Today's Content

→ 2D matrices questions

↳ → No. of ways from (0,0) → BR → Blocked cells

TODO → min cost to reach BR

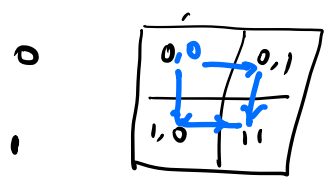
↳ → Dungeons & Dragon.

Q. No of ways to go from (0,0) → BR cell (n-1, m-1)

n, m.

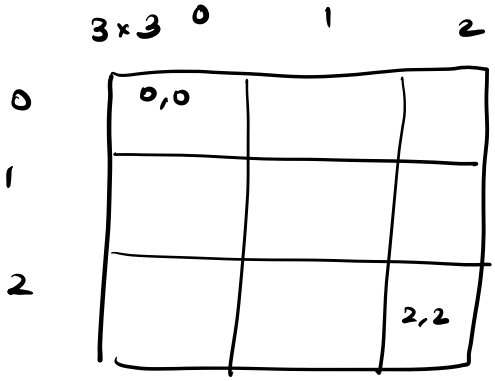
Cell → right, bottom movement is only allowed.

n = 2 m = 2



ans = 2

3×3



ans = 6

Path

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

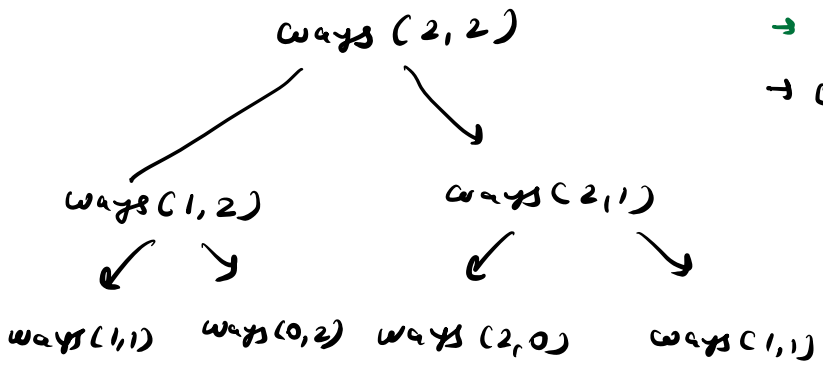
(0,0) (0,1) (1,1) (1,2) (2,2)

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

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

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

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

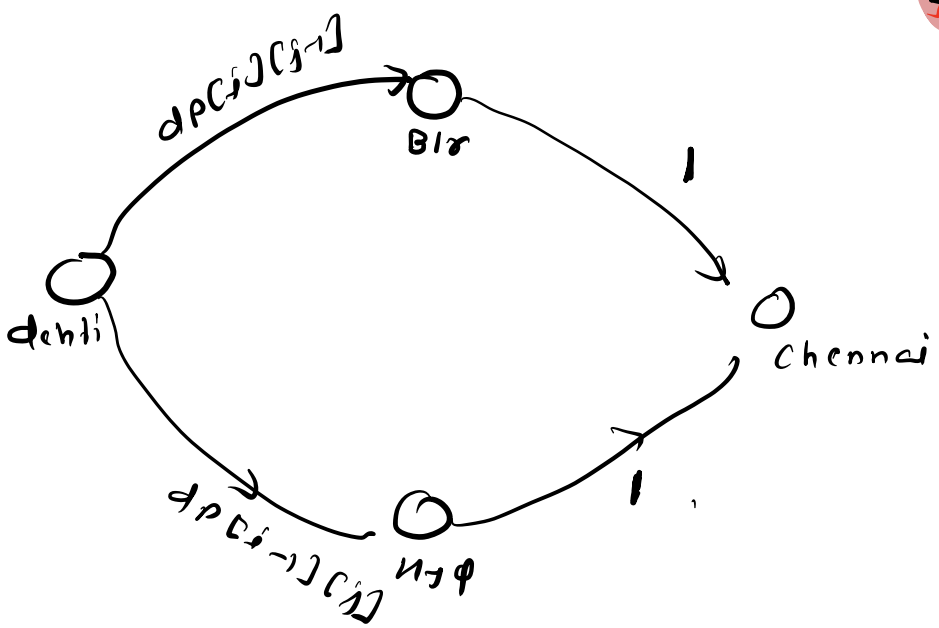


→ optimal substructure.

→ overlapping subproblems

dp[i][j] = # No. of ways to reach (0,0) → (i,j)

dp[i][j] = dp[i][j-1] + dp[i-1][j] + 1 + 2



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

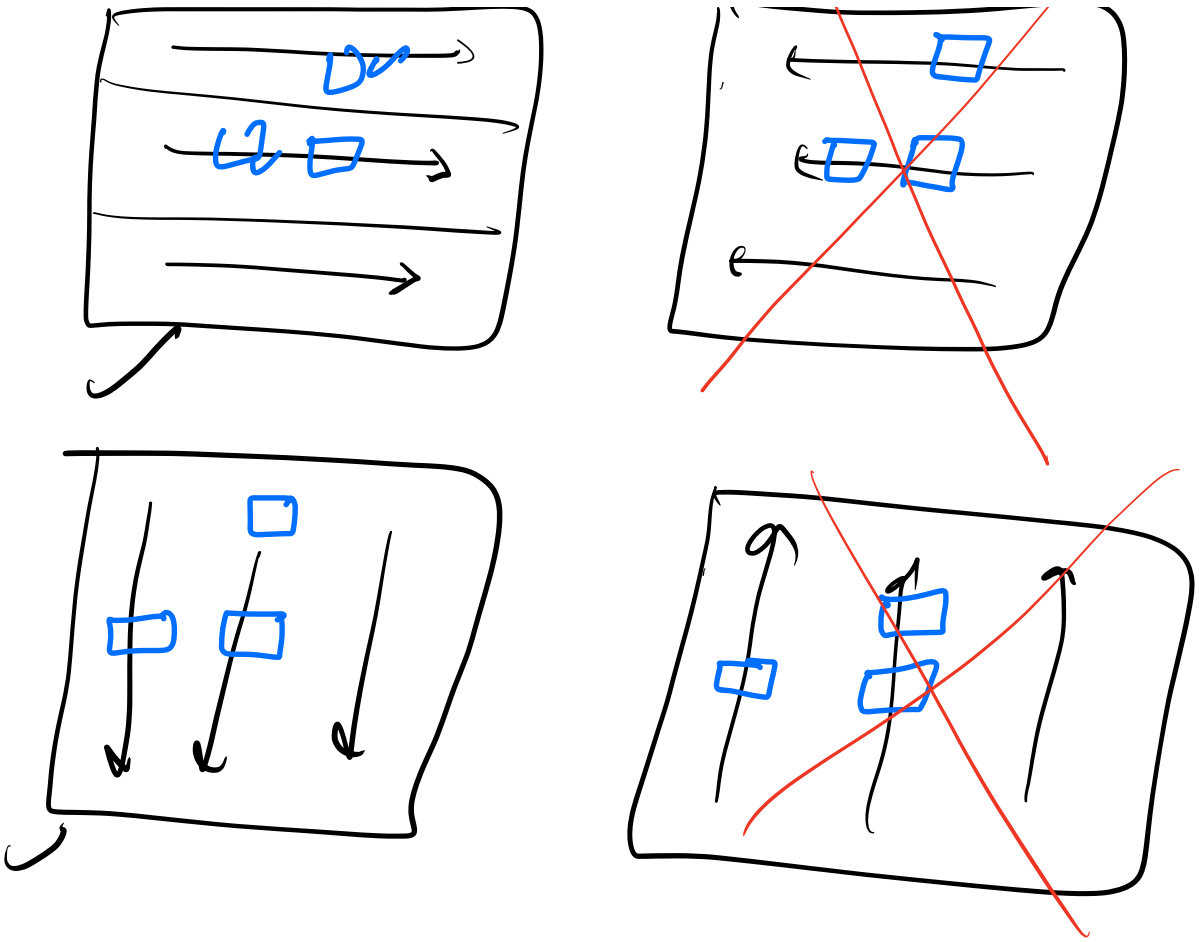
Base condition

↳ if (i==0 || j==0) dp[i][j] = 1

```
int ways(int N, int M)
{
    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]
            }
        }
    }
    return dp[n-1][m-1]
}
```

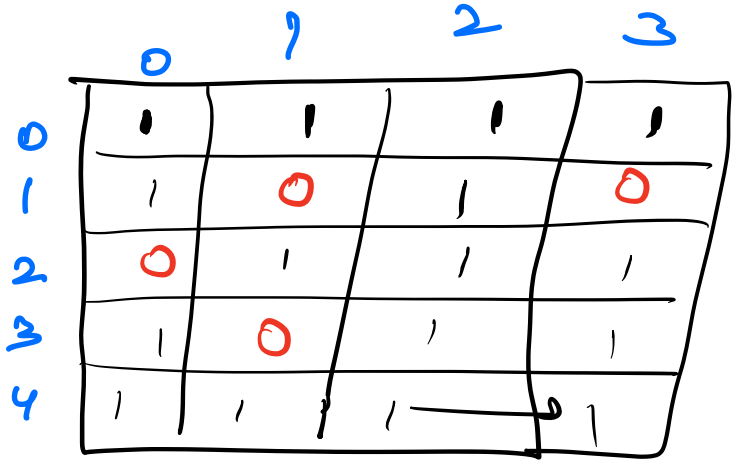
TC : O(N×M)

SC : O(NM)



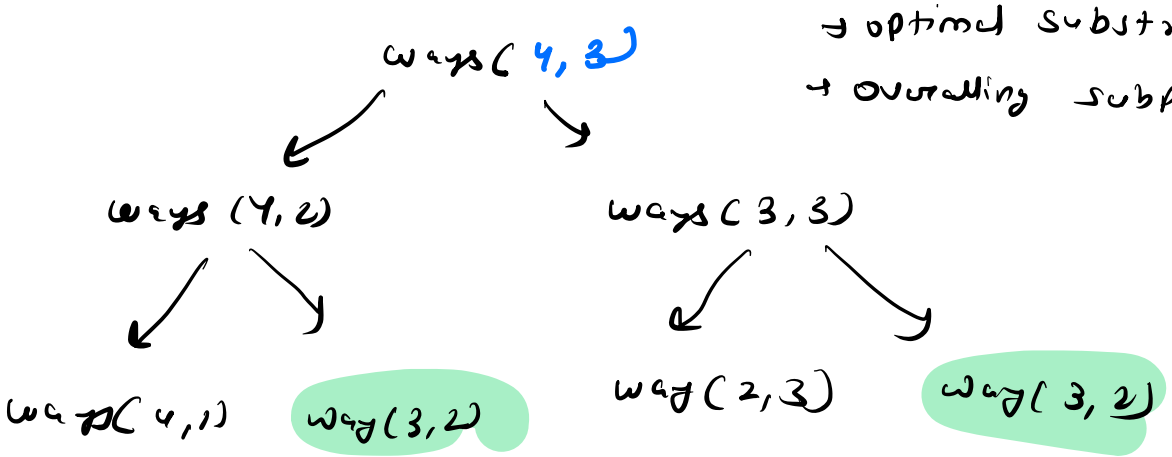
Q. No of ways to go from 0,0 → BR cell

cell → right / bottom.



ans = 3.

→ optimal substructure
→ overlapping subproblems

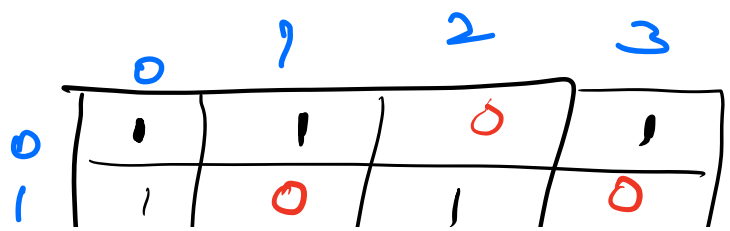


$$dp[i][j] = \begin{cases} \text{if } (mat[i][j] == 0) \quad dp[i][j] = 0 \\ \text{else} \\ \{\ dp[i-1][j] + dp[i][j-1]\ \} \end{cases}$$

Base conditions,

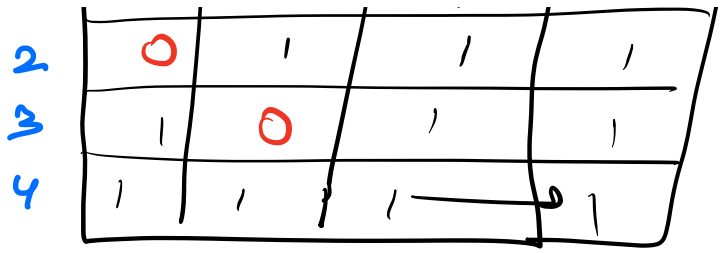
$i == 0 \quad || \quad j == 0$ code fails.

$j = 0$



$dp[0][3] = ?$

$= 0$



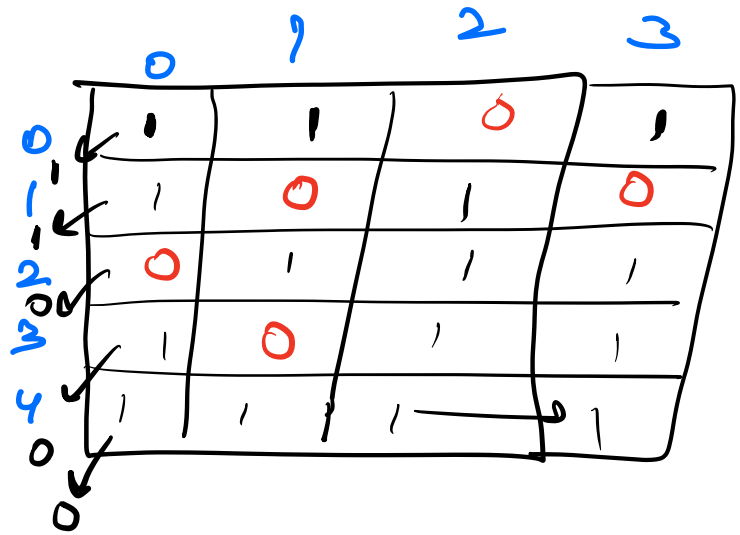
dp[0][2] = 0
dp[0][3] = 1
dp[0][4] = 1

dp[N][M] = {0}.

for(i = 0 ; i < m ; i++)

{ if (mat[0][i] == 1) dp[0][i] = 1
else break

i = 0



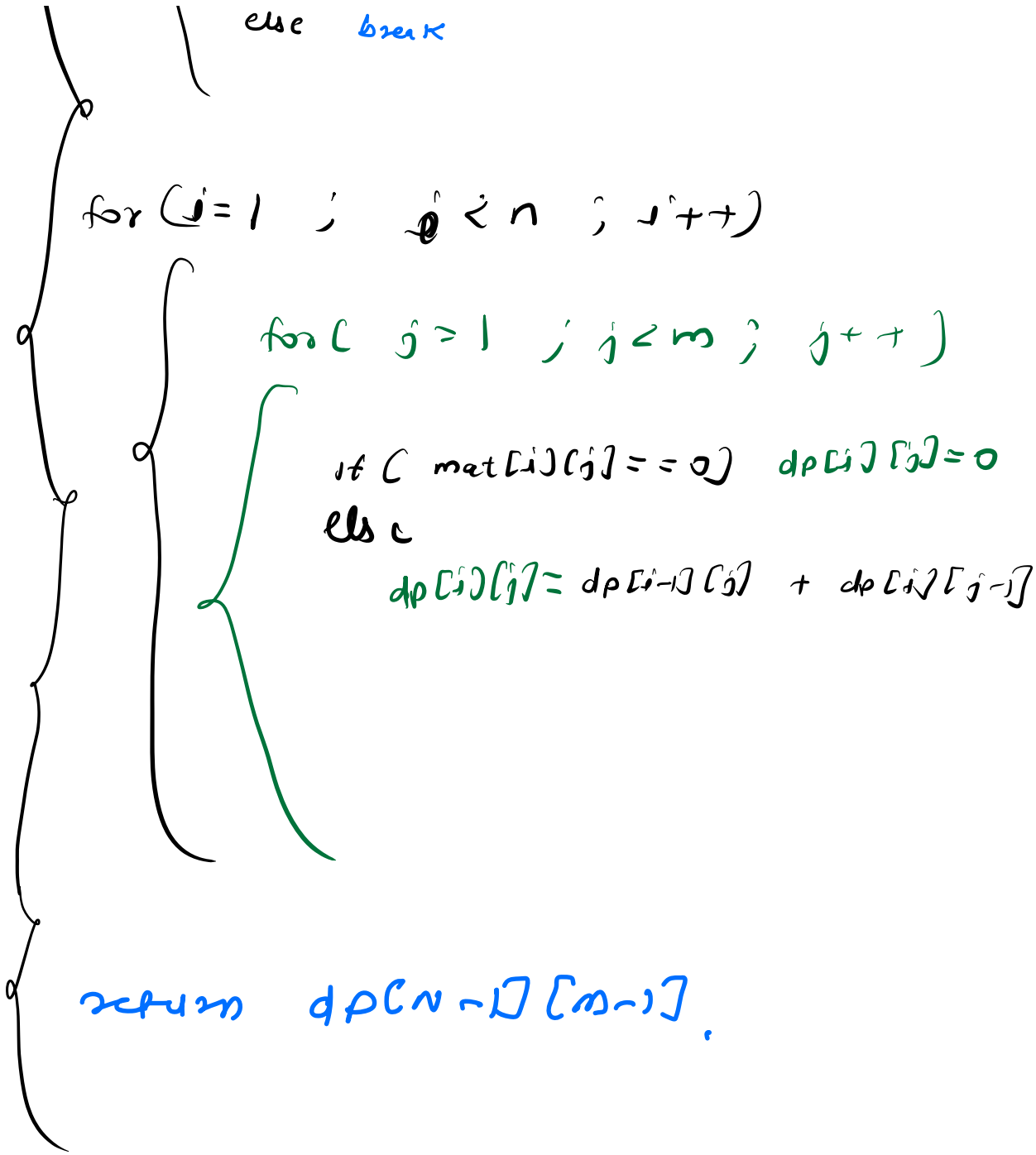
dp[N][M] = x or y.

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

```
int ways ( int mat[][], int N, int m)
{
    int dp[N][M] = {0};

    // row 0 is taken care
    for( i=0 ; i<m ; i++)
    {
        if( mat[0][i] == 1)      dp[0][i] = 1
        else    break
    }

    // col 0 is taken care
    for( i=0 ; i<n ; i++)
    {
        if( mat[i][0] == 1)      dp[i][0] = 1
```



TC : O(N M)

SC : O(N x M)

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

    return func (mat, dp, N-1, M-1)
}

int func (int mat[][], int dp[][], i, j)
{
    If (i<0 || j<0)   return 0
    If (mat[i][j] == 0)   return 0

    If (i+j == 0)   return 1
    same <-  i==0 && j==0

    If (dp[i][j] == -1)
    {
        dp[i][j] = fun (mat, dp, i-1, j)
                   +
                   fun (mat, dp, i, j-1)
    }

    return dp[i][j]
}
```

In recursive code, writing base case

is generally relatively easier.

TC : $O(N \times M)$

SC : $O(NM)$.

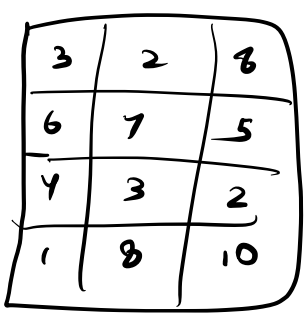
Break

10 : 23 — 10 : 32

TODO

Q. Mincost to reach 0,0 to BR cell

cell → right / bottom.



3	2	8
6	7	5
4	3	2
1	8	10

Q. Given mat[N][M], where mat[i][j] indicate health gain at this cell.

find min health you should start with at (0,0) so that you can reach N-1, M-1

→ cell → bottom / right

→ If your health reach zero, game over

→ (0,0) → n-1, m-1

~~H=4~~ ~~H=5~~ H = 6

-3	-5
-2	1

10

-3	2		
-6			
		1	

minH (0,2)

min (1,0)

Expression.

Case I

x

-5

x - 5 = 1

x = 6

Case II

x

-2

-5 6

x - 2 = 6

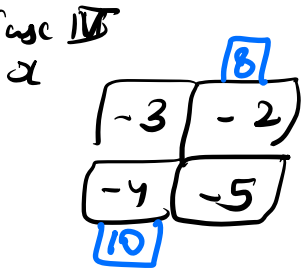
x = 8

Case III

$x - y = 6$

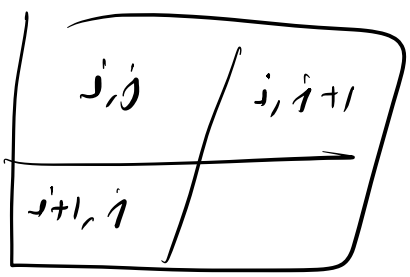
$x = 10$

Case IV



$x - 3 = \min(8, 10)$

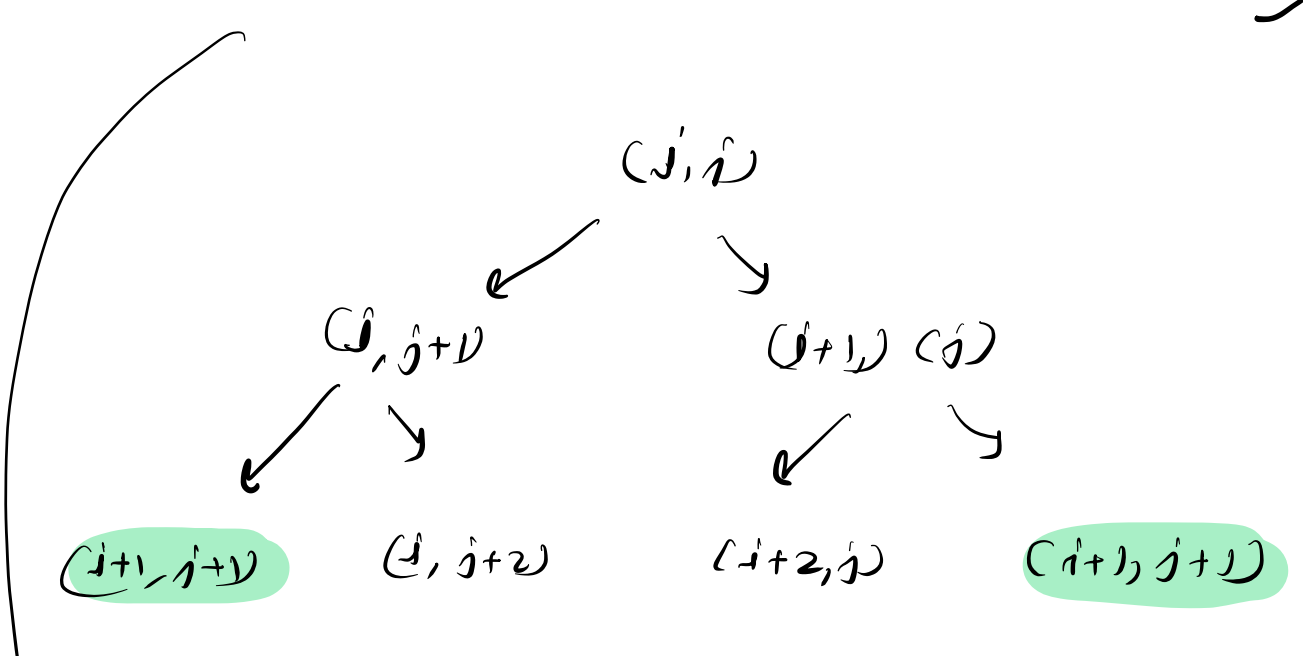
$x = 11$



→ optimal substructure.

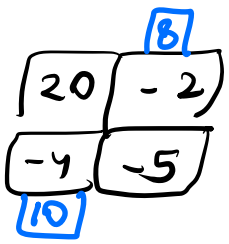
→ overlapping subproblems.

$$x + mat[i][j] = \min(dp[i+1][j], dp[i][j+1])$$



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

$Dp[i][j]$ = min health you will need start (i, j) and reach $(N-1, M-1)$



$x + 20 = 8$

$x = -12$

$dp[i][j] = max(1, min(dp[i+1][j], dp[i][j+1]) - mat[i][j])$

Base cases

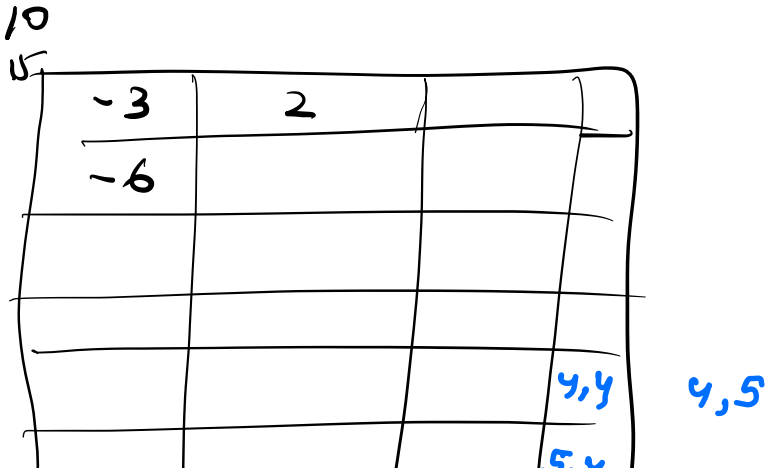
$i == N-1 \;||\; j == M-1 \Rightarrow$ code fail.

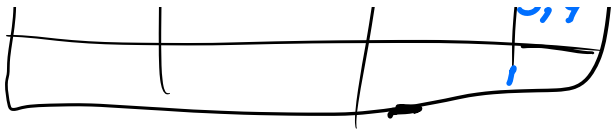
```
int minH(int mat[][], N, M)
{
    int dp[N][M] = {-1}
    dp[N-1][M-1] = max(1, 1 - mat[N-1][M-1])
```

return fun(mat, N, M, dp, 0, 0)

```
int fun( mat, N, M, dp, i, j)
{
    if( i >= N || j >= M) return INT_MAX

    if( dp[i][j] == -1)
    {
        int x = fun( mat, N, M, dp, i+1, j)
        int y = fun( mat, N, M, dp, i, j+1)
        dp[i][j] = max( 1, min(x, y) - mat[i][j])
    }
    return dp[i][j]
}
```





Passing by value

Passing by reference

↓

sending array will not create copies.