Dynamic Programming

Video - 82

Note :- This playlist is only for explanation of dps & solutions.

See my "DP Concepts & dps" playlist for understanding DP from scratch...

Facebook, Instagram → codestorywithMIK

Twitter → cswithMIK

→ codestorywithMIK

Leetcode - 576 Medium

Recursion + Memo → Bottom up

Better Bottom up

Company :- Baidu , Amazon

576. Out of Boundary Paths

Medium 3246 246 Add to List Share

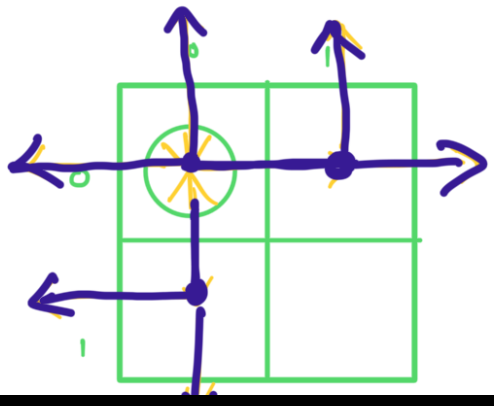
There is an `m x n` grid with a ball. The ball is initially at the position `[startRow, startColumn]`. You are allowed to move the ball to one of the four adjacent cells in the grid (possibly out of the grid crossing the grid boundary). You can apply **at most** `maxMove` moves to the ball.

Given the five integers `m`, `n`, `maxMove`, `startRow`, `startColumn`, return the number of paths to move the ball out of the grid boundary. Since the answer can be very large, return it **modulo** $10^9 + 7$.

Example :- m = 2 , n = 2 , maxMove = 2 ,

startRow = 0, startCol = 0

output = 6

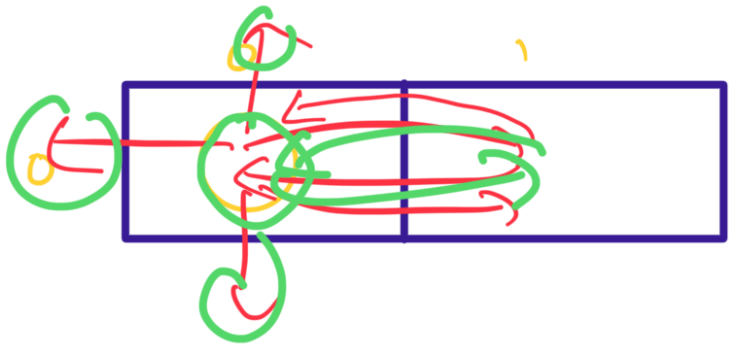


1+1+1
+1+1+1

Intuition Building

(How you should talk out loud during interviews)

+1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1 +1



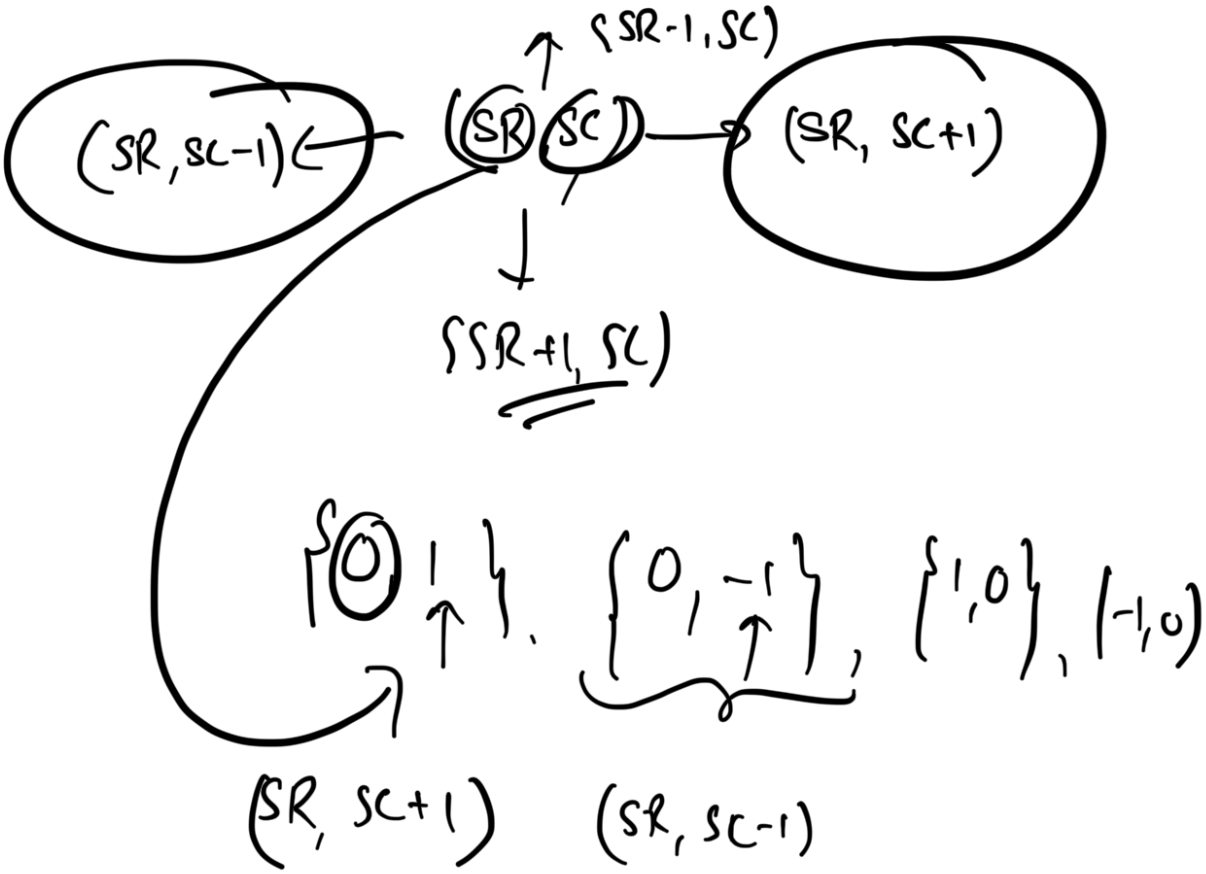
m = 1
n = 2
maxmoves = 5
(0,0)

moves = 4

solve(SR, SC, moves) {

if(SR<0 || SR>=m || SC<0 || SR>2n)
return 1;

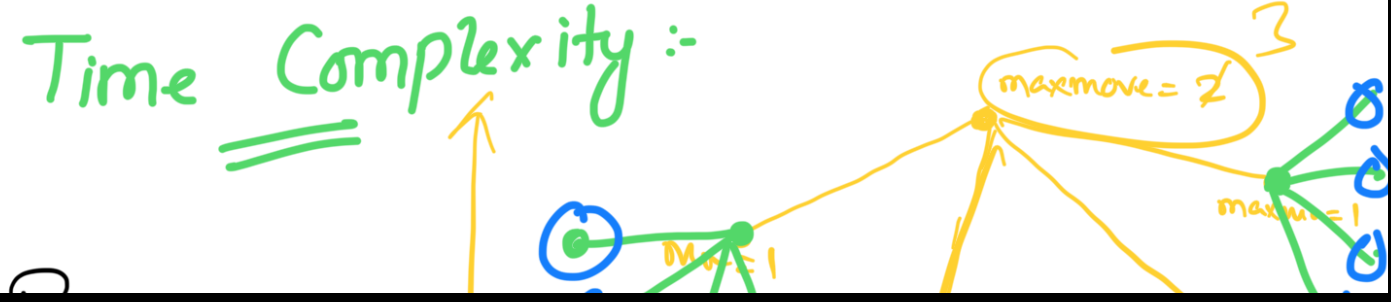
if(moves<=0)
return 0;



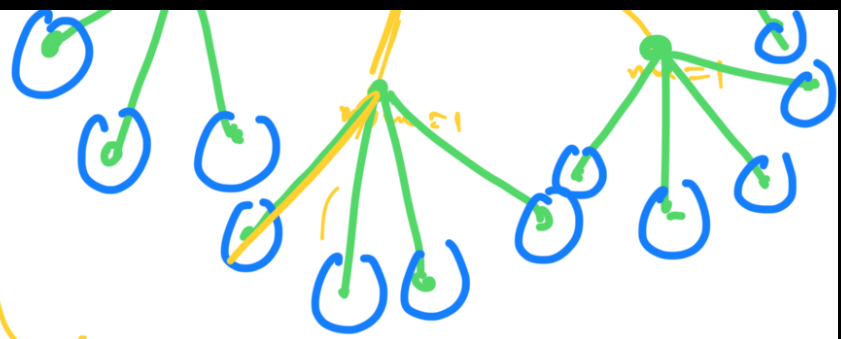

```
a = Solve(SR, SC+1);
b = Solve(SR, SC-1);
c = Solve(SR-1, SC);
d = Solve(SR+1, SC);
```

$\sim (a+b+c+d)$.

Time Complexity :-



Recursion ⇒



4^2 | 4^3 → $O(4^{maxMoves})$.

Space Complexity = O(height in recursion tree)

= O(maxMoves)

Recu + Memo:-

T.C = O(m * n * maxMoves)

S.C = O(m * n)

Bottom UP...

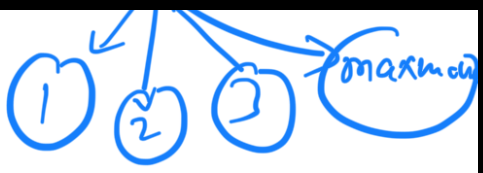
(like always)

→ Derive from Recursion code

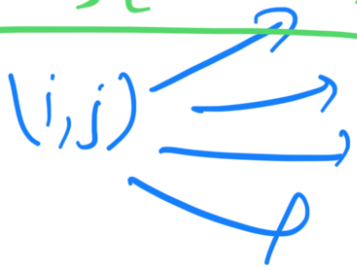
```
if(startRow < 0 || startRow >= M || startColumn < 0 || startColumn >= N) {
    return 1; //Found one path out of grid
}

if(maxMove <= 0) {
    return 0;
}

if(t[startRow][startColumn][maxMove] != -1) {
    return t[startRow][startColumn][maxMove];
}

int result = 0;
for(vector<int>& dir : directions) {
    int new_row = startRow    + dir[0];
    int new_col = startColumn + dir[1];

    result = (result + solve(new_row, new_col, maxMove-1)) % MOD;
}
```

dp[i][j][k] = no of possible paths out of grid, when you are standing at [i][j] & you have K moves

at max.



dp[startRow][startCol][maxMoves];



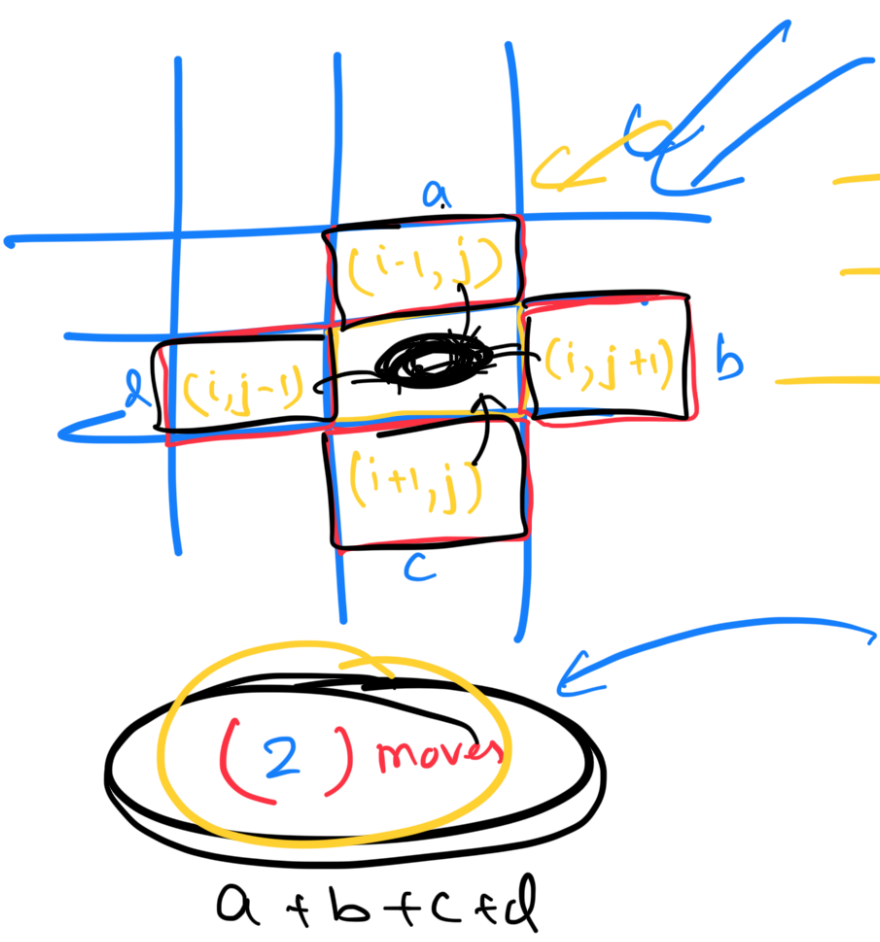

```
for(K = 1; K <= maxMoves; K++) {
    for( i = 0 ; i<m; i++) {
        for( j=0; j<n; j++) {
            for( dir : directions) {
                x = i + dir[0];
                y = j + dir[1];
                if(x<0 || x>=m || y<0 || y>=n){
                    dp[i][j][K] += 1 ;
                } else {
                    dp[i][j][K] += dp[x][y][K-1];
                }
            }
        }
    }
}
```

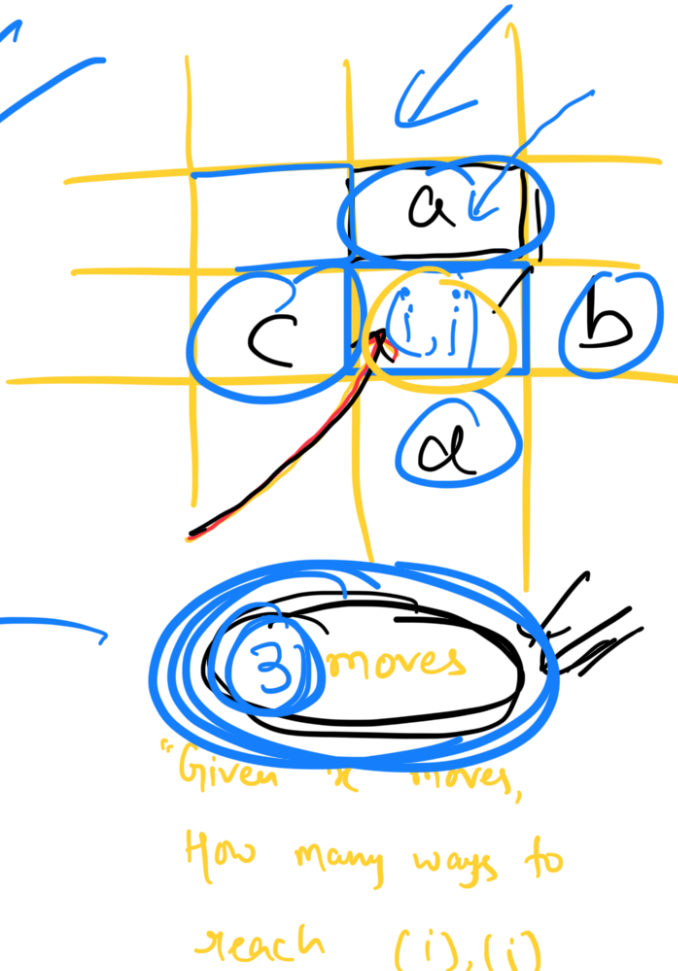
T.C = O(m*n*maxMoves)

S.C = O(m*n*maxMoves).

return dp[startR][startCol][maxMove];

Optimised Bottom UP :-

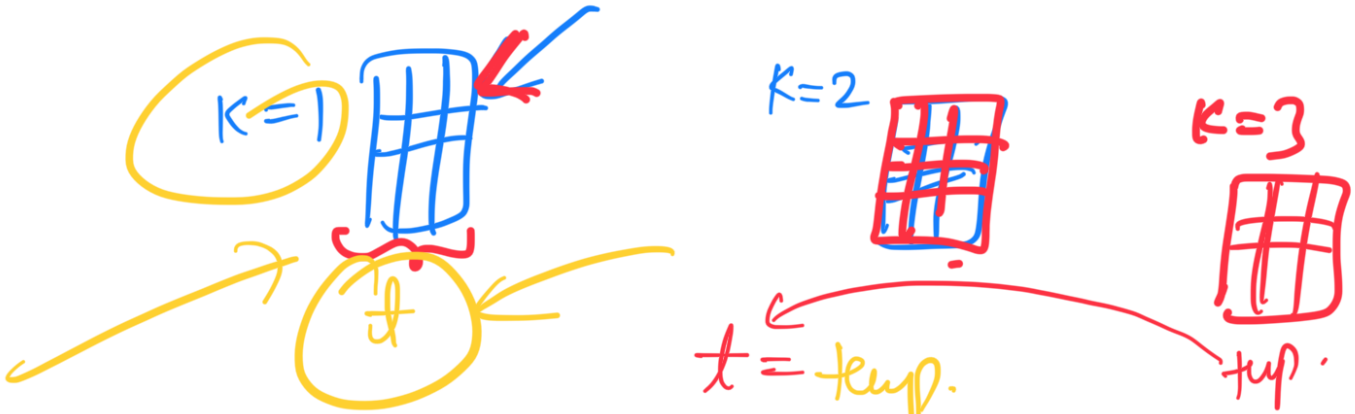




dp[i][j] = no of ways of reaching [i][j] given some no of moves (K=1, K=2, K=3)

K moves

$$dp[i][j]_{(K)} = \underbrace{(i-1,j)}_{(K-1)} + \underbrace{(i+1,j)}_{(K-1)} + \underbrace{(i,j-1)}_{(K-1)} + \underbrace{(i,j+1)}_{(K-1)}$$



t[50][50]; (max).

for (K=1; K<= maxMove; K++) {

```
temp[50][50];                    main
for(i=0; i<m; i++) {

    for(j=0; j<n; j++) {

        for( dir : directions) {
            x = i + dir[0];
            y = j + dir[1];
            if(x || y out of bound) {
                result = (result + dp[i][j]);
            } else {
                temp[x][y] = (temp[x][y] +
                              f[i][j])
            }
        }

    }

    f = temp;

}

}

Return: result;
```