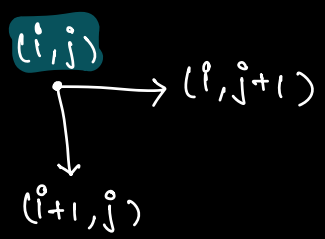
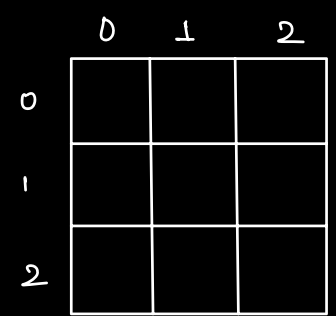
2D DP Problems.

Q. N x M matrix.

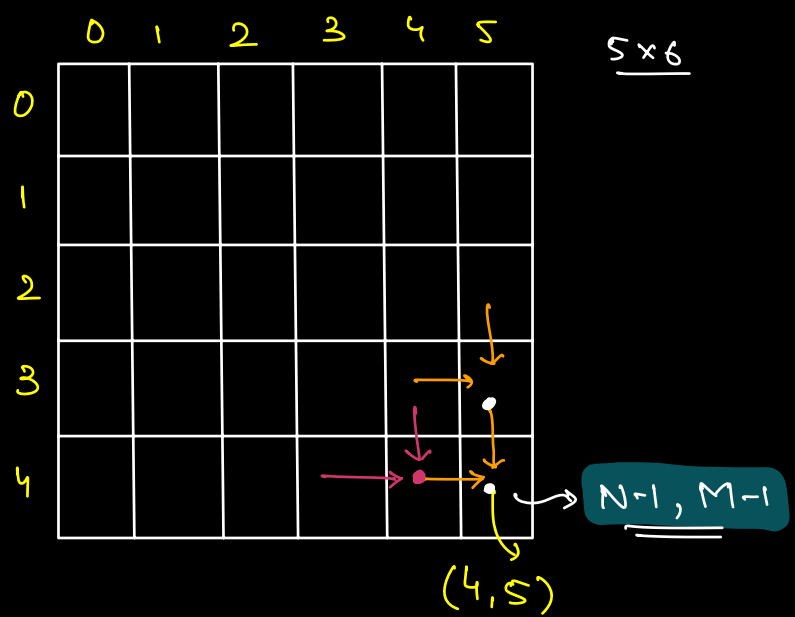
Number of ways to reach from $(0,0) \rightarrow (N-1, M-1)$





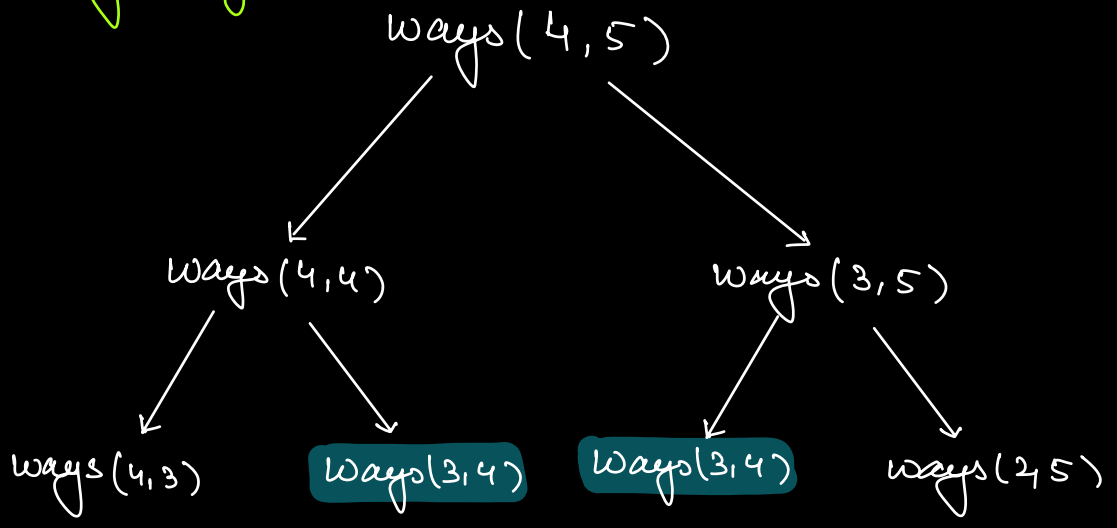
(0,0) (0,1) (0,2) (1,2) (2,2)
(0,0) (0,1) (1,1) (2,1) (2,2)
(0,0) (0,1) (1,1) (1,2) (2,2)
(0,0) (1,0) (2,0) (2,1) (2,2)
(0,0) (1,0) (1,1) (1,2) (2,2)
(0,0) (1,0) (1,1) (2,1) (2,2)

6



5 x 6

# of ways to reach from (0,0) to (4,5)



⇒ Optimal Substructure. } DP.
⇒ Overlapping subproblems }

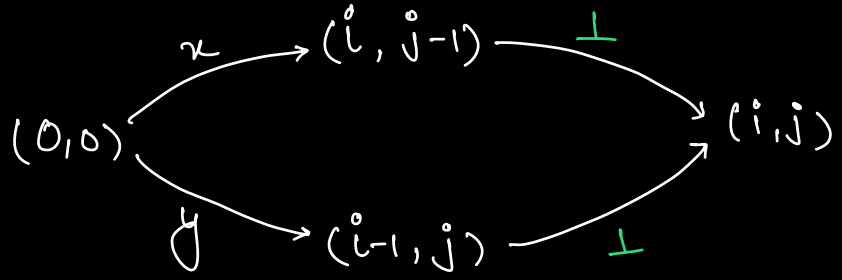
dp state.

dp[i][j] = # of ways to reach (i,j) from (0,0).

dp table :-

int dp[N][M];

dp Expression :-



$$dp[i,j] = x \times 1 + y \times 1$$

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

Base Conditions

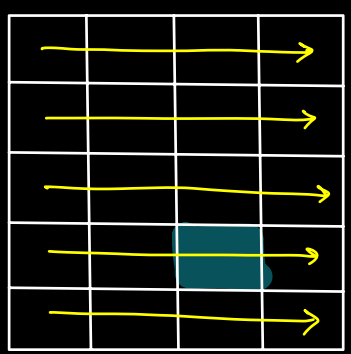
↳ $i == 0$ OR $j == 0$

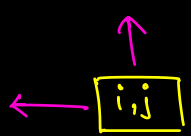
	0	1	2	3	4	5
0	1	1	1	1	1	1
1	1					
2	1					
3	1					
4	1					

int dp[N][M];

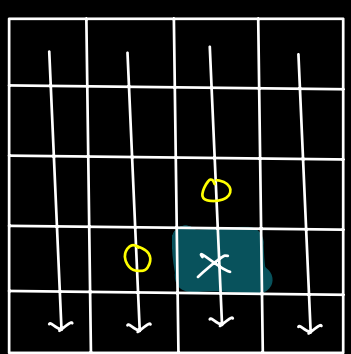
```
for(i=0; i<N; i++){
    for(j=0; j<M; j++){
        if(i==0 || j==0)
            dp[i][j] = 1;
        else{
            dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }
    }
}
```

Base Case





left to right
+
top to bottom



top to bottom
+
left to right

TC: O(NM)
SC: O(NM)

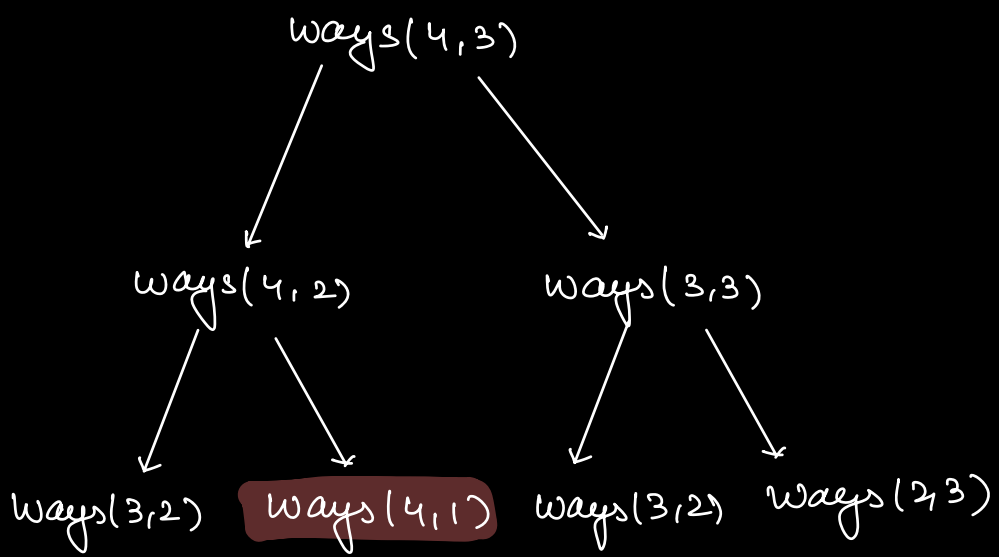
Q. No. of ways to reach to cell N-1, M-1.

mat[i][j] = 0 ⇒ Blocked

mat[i][j] = 1 ⇒ Empty

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

Ways to reach from (0,0) to (4,3).



→ Optimal substructure

→ Overlapping subproblems.

} DP.

int dp[N][M];

#

dp[i][j] ⇒ # of ways to reach from (0,0) to (i,j)

$$dp[i][j] = \begin{cases} dp[i][j] = 0 & \text{if } mat[i][j] = 0 \\ dp[i-1][j] + dp[i][j-1] \end{cases}$$

Base Cond^n

i = 0 || j = 0.

dp matrix

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

⟹

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

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

} 0th row

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

0th col

TC : O(NM)

SC : O(NM)

⇒ Bottom Up DP.

Recursive DP Solution :

↳ Recursion + Memoization.

```
int ways(int mat[][], int N, int M) {
    int dp[N][M] = {-1};
    if (mat[0][0] == 0) return 0;
    dp[0][0] = 1
    fun(mat, dp, N-1, M-1);
}
```

↳ # of ways to reach to cell N-1, M-1.

```
int fun(int mat[][], int dp[][], i, j) {
    if(i < 0 || j < 0) return 0;
    if(mat[i][j] == 0) return 0;
```

(i,j) is blocked.

```
    if (dp[i][j] == -1) {
```

i,j cell hasn't been calculated yet.

```
        dp[i][j] = fun(mat, dp, i-1, j)
                 +
                   fun(mat, dp, i, j-1);
    }
    return dp[i][j];
}
```

TC: O(NM) | SC: O(NM)