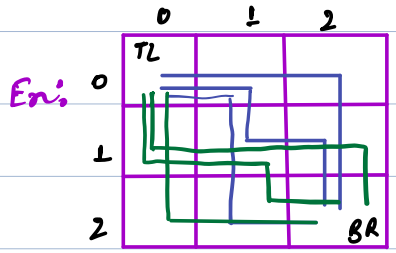
Today's agenda → 2d arrays dP

- unique Paths in grid
- Min Sum Path
- Dungeon Princess

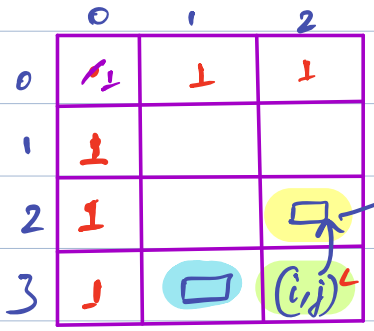
- Subhesh
- DTV graduate
- 5* on codechef → ICPC regional
- 3 yost teaching emP
- youtuber → PeRcoding → level-3 (CP)
  → Graphs (level-2) } 100K+ views
- SDE @ instructor
  ↓
  Advanced module

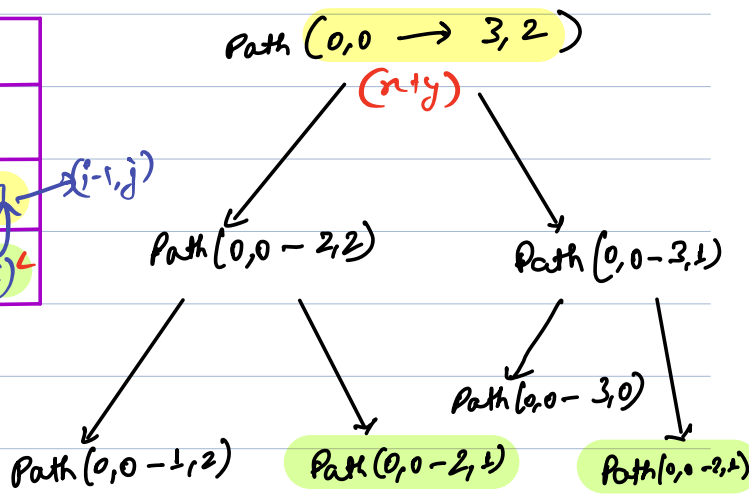
Q) Number of ways to go from (0,0) → (n-1, m-1) cell.

Movement allowed: cell → 1 step right or ↓ 1 step bottom



Ans: 6



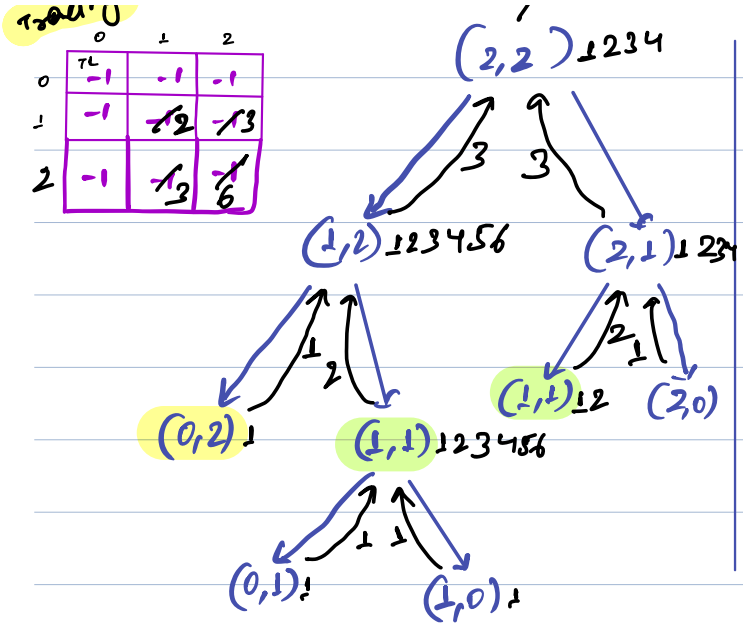


//Pseudo Code

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

```
int ways (int i, int j) {
    if (i == 0 || j == 0) {return 1;}
    if (dp[i][j] != -1) {return dp[i][j];}

    int a = ways (i-1, j);
    int b = ways (i, j-1);

    dp[i][j] = a+b;
    return a+b;

}
```

DRY




```
int ways (int i, int j) {
① if (i==0 || j==0) {return 1;}
② if (dp[i][j]!=-1) {return dp[i][j];}
③ int a = ways (i-1,j);
④ int b = way (i,j-1);
⑤ dp[i][j] = a+b;
⑥ return a+b;
}
```

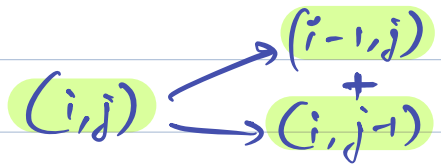
T.C : O(n*m) S.C : O(n*m) + Stack Space

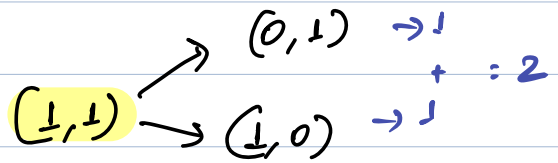
// Iterative idea

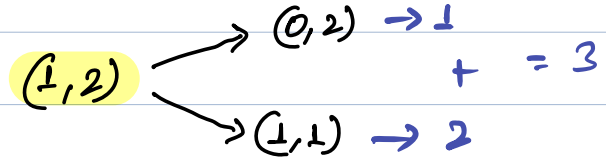
dp

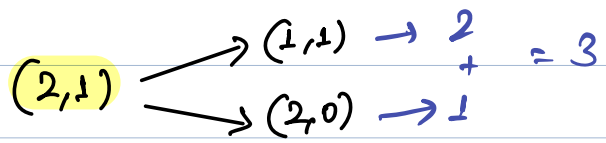
	0	1	2
0	1 (TL)	1	1
1	1	2	3
2	1	3	6 (BR)

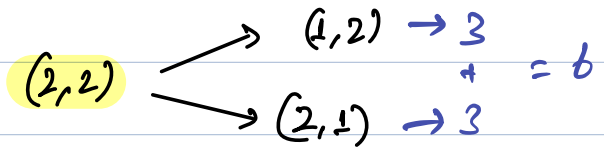
Ans = dp[2][2] = 6











//Pseudo code

```
int countPath ( int N, int M) {
    int dp[N][m];

    for(int j=0; j<m; j++) { //0th row
        dp[0][j] = 1
    }

    for(int i=0; i<N; i++){ // 0th col
        dp[i][0] = 1
    }

    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];
}
```

T.C: O(N*m) S.C: O(N*m)

Q) Minimum Path Sum

↳ Given a 2d matrix, filled with non-negative numbers, find a Path from (0,0) ↔ (N-1, m-1) which minimizes the total cost of Path.

↳ Sum of all the elements on that Path.

Note: move right or down

Ex:

	0	1	2
0	2	3	5
1	1	6	4
2	9	2	7

(i-1, j) → 4, (i, j) → 7, (i, j-1) → 2

dp State: dp[i][j] = min cost to reach (i,j) from (0,0)

dp expression:

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

arr

	0	1	2
0	2	3	5
1	1	6	4
2	9	2	7

dp

	0	1	2
0	2	5	10
1	3	9	
i→2	12		

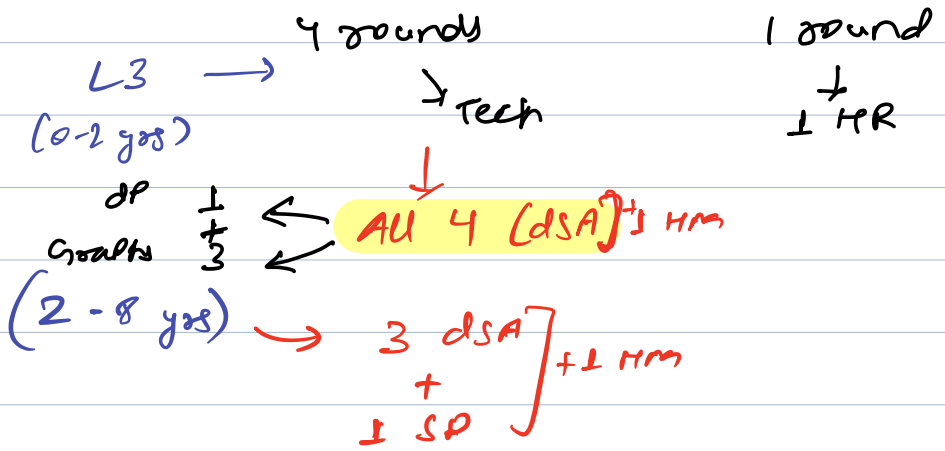
iterative

//Pseudo Code

```
int minPathSum(int arr[N][M]){
    int dp[N][M];
    dp[0][0] = arr[0][0];
    for(int j=1; j<M; j++){ //0th row
        dp[0][j] = dp[0][j-1] + arr[0][j]
    }

    for(int i=1; i<N; i++){
        dp[i][0] = dp[i-1][0] + arr[i][0];
    }

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

T.C : O(N*M) S.C: O(N*M)

Break till 10:30 Pm


4 rounds
1 round
L3
(0-2 yrs)
Tech
1 HR
DP
Graphs
1
+
3
All 4 (dSA) + 1 HM
(2 - 8 yrs)
3 dSA
+
1 SD
+1 HM

Q) Dungeon Princess

↳ Given mat[n][m] where each cell indicates health gained.

→ Find out min health required at (0,0) so that we can reach [n-1, m-1].

Note: ① movements : right or bottom.

② if health reaches 0, at any place you are dead.

③ we are starting at (0,0).

Ex:

	0	1
0	-3	-5
1	-2	1

~~4~~ ~~5~~ 6

Ex:

	0	1	2	3
0	-3	2	4	-7
1	-6	5	-4	6
2	-15	-8	3	-4
3	7	4	-2	-7

$\min(a,b)$ - arr[i][j]

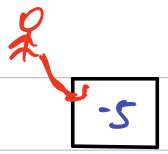
minH[0-0 → 3,3]

right

minH[0,1 - 3,3] (a)

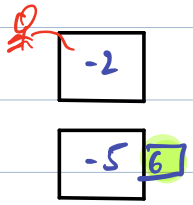
minH[1,0 → 3,3] (b)

Case 1:



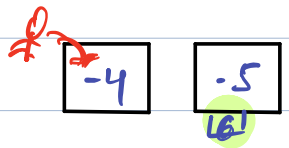
$H + (-5) = 1 \Rightarrow H = 1 - (-5) = 6$

Case 2:



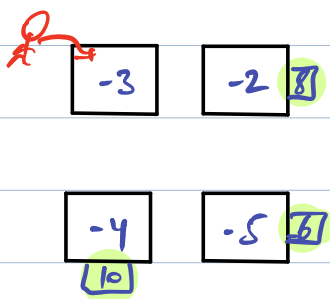
$H + (-2) = 6 \Rightarrow H = 6 - (-2) = 8$

Case 3:



$H + (-4) = 6 \Rightarrow H = 6 - (-4) = 10$

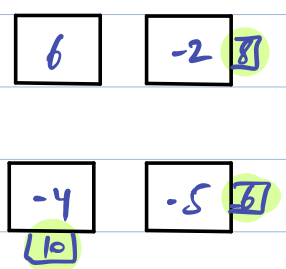
Case 4:



$H + (-3) = 8$

$H = 8 - (-3) = 11$

Case 5:

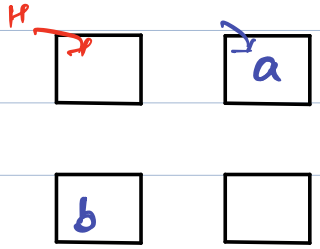


$H + 6 = \min(8, 10)$

$H = 8 - 6 = 2$

$\min(a, b) - arr[i][j]$

generic case:



$H + arr[i][j] = \min(a, b)$

$H = \min(a, b) - arr[i][j]$

$H = \max(\min(a,b) - arr[i][j], 1)$

Case 6:

16	-2
-4	-5

-8

$H = (\min(8, 10) - 16)$

$= \max(-8, 1) = 1$

$H + 16 = 18 \Rightarrow H = 18 - 16 = 2$

	0	1
0	16	2
1	-4	-19

$H + 2 = 20 \Rightarrow H = 20 - 2 = 18$

$H + (-4) = 20$

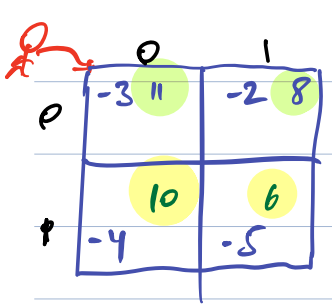
$H = 20 - (-4) = 24$

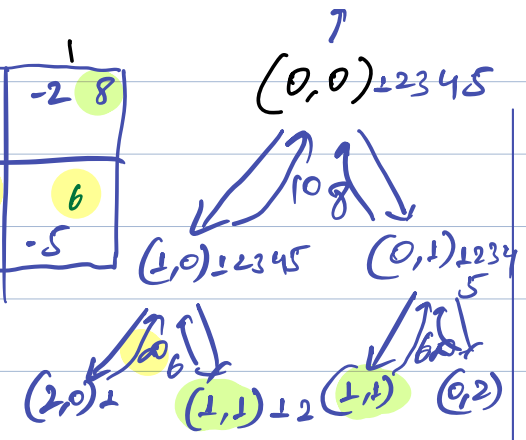
	0	1	2	3
0	-3	2	4	-7
1	-6	5	-4	6
2	-15	-8	3	-4
3	7	4	-2	-7

//Pseudo code

```
int dp[N][M] = {-1};

int minH (int mat[N][M], int i, int j){
    if (i>=N || j>=M) {return INT_MAX;}
    if (i==N-1 && j==M-1) {return mat[i][j];}
    if (dp[i][j]!=-1) {return dp[i][j];}

    int a = minH(mat, i+1, j);
    int b = minH(mat, i, j+1);

    dp[i][j]= max(min(a,b) - mat[i][j],1);
    return max(min(a,b) - mat[i][j],1);

}
```

T.C: O(N*M) S.C: O(N*M) + Stack space






```
int minH (int mat[n][m], int i, int j){
① if (i>=N || j>=m) {return INT_MAX;}
② if (i==N-1 && j==M-1) {return max(1, 1-mat[i][j]);}
③ int a = minH(mat, i+1, j);
④ int b = minH(mat, i, j+1);
⑤ return max(min(a, b) - mat[i][j], 1)
}
```

6 - (-4) = 1

8 - (-3) = 11

Doubt

arr

	0	1	2	3
0	-3	2	4	-7
1	-6	5	-4	6
2	-15	-8	3	-4
3	7	4	-2	-7

dp

	0	1	2	3
0				13
1				6
2		14	7	12
3	1	6	10	8

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

8 - (-2) = 10