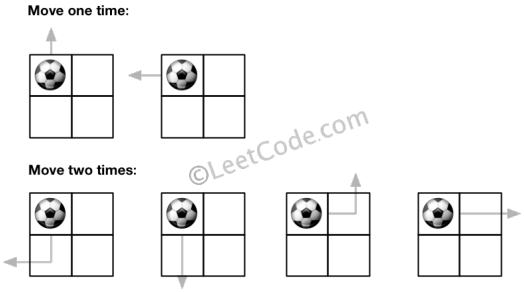
Out of Boundary Paths

There is an **m** by **n** grid with a ball. Given the start coordinate **(i,j)** of the ball, you can move the ball to **adjacent** cell or cross the grid boundary in four directions (up, down, left, right). However, you can **at most** move **N** times. Find out the number of paths to move the ball out of grid boundary. The answer may be very large, return it after mod $10^9 + 7$.

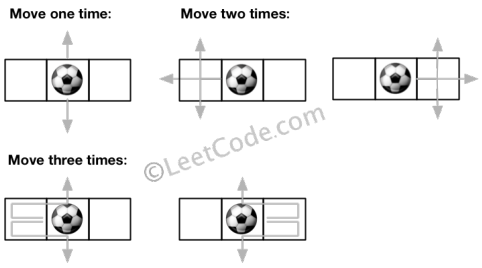
Example 1:

Input: `m = 2, n = 2, N = 2, i = 0, j = 0`
Output: `6`
Explanation:



Example 2:

Input: `m = 1, n = 3, N = 3, i = 0, j = 1`
Output: `12`
Explanation:



Note:

1. Once you move the ball out of boundary, you cannot move it back.
2. The length and height of the grid is in range [1,50].
3. N is in range [0,50].

Solution 1

The number of paths for N moves is the sum of number of paths for N - 1 moves from the adjacent cells. If an adjacent cell is out of the borders, the number of paths is 1.

```
int findPaths(int m, int n, int N, int i, int j) {
  int dp[51][50][50] = {};
  for (auto Ni = 1; Ni <= N; ++Ni)
    for (auto mi = 0; mi < m; ++mi)
      for (auto ni = 0; ni < n; ++ni)
        dp[Ni][mi][ni] = ((long long)(mi == 0 ? 1 : dp[Ni - 1][mi - 1][ni]) + (mi =
= m - 1? 1 : dp[Ni - 1][mi + 1][ni])
            + (ni == 0 ? 1 : dp[Ni - 1][mi][ni - 1]) + (ni == n - 1 ? 1 : dp[Ni - 1
][mi][ni + 1])) % 1000000007;
  return dp[N][i][j];
}
```

written by votrubac original link here

Solution 2

```
class Solution {
int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0 ,0};

public:
    long findPaths(int m, int n, int N, int i, int j) {
        int mod = (10E8) + 7;
        vector<vector<vector<long>>> dp(m, vector<vector<long>>(n, vector<long>(2))
);

        for(int i=0; i<m; ++i){
            ++dp[i][0][1];
            ++dp[i][n-1][1];
        }

        for(int j=0; j<n; ++j){
            ++dp[0][j][1];
            ++dp[m-1][j][1];
        }


        for(int k=2; k<=N; ++k){
            for(int i=0; i<m; ++i){
                for(int j=0; j<n; ++j){
                    dp[i][j][k%2] = 0;
                    for(int t=0; t<4; ++t){
                        if(outOfBounds(m,n, i+dx[t], j+dy[t])){
                            dp[i][j][k%2] = (dp[i][j][k%2]+1L) % mod;
                        } else {
                            dp[i][j][k%2] = (dp[i][j][k%2] + dp[i+dx[t]][j+dy[t]][(
k-1)%2]) % mod;
                        }
                    }
                }
            }
        }

        return dp[i][j][N%2] % mod;
    }

    bool outOfBounds(int m, int n, int i, int j){
        return i < 0 || j < 0 || i >= m || j >= n;
    }
};
```

written by kevin36 original link here

Solution 3

At time t, let's maintain `cur[r][c]` = the number of paths to `(r, c)` with `t` moves, and `nxt[r][c]` = the number of paths to `(r, c)` with `t+1` moves.

A ball at `(r, c)` at time `t`, can move in one of four directions. If it stays on the board, then it contributes to a path that takes `t+1` moves. If it falls off the board, then it contributes to the final answer.

```
def findPaths(self, R, C, N, sr, sc):
    MOD = 10**9 + 7
    nxt = [[0] * C for _ in xrange(R)]
    nxt[sr][sc] = 1

    ans = 0
    for time in xrange(N):
        cur = nxt
        nxt = [[0] * C for _ in xrange(R)]
        for r, row in enumerate(cur):
            for c, val in enumerate(row):
                for nr, nc in ((r-1, c), (r+1, c), (r, c-1), (r, c+1)):
                    if 0 <= nr < R and 0 <= nc < C:
                        nxt[nr][nc] += val
                        nxt[nr][nc] %= MOD
                    else:
                        ans += val
                        ans %= MOD

    return ans
```

written by awice original link here

From Leetcoder.