Shortest Distance from All Buildings

You want to build a house on an *empty* land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values **0**, **1** or **2**, where:

- Each **0** marks an empty land which you can pass by freely.
- Each **1** marks a building which you cannot pass through.
- Each **2** marks an obstacle which you cannot pass through.

For example, given three buildings at `(0,0)`, `(0,4)`, `(2,2)`, and an obstacle at `(0,2)`:

```
1 - 0 - 2 - 0 - 1
|   |   |   |   |
0 - 0 - 0 - 0 - 0
|   |   |   |   |
0 - 0 - 1 - 0 - 0
```

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.

Note:
There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

Solution 1

I also tested the other three C++ solutions posted so far, they took 340-1812 ms. I think mine is faster because I don't use a fresh "`visited`" for each BFS. Instead, I walk only onto the cells that were reachable from all previous buildings. From the first building I only walk onto cells where `grid` is 0, and make them -1. From the second building I only walk onto cells where `grid` is -1, and I make them -2. And so on.

```
int shortestDistance(vector<vector<int>> grid) {
    int m = grid.size(), n = grid[0].size();
    auto total = grid;
    int walk = 0, mindist, delta[] = {0, 1, 0, -1, 0};
    for (int i=0; i<m; ++i) {
        for (int j=0; j<n; ++j) {
            if (grid[i][j] == 1) {
                mindist = -1;
                auto dist = grid;
                queue<pair<int, int>> q;
                q.emplace(i, j);
                while (q.size()) {
                    auto ij = q.front();
                    q.pop();
                    for (int d=0; d<4; ++d) {
                        int i = ij.first + delta[d];
                        int j = ij.second + delta[d+1];
                        if (i >= 0 && i < m && j >= 0 && j < n && grid[i][j] == w
alk) {
                            grid[i][j]--;
                            dist[i][j] = dist[ij.first][ij.second] + 1;
                            total[i][j] += dist[i][j] - 1;
                            q.emplace(i, j);
                            if (mindist < 0 || mindist > total[i][j])
                                mindist = total[i][j];
                        }
                    }
                }
                walk--;
            }
        }
    }
    return mindist;
}
```

written by StefanPochmann original link here

Solution 2

Inspired by previous solution. The main idea is the following:

Traverse the matrix. For each building, use BFS to compute the shortest distance from each '0' to this building. After we do this for all the buildings, we can get the sum of shortest distance from every '0' to all reachable buildings. This value is stored in 'distance[][]'. For example, if grid[2][2] == 0, distance[2][2] is the sum of shortest distance from this block to all reachable buildings. Time complexity: O(number of 1)*O(number of 0) ~ O(m^2n^2)*

We also count how many building each '0' can be reached. It is stored in reach[][]. This can be done during the BFS. We also need to count how many total buildings are there in the matrix, which is stored in 'buildingNum'.

Finally, we can traverse the distance[][] matrix to get the point having shortest distance to all buildings. O(m*n)

The total time complexity will be O(m^2*n^2), which is quite high!. Please let me know if I did the analysis wrong or you have better solution.

```
public class Solution {
    public int shortestDistance(int[][] grid) {
        if (grid == null || grid[0].length == 0) return 0;
        final int[] shift = new int[] {0, 1, 0, -1, 0};

        int row  = grid.length, col = grid[0].length;
        int[][] distance = new int[row][col];
        int[][] reach = new int[row][col];
        int buildingNum = 0;

        for (int i = 0; i < row; i++) {
            for (int j =0; j < col; j++) {
                if (grid[i][j] == 1) {
                    buildingNum++;
                    Queue<int[]> myQueue = new LinkedList<int[]>();
                    myQueue.offer(new int[] {i,j});

                    boolean[][] isVisited = new boolean[row][col];
                    int level = 1;

                    while (!myQueue.isEmpty()) {
                        int qSize = myQueue.size();
                        for (int q = 0; q < qSize; q++) {
                            int[] curr = myQueue.poll();

                            for (int k = 0; k < 4; k++) {
                                int nextRow = curr[0] + shift[k];
                                int nextCol = curr[1] + shift[k + 1];

                                if (nextRow >= 0 && nextRow < row && nextCol >= 0
&& nextCol < col
                                    && grid[nextRow][nextCol] == 0 && !isVisited[
nextRow][nextCol]) {
                                    //The shortest distance from [nextRow][ne
```

```
xtCol] to this building
                                        // is 'level'.
                                        distance[nextRow][nextCol] += level;
                                        reach[nextRow][nextCol]++;

                                        isVisited[nextRow][nextCol] = true;
                                        myQueue.offer(new int[] {nextRow, nextCol
});
                                    }
                                }
                            }
                            level++;
                        }
                    }
                }
            }

            int shortest = Integer.MAX_VALUE;
            for (int i = 0; i < row; i++) {
                for (int j = 0; j < col; j++) {
                    if (grid[i][j] == 0 && reach[i][j] == buildingNum) {
                        shortest = Math.min(shortest, distance[i][j]);
                    }
                }
            }

            return shortest == Integer.MAX_VALUE ? -1 : shortest;


        }
}
```

written by shuoshankou original link here

Solution 3

```
    int shortestDistance(vector<vector<int>>& grid) {
    const int row = grid.size();
    if (0 == row) return -1;
    const int col = grid[0].size();

    vector<vector<int> > distance(row, vector<int>(col, 0));
    vector<vector<int>> reach(row, vector<int>(col, 0));
    int building = 0, res = INT_MAX;

    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++) {
        // check from the building node, extend to all 0 node with distance
        if (1 == grid[i][j]) {
            ++building;
            int dist = 0;
            vector<vector<bool>> visited(row, vector<bool>(col, false));
            queue<pair<int, int>> curLevel, nextLevel;
            curLevel.emplace(i, j);
            // bfs search for each current building
            while (!curLevel.empty()) {
                ++dist;
                while (!curLevel.empty()) {
                    pair<int, int> cur = curLevel.front();
                    curLevel.pop();
                    int x = cur.first, y = cur.second;
                    ++reach[x][y];
                    vector<pair<int, int>> dirs = {{1, 0}, {-1, 0}, {0, 1}, {0,
-1}};
                    for (auto dir : dirs) {
                        int i = x + dir.first, j = y + dir.second;
                        if (i >= 0 && i < grid.size() && j >= 0 && j < grid[0].si
ze() && 0 == grid[i][j] && !visited[i][j])
                        {
                            distance[i][j] += dist;
                            nextLevel.emplace(i, j);
                            visited[i][j] = true;
                        }
                    }
                }
                swap(curLevel, nextLevel);
            }
        }
    }
for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
        if (0 == grid[i][j] && reach[i][j] == building) {
            res = min(res, distance[i][j]);
        }
    }
}
return res == INT_MAX ? -1 : res;
```

}

written by lchen77 original link here