Problem statement

> Given a ‘N’ * ’M’ maze with obstacles, count and return the number of paths to reach the right-bottom cell from the top-left cell. A cell in the given maze has a value -1 if it is a blockage or dead-end, else 0. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only. Since the answer can be large, print it modulo 10^9 + 7.

Example :

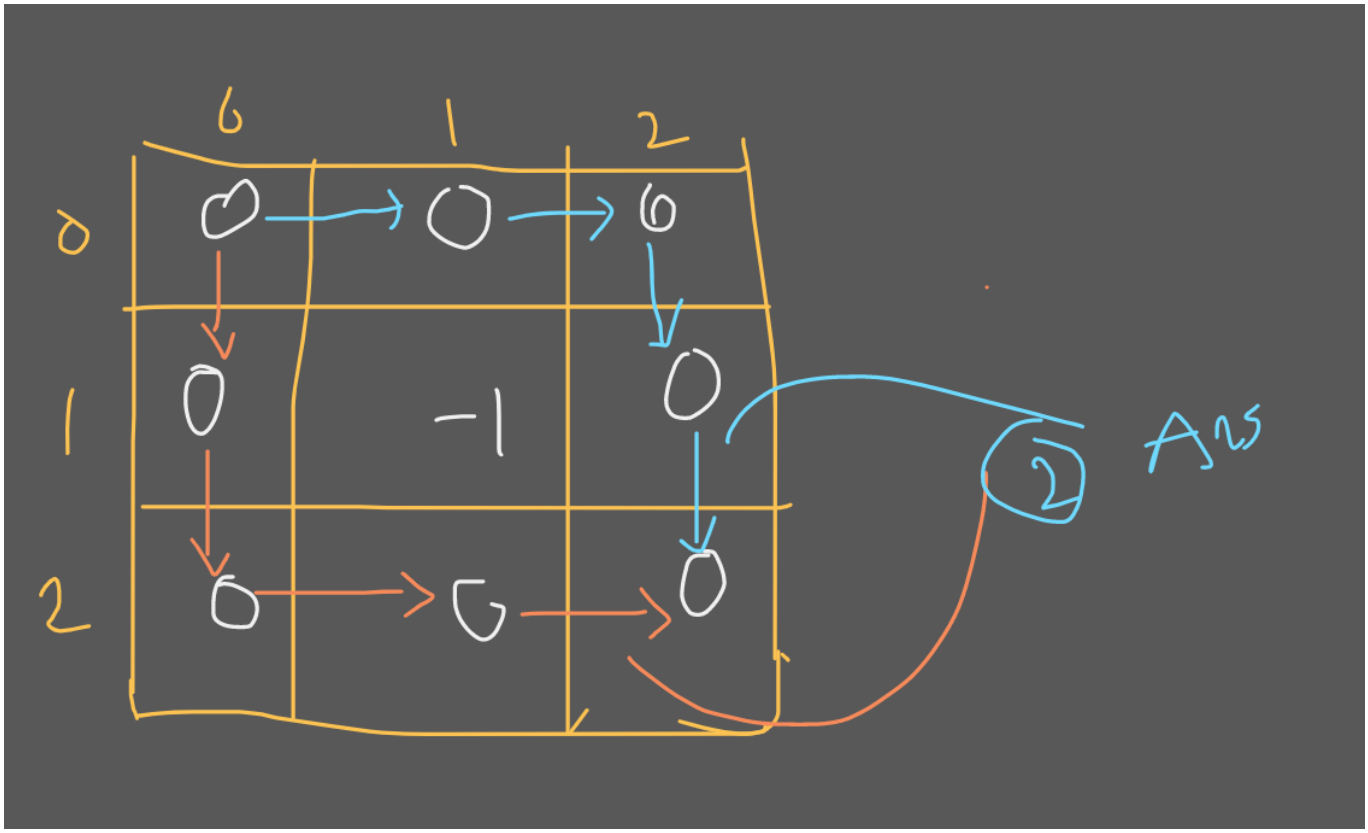
```
Consider the maze below :
0 0 0
0 -1 0
0 0 0

There are two ways to reach the bottom left corner -

(1, 1) -> (1, 2) -> (1, 3) -> (2, 3) -> (3, 3)
(1, 1) -> (2, 1) -> (3, 1) -> (3, 2) -> (3, 3)

Hence the answer for the above test case is 2.
```

⟶ Diagram:



Approach:

⟶ we know that we can find unique path as recursive call with the code for uniquePath question. Now we just have to add one more condition for obstacle.

The condition will be like this:

```
if(i >=0 && j>=0 && arr[i][j] == -1){
        return 0;
}
```

⟶ Also other base cases will be like this:

```
if(i == 0 && j == 0){
        return 1;
}
if(i < 0 && j < 0){
```

```
        return 0;
}
```

Tabulation Code:

```
int solve(int n,int m){
        for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                        // Condition for obstacles
                        if(arr[i][j] == -1){
                                dp[i][j] = 0;
                        }
                        // First base case
                        else if(i==0 && j == 0){
                                dp[i][j] = 1;
                        }else{
                                int up=0,left=0;
                                // second base case to prevent
from going outside of row or column
                                if(i > 0){
                                        up = dp[i-1][j];
                                }
                                if(j > 0){
                                        left = dp[i][j-1];
                                }
                                dp[i][j] = up+left;
                        }
                }
        }
        return dp[n-1][m-1];
}
```