https://course.acciojob.com/idle?question=d8b0a8ae-b9f4-4647-9d7b-eb783e5e5781

- HARD
- Max Score: 50 Points

Path With Minimum Effort

You are given an `nxm` grid, where each cell has a value representing the height of the terrain at that point. A path from `[x1, y1]` to `[x2, y2]` is a sequence of cells such that `x1 <= x2` and `y1 <= y2` and for each `(x, y)` pair in the sequence, `x + 1 = x2` or `y + 1 = y2` and `abs(grid[x1][y1] - grid[x2][y2]) <= d` where `d` is the maximum absolute difference in height of any two consecutive cells in the path. You want to find a path from the top-left cell to the bottom-right cell such that the maximum absolute difference in height between any two consecutive cells in the path is minimized, and return this minimum difference.

Input Format

The first line contains two integers `n` and `m`, the number of rows and columns in the grid.

The next `n` lines each contain `m` integers, where the `j`th integer in the `i`th line is `grid[i][j]`.

Output Format

Return the minimum difference in height between any two consecutive cells in the path from the top-left cell to the bottom-right cell.

Example 1

Input

```
3 3
1 2 3
3 8 4
5 3 5
```

Output

```
1
```

Explanation

The path from `[0, 0]` **to** `[2, 2]` **is** `[0, 0] -> [0, 1] -> [0, 2] -> [1, 2] -> [2, 2]`**. The maximum absolute difference in height is** `1` **(between** `[1, 2]` **and** `[2, 2]`**). This is the minimum possible value, so return** `1`**.**

Example 2

Input

```
2 2
1 2
2 10
```

Output

```
8
```

Explanation

All paths have a maximum absolute difference in height of at least `8`**.**

Constraints

- `1 <= n,m <= 100`
- `1 <= heights[i][j] < 1e6`

Topic Tags

- Recursion
- Graphs
- 2D-Arrays
- DP

My code

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        int[][] grid = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = sc.nextInt();
            }
        }
        Solution solution = new Solution();
        System.out.println(solution.minimumEffortPath(grid));
    }
}

class Solution {
    public int minimumEffortPath(int[][] heights) {
                    PriorityQueue<int[]> pq = new PriorityQueue<>(new Comparator<int[]>() {
                            public int compare(int[] a, int[] b) {
                                    return Integer.compare(a[2], b[2]);
                            }
                    });

                    int m = heights.length;
                    int n = heights[0].length;

                    boolean vis[][] = new boolean[m][n];

                    for(int i = 0; i < m; i++) for(int j = 0; j < n; j++) vis[i][j] = false;


                    pq.add(new int[]{0, 0, 0});

                    int di[] = new int[]{-1, 1, 0, 0};
                    int dj[] = new int[]{0, 0, -1, 1};
```

```
        while(pq.size() > 0){
            int[] curr = pq.poll();
            int i = curr[0], j = curr[1], dist_ij = curr[2];

            if(vis[i][j] == true) continue;

            if(i == m-1 && j == n-1) return dist_ij;

            vis[i][j] = true;

            // for all nbrs, put in pq
            for(int k = 0; k < 4; k++) {
                int new_i = i + di[k];
                int new_j = j + dj[k];

                if(new_i < 0 || new_i >= m || new_j < 0 || new_j >= n) continue;
                if(vis[new_i][new_j] == true) continue;

                int g_uv = Math.abs(heights[new_i][new_j] - heights[i][j]);

                pq.add(new int[]{new_i, new_j, Math.max(dist_ij, g_uv)});
            }
        }

        return -1;
    }
}
```