = 0 \\ \text{else} \\ \quad dP[i][j] = dP[i][j-1] + dP[i-1][j] \end{cases}$$

```
int dP[N][M] = {-1}
int    ways( int i, int j) {
        if(mat[i][j]==0) return 0 ✓
        if(i==0 or j==0)  return 1
        if(dP[i][j] != -1)  return dP[i][j]
        dP[i][j] = ways(i-1, j) + ways(i, j-1)
        return dP[i][j];
}
```

Base Condition :-

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

dP[N][M]

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

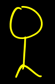
Try writing bottom up DP.

Break of 8 Min

HQ) Dungeons & Princess (Hard)

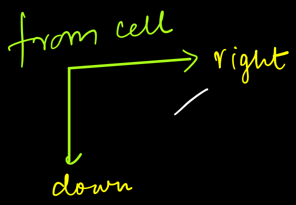
1) ✓
2) ✓

Prince



$n = 4$

	0	1	2	3
0	-3	+2	+4	-5
1	-6	+5	-4	+6
2	-15	-7	+5	-2
3	+2	+10	-3	-4



[health level ≤ 0], prince dies

Find the minimum health level to start with, so that you can save the princess.

$$x \downarrow \boxed{-6}$$

$x - 6 > 0$

$x - 6 = 1$ ✓

$\boxed{x_{min} = 7}$ ✓

Ex2 $\boxed{h = 7}$

-2	-8	100
-1	-3	1

Problem :- Find min health to enter (0,0) & save the

x = Min Health to enter i, j & save the princess.

$$\begin{tikzcd} x \arrow[d] & \\ (i,j) \arrow[r] \arrow[d] & (i,j+1) \\ (i+1,j) & \end{tikzcd}$$

$$(x + mat[i][j]) = \min \left\{ \begin{array}{l} \text{Min health to enter } (i, j+1)\ \&\ \text{save princess,} \\ \text{Min health to enter } (i+1, j)\ \&\ \text{save princess} \end{array} \right\}$$

<u>minimize</u>

ans = 4

	0	1	2	3
← 0	4 -3	1 +2	1 +4	6 -5
← 1	7 -6	1 +5	5 -4	1 +6
← 2	16 -15	8 -7	2 +5	7 -2
3	1 +2	1 +10	8 -3	5 -4

$x - 4 = 1$

$x - 3 = 5$

$x = 8$

last row

last col

$x + 5 = \min(7, 8)$

$x = 2$

$x - 7 = 1$

dP state

dP[i][j] = Min health to enter (i, j) & save princess

dP expression

$$(x + mat[i][j]) = \min \begin{cases} \text{Min health to enter } (i, j+1) \text{ \& save princess,} \xrightarrow{} dP[i][j+1] \\ \text{Min health to enter } (i+1, j) \text{ \& save princess} \xrightarrow{} dP[i+1][j] \end{cases}$$

<u>minimize</u>

$$\underbrace{x}_{dP[i][j]} + mat[i][j] = \min(dP[i][j+1], dP[i+1][j])$$

$$\boxed{dP[i][j] = \max(1, \min(dP[i][j+1], dP[i+1][j]) - mat[i][j])}$$

→ We are taking max to ensure that health required > 0

$i = N-1$
&
$j = M-1$

dP initialization

Iterative

```
int dP[N][M]

if(mat[N-1][M-1] > 0){
    dP[N-1][M-1] = 1
}
else{
    dP[N-1][M-1] = abs(mat[N-1][M-1]) + 1
}

//fill last row
for(int j = M-2; j >=0; j--){
    dP[N-1][j] = max(1,dP[N-1][j+1]-mat[N-1][j])
}
//fill last col
for(int i = N-2; i >=0; i--){
    dP[i][M-1] = max(1, dP[i+1][M-1] -mat[i][M-1])
}

for(int i = N-2; i >=0; i--){
    for(int j = M-2; j >=0; j--){
        dP[i][j] = max(1, min( dP[i][j+1], dP[i+1][j]) - mat[i][j])
    }
}
```

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

return dP[0][0]

④ ①
-3

mat ← j M-1

	0	1	2	3
0	-3	+2	+4	-5
1	-6	+5	-4	+6
N-2th 2	-15	-7	+5	-2
N-1th 3	+2	+10	-3	-4

i

dP j M-1

	0	1	2	3
0	4	1	1	6
1	7	1	5	1
2	16	8	2	7
3	1	1	8	5

Q4) Given N arr[ ] elements, find max Subsequence Sum.

Note - In a subsequence, 2 adjacent elements cannot be present.

<u>all ele > 0</u>

<u>Ex</u> {9 14 3} : ans = 14

<u>Ex</u> {9 4 13 24} : ans = 33

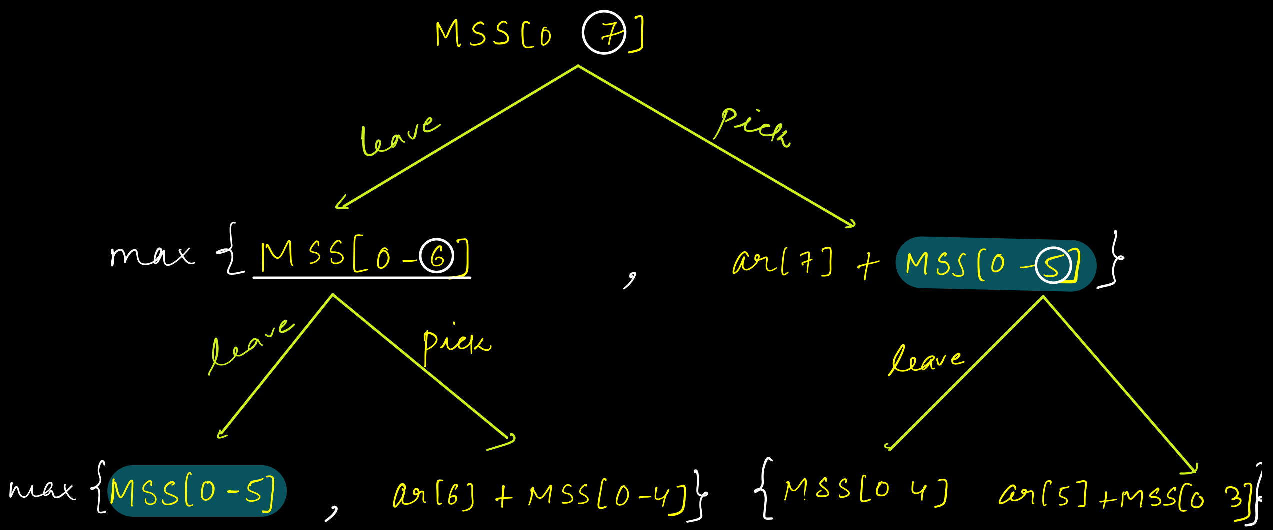
<u>Ex</u> {13 14 2} : ans = 15

<u>Ex</u> $\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 5 & 6 & 3 & 8 \end{array}$ $\cancel{9}$

15]

15

N = 8



<u>N</u>

$$\frac{\text{MSS}[0 \quad N-1]}{\downarrow}$$

dp[N-1]

dP state :-

max N-1

dP[i] = Max Sum Subsequence from [0 i] without adjacent elements.

dP expression :-

$\underline{[0 \quad i]}$ dP[i]

leave

$$\max\left\{\underbrace{MSS[0 \quad i-1]}_{dP[i-1]} \;,\; ar[i] + \underbrace{MSS[0 \quad i-2]}_{dP[i-2]}\right\}$$

$$\boxed{dP[i] = \max(dP[i-1],\ ar[i] + dP[i-2])}$$ ✓

i=0

i=1

dP initialization :-

MSS[0 0]

for i=0, $\underline{dP[0]}$ = ar[0]

for i=1, dP[1] = max(ar[0], ar[1])

Pseudo Code :-

```
int MSS(int ar[N]) {
    int dP[N]
    dP[0] = ar[0]
    dP[1] = max(ar[0], ar[1])
    for(int i=2; i<N; i++){
        dP[i] = max (dP[i-1]  ar[i] + dP[i-2])
    }
    return dP[N-1]
}
```

TC : O(N)
SC : O(N)

Bottom up DP
Iterative DP
Tabulation

MSS[0 N-1]

dP[N-1]

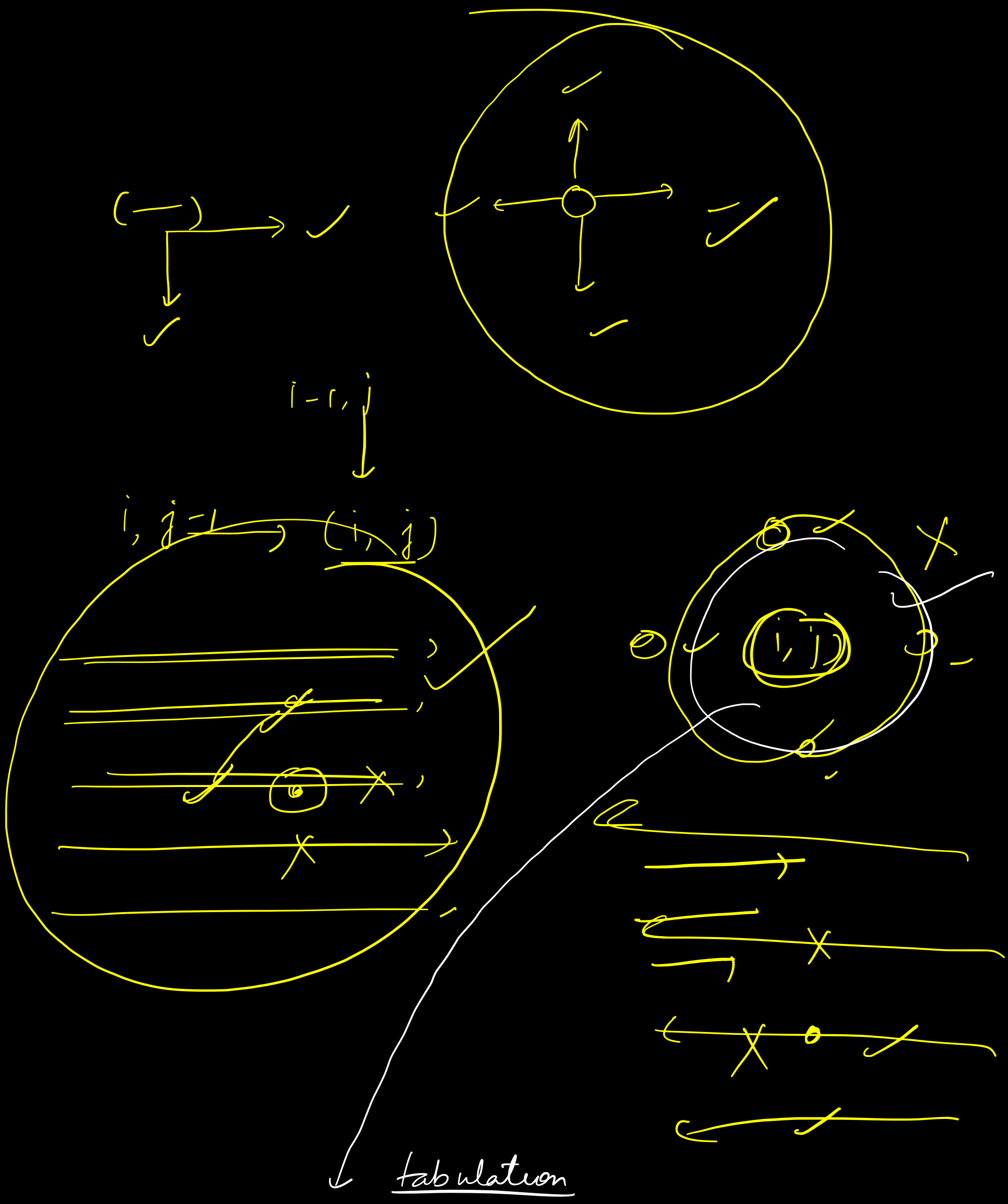
```
int dP[N] = {-1}

int MSS( int ar[], int i) {
    if (i==0) return ar[0]
    if (i==1) return max(ar[0], ar[1])
    if(dP[i] != -1) return dP[i]
    dP[i] = max (MSS(ar, i-1),
                 ar[i] + MSS(ar, i-2))
    return dP[i]
}
```

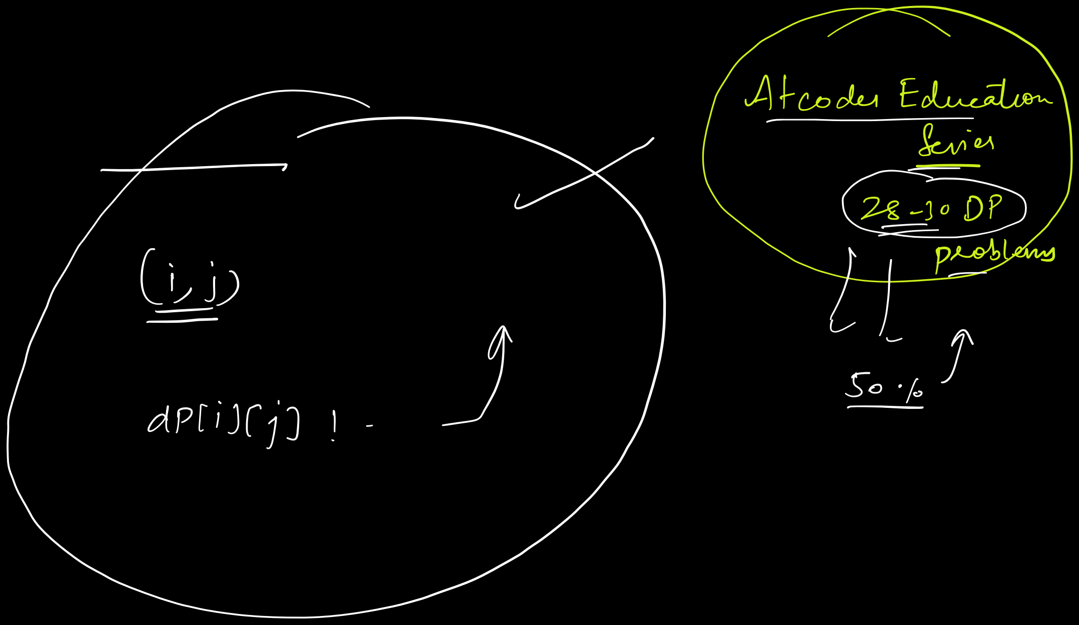
MSS[0 i]

MSS(ar, N-1) ✓

Top Down
Recursive
Memoization

9521279429





(i, j)

dp[i][j] ! -

Atcoder Education Series

28-30 DP problems

50%