# 1.11 DYNAMIC PROGRAMMING

**Question:**

Given an m x n grid and a ball at a starting cell, find the number of ways to move the ball out of the grid boundary in exactly N steps.

**AIM**:

To find the number of ways to move the ball out of the grid boundary in exactly N steps from m x n grid.

**ALGORITHM:**

1. Define a recursive function dfs(m, n, N, i, j):
2. If (i, j) is outside the grid → return 1 (valid way).
3. If N == 0 → return 0 (no moves left, but not outside).
4. Otherwise, recursively try 4 directions: up, down, left, right.
5. Use memorization (cache results) to avoid re-computation.
6. Final answer = dfs(m, n, N, i, j).

**PROGRAM:**

```
def find_paths(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
    count = 0
    for step in range(1, N+1):
        for x in range(m):
            for y in range(n):
                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-1][x][y]
                    else:
                        count += dp[step-1][x][y]
    print("Number of ways to exit:", count % MOD)

def run_find_paths():
    m = int(input("Enter number of rows (m): "))
    n = int(input("Enter number of columns (n): "))
    N = int(input("Enter number of steps (N): "))
    i = int(input("Enter starting row index (i): "))
    j = int(input("Enter starting column index (j): "))
    find_paths(m, n, N, i, j)

run_find_paths()
```

Input:

m = 2, n = 2, N = 2, i = 0, j = 0

Output:

```
Enter number of rows (m): 2
Enter number of columns (n): 2
Enter number of steps (N): 2
Enter starting row index (i): 0
Enter starting column index (j): 0
Number of ways to exit: 6
>>>
```

**RESULT:**

Thus the program is successfully executed, and the output is verified.

**PERFORMANCE ANALYSIS:**

- States = N × m × n
- Transitions = 4 per state
- Time Complexity = O(N × m × n)
- Space Complexity = O(N × m × n) (due to memorization table)