1. def findPaths(m, n, N, i, j):

MOD = 10\*\*9 + 7

dp = [[[0] \* n for \_ in range(m)] for \_ in range(N+1)]

dp[0][i][j] = 1

result = 0

for step in range(1, N+1):

for x in range(m):

for y in range(n):

if dp[step-1][x][y] > 0:

for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

nx, ny = x + dx, y + dy

if 0 <= nx < m and 0 <= ny < n:

dp[step][nx][ny] = (dp[step][nx][ny] + dp[step-1][x][y]) % MOD

else:

result = (result + dp[step-1][x][y]) % MOD

return result

print(findPaths(2, 2, 2, 0, 0))

print(findPaths(1, 3, 3, 0, 1))

**Output:** 6 and 12.

**2**. def rob(nums):

if not nums:

return 0

if len(nums) == 1:

return nums[0]

# Standard House Robber function

def house\_robber(arr):

n = len(arr)

if n == 0:

return 0

if n == 1:

return arr[0]

dp = [0] \* n

dp[0] = arr[0]

dp[1] = max(arr[0], arr[1])

for i in range(2, n):

dp[i] = max(dp[i-1], dp[i-2] + arr[i])

return dp[-1]

case1 = house\_robber(nums[:-1])

case2 = house\_robber(nums[1:])

return max(case1, case2)

nums = [2, 3, 2]

print("The maximum money you can rob without alerting the police is", rob(nums))

**Output:** The maximum money you can rob without alerting the police is 3.

2. def climbStairs(n):

if n == 0:

return 1

if n == 1:

return 1

prev1 = 1 # ways(n-1)

prev2 = 1 # ways(n-2)

for i in range(2, n + 1):

current = prev1 + prev2

prev2 = prev1

prev1 = current

return prev1

n = 4

print("The number of distinct ways to climb", n, "steps is", climbStairs(n))

**Output:** The number of distinct ways to climb 4 steps is 5.

4. from math import factorial

def unique\_paths(m, n):

return factorial(m + n - 2) // (factorial(m - 1) \* factorial(n - 1))

print(unique\_paths(7, 3))

**Output:** 28.

5. def large\_group\_positions(s):

res = []

i = 0 # Start of a group

for j in range(len(s)):

if j == len(s) - 1 or s[j] != s[j + 1]:

if j - i + 1 >= 3:

res.append([i, j])

i = j + 1

return res

print(large\_group\_positions("abbxxxxzzy"))

**Output:** [[3, 6]]

6. def game\_of\_life(board):

m, n = len(board), len(board[0])

directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]

for i in range(m):

for j in range(n):

live\_neighbors = 0

for dx, dy in directions:

ni, nj = i + dx, j + dy

if 0 <= ni < m and 0 <= nj < n and abs(board[ni][nj]) == 1:

live\_neighbors += 1

if board[i][j] == 1 and (live\_neighbors < 2 or live\_neighbors > 3):

board[i][j] = -1

if board[i][j] == 0 and live\_neighbors == 3:

board[i][j] = 2

for i in range(m):

for j in range(n):

if board[i][j] == -1:

board[i][j] = 0 # Dead

elif board[i][j] == 2:

board[i][j] = 1 # Alive

board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]

game\_of\_life(board)

print(board)

**Output:** [[0, 0, 0], [1, 0, 1], [0, 1, 1], [0, 1, 0]]

7. def champagneTower(poured: int, query\_row: int, query\_glass: int) -> float:

dp = [[0] \* (i + 1) for i in range(query\_row + 1)]

dp[0][0] = poured

for i in range(query\_row):

for j in range(len(dp[i])):

overflow = (dp[i][j] - 1) / 2

if overflow > 0:

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

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

return min(1, dp[query\_row][query\_glass])

print(champagneTower(1, 1, 1))

**Output:** 0.