Today's Agenda :-

Welcome everyone

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 ✓

ASSUME

---

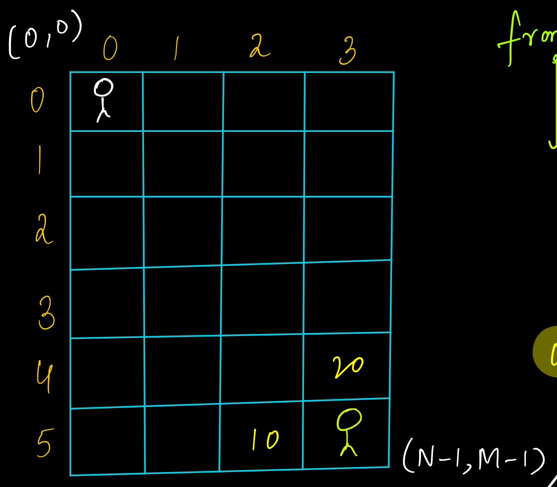
3) dp state   dp[i] = ? (Assumption)

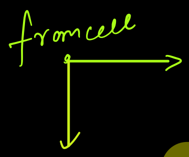
4) dp expression   (Main Logic)

5) dp intialization (Base Condition)

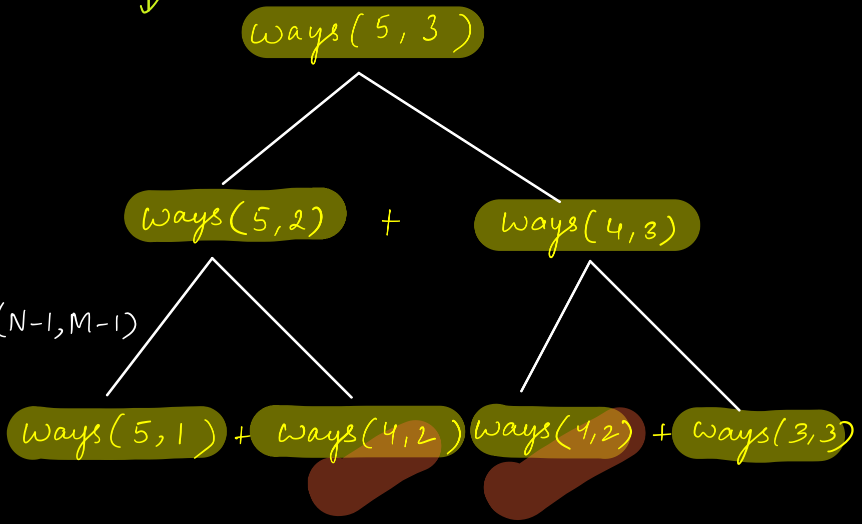
$N * M$

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





1) ✓
2) ✓



3) dP state

dP[i][j] = No of ways to reach (i, j)

4) dP expression

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

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

5) Base Condition

∀ i = 0 ✓

&&

∀ j = 0 ✓

!

for i=0 & j=0, formula fails

```
int dP[N][M]

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

$$\overset{M-1}{\underset{j=0}{\forall}} dP[0][j] = 1$$

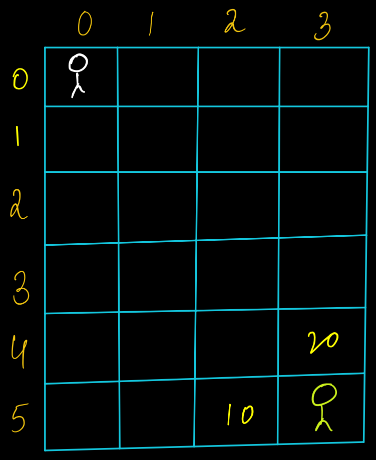
```
for (int i = 0; i < N; i++){
    dP[i][0] = 1
}
```

$$\overset{N-1}{\underset{i=0}{\forall}} dP[i][0] = 1$$

TC : O(N*M)

SC : O(N*M)

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



dP[ ][ ]	0	1	2	3
0	1	1	1	1
1	1			
2	1			
3	1			
4	1			
5	1			

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

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

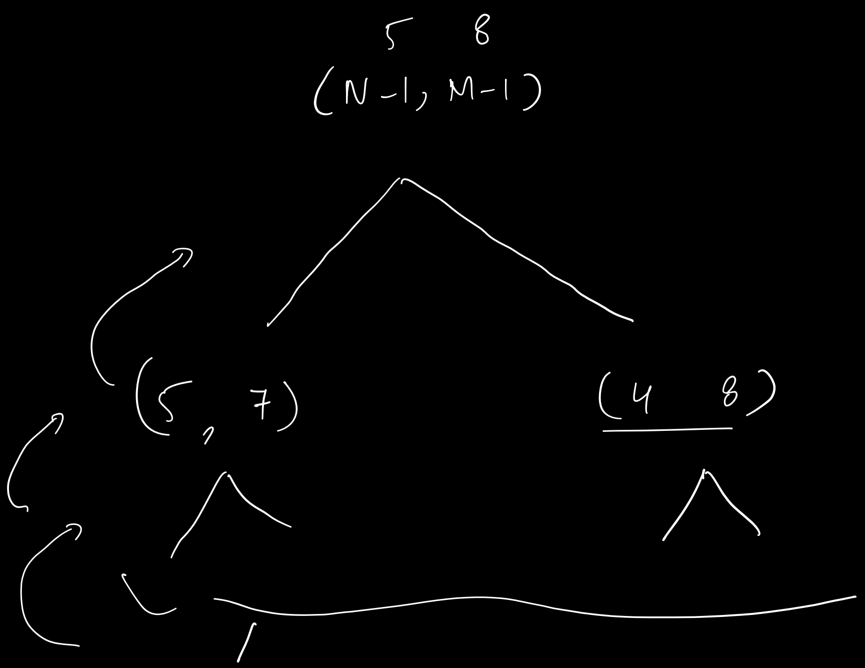
Update

X

(0,0) - 1 ✓
(0,0) - 0 ✓

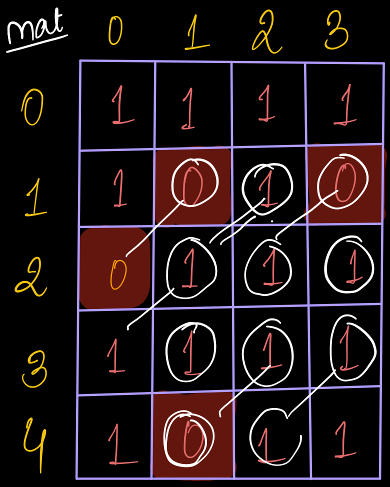
```
int  dP[N][M] = {-1}
int  ways( int , int  j){  if(i==0 && j==0) return
                                          mat[0][0]
   if(mat[i][j] == 0) return  0 ;
   if( i == 0)  return  ways(i, j-1)
   if (j == 0)  return  ways(i-1, j)
   if (dP[i][j] != -1) return dP[i][j] ;

   else return  dP[i][j] = ways(i, j-1) +
                            ways(i-1, j);
}
```

ways (N-1, M-1)



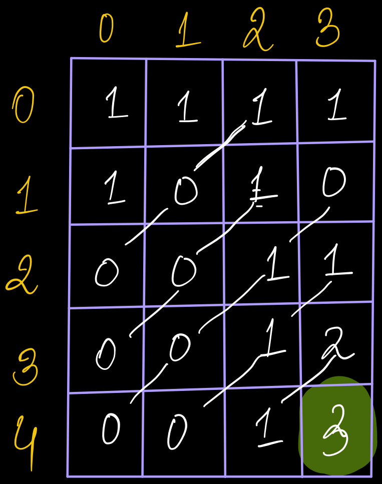
Iteration
Bottom

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

✓

```
int dP[N][M] = {0}

for(int j = 0; j < M; j++){
    if (mat[0][j] == 0)
        break
    }
    else{
        dP[0][j] = 1
    }
}
```

✓

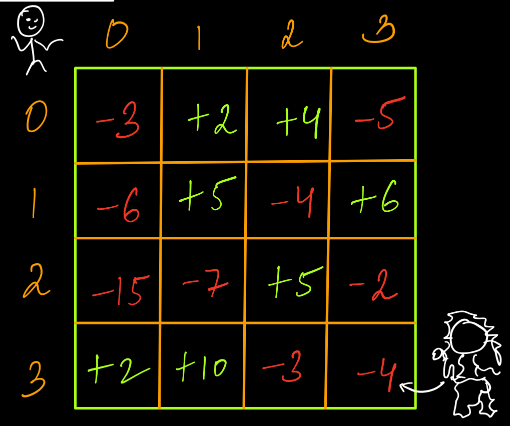
```
for(int i=0; i<N; i++){
    if (mat[i][0]==0)
        break
    }
    else{
        dP[i][0]=1
    }
}

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

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

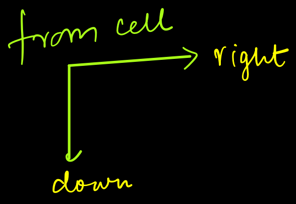
Break of
8 Min

HQ) Dungeons & Princess (Hard)

$\boxed{h = 4}$ ✓

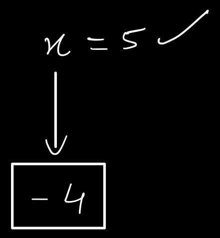


	0	1	2	3
0	-3	+2	+4	-5
1	-6	+5	-4	+6
2	-15	-7	+5	-2
3	+2	+10	-3	-4



$\boxed{h \le 0, \text{ prince dies}}$

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

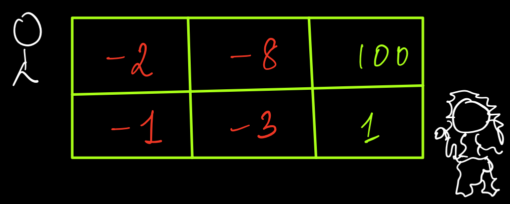


$x - 4 > 0$

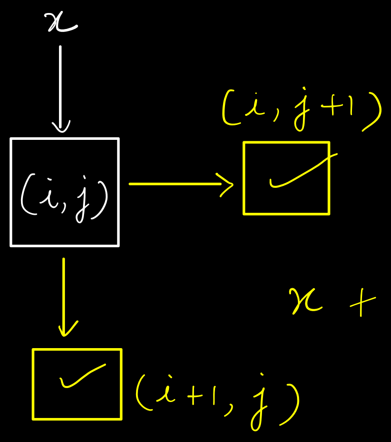
$x - 4 = 1$

$x = 5$

Ex2 $\boxed{h = 7}$



-2	-8	100
-1	-3	1



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

the princess

dP state

$$\boxed{dP[i][j] = \{\text{Min health to enter } (i,j) \text{ \& save the princess}\}}$$

dP expression

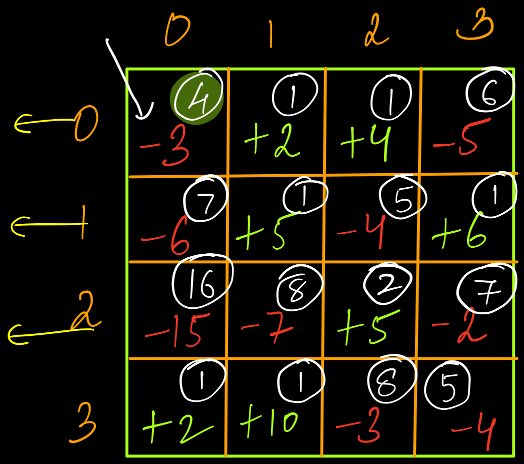
$$\underline{ⓧ + mat[i][j]} = \min(dP[i][j+1],\ dP[i+1][j])$$

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

max(1,

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

✓



	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

Let's fill the last row

last col

$$\boxed{\text{final ans} = dP[0][0]}$$

$$x + \underline{mat[\,][\,]} = \underline{dP[N-1][j+1]}$$

$$x =$$

```
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

// for filling last row

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

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])
    }
}
return dP[0][0]
```

TC : O(N*M)

SC : O(N*M)

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

int minHealth( int i, int j) {
    if(i >= N || j >= M) return ∞
    if(i == N-1 && j == M-1) {
        if (mat[N-1][M-1] > 0) return 1
        else return abs(mat[N-1][M-1]) + 1
    }
    if(dP[i][j] != -1) return dP[i][j]

    int a = minHealth(i+1, j)
    int b = minHealth(i, j+1)

    return dP[i][j] =
              max(1, min(a,b) - mat[i][j])
}
```

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

	0	1	2	3
0				
1				
2				
3				

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

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

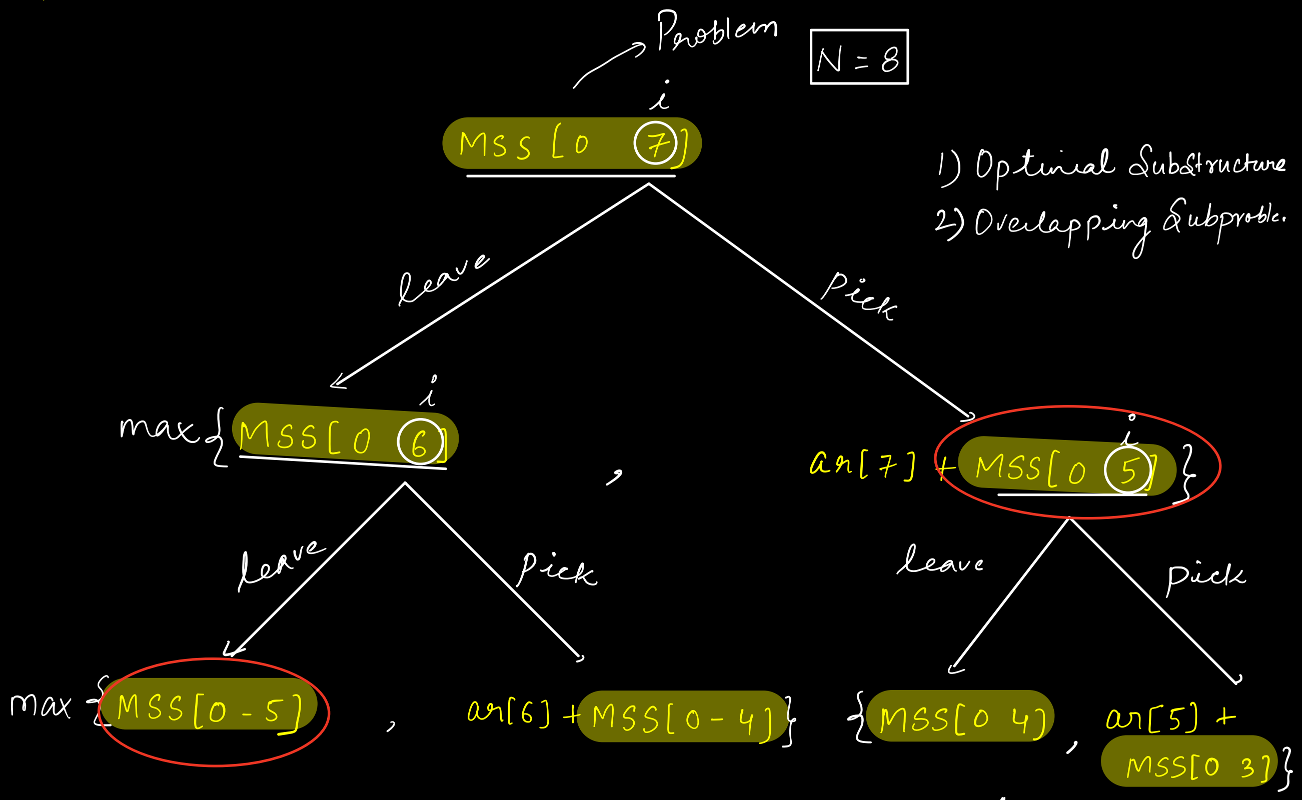
all ele > 0

Ex { 9 14 3 } : ans = 14

Ex { 9 4 13 24 } : ans = 33

Ex { 13 14 2 } : ans = 15

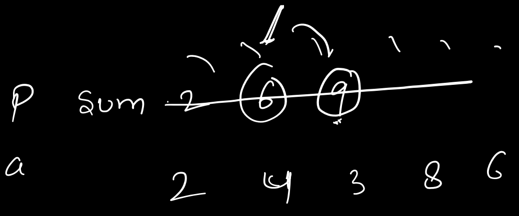
MSS [0 7]



N = 8

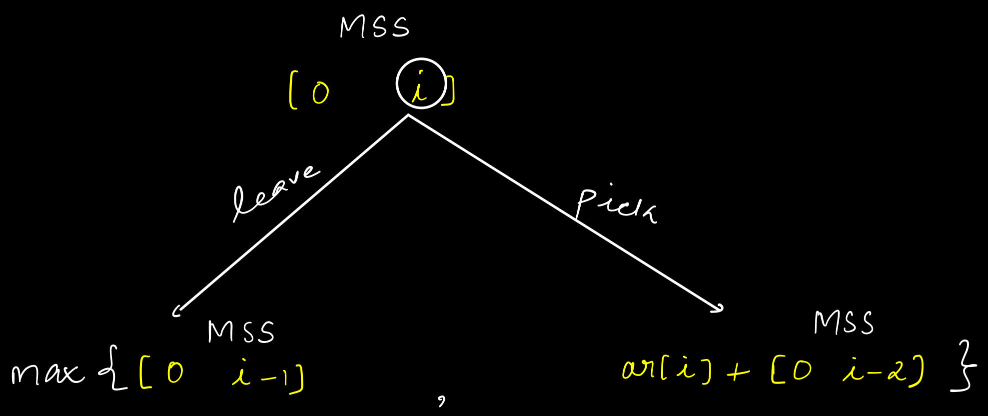
1) Optimal Substructure

2) Overlapping Subproble.



3) dP state

dP[i] = MSS from [0 i] such that adjacent elements are not present



4) dP expression

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

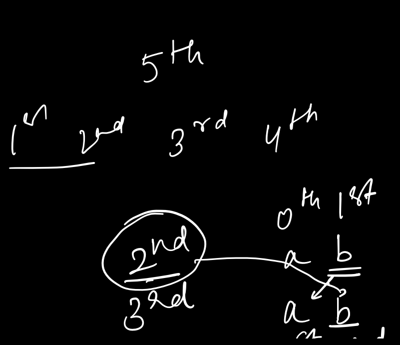
$\boxed{i = 0, 1}$ ✓

5) Base Condition

MSS[0 0]

MSS[0 1]

MSS[0 N-1]



$\overset{i}{dP[5]} = dP[4]$ dP[3]

dP[0] * ✓

dP[1] ×

dP[2] ×

ith

Nth

N-1 N-2

```
       1st 2nd
    if(N==1) return ar[0]

c   int dP[N];                          a
a | dP[0] = ar[0]                       b
b | 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])
    |    c            a                  b
    }
    return dP[N-1];      a = b
              c          b = c  ✓
```

MSS [0 ... N-1]

dP[N-1]

MSS[0 N-2]

TC : O(N)
SC : O(N) ✓

Bottom Up DP ✓

Iterative

{9 4 13 24} :   max(22, 24+9)

33

	0	1	2	3
dP	9	9	22	33

33 ans ✓

Ex {9 4 13 24}

inc	9	4	22	33
exc	0	9	9	29
ninc		4	22	33
nexc		9	9	22

→ max 33

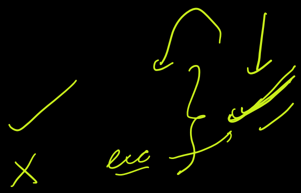
0	1	2	3	4
3	4	5	8	6

dP[0] = ar[0]

inc	3	4	8	12
exc	0	3	4	8
ninc		4	8	12
nexc		3	4	8

Recursive
Memoization
Top Down

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

MSS(ar, N-1) : ans
```



inc

<u>exc</u>

6 10 2 1 1 15

6 10 8

0 6