2812. Find the Safest Path in a Grid

My Submissions (/contest/weekly-contest-357/problems/find-the-safest-path-in-a-grid/submissions/) Back to Contest (/contest/weekly-contest-357/)

User Accepted:	1506
User Tried:	4740
Total Accepted:	1566
Total Submissions:	12028
Difficulty:	Medium

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.

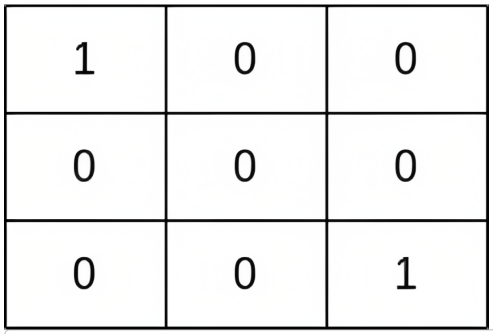
The **safeness factor** of a path on the grid is defined as the **minimum** manhattan distance from any cell in the path to any thief in the grid.

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.

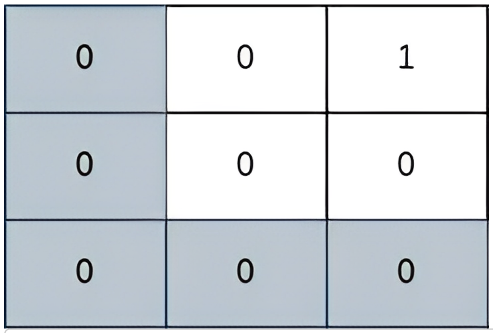
The **Manhattan distance** between two cells `(a, b)` and `(x, y)` is equal to `|a - x| + |b - y|`, where `|val|` denotes the absolute value of val.

Example 1:



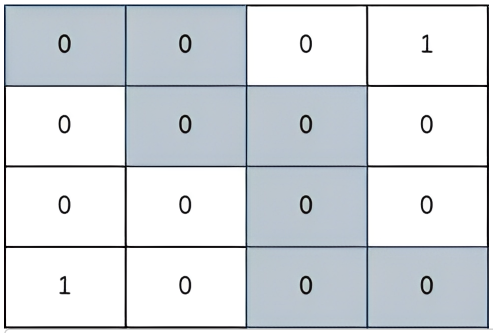
```
Input: grid = [[1,0,0],[0,0,0],[0,0,1]]
Output: 0
Explanation: All paths from (0, 0) to (n - 1, n - 1) go through the thieves in cells (0, 0) and (n - 1, n - 1).
```

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

Example 3:



0	0	0	1
0	0	0	0
0	0	0	0
1	0	0	0

```
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 | =
- 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 | =
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`.

Discuss (https://leetcode.com/problems/find-the-safest-path-in-a-grid/discuss)

JavaScript

```javascript
function DJSet(n) {
    let parent = Array(n).fill(-1);
    return { find, union, count, equiv, par, grp }
    function find(x) {
        return parent[x] < 0 ? x : parent[x] = find(parent[x]);
    }
    function union(x, y) {
        x = find(x);
        y = find(y);
        if (x == y) return false;
        if (parent[x] < parent[y]) [x, y] = [y, x];
        parent[x] += parent[y];
        parent[y] = x;
        return true;
    }
    function count() { // total groups
        return parent.filter(v => v < 0).length;
    }
    function equiv(x, y) { // isConnected
        return find(x) == find(y);
    }
    function par() {
        return parent;
    }
    function grp() { // generate all groups (nlogn)
        let groups = [];
        for (let i = 0; i < n; i++) groups.push([]);
        for (let i = 0; i < n; i++) groups[find(i)].push(i); // sorted and unique
        return groups;
    }
}

const initialize2DArray = (n, m) => [...Array(n)].map(() => Array(m).fill(Number.MAX_SAFE_INTEGER));
const maximumSafenessFactor = (g) => {
    let n = g.length, m = g[0].length, dis = minDisGlobal(g), ds = new DJSet(n * m), es = [];
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) es.push([i, j, dis[i][j]]);
    }
    es.sort((x, y) => y[2] - x[2]);
    let path = initialize2DArray(n, m);
    for (let i = 0; i < n; i++) path[i].fill(false);
    for (const [x, y, d] of es) {
        path[x][y] = true;
        for (let k = 0; k < 4; k++) {
            let nx = x + dx[k], ny = y + dy[k];
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            if (path[nx][ny]) ds.union(x * m + y, nx * m + ny);
            if (ds.equiv(0, (n - 1) * m + (m - 1))) return d;
        }
    }
    return 0;
}

const dx = [-1, 1, 0, 0], dy = [0, 0, -1, 1];
const minDisGlobal = (g) => {
    let n = g.length, m = g[0].length, dis = initialize2DArray(n, m), q = [], thief = 1;
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            if (g[i][j] == thief) {
                dis[i][j] = 0;
                q.push([i, j]);
            }
        }
    }
    while (q.length) {
        let [x, y] = q.shift();
        for (let k = 0; k < 4; k++) {
            let nx = x + dx[k], ny = y + dy[k];
            if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
            if (dis[nx][ny] > dis[x][y] + 1) {
                dis[nx][ny] = dis[x][y] + 1;
                q.push([nx, ny]);
            }
        }
    }
    return dis;
};
```

☐ **Custom Testcase** Use Example Testcases

Run Submit

Submission Result: Accepted (/submissions/detail/1015297696/) More Details (/submissions/detail/1015297696/)

Share your acceptance!