489. Robot Room Cleaner

Hard 1869 109 Add to List Share

You are controlling a robot that is located somewhere in a room. The room is modeled as an `m x n` binary grid where `0` represents a wall and `1` represents an empty slot.

The robot starts at an unknown location in the root that is guaranteed to be empty, and you do not have access to the grid, but you can move the robot using the given API `Robot`.

You are tasked to use the robot to clean the entire room (i.e., clean every empty cell in the room). The robot with the four given APIs can move forward, turn left, or turn right. Each turn is `90` degrees.

When the robot tries to move into a wall cell, its bumper sensor detects the obstacle, and it stays on the current cell.

Design an algorithm to clean the entire room using the following APIs:

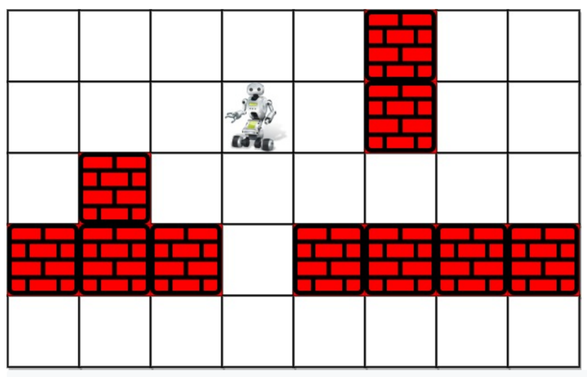
```
interface Robot {
  // returns true if next cell is open and robot moves into the cell.
  // returns false if next cell is obstacle and robot stays on the current cell.
  boolean move();

  // Robot will stay on the same cell after calling turnLeft/turnRight.
  // Each turn will be 90 degrees.
  void turnLeft();
  void turnRight();

  // Clean the current cell.
  void clean();
}
```

Note that the initial direction of the robot will be facing up. You can assume all four edges of the grid are all surrounded by a wall.

Custom testing:

The input is only given to initialize the room and the robot's position internally. You must solve this problem "blindfolded". In other words, you must control the robot using only the four mentioned APIs without knowing the room layout and the initial robot's position.

Example 1:



```
Input: room = [[1,1,1,1,1,0,1,1],[1,1,1,1,1,0,1,1],[1,0,1,1,1,1,1,1],
[0,0,0,1,0,0,0,0],[1,1,1,1,1,1,1,1]], row = 1, col = 3
Output: Robot cleaned all rooms.
Explanation: All grids in the room are marked by either 0 or 1.
0 means the cell is blocked, while 1 means the cell is accessible.
The robot initially starts at the position of row=1, col=3.
From the top left corner, its position is one row below and three columns right.
```

Example 2:

```
Input: room = [[1]], row = 0, col = 0
Output: Robot cleaned all rooms.
```

Constraints:

- `m == room.length`
- `n == room[i].length`
- `1 <= m <= 100`
- `1 <= n <= 200`
- `room[i][j]` is either `0` or `1`.
- `0 <= row < m`
- `0 <= col < n`
- `room[row][col] == 1`
- All the empty cells can be visited from the starting position.

Accepted 98,946 | Submissions 132,384

Seen this question in a real interview before? Yes No

Companies

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

Google | 8 Facebook | 7 Microsoft | 5 ByteDance | 2

Related Topics

Backtracking Interactive

Similar Questions

Walls and Gates Medium

Shortest Path in a Hidden Grid Medium

Minimum Path Cost in a Hidden Grid Medium

Number of Spaces Cleaning Robot Cleaned Medium

Java | Autocomplete

```java
class Solution {
  // going clockwise : 0: 'up', 1: 'right', 2: 'down', 3: 'left'
  int[][] directions = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};
  Set<Pair<Integer, Integer>> visited = new HashSet();
  Robot robot;

  public void goBack() {
    robot.turnRight();
    robot.turnRight();
    robot.move();
    robot.turnRight();
    robot.turnRight();
  }

  public void backtrack(int row, int col, int d) {
    visited.add(new Pair(row, col));
    robot.clean();
    // going clockwise : 0: 'up', 1: 'right', 2: 'down', 3: 'left'
    for (int i = 0; i < 4; ++i) {
      int newD = (d + i) % 4;
      int newRow = row + directions[newD][0];
      int newCol = col + directions[newD][1];

      if (!visited.contains(new Pair(newRow, newCol)) && robot.move()) {
        backtrack(newRow, newCol, newD);
        goBack();
      }
      // turn the robot following chosen direction : clockwise
      robot.turnRight();
    }
  }

  public void cleanRoom(Robot robot) {
    this.robot = robot;
    backtrack(0, 0, 0);
  }
}
```