2257. Count Unguarded Cells in the Grid

Approach 1: Iterative Simulation

Intuition

We want to determine which unoccupied cells are unguarded given a grid populated by guards, walls, and empty cells.

The vision of each guard is limited:

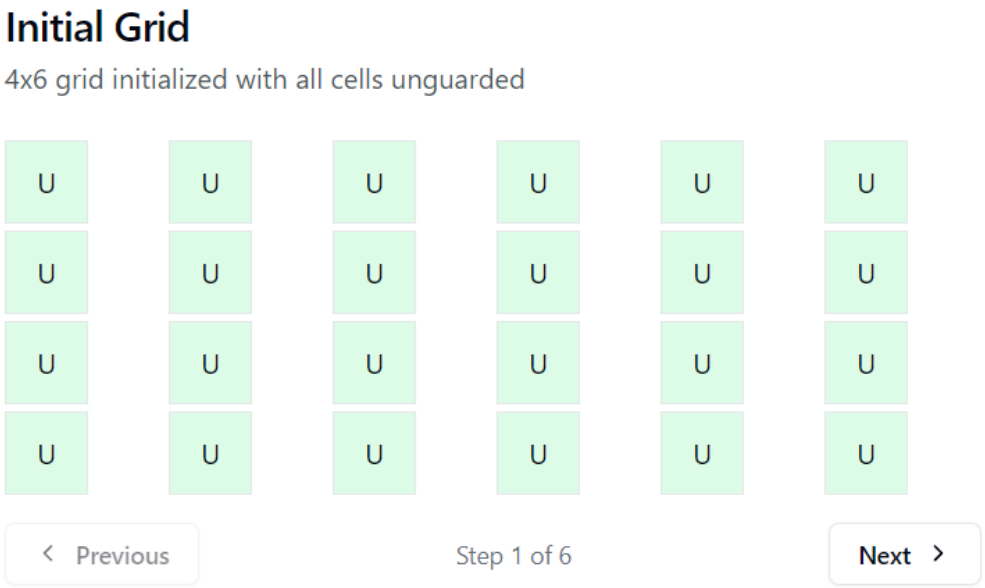
- They can see every cell in the four cardinal directions from their position: north, east, south, and west. In other words, they cannot see diagonally.
- They cannot see past walls.

Since we are given the locations of the guards and walls in the grid, the simplest approach will be to simulate each guard's range of vision. We can iterate through each direction from the guard's position until we either reach the grid's boundary, encounter another guard, or a wall, which blocks the guard's line of sight.

The key things to keep in mind are:

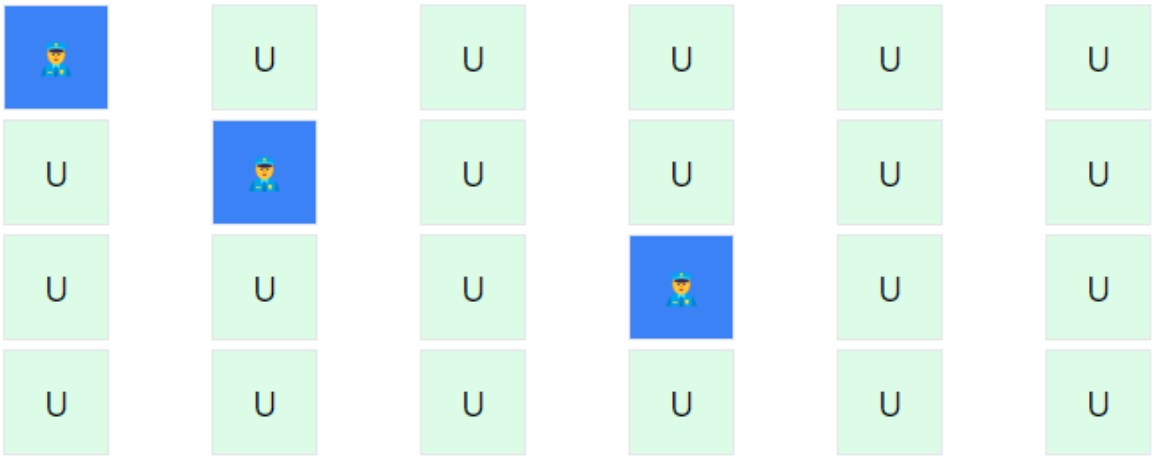
- Guards and walls occupy cells that cannot be guarded, so these should be distinctly marked.
- For each guard, visibility should be checked in all four directions until an obstruction or the grid’s edge is reached.
- Once all guarded cells are marked, any unmarked cells represent unguarded areas, which can then be counted to find the solution.

The following is a simulation of the approach, resulting in a final answer of 7 unguarded cells.



Place Guards

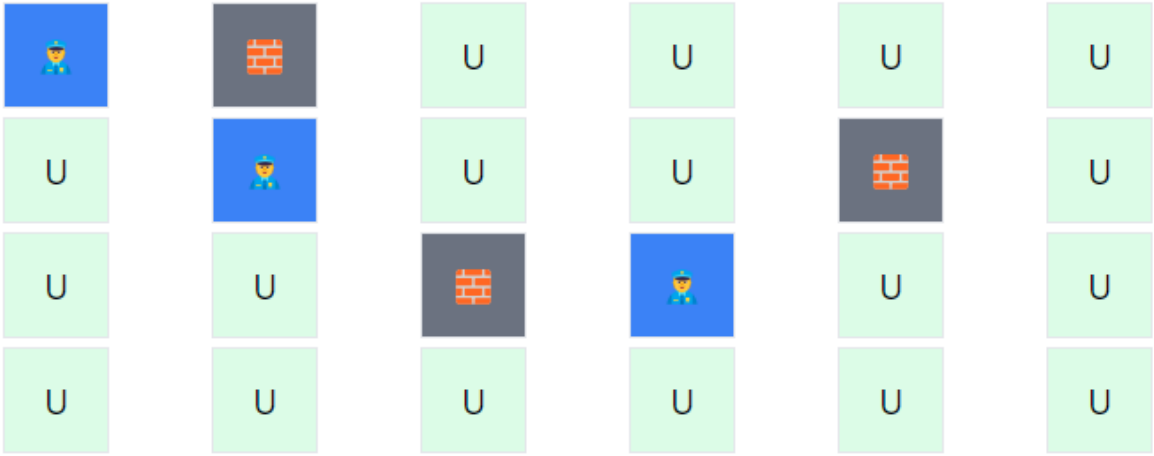
Mark guards' positions at [0,0], [1,1], and [2,3]



Previous Step 2 of 6 Next

Place Walls

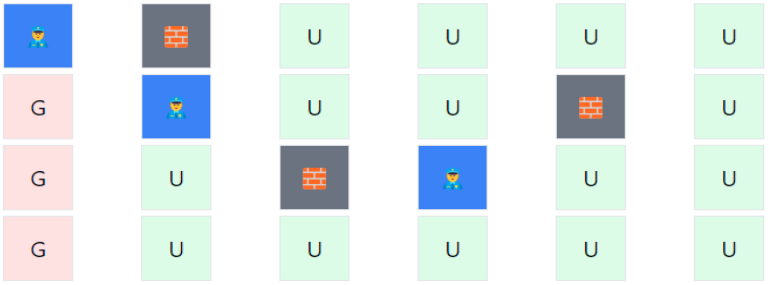
Mark walls at [0,1], [2,2], and [1,4]



Previous Step 3 of 6 Next

Process Guard at [0,0]

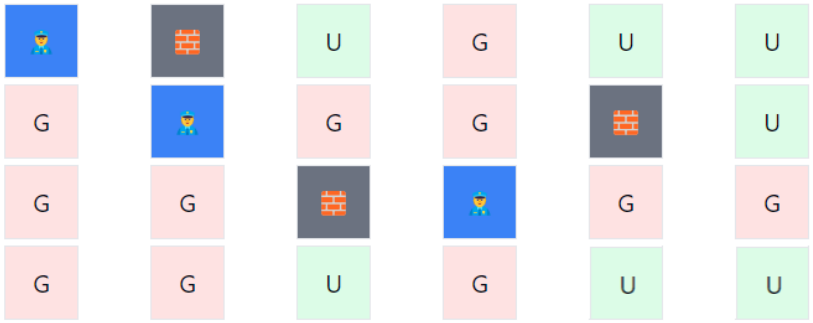
Mark cells guarded by first guard



Previous Step 4 of 6 Next

Process Guard at [2,3]

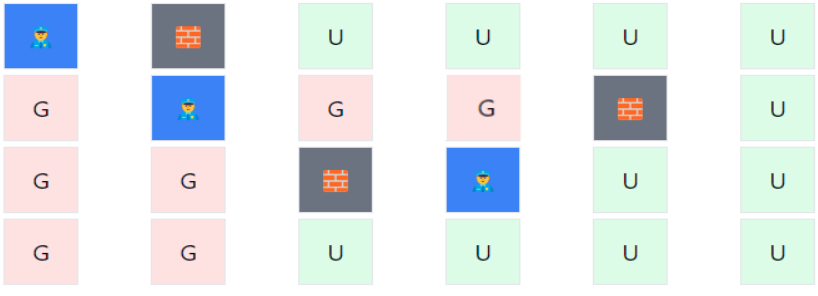
Mark cells guarded by third guard



Previous Step 6 of 6 Next

Process Guard at [1,1]

Mark cells guarded by second guard



Previous Step 5 of 6 Next

2257. Count Unguarded Cells in the Grid

Approach 2: Recursive Way

Intuition

We begin by marking the positions of the guards and walls in the grid, just like in the first approach. Then, for each guard, we trigger recursion in all four directions. Each recursive call will explore one direction as far as possible, marking all the reachable cells as "guarded." The exploration stops when it encounters a wall or another guard, and we repeat this process with other guards.

There is not much difference between Approach 1 and Approach 2 on a fundamental level, apart from their implementation, so this is meant to showcase a different implementation.

```
class Solution {
public:
    const int UNGUARDED = 0;
    const int GUARDED = 1;
    const int GUARD = 2;
    const int WALL = 3;

    void recurse(int row, int col, vector<vector<int>>& grid, char direction) {
        if (row < 0 || row >= grid.size() || col < 0 ||
            col >= grid[row].size() || grid[row][col] == GUARD ||
            grid[row][col] == WALL) {
            return;
        }
        grid[row][col] = GUARDED;  // Mark cell as guarded
        if (direction == 'U') recurse(row - 1, col, grid, 'U');  // Up
        if (direction == 'D') recurse(row + 1, col, grid, 'D');  // Down
        if (direction == 'L') recurse(row, col - 1, grid, 'L');  // Left
        if (direction == 'R') recurse(row, col + 1, grid, 'R');  // Right
    }
```

```
int countUnguarded(int m, int n, vector<vector<int>>& guards,
                   vector<vector<int>>& walls) {
        vector<vector<int>> grid(m, vector<int>(n, UNGUARDED));

        // Mark guards' positions
        for (const auto& guard : guards) {
            grid[guard[0]][guard[1]] = GUARD;
        }

        // Mark walls' positions
        for (const auto& wall : walls) {
            grid[wall[0]][wall[1]] = WALL;
        }

        // Mark cells as guarded by traversing from each guard
        for (const auto& guard : guards) {
            recurse(guard[0] - 1, guard[1], grid, 'U');  // Up
            recurse(guard[0] + 1, guard[1], grid, 'D');  // Down
            recurse(guard[0], guard[1] - 1, grid, 'L');  // Left
            recurse(guard[0], guard[1] + 1, grid, 'R');  // Right
        }

        // Count unguarded cells
        int count = 0;
        for (const auto& row : grid) {
            for (const auto& cell : row) {
                if (cell == UNGUARDED) count++;
            }
        }
        return count;
    }
};
```