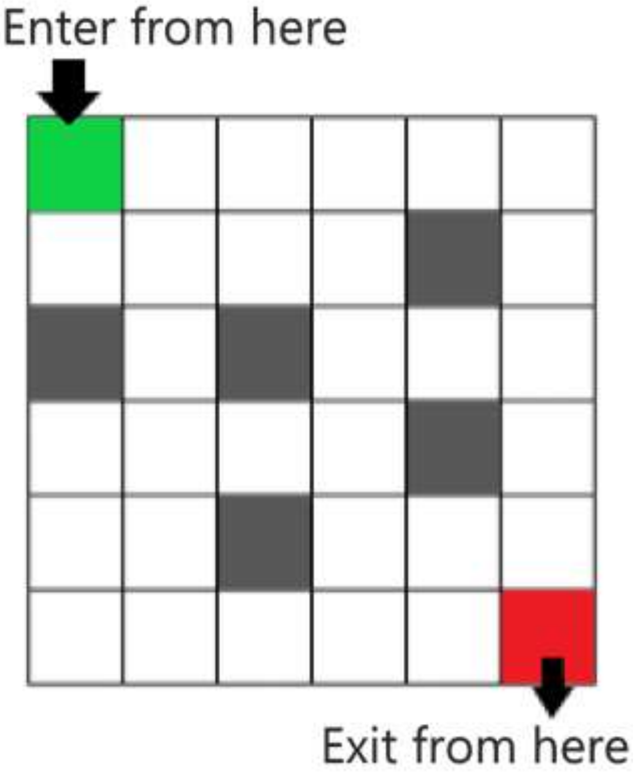
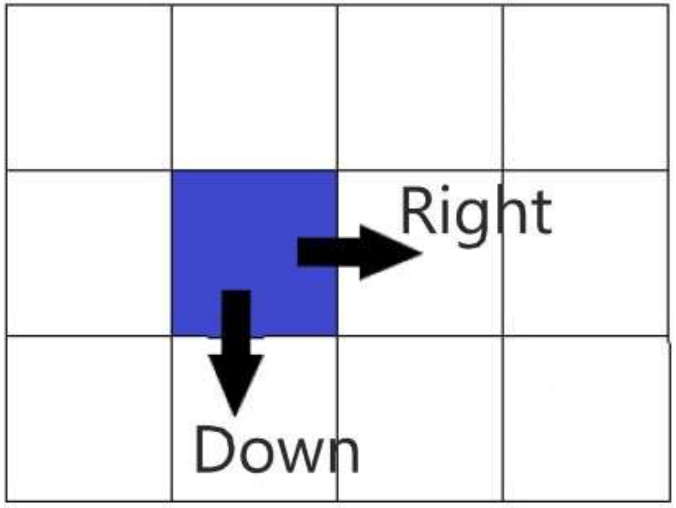
Dynamic Programming

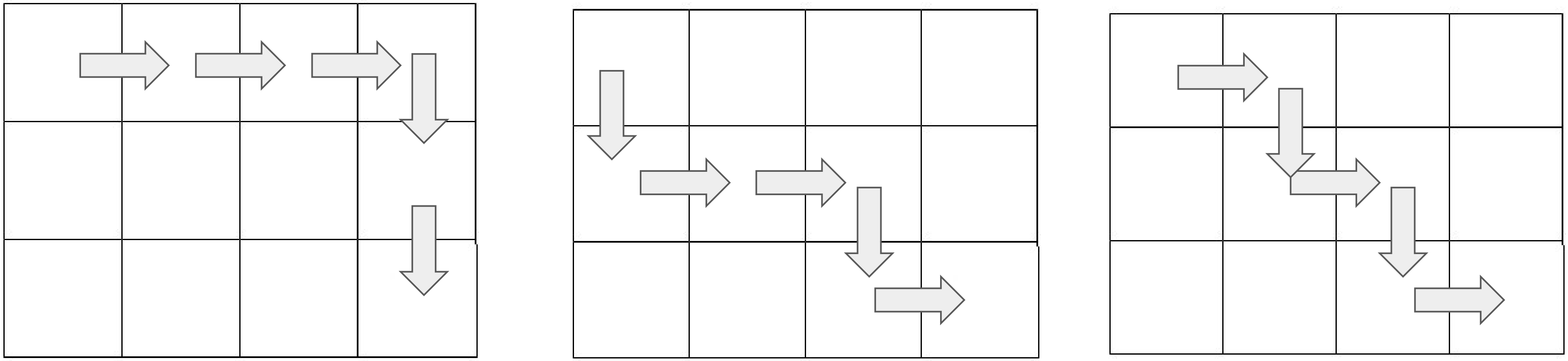
Q.NO. How many ways to go from start to end of maze with some boxes blocked and we can only move to right or down.





Left and down movements

Let’s start with easy version (without any box blocked)



Observation

One thing to observe was every figure had-
3 Arrows with this shape ⇨

And 2 Arrows with this shape ⇩

So we can calculate our answer via P&C….

Q. No. of ways to arrange 3 ⇨ and 2 ⇩ is

Ans. (3+2)!/(3!*2!)

So Basically for M x N Grid no. of ways is equal to

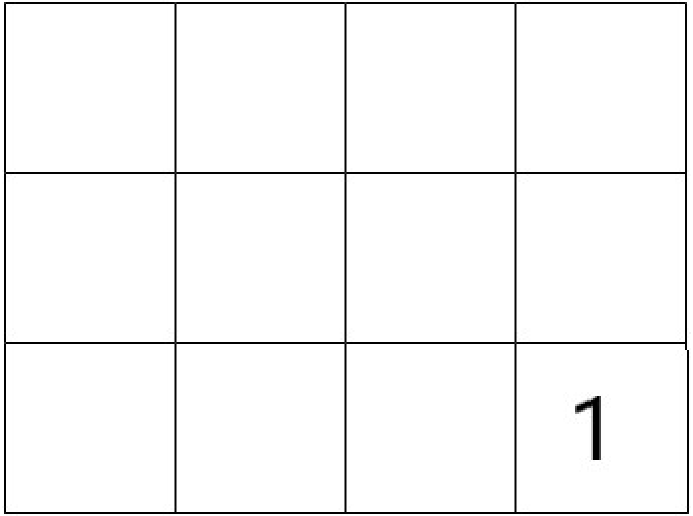
$$\frac{(M-1 + N-1)!}{(M-1)!\,(N-1)!}$$

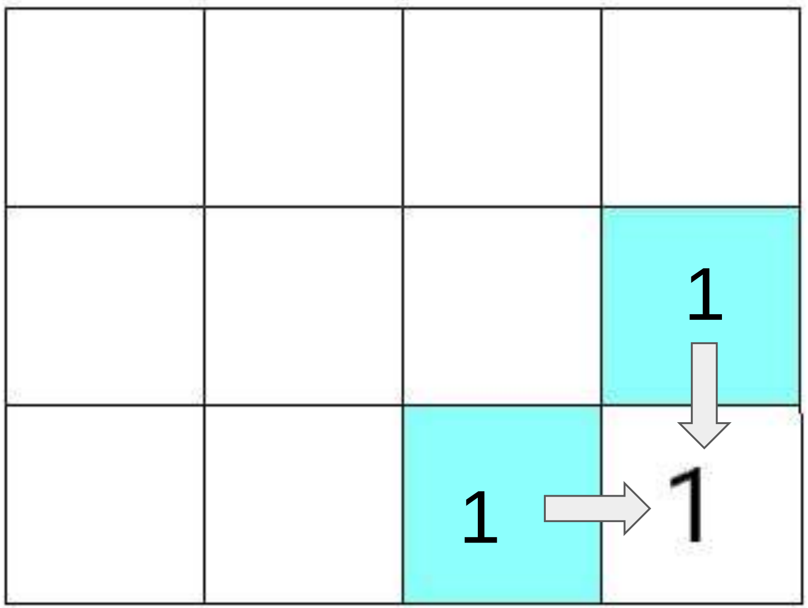
DP Approach

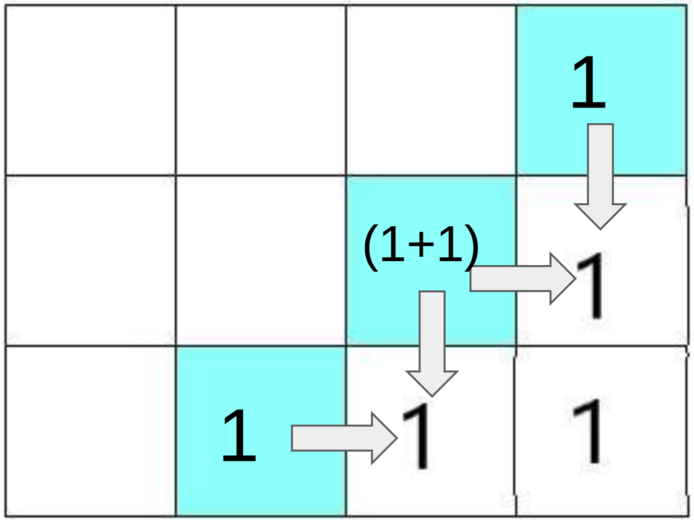
Lets Calculate for each cell the no. ways from there to exit

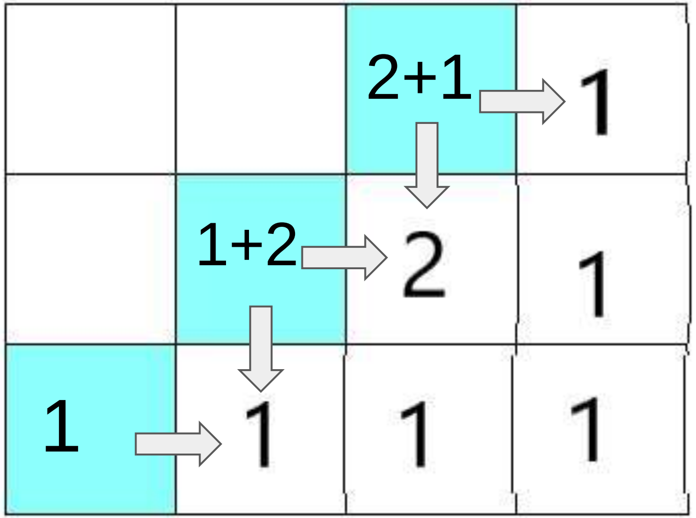
BASE CASE-

If I am in last cell then I have only one move that is exit this cell. So for last cell answer is 1




1
1
1


1
(1+1)
1
1
1
1


2+1
1
1+2
2
1
1
1
1
1

	(6+6)	3	1
(1+3)	3	2	1
1	1	1	1

(4+6)	6	3	1
4	3	2	1
1	1	1	1

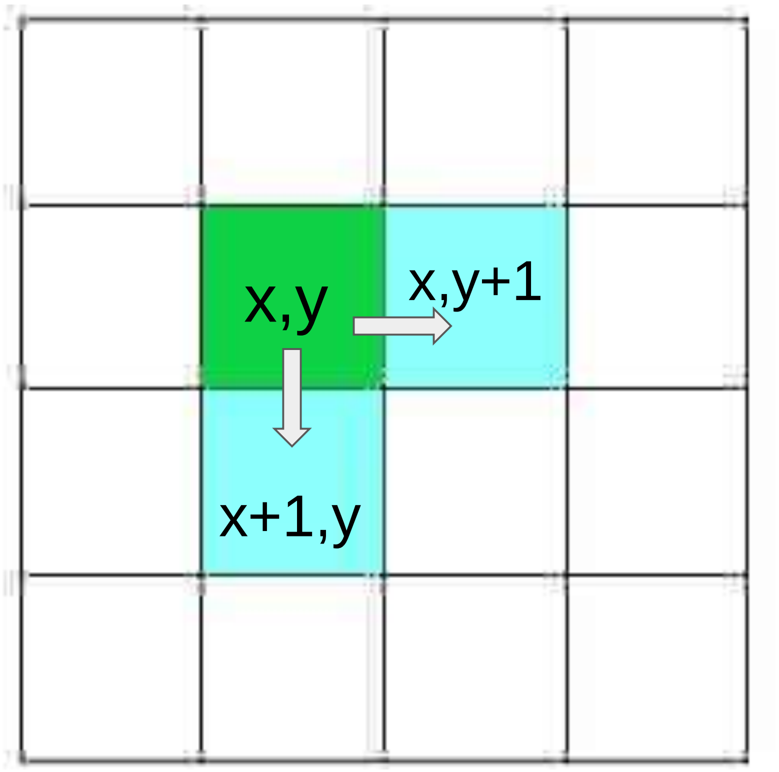
So for a general Case

ans(x,y) = ans(x+1,y) + ans(x,y+1)

Let's say I want to calculate answer for (X,Y) .

Then I will first Calculate for them -

(X+1,Y) and (X,Y+1)



Recurrence relation

(Pseudo Code)

```
// Assume N * M Grid

Calculate_Ways ( X,Y )
{
    // Base Case
    if( X==N && Y== M )
        return 1;


    Ans=0;

    if(X+1<=N)
        Ans += Calculate_Ways(X+1,Y);

    if(Y+1<=M)
        Ans += Calculate_Ways(x,Y+1);

    return Ans;
}
```

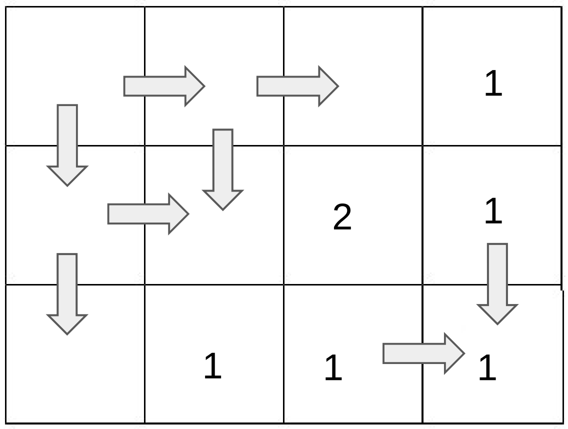
What happens

We are in block (1,1).. So to find answer for it we need to calculate answer for (1,2) and (2,1).

Now similarly (1,2) calls (1,3) and (2,2)... and (2,1) calls (1,2) and (1,3) and this goes on….

Sometime we reach base Case and then we actually have answers.

In actual this happens…



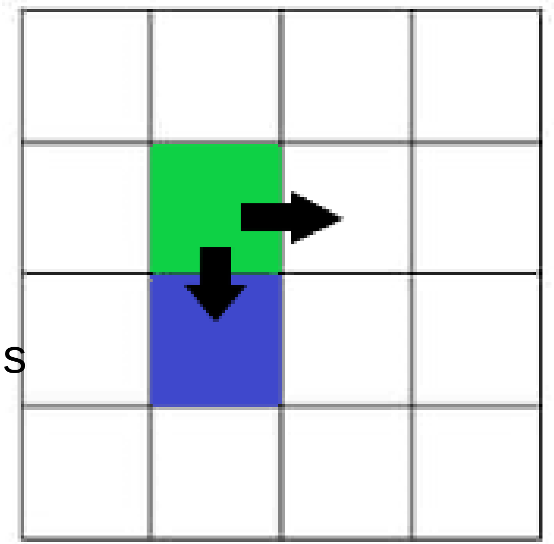
This Goes on….

In some time, we reach end.

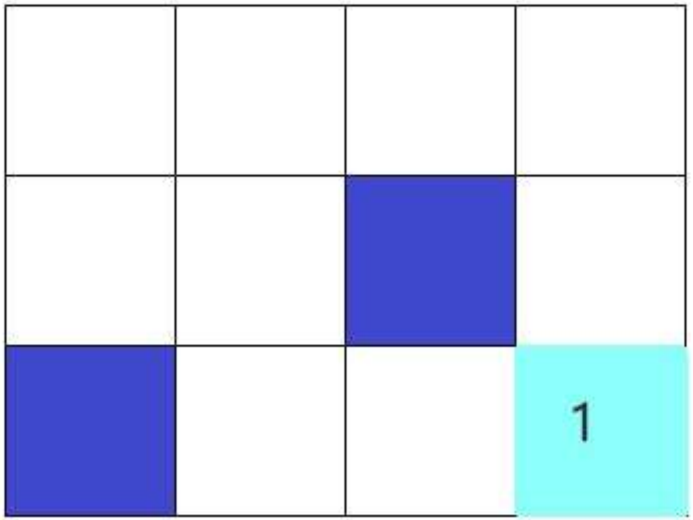
So, now I will soon have answers

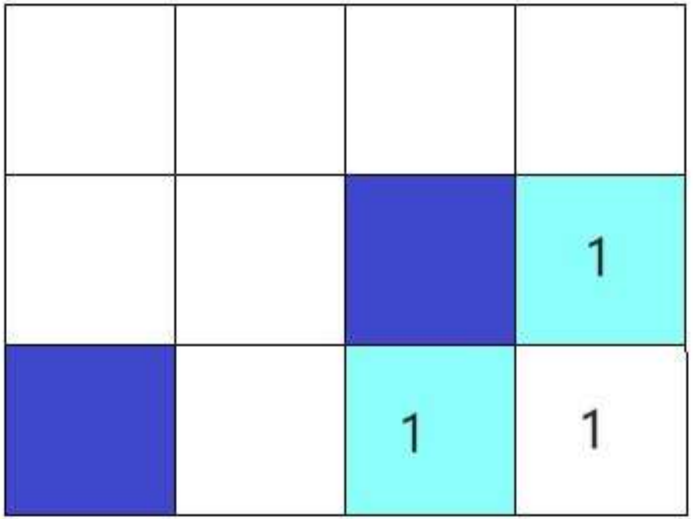
So.. soon I will have answer for every box.

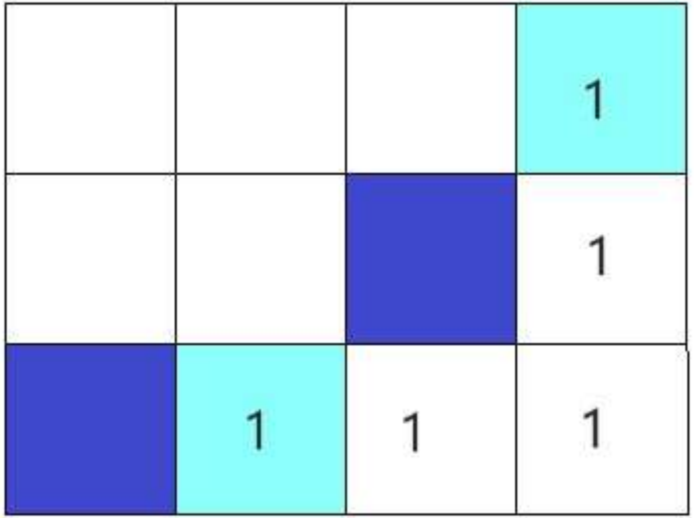
Now Let's move on to block cells.

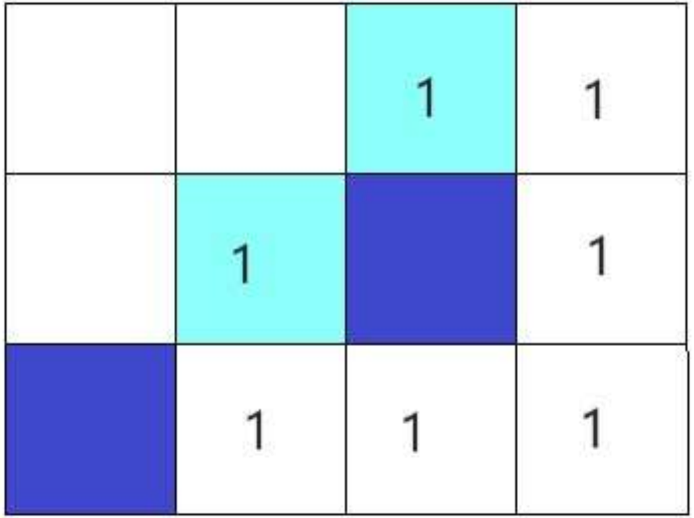


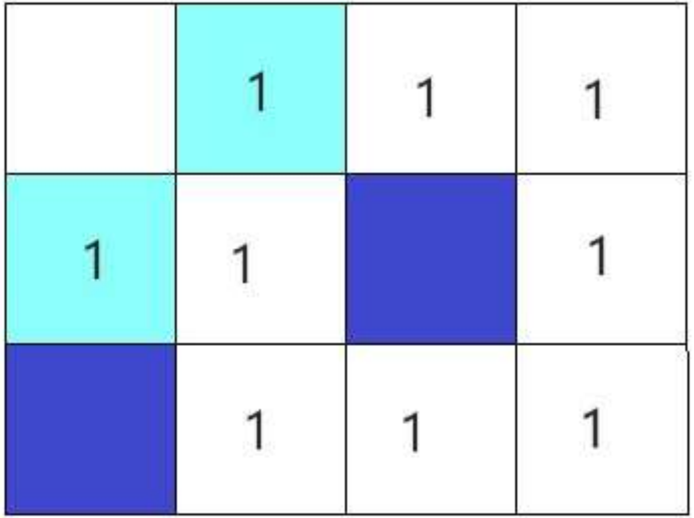
This cell is blocked. So we shouldn’t go here.

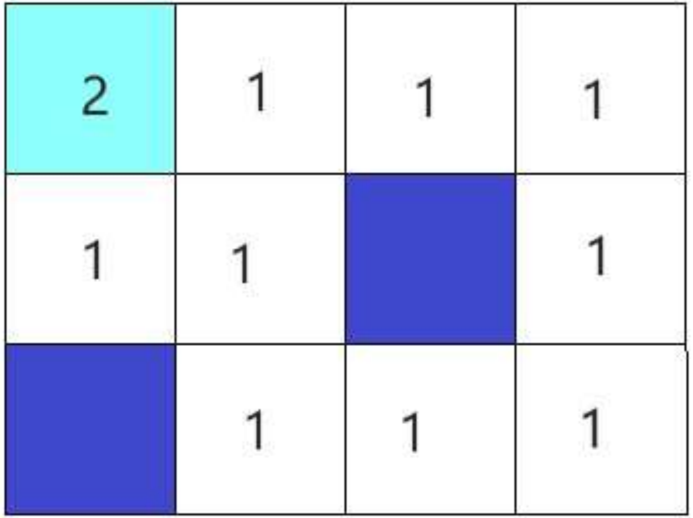

1


1
1
1


1
1
1
1
1


1
1
1
1
1
1
1

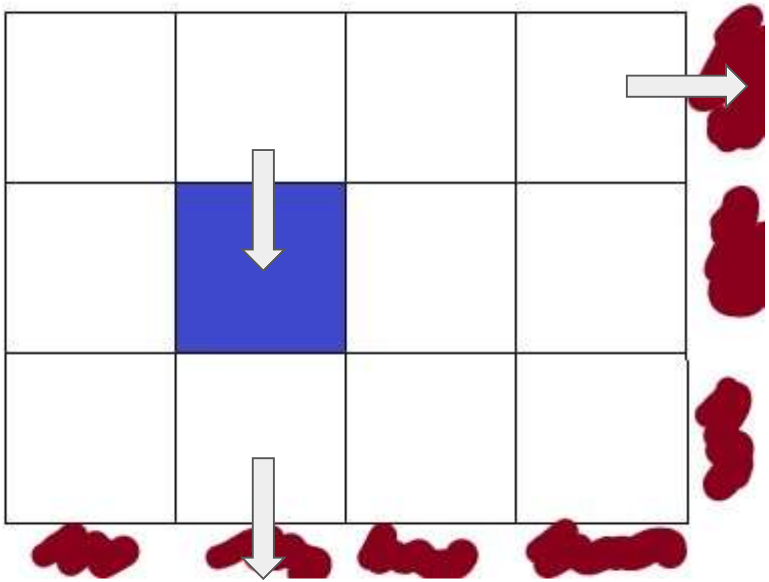

1 1 1
1 1 1
1 1 1


2 1 1 1
1 1 1
1 1 1

Like we are checking for boundary condition,
Similarly check for if the cell is blocked

```
// Assume N * M Grid
// Assume Is_blocked [X][Y] is 1 if cell is blocked otherwise 0.

Calculate_Ways ( X,Y )
{
    // Base Case
    if( X==N && Y== M )
        return 1;

    Ans=0;

    if(X+1<=N && Is_blocked [X+1][y] == 0)
        Ans += Calculate_Ways(X+1,Y);

    if(Y+1<=M && Is_blocked [X][Y+1] == 0)
        Ans += Calculate_Ways(X,Y+1);

    return Ans;
}
```

More better way to write code….



Invalid Blocks have following property

X>N

Y>M

Is_blocked [X] [Y] == 1

I have 0 way to reach to the end from these places.

More better way to write code….

```
// Assume N * M Grid
// Assume Is_blocked [X][Y] is 1 if cell is blocked otherwise 0.

Calculate_Ways ( X,Y )
{
    // check if block is invalid
    if( X > N  ||  Y > M  || Is_blocked[X][Y] ==1 )
        return 0;

    // Base Case
    if( X==N && Y== M )
        return 1;

    // Recursion
    Ans = Calculate_Ways(X+1,Y) + Calculate_Ways(X,Y+1);

    return Ans;
 }
```

Relation to PMI (just for understanding)

Dynamic Programming

Base Case- Like Ans[M][N] =1

Or Our invalid Cases X>N ,Y>M or if the cell is blocked Ans is 0.

Recursion Step - We want answer for (X,Y), so we first find for (X+1,Y) and (X,Y+1)

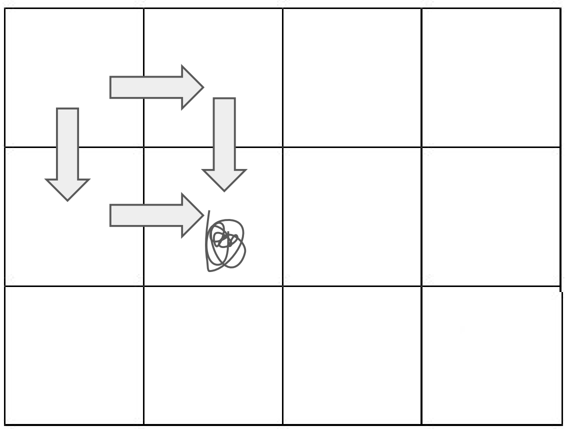
PMI

Base Case- Like we had for k=0, or k=1, we had proof.

Recursion Step - If we want to prove for K+1, then we need to have proof for K.

Memoization…

Problem is we might be reaching the same Box again… like…



So why calculate again and again...
Lets store the answer…

If ans[X][Y] == -1 ….then we have not calculated for it yet otherwise we had.

```
// Assume N * M Grid
// Assume Is_blocked [X][Y] is 1 if cell is blocked otherwise 0.
// Lets Store Answer in DP[X][Y]
// initially all the values in DP[X][Y] is -1

Calculate_Ways ( X,Y )
{
    // check if block is invalid
    if( X > N  ||  Y > M  || Is_blocked[X][Y] ==1 )
        return 0;

    // if Answer is already there
    if( DP[X][Y] != -1 )
        return DP[X][Y];

    // Base Case
    if( X==N && Y== M )
        return 1;

    // Recursion
    Ans = Calculate_Ways(X+1,Y) + Calculate_Ways(X,Y+1);

    return DP[X][Y] = Ans;
 }

```