LeetCode
Explore
Problems
New
Interview
Contest
Discuss
Store
🎁 LeetCoding Challenge + GIVEAWAY! 🎁
Description
Solution
Discuss (420)
Submissions
Java
Autocomplete

317. Shortest Distance from All Buildings

Hard 1297 101 Add to List Share

You are given an `m x n` grid `grid` of values `0`, `1`, or `2`, where:

- each `0` marks **an empty land** that you can pass by freely,
- each `1` marks **a building** that you cannot pass through, and
- each `2` marks **an obstacle** that you cannot pass through.

You want to build a house on an empty land that reaches all buildings in the **shortest total travel** distance. You can only move up, down, left, and right.

Return *the **shortest travel distance** for such a house*. If it is not possible to build such a house according to the above rules, return `-1`.

The **total travel distance** is the sum of the distances between the houses of the friends and the meeting point.

The distance is calculated using Manhattan Distance, where `distance(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|`.

Example 1:

1	0	2	0	1
0	0	0	0	0
0	0	1	0	0

```
Input: grid = [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]
Output: 7
Explanation: Given three buildings at (0,0), (0,4), (2,2),
and an obstacle at (0,2).
The point (1,2) is an ideal empty land to build a house, as
the total travel distance of 3+3+1=7 is minimal.
So return 7.
```

Example 2:

```
Input: grid = [[1,0]]
Output: 1
```

Example 3:

```
Input: grid = [[1]]
Output: -1
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 50`
- `grid[i][j]` is either `0`, `1`, or `2`.
- There will be **at least one** building in the `grid`.

Accepted 114,743 Submissions 262,370

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Company	Count
Facebook	27
DoorDash	8
Google	2
Oracle	2

Related Topics

Array Breadth-First Search Matrix

Similar Questions

Walls and Gates	Medium
Best Meeting Point	Hard
As Far from Land as Possible	Medium

```java
class Solution {
    private void bfs(int[][] grid, int[][][] distances, int row, int col) {
        int dirs[][] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

        int rows = grid.length;
        int cols = grid[0].length;

        // Use a queue to do a bfs, starting from each cell located at (row, col).
        Queue<int[]> q = new LinkedList<>();
        q.offer(new int[]{ row, col });

        // Keep track of visited cells.
        boolean[][] vis = new boolean[rows][cols];
        vis[row][col] = true;

        int steps = 0;

        while (!q.isEmpty()) {
            for (int i = q.size(); i > 0; --i) {
                int[] curr = q.poll();
                row = curr[0];
                col = curr[1];

                // If we reached an empty cell, then add the distance
                // and increment the count of houses reached at this cell.
                if (grid[row][col] == 0) {
                    distances[row][col][0] += steps;
                    distances[row][col][1] += 1;
                }

                // Traverse the next cells which is not a blockage.
                for (int[] dir : dirs) {
                    int nextRow = row + dir[0];
                    int nextCol = col + dir[1];

                    if (nextRow >= 0 && nextCol >= 0 && nextRow < rows && nextCol < cols) {
                        if (!vis[nextRow][nextCol] && grid[nextRow][nextCol] == 0) {
                            vis[nextRow][nextCol] = true;
                            q.offer(new int[]{ nextRow, nextCol });
                        }
                    }
                }
            }

            // After traversing one level cells, increment the steps by 1.
            steps++;
        }
    }

    public int shortestDistance(int[][] grid) {
        int minDistance = Integer.MAX_VALUE;
        int rows = grid.length;
        int cols = grid[0].length;
        int totalHouses = 0;
```