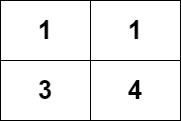
<https://leetcode.com/problems/number-of-increasing-paths-in-a-grid/description/>

You are given an m x n integer matrix grid, where you can move from a cell to any adjacent cell in all 4 directions.

Return the number of **strictly increasing** paths in the grid such that you can start from any cell and end at any cell. Since the answer may be very large, return it modulo 10^9 + 7.

Two paths are considered different if they do not have exactly the same sequence of visited cells.

**Example 1:**



**Input:** grid = [[1,1],[3,4]]

**Output:** 8

**Explanation:** The strictly increasing paths are:

- Paths with length 1: [1], [1], [3], [4].

- Paths with length 2: [1 -> 3], [1 -> 4], [3 -> 4].

- Paths with length 3: [1 -> 3 -> 4].

The total number of paths is 4 + 3 + 1 = 8.

**Example 2:**

**Input:** grid = [[1],[2]]

**Output:** 3

**Explanation:** The strictly increasing paths are:

- Paths with length 1: [1], [2].

- Paths with length 2: [1 -> 2].

The total number of paths is 2 + 1 = 3.

**Constraints:**

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m \* n <= 10^5

1 <= grid[i][j] <= 10^5

**Attempt 1: 2025-02-08**

**Solution 1: DFS + Memoization (10 min, similar to** [L329.Longest Increasing Path in a Matrix (Ref.L2328)](note://WEBf343741b03b68344176ed66cc6bb4808)**)**

**Style 1: All reject condition at the beginning of helper method**

class Solution {

    int MOD = (int)1e9 + 7;

    public int countPaths(int[][] grid) {

        int m = grid.length;

        int n = grid[0].length;

        Integer[][] memo = new Integer[m][n];

        int totalPaths = 0;

        for(int i = 0; i < m; i++) {

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

                totalPaths = (totalPaths + helper(grid, i, j, m, n, 0, memo)) % MOD;

            }

        }

        return totalPaths;

    }

    int[] dx = new int[]{0, 0, 1, -1};

    int[] dy = new int[]{1, -1, 0, 0};

    private int helper(int[][] grid, int x, int y, int m, int n, int prev, Integer[][] memo) {

        if(x < 0 || x >= m || y < 0 || y >= n || grid[x][y] <= prev) {

            return 0;

        }

        if(memo[x][y] != null) {

            return memo[x][y];

        }

        int paths = 1;

        for(int k = 0; k < 4; k++) {

            int new\_x = x + dx[k];

            int new\_y = y + dy[k];

            paths = (paths + helper(grid, new\_x, new\_y, m, n, grid[x][y], memo)) % MOD;

        }

        return memo[x][y] = paths;

    }

}

Time Complexity: O(m \* n) for both approaches, where m is the number of rows and n is the number of columns.

Space Complexity: O(m \* n) for the dp table and auxiliary data structures.

**Style 2: All reject condition inside for loop logic**

class Solution {

    int MOD = (int)1e9 + 7;

    public int countPaths(int[][] grid) {

        int m = grid.length;

        int n = grid[0].length;

        Integer[][] memo = new Integer[m][n];

        int totalPaths = 0;

        for(int i = 0; i < m; i++) {

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

                totalPaths = (totalPaths + helper(grid, i, j, m, n, 0, memo)) % MOD;

            }

        }

        return totalPaths;

    }

    int[] dx = new int[]{0, 0, 1, -1};

    int[] dy = new int[]{1, -1, 0, 0};

    private int helper(int[][] grid, int x, int y, int m, int n, int prev, Integer[][] memo) {

        if(memo[x][y] != null) {

            return memo[x][y];

        }

        int paths = 1;

        for(int k = 0; k < 4; k++) {

            int new\_x = x + dx[k];

            int new\_y = y + dy[k];

                paths = (paths + helper(grid, new\_x, new\_y, m, n, grid[x][y], memo)) % MOD;

            }

        }

        return memo[x][y] = paths;

    }

}

Time Complexity: O(m \* n) for both approaches, where m is the number of rows and n is the number of columns.

Space Complexity: O(m \* n) for the dp table and auxiliary data structures.

**Solution 2: Topological Sort (10 min, similar to** [L329.Longest Increasing Path in a Matrix (Ref.L2328)](note://WEBf343741b03b68344176ed66cc6bb4808)**)**

class Solution {

    public int countPaths(int[][] grid) {

        int MOD = (int)1e9 + 7;

        int[] dx = new int[]{0, 0, 1, -1};

        int[] dy = new int[]{1, -1, 0, 0};

        int m = grid.length;

        int n = grid[0].length;

        // Generate the indegree array

        int[][] indegree = new int[m][n];

        for(int i = 0; i < m; i++) {

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

                for(int k = 0; k < 4; k++) {

                    int new\_x = i + dx[k];

                    int new\_y = j + dy[k];

                        && grid[new\_x][new\_y] > grid[i][j]) {

                        indegree[new\_x][new\_y]++;

                    }

                }

            }

        }

        // Initialize the queue with nodes having indegree == 0

        Queue<int[]> q = new LinkedList<>();

        for(int i = 0; i < m; i++) {

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

                if(indegree[i][j] == 0) {

                    q.offer(new int[]{i, j});

                }

            }

        }

        // Initialize dp table

        int[][] dp = new int[m][n];

        for(int i = 0; i < m; i++) {

            Arrays.fill(dp[i], 1);

        }

        // Perform BFS to count paths

        while(!q.isEmpty()) {

            int[] cur = q.poll();

            int x = cur[0];

            int y = cur[1];

            for(int k = 0; k < 4; k++) {

                int new\_x = x + dx[k];

                int new\_y = y + dy[k];

                    && grid[new\_x][new\_y] > grid[x][y]) {

                    dp[new\_x][new\_y] = (dp[new\_x][new\_y] + dp[x][y]) % MOD;

                    indegree[new\_x][new\_y]--;

                    if(indegree[new\_x][new\_y] == 0) {

                        q.offer(new int[]{new\_x, new\_y});

                    }

                }

            }

        }

        // Sum of all paths

        int totalPaths = 0;

        for(int i = 0; i < m; i++) {

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

                totalPaths = (totalPaths + dp[i][j]) % MOD;

            }

        }

        return totalPaths;

    }

}

Time Complexity: O(m \* n) for both approaches, where m is the number of rows and n is the number of columns.

Space Complexity: O(m \* n) for the dp table and auxiliary data structures.

**Refer to Deepseek**

**LeetCode problem 2328. Number of Increasing Paths in a Grid is similar to 329. Longest Increasing Path in a Matrix, but instead of finding the length of the longest increasing path, we need to count the total number of strictly increasing paths in the grid. This problem can also be solved using DFS with memoization or Topological Sorting, but with a slight modification to count the paths.**

**Problem Summary:**

You are given an m x n grid of integers.

A path is strictly increasing if the values along the path are in increasing order.

You can move in four directions: left, right, up, or down.

Return the total number of strictly increasing paths in the grid. Since the answer may be large, return it modulo 10^9 + 7.

**Key Observations:**

Strictly Increasing Paths:

Similar to LeetCode 329, the path must be strictly increasing, so cycles are impossible.

This means the problem can be modeled as a Directed Acyclic Graph (DAG).

Counting Paths:

Instead of finding the length of the longest path, we need to count all possible paths.

Each cell (i, j) can be the starting point of one or more paths.

Overlapping Subproblems:

The number of paths starting from a cell (i, j) depends on the number of paths starting from its neighboring cells with larger values.

This suggests that memoization can be used to avoid redundant calculations.

**Approach 1: DFS with Memoization**

**Steps:**

DFS Traversal:

For each cell (i, j), perform a DFS to count the number of increasing paths starting from that cell.

If the result for a cell is already computed (memoized), return it directly.

Memoization:

Use a dp table to store the number of paths starting from each cell (i, j).

**Base Case:**

**If a cell has no neighbors with larger values, the number of paths starting from it is 1 (the cell itself).**

**Recursive Case:**

**For each neighbor (x, y) with a larger value, recursively compute the number of paths starting from (x, y) and add it to the result for (i, j).**

Modulo Operation:

Since the result can be large, take the result modulo 10^9 + 7 at each step.

**Solution Code (DFS with Memoization):**

class Solution {

private static final int MOD = 1\_000\_000\_007;

private static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

public int countPaths(int[][] grid) {

int m = grid.length;

int n = grid[0].length;

int[][] dp = new int[m][n]; // Memoization table

int totalPaths = 0;

// Iterate through each cell and compute the number of paths starting from it

for (int i = 0; i < m; i++) {

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

totalPaths = (totalPaths + dfs(grid, i, j, dp)) % MOD;

}

}

return totalPaths;

}

private int dfs(int[][] grid, int i, int j, int[][] dp) {

// If the result for this cell is already computed, return it

if (dp[i][j] != 0) {

return dp[i][j];

}

int paths = 1; // The cell itself is a valid path

// Explore all four directions

for (int[] dir : DIRECTIONS) {

int x = i + dir[0];

int y = j + dir[1];

// Check if the neighboring cell is within bounds and has a larger value

paths = (paths + dfs(grid, x, y, dp)) % MOD;

}

}

// Store the result in the dp table to avoid recomputation

dp[i][j] = paths;

return paths;

}

}

**Approach 2: Topological Sorting (BFS with Indegree)**

**Steps:**

Graph Representation:

Treat each cell as a node.

Draw a directed edge from (i, j) to (x, y) if grid[x][y] > grid[i][j].

Indegree Calculation:

Compute the indegree of each node (number of smaller neighbors).

Topological Sorting:

Use BFS to process nodes level by level, starting from nodes with indegree == 0.

For each node, update the number of paths for its neighbors.

Counting Paths:

Use a dp table to store the number of paths starting from each node.

Initialize dp[i][j] = 1 for all nodes (each cell is a valid path by itself).

During BFS, propagate the path counts to neighbors.

**Solution Code (Topological Sorting):**

class Solution {

private static final int MOD = 1\_000\_000\_007;

private static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

public int countPaths(int[][] grid) {

int m = grid.length;

int n = grid[0].length;

// Initialize dp table

int[][] dp = new int[m][n];

for (int i = 0; i < m; i++) {

Arrays.fill(dp[i], 1); // Each cell is a valid path by itself

}

// Generate the indegree array

int[][] indegree = new int[m][n];

for (int i = 0; i < m; i++) {

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

for (int[] dir : DIRECTIONS) {

int x = i + dir[0];

int y = j + dir[1];

indegree[x][y]++;

}

}

}

}

// Initialize the queue with nodes having indegree == 0

Queue<int[]> queue = new LinkedList<>();

for (int i = 0; i < m; i++) {

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

if (indegree[i][j] == 0) {

queue.offer(new int[]{i, j});

}

}

}

// Perform BFS to count paths

while (!queue.isEmpty()) {

int[] cur = queue.poll();

int i = cur[0];

int j = cur[1];

for (int[] dir : DIRECTIONS) {

int x = i + dir[0];

int y = j + dir[1];

dp[x][y] = (dp[x][y] + dp[i][j]) % MOD;

indegree[x][y]--;

if (indegree[x][y] == 0) {

queue.offer(new int[]{x, y});

}

}

}

}

// Sum all paths

int totalPaths = 0;

for (int i = 0; i < m; i++) {

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

totalPaths = (totalPaths + dp[i][j]) % MOD;

}

}

return totalPaths;

}

}

**Complexity Analysis:**

Time Complexity: O(m \* n) for both approaches, where m is the number of rows and n is the number of columns.

Space Complexity: O(m \* n) for the dp table and auxiliary data structures.

Both approaches are efficient and suitable for solving the problem. The choice between DFS with memoization and topological sorting depends on your familiarity with the techniques.

**Refer to**

[L329.Longest Increasing Path in a Matrix (Ref.L2328)](note://WEBf343741b03b68344176ed66cc6bb4808)

[L797.All Paths From Source to Target](note://WEB9a09691e194cc80216d86222e3e676f6)