Path with Minimum Effort

Shishir Sharma Rijal

1 Description

You are a hiker preparing for an upcoming hike. You are given heights, a 2D array of size rows x columns, where heights[row][col] represents the height of cell (row, col). You are situated in the top-left cell, (0, 0), and you hope to travel to the bottom-right cell, (rows-1, columns-1) (i.e., 0-indexed). You can move up, down, left, or right, and you wish to find a route that requires the minimum effort.

A route's effort is the maximum absolute difference in heights between two consecutive cells of the route.

Return the minimum effort required to travel from the top-left cell to the bottom-right cell.

2 Code

```
class Solution(object):
    def minimumEffortPath(self, heights):
        """
        :type heights: List[List[int]]
        :rtype: int
        """
        import heapq, math
        rows, cols = len(heights), len(heights[0])
        dist = [[math.inf] * cols for _ in range(rows)]  # Initialize distances to infinity
        dist[0][0] = 0  # Starting cell has 0 effort

        minHeap = [(0, 0, 0)]  # (effort, row, col)

        while minHeap:
            d, i, j = heapq.heappop(minHeap)  # Get cell with min effort
            if i == rows - 1 and j == cols - 1:  # Reached destination
                return d

            for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:  # Explore neighbors
                x, y = i + dx, j + dy
                if 0 <= x < rows and 0 <= y < cols:  # Check if neighbor is valid
                    new_effort = max(d, abs(heights[i][j] - heights[x][y]))  # Calculate effort
                    if new_effort < dist[x][y]:  # If lower effort found
                        dist[x][y] = new_effort
                        heapq.heappush(minHeap, (new_effort, x, y))  # Add to heap
```

3 Explanation

3.1 Example 1

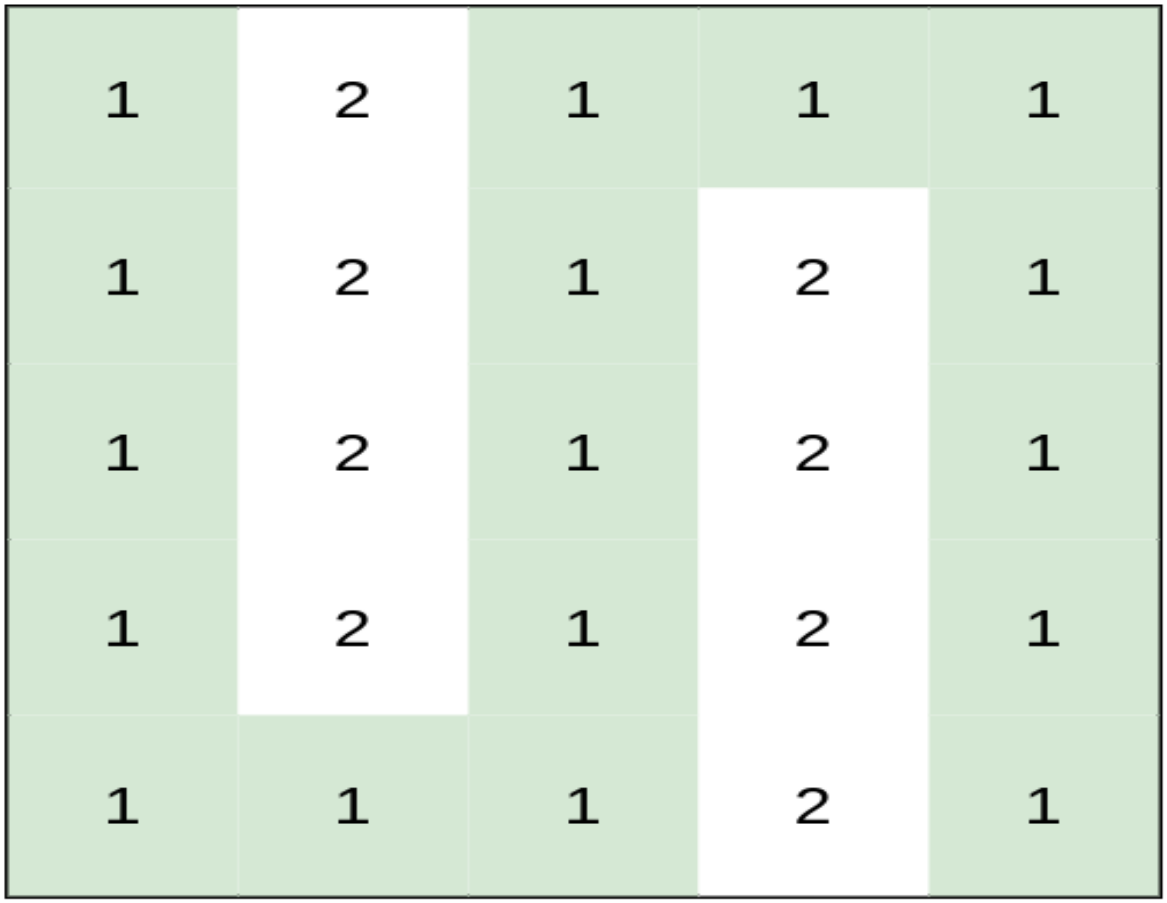

Figure 1: Organizational Structure of JankariTech

Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]
Output: 0
Explanation: This route does not require any effort.

3.2 Example 2

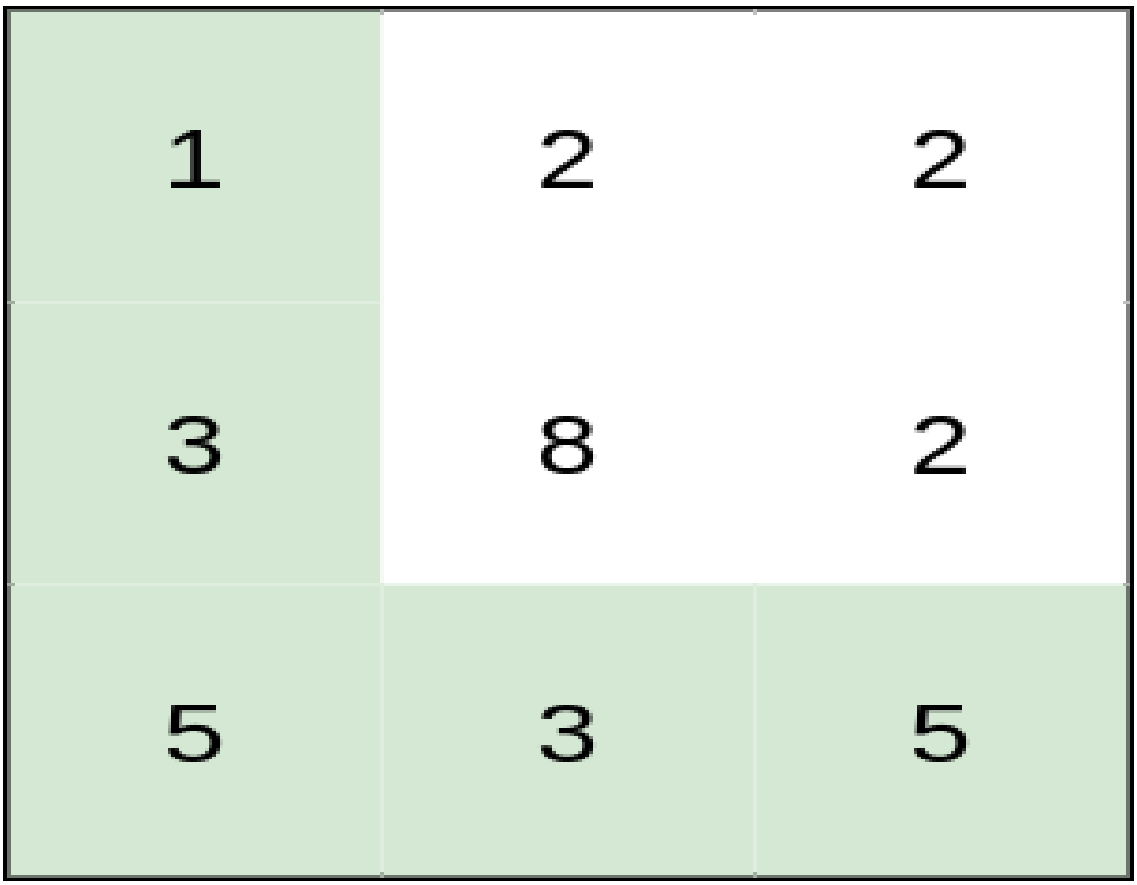

Figure 2: Organizational Structure of JankariTech

Input: heights = [[1,2,2],[3,8,2],[5,3,5]]

Output: 2

Explanation: The route of [1,3,5,3,5] has a maximum absolute difference of 2 in consecutive cells. This is better than the route of [1,2,2,2,5], where the maximum absolute difference is 3.

3.3 Variable Updates

Table 1: Variable Updates for Example 2

Iteration	(i, j)	d	(x, y)	new_effort	$dist[x][y]$	`minHeap`
1	(0, 0)	0	(0, 1)	1	1	[(1, 0, 1)]
2	(0, 1)	1	(0, 2)	0	1	[(1, 0, 2)]
3	(0, 2)	1	(1, 2)	1	1	[(1, 1, 2)]
4	(1, 2)	1	(2, 2)	3	3	[(3, 2, 2)]
5	(2, 2)	3	(2, 1)	2	2	[(2, 2, 1)]
6	(2, 1)	2	(2, 0)	3	3	[(3, 2, 0)]
7	(2, 0)	3	(1, 0)	4	4	[(4, 1, 0)]
8	(1, 0)	4	(1, 1)	6	6	[(6, 1, 1)]
9	(1, 1)	6	(0, 1)	6	6	[]

4 Result

The minimum effort required to travel from the top-left cell to the bottom-right cell is 2. This is the maximum absolute difference in heights between two consecutive cells in the path.

5 Conclusion

This problem can be solved using a variation of Dijkstra's algorithm. However, the given code is based on the Bellman-Ford algorithm. The algorithm maintains a min-heap (priority queue) to always extend the path with the current minimum effort. The `dist` array keeps track of the minimum effort required to reach each cell, ensuring that we explore the least effort paths first. By doing so, the algorithm efficiently finds the minimum effort path from the top-left to the bottom-right cell.