Data Backup Scheduler

Problem Statement:
You're the network administrator at a large company where daily data backups are crucial. Each backup affects the network differently, measured by a "load score". A higher load score could indicate more crucial data being backed up, even though it strains the network more.

The strategy aims to maximize important data being backed up(important data means the one having higher load score) over several days while avoiding backups on consecutive days to prevent overwhelming the network. This balances crucial data backup with maintaining network performance.

Here's an example of how you might approach this over any number of days, say N=10 days, with the following load scores:

Day 1: `Load score = 45`
Day 2: `Load score = 55`
Day 3: `Load score = 40`
Day 4: `Load score = 65`
Day 5: `Load score = 60`
Day 6: `Load score = 35`
Day 7: `Load score = 75`
Day 8: `Load score = 50`
Day 9: `Load score = 80`
Day 10: `Load score = 70`

Select days - `1, 3, 5, 7, 9` to get max score of `300`.

Q → Find the maximum subsequence sum where selecting adjacent elements is not allowed.

$$A = \begin{array}{ccc} 0 & 1 & 2 \\ \end{array}$$
$$A = [\underset{\checkmark}{5} \quad 8 \quad \underset{\checkmark}{4}] \qquad Ans = \underline{9}$$

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}$$
$$A = [10 \quad \underset{\checkmark}{20} \quad 30 \quad \underset{\checkmark}{40}] \qquad Ans = \underline{60}$$

Bruteforce → Consider all subsets / subsequence.

$TC = \underline{O(2^N)}$

$$\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array}$$
$$A = [5 \quad \underset{\checkmark}{18} \quad 10 \quad 2 \quad \underset{\checkmark}{15}] \qquad Ans = \underline{33}$$

(index) → 0 / N-1 maxSum = S

(N-1) S = 0

✓ (N-3) x (N-2)

S = A[N-1] S = 0

(N-5)   (N-4)   (N-4)   (N-3)

sum[i] $\xrightarrow{\times}$ sum[i-1]
sum[i] $\xrightarrow{\checkmark}$ sum[i-2] + A[i]
} max

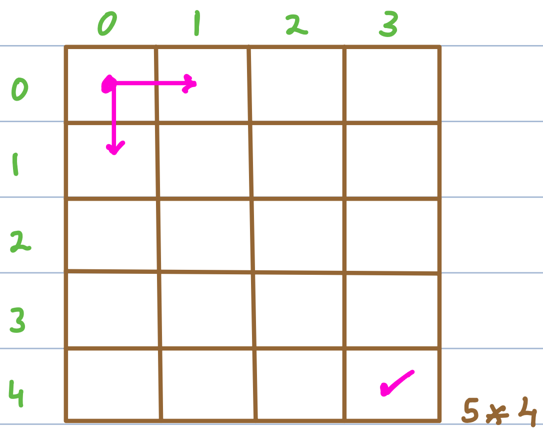
```
sum[0] = max(0, A[0])
sum[1] = max(sum[0], A[1])
for i → 2 to (N-1) {
|    sum[i] = max(sum[i-1], sum[i-2] + A[i])
}
return sum[N-1]
```

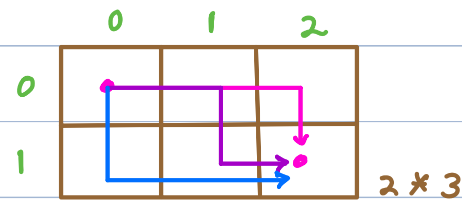
TC = O(N)   SC = O(N) → O(1) ✓

Q → Given a 2D matrix.

1 step → move right or down.

Find # ways to move from (0,0) to (N-1, M-1).





R R D

R D R

D R R   Ans = 3

→ (i, j) →

$$\begin{array}{ccc} & (i-1, j)_y & \\ & \downarrow & \\ (i, j-1)_x & \longrightarrow & (i, j) \end{array}$$

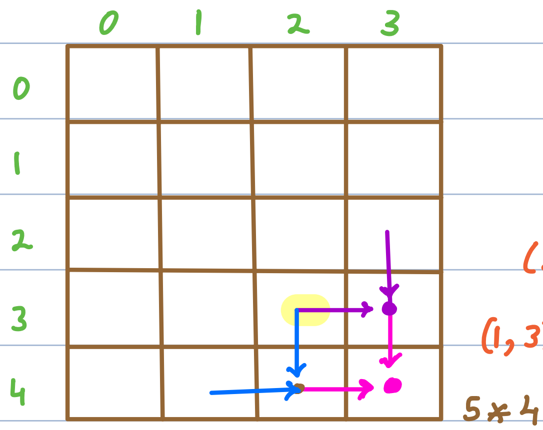
ways = x + y

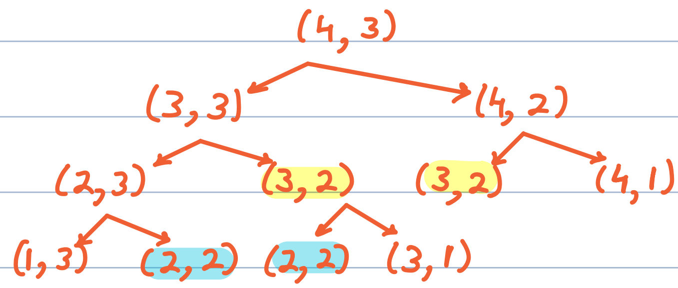
ways(i,j) = #ways to reach i,j from 0,0.

ways (0,0) = 1

N-1 M-1

```
int ways (i, j) {
    if (i==0 && j==0) return 1
    if ( i<0 || j<0 ) return 0
    return ways(i-1, j) + ways (i, j-1)
}
```





∀i, j → W[i][j] = -1

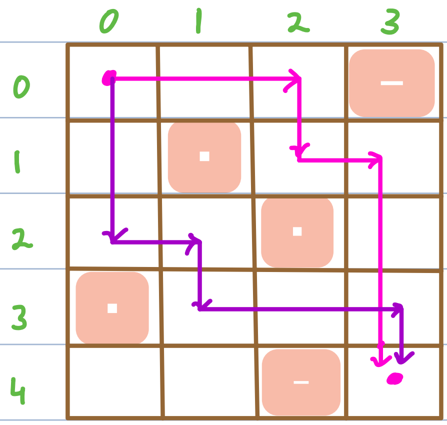
```
int ways (i, j) {
    if (i==0 && j==0) return 1
    if ( i<0 || j<0 ) return 0
    if ( W[i][j] != -1)   return W[i][j]
    W[i][j] = ways (i-1, j) + ways (i, j-1)
    return W[i][j]
}
```

prev. row   cur. row

TC = O(N*M)   SC = O(N*M)

H.W → Solve in linear SC.

Solve the above with blocked cells.



Ans = 2

5*4

$\forall i, j \rightarrow W[i][j] = -1$

```
int ways (i, j) {                                   → end
    if (i==0 && j==0) return 1                      → start
    if ( i<0 || j<0 ) return 0
    if ( A[i][j] == 0 ) return 0   // blocked cell
    if (W[i][j] != -1)  return W[i][j]
    W[i][j] = ways (i-1, j) + ways (i, j-1)
    return W[i][j]
}
```

Q → <u>Dungeons & Princess</u>

start ↘

-3	2	4	-5
-6	5	-4	6
-15	-7	5	-2
2	10	-3	-4

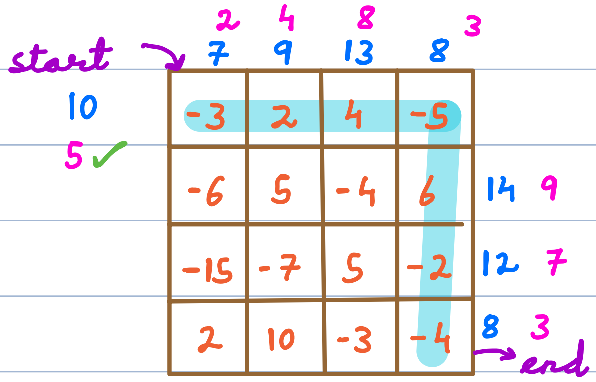
→ end

A[i][j] → +ve, health increas by A[i][j]
→ 0, no change
→ -ve, health decrease by |A[i][j]|

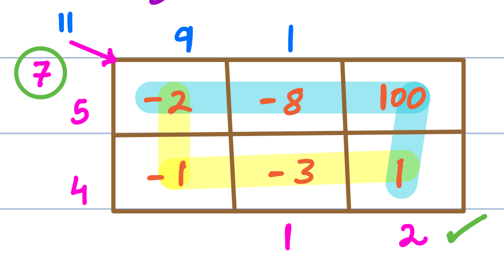
Move → right, ↓ down

Find min initial health to start s.t we can reach the last cell alive.

If at any point health <= 0 ⇒ dead.



X Greedy → Path with max sum. X



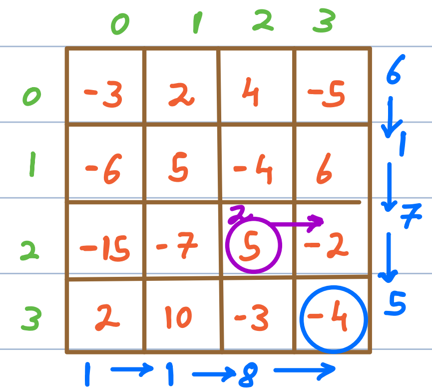
Sum = 91

health (i,j) → min health to travel from (0,0) to (i,j)
→ min health to travel from (i,j) to (N-1, M-1) ✓

Ans = health (0,0)

Base Case → health (N-1, M-1)

max (1, 1 - A[N-1][M-1])



health[i][j] + A[i][j] = min (health[i+1][j],
health[i][j+1])

max (1, ↓)

health[i][j] = (min (health[i+1][j], health[i][j+1]) - A[i][j])

7 ✓ 8 - 5 = 2

7 - 10 = -3

```
for i → (N-1) to 0 {
    for j → (M-1) to 0 {
        if ( i == N-1 && j == M-1)
            h[i][j] = max (1, 1 - A[N-1][M-1])
        else if ( i == N-1)
            h[i][j] = max (1, h[i][j+1] - A[i][j])
```

```
            else if (j == M-1)
                h[i][j] = max(1, h[i+1][j] - A[i][j])
            else
                h[i][j] = max(1, min(h[i+1][j], h[i][j-1]) - A[i][j])
        }
}
return h[0][0]
```

TC = O(N * M) SC = O(N * M) → O(2 * M)

store only 2 rows.