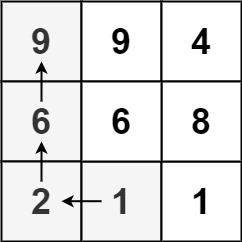
Given an m x n integers matrix, return *the length of the longest increasing path in*matrix.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

**Example 1:**

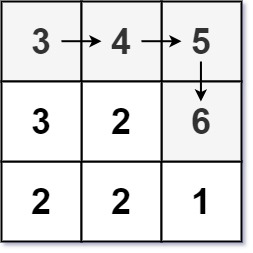


**Input:** matrix = [[9,9,4],[6,6,8],[2,1,1]]

**Output:** 4

**Explanation:** The longest increasing path is [1, 2, 6, 9].

**Example 2:**



**Input:** matrix = [[3,4,5],[3,2,6],[2,2,1]]

**Output:** 4

**Explanation:** The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

**Example 3:**

**Input:** matrix = [[1]]

**Output:** 1

Solution:

class Solution {

public static final int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

public int longestIncreasingPath(int[][] matrix) {

if(matrix.length == 0) return 0;

int m = matrix.length, n = matrix[0].length;

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

int max = 1;

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

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

int len = dfs(matrix, i, j, m, n, dp);

max = Math.max(max, len);

}

}

return max;

}

public int dfs(int[][] matrix, int i, int j, int m, int n, int[][] dp) {

if(dp[i][j] != 0) return dp[i][j];

int max = 1;

for(int[] dir: dirs) {

int x = i + dir[0], y = j + dir[1];

if(x < 0 || x >= m || y < 0 || y >= n || matrix[x][y] <= matrix[i][j]) continue;

int len = 1 + dfs(matrix, x, y, m, n, dp);

max = Math.max(max, len);

}

dp[i][j] = max;

return max;

}

}