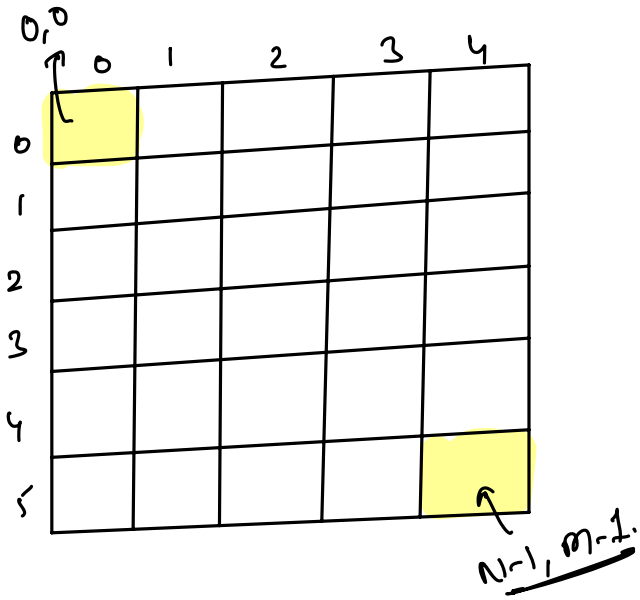
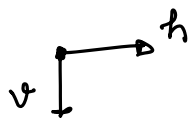
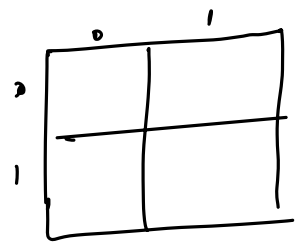
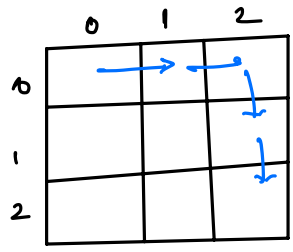
Q: Given mat[N][m], Total no. of ways from (0,0) to (N-1, m-1).







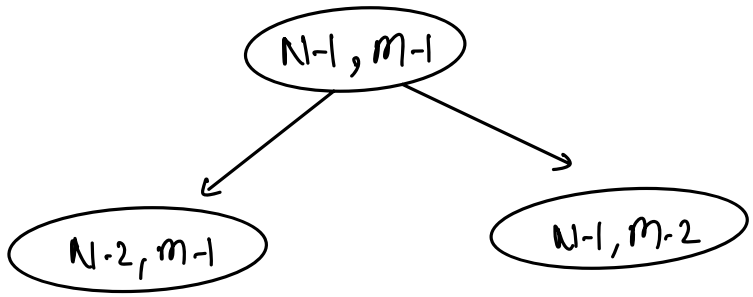
hv, vh.
ans = 2



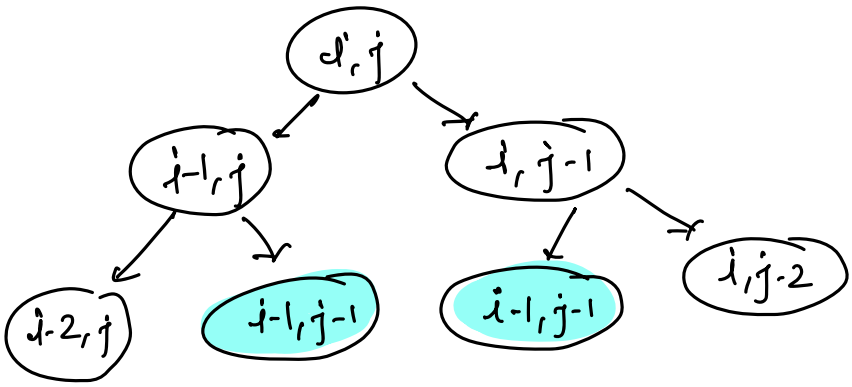
hhvv
hvhv
hvvh
vhhv
vhvh
vvhh

ans = 6.

Bf idea → Try all possible ways to reach N-1, m-1 starting from (0,0).



ways(i, j) = ways(i-1, j) + ways(i, j-1)



optimal substructure ✓

overlapping sub-problems ✓

∴

D.P.

code :-

```
                    N-1   m-1
                     ↑     ↑
int  ways ( i , j ){
    if ( i == 0 || j == 0) {return 1 }
    return ways( i-1, j) + ways( i, j-1) ;
}
```

T.C → $O(2^{N+m})$
S.C → $O(N+m)$

```
dp [N][m]  // ∀ dp[i][j] = -1
```

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

T.C → $O(N*m)$
S.C → $O(N*m)$

	0	1	2	3	4
0	1	1	1	1	1
1	1	2	3	4	5
2	1	3	6	10	15
3	1	4	10	20	35
4	1	5	15	35	70
5	1	6	21	56	126

ans.

total no. of ways to reach (0,0) starting from (0,0)?

⇓

0 → You can't reach (0,0) starting from (0,0)

1 → Do nothing

dp[i][j] = total no. of ways to reach (i,j) starting from (0,0)

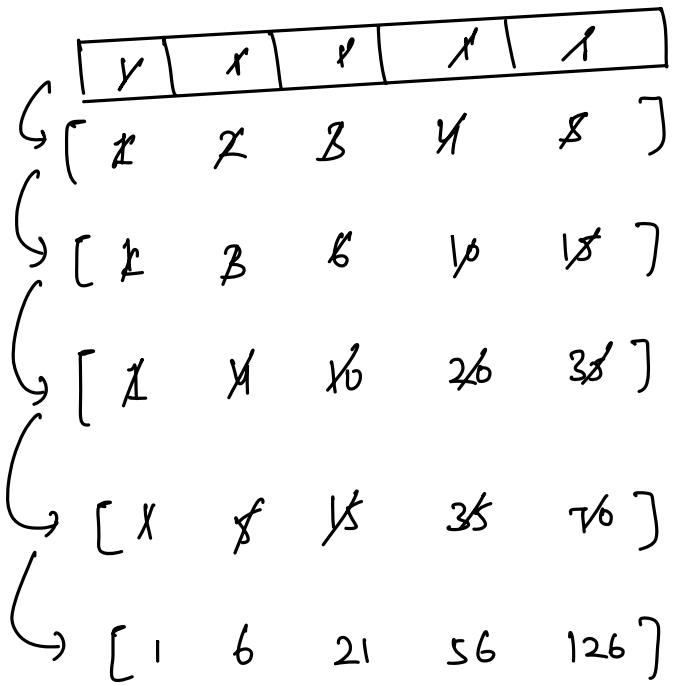
#code:-

```
initialise row->0 with 1
initialise col->0 with 1

for( i=1; i<N; i++){
    for( 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)

further SC optimisation ?



Apply psum n-1 times.

final code

```
int dp[m];    ∀i, dp[i]=1;

for( i=1; i<N; i++){
    for( j=1; j<m; j++){
        dp[j] = dp[j-1] + dp[j];
    }
}

return dp[m-1];
```

T.C → O(N*m)
S.C → O(m)

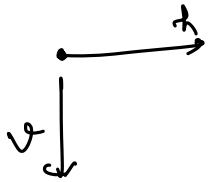
Q2

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

arr[N][m]

arr[i][j] = 0 ⇒ cell is blocked

arr[i][j] = 1 ⇒ cell is empty.



initialise row → 0 with 1

initialise col → 0 with 1

```
for( i = 1; i < N; i++){
    for( j = 1; j < m; j++){
        if( arr[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];
```

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

→

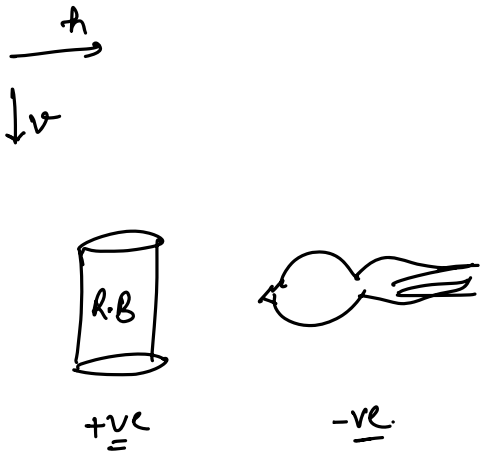
	0	1	2	3
0	1	1	1	1
1	1	0	1	0
2	0	0	1	1
3	0	0	1	2
4	0	0	1	3

→ #todo

s.c optimisation

Dungeons & Princess

	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 drops to less than or equal to 0, then the person dies.

→ What should the minimum health level with which you can enter top-left cell so that you save the princess.

①

-3 →	-2
-4	1

✗ h.l = 1

1 - 2 = -1 ✗

1 - 4 = -3 ✗

② Path with minimum sum ✗

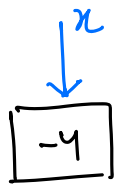
	0	1	2	3
0	-3	+2	+4	-5
1	-6	+5	-4	+6
2	-15	-7	+5	-2
3	+2	+10	-3	-4

③ Path with maximum Sum

-2	-8	100
-1	-3	1

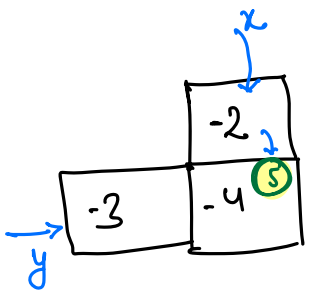
q1 ≡ ✗

Let's start with smallest problem.



$x + (-4) = 1$

$x = 1 + 4 = 5$

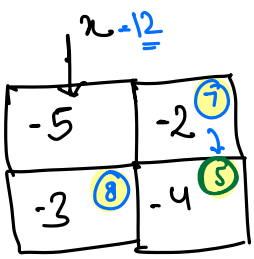


$x + (-2) = 5$

$x = 5 + 2 = 7$

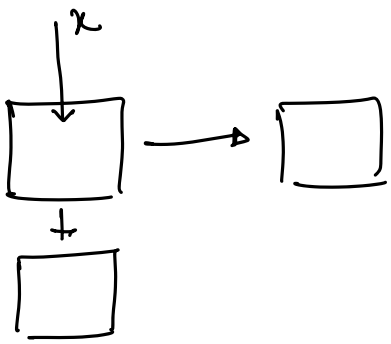
$y + (-3) = 5$

$y = 5 + 3 = 8$



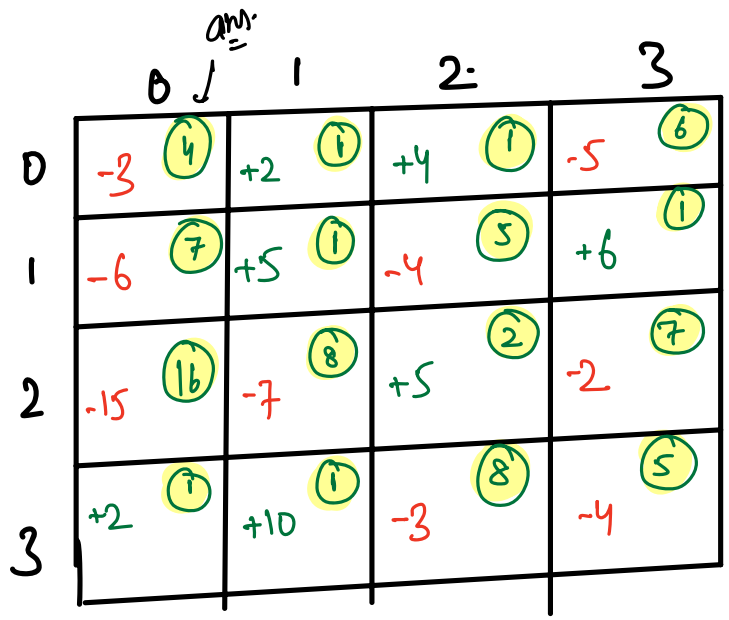
$x + (-5) = \min(7, 8)$

$x = 7 + 5 = 12$



$$x + arr[i][j] = \text{Min}\begin{bmatrix} \text{min Energy required to enter } [i+1][j], \\ \text{min Energy required to enter } [i][j+1] \end{bmatrix}$$

$$x = \text{Max}\left(1, \text{Min}\begin{bmatrix} \text{min Energy required to enter } [i+1][j], \\ \text{min Energy required to enter } [i][j+1] \end{bmatrix} - arr[i][j]\right)$$



$x + 5 = \text{Min}(7, 8)$

$x = 7 - 5 = 2$

codes

```
dp[N][M];

if( arr[N-1][M-1] > 0) { dp[N-1][M-1] = 1 }
else { dp[N-1][M-1] = Abs(arr[N-1][M-1]) + 1 }

//last row
for( j = M-2; j >=0; j--){
{
    dp[i][j] = Max( 1, dp[N-1][j+1] - arr[N-1][j]
}
```

// last col

```
for( i = N-2 ; i >= 0 ; i--){
    dp[i][m-1] = Max(1, dp[i+1][m-1] - arr[i][m-1]);
}
```

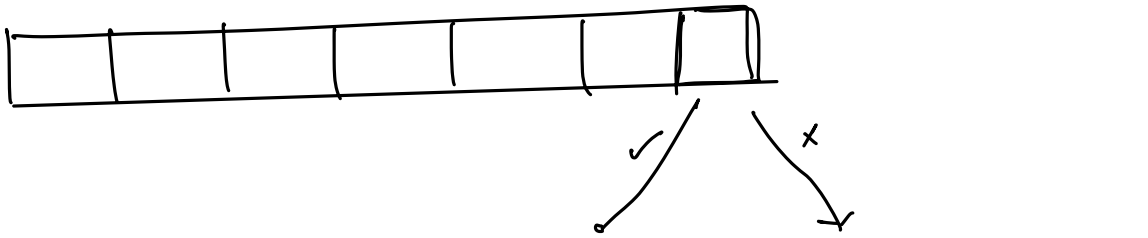
```
for( i = N-2 ; i >= 0 ; i--){
    for( j = m-2 ; j >= 0 ; j--){
        dp[i][j] = Max( 1 , Min( dp[i+1][j] , dp[i][j+1]) - arr[i][j]);
    }
}
```

```
return dp[0][0];
```

T.C → O(N*m)
S.C → O(N*m)

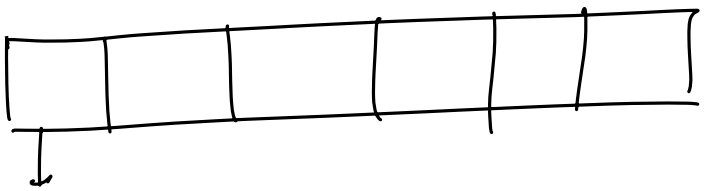
↓

SC → O(m)



arr[N-1] +
maxSum(arr, N-2)

maxSum(arr, N-1)



(0, arr[0])

$\forall$
$i \rightarrow 0, N-1$

$$dp[i] = \max \begin{pmatrix} 0 + dp[i-1] \\ arr[i] + dp[i-2] \end{pmatrix}$$

Maximum XOR Subarray - https://codeshare.io/X86YgM