**[Find the Safest Path in a Grid](https://leetcode.com/problems/find-the-safest-path-in-a-grid/)**

You are given a **0-indexed** 2D matrix grid of size n x n, where (r, c) represents:

* A cell containing a thief if grid[r][c] = 1
* An empty cell if grid[r][c] = 0

You are initially positioned at cell (0, 0). In one move, you can move to any adjacent cell in the grid, including cells containing thieves.

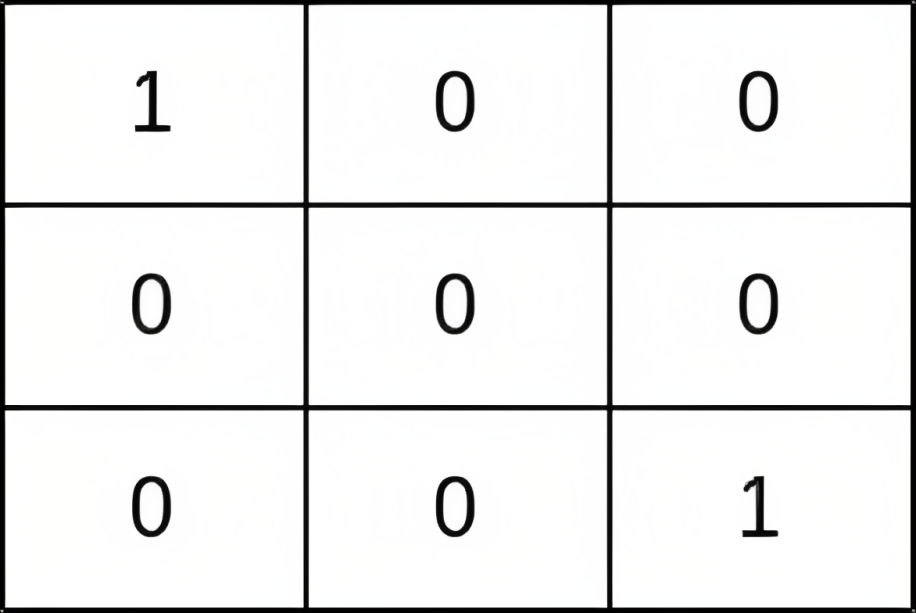
The **safeness factor** of a path on the grid is defined as the **minimum** manhattan distance from any cell in the path to any thief in the grid.

Return *the****maximum safeness factor****of all paths leading to cell*(n - 1, n - 1)*.*

An **adjacent** cell of cell (r, c), is one of the cells (r, c + 1), (r, c - 1), (r + 1, c) and (r - 1, c) if it exists.

The **Manhattan distance** between two cells (a, b) and (x, y) is equal to |a - x| + |b - y|, where |val| denotes the absolute value of val.

**Example 1:**

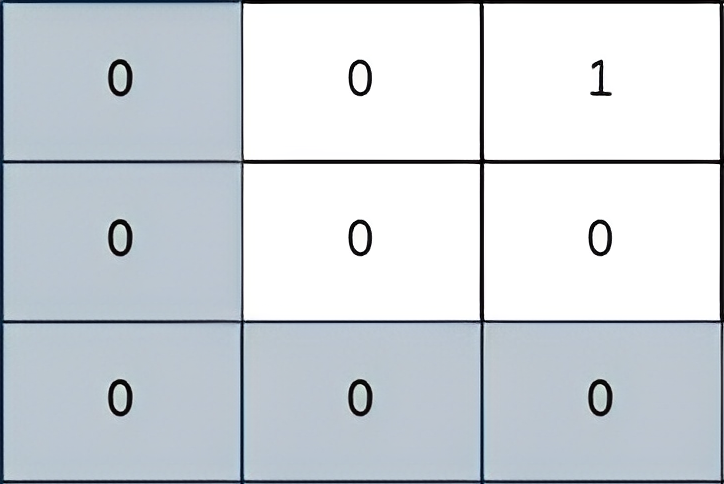


**Input:** grid = [[1,0,0],[0,0,0],[0,0,1]]

**Output:** 0

**Explanation:** All paths from (0, 0) to (n - 1, n - 1) go through the thieves in cells (0, 0) and (n - 1, n - 1).

**Example 2:**



**Input:** grid = [[0,0,1],[0,0,0],[0,0,0]]

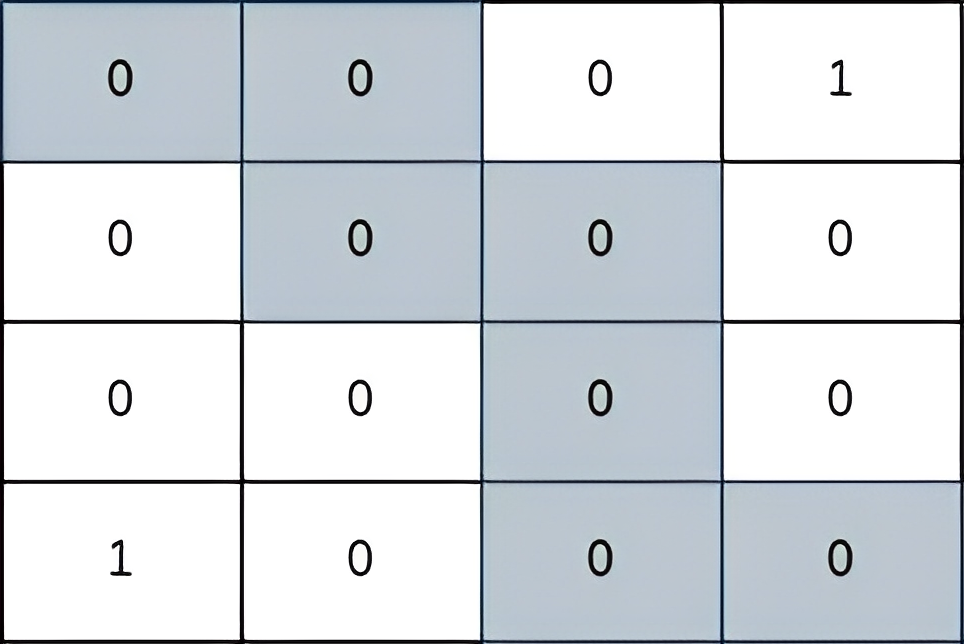
**Output:** 2

**Explanation:** The path depicted in the picture above has a safeness factor of 2 since:

- The closest cell of the path to the thief at cell (0, 2) is cell (0, 0). The distance between them is | 0 - 0 | + | 0 - 2 | = 2.

It can be shown that there are no other paths with a higher safeness factor.

**Example 3:**



**Input:** grid = [[0,0,0,1],[0,0,0,0],[0,0,0,0],[1,0,0,0]]

**Output:** 2

**Explanation:** The path depicted in the picture above has a safeness factor of 2 since:

- The closest cell of the path to the thief at cell (0, 3) is cell (1, 2). The distance between them is | 0 - 1 | + | 3 - 2 | = 2.

- The closest cell of the path to the thief at cell (3, 0) is cell (3, 2). The distance between them is | 3 - 3 | + | 0 - 2 | = 2.

It can be shown that there are no other paths with a higher safeness factor.

**Constraints:**

* 1 <= grid.length == n <= 400
* grid[i].length == n
* grid[i][j] is either 0 or 1.
* There is at least one thief in the grid.

class Solution {

public:

    vector<int> roww = {0,0,-1,1};

    vector<int> coll = {-1,1,0,0};

    void bfs(vector<vector<int>>& grid,vector<vector<int>>& score,int n) {

        queue<pair<int, int>> q;

        for(int i = 0; i < n; i++) {

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

                if(grid[i][j]) {

                    score[i][j] = 0;

                    q.push({i, j});

                }

            }

        }

        while(!q.empty()){

            auto t = q.front();

            q.pop();

            int x = t.first, y = t.second;

            int s = score[x][y];

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

                int newX = x + roww[i];

                int newY = y + coll[i];

                if(newX >= 0 && newX < n && newY >= 0 && newY < n && score[newX][newY] > 1 + s) {

                    score[newX][newY] = 1 + s;

                    q.push({newX, newY});

                }

            }

        }

    }

    int maximumSafenessFactor(vector<vector<int>>& grid) {

        ios\_base::sync\_with\_stdio(false);cin.tie(nullptr);cout.tie(nullptr);

        int n = grid.size();

        if(grid[0][0] || grid[n - 1][n - 1]) return 0;

        vector<vector<int>> score(n,vector<int>(n,INT\_MAX));

        bfs(grid, score, n);

        vector<vector<bool>> vis(n, vector<bool>(n, false));

        priority\_queue<pair<int,pair<int,int>>> pq;

        pq.push({score[0][0], {0,0}});

        while(!pq.empty()){

            auto temp = pq.top().second;

            auto safe = pq.top().first;

            pq.pop();

            if(temp.first == n - 1 && temp.second == n - 1) return safe;

            vis[temp.first][temp.second] = true;

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

                int newX = temp.first + roww[i];

                int newY = temp.second + coll[i];

                if(newX >= 0 && newX < n && newY >= 0 && newY < n && !vis[newX][newY]){

                    int s = min(safe, score[newX][newY]);

                    pq.push({s, {newX, newY}});

                    vis[newX][newY] = true;

                }

            }

        }

        return -1;

    }

};

Link : https://leetcode.com/problems/find-the-safest-path-in-a-grid/submissions/1258956680/?envType=daily-question&envId=2024-05-15