778. Swim in Rising Water

DescriptionHintsSubmissionsDiscussSolution

On an N x N `grid`, each square `grid[i][j]` represents the elevation at that point `(i,j)`.

Now rain starts to fall. At time `t`, the depth of the water everywhere is `t`. You can swim from a square to another 4-directionally adjacent square if and only if the elevation of both squares individually are at most `t`. You can swim infinite distance in zero time. Of course, you must stay within the boundaries of the grid during your swim.

You start at the top left square `(0, 0)`. What is the least time until you can reach the bottom right square `(N-1, N-1)`?

Example 1:

```
Input: [[0,2],[1,3]]
Output: 3
Explanation:
At time 0, you are in grid location (0, 0).
You cannot go anywhere else because 4-directionally adjacent neighbors have a
higher elevation than t = 0.

You cannot reach point (1, 1) until time 3.
When the depth of water is 3, we can swim anywhere inside the grid.
```

Example 2:

```
Input: [[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]]
Output: 16
Explanation:
 0  1  2  3  4
24 23 22 21  5
12 13 14 15 16
11 17 18 19 20
10  9  8  7  6

The final route is marked in bold.
We need to wait until time 16 so that (0, 0) and (4, 4) are connected.
```

Note:

1. `2 <= N <= 50`.
2. grid[i][j] is a permutation of [0, ..., N*N - 1].

```
class Solution {
public:
    int swimInWater(vector<vector<int>>& grid) {
        int n = grid.size(), ans = max(grid[0][0], grid[n-1][n-1]);
        priority_queue<vector<int>, vector<vector<int>>, greater<vector<int>>> pq;
        vector<vector<int>> visited(n, vector<int>(n, 0));
        visited[0][0] = 1;
        vector<int> dir({-1, 0, 1, 0, -1});
        pq.push({ans, 0, 0});
        while (!pq.empty()) {
            auto cur = pq.top();
            pq.pop();
            ans = max(ans, cur[0]);
            for (int i = 0; i < 4; ++i) {
                int r = cur[1] + dir[i], c = cur[2] + dir[i+1];
                if (r >= 0 && r < n && c >= 0 && c < n && visited[r][c] == 0) {
                    if (r == n-1 && c == n-1) return ans;
                    pq.push({grid[r][c], r, c});
                    visited[r][c] = 1;
                }
            }
        }
        return -1;
    }
};
```