Problem 489: Robot Room Cleaner

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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 room 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(); }

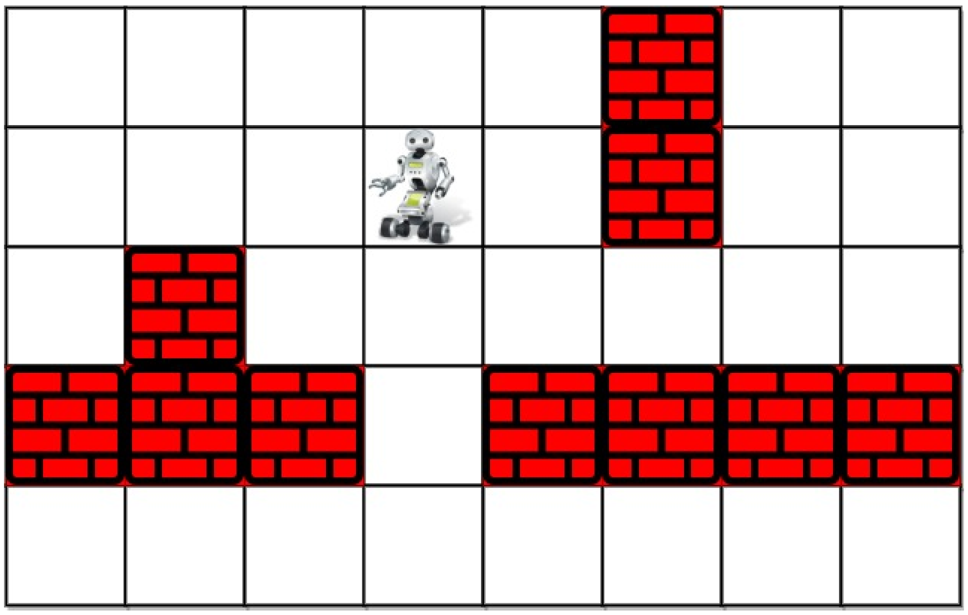
Note

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.

Code Snippets

C++:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
```

```
* public:
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* bool move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* void turnLeft();
* void turnRight();
*
* // Clean the current cell.
* void clean();
* };
*/

class Solution {
public:
void cleanRoom(Robot& robot) {

}
};
```

Java:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* interface Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public boolean move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public void turnLeft();
* public void turnRight();
*
* // Clean the current cell.
* public void clean();
* }
```

```
*/

class Solution {
public void cleanRoom(Robot robot) {

}
}
```

Python3:

```
# """
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# """
#class Robot:
# def move(self):
# """
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
# :rtype bool
# """
#
# def turnLeft(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def turnRight(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def clean(self):
# """
# Clean the current cell.
# :rtype void
# """
```

```
class Solution:
def cleanRoom(self, robot):
"""
:type robot: Robot
:rtype: None
"""
```

Python:

```
# """
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# """
#class Robot(object):
# def move(self):
# """
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
# :rtype bool
# """
#
# def turnLeft(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def turnRight(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def clean(self):
# """
# Clean the current cell.
# :rtype void
# """
```

```
class Solution(object):
def cleanRoom(self, robot):
"""
:type robot: Robot
:rtype: None
"""
```

JavaScript:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* function Robot() {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* @return {boolean}
* this.move = function() {
* ...
* };
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* @return {void}
* this.turnLeft = function() {
* ...
* };
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* @return {void}
* this.turnRight = function() {
* ...
* };
*
* // Clean the current cell.
* @return {void}
* this.clean = function() {
* ...
* };
* };
```

```
*/

/**
* @param {Robot} robot
* @return {void}
*/
var cleanRoom = function(robot) {

};
```

TypeScript:

```
/**
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* move(): boolean {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* turnRight() {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* turnLeft() {}
*
* // Clean the current cell.
* clean(): {}
* }
*/

function cleanRoom(robot: Robot) {

};
```

C#:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* interface Robot {
```

```
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public bool Move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public void TurnLeft();
* public void TurnRight();
*
* // Clean the current cell.
* public void Clean();
* }
*/

class Solution {
public void CleanRoom(Robot robot) {

}
}
```

Go:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* type Robot struct {
* }
*
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* func (robot *Robot) Move() bool {}
*
* // Robot will stay in the same cell after calling TurnLeft/TurnRight.
* // Each turn will be 90 degrees.
* func (robot *Robot) TurnLeft() {}
* func (robot *Robot) TurnRight() {}
*
* // Clean the current cell.
* func (robot *Robot) Clean() {}
*/
```

```
func cleanRoom(robot *Robot) {

}
```

Kotlin:

```
/**
* // This is the Robot's API interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* fun move(): Boolean {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* fun turnLeft() {}
* fun turnRight() {}
*
* // Clean the current cell.
* fun clean() {}
* }
*/

class Solution {
fun cleanRoom(robot: Robot) {

}
}
```

Swift:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* public class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public func move() -> Bool {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public func turnLeft() {}
* public func turnRight() {}
*
* // Clean the current cell.
* public func clean() {}
* }
*/

class Solution {
func cleanRoom(_ robot: Robot) {

}
}
```

Ruby:

```
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# class Robot
# def move():
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
# end
#
# def turnLeft():
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# end
#
# def turnRight():
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# end
#
# def clean():
# Clean the current cell.
# end
# end
# @param {Robot} robot
# @return {}
def cleanRoom(robot)

end
```

PHP:

```
/**
* // This is the Robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* function move() {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* function turnLeft() {}
* function turnRight() {}
*
* // Clean the current cell.
* function clean() {}
* }
*/

class Solution {
/**
* @param Robot $robot
* @return
*/
function cleanRoom($robot) {

}
}
```

Scala:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* def move(): Boolean = {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* def turnLeft(): Unit = {}
* def turnRight(): Unit = {}
*
* // Clean the current cell.
* def clean(): Unit = {}
* }
*/

object Solution {
def cleanRoom(robot: Robot): Unit = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Robot Room Cleaner
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* public:
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* bool move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* void turnLeft();
* void turnRight();
*
* // Clean the current cell.
* void clean();
* };
*/

class Solution {
public:
void cleanRoom(Robot& robot) {

}
};
```

Java Solution:

```
/**
* Problem: Robot Room Cleaner
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* interface Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public boolean move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public void turnLeft();
* public void turnRight();
*
* // Clean the current cell.
* public void clean();
* }
*/

class Solution {
public void cleanRoom(Robot robot) {

}
}
```

Python3 Solution:

```
# """
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# """
#class Robot:
# def move(self):
# """
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
# :rtype bool
# """
#
# def turnLeft(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
```

```
#
# def turnRight(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def clean(self):
# """
# Clean the current cell.
# :rtype void
# """

class Solution:
def cleanRoom(self, robot):
"""
:type robot: Robot
:rtype: None
"""
```

Python Solution:

```
# """
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# """
#class Robot(object):
# def move(self):
# """
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
# :rtype bool
# """
#
# def turnLeft(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def turnRight(self):
# """
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# :rtype void
# """
#
# def clean(self):
# """
# Clean the current cell.
# :rtype void
# """

class Solution(object):
def cleanRoom(self, robot):
"""
:type robot: Robot
:rtype: None
"""
```

JavaScript Solution:

```
/**
* Problem: Robot Room Cleaner
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* function Robot() {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* @return {boolean}
* this.move = function() {
* ...
* };
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* @return {void}
* this.turnLeft = function() {
* ...
* };
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* @return {void}
* this.turnRight = function() {
* ...
* };
*
* // Clean the current cell.
* @return {void}
* this.clean = function() {
* ...
* };
* };
*/

/**
* @param {Robot} robot
* @return {void}
*/
var cleanRoom = function(robot) {

};
```

TypeScript Solution:

```
/**
* Problem: Robot Room Cleaner
* Difficulty: Hard
* Tags: tree
```

```
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* move(): boolean {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* turnRight() {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* turnLeft() {}
*
* // Clean the current cell.
* clean(): {}
* }
*/

function cleanRoom(robot: Robot) {

};
```

C# Solution:

```
/*
* Problem: Robot Room Cleaner
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* interface Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public bool Move();
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public void TurnLeft();
* public void TurnRight();
*
* // Clean the current cell.
* public void Clean();
* }
*/

class Solution {
public void CleanRoom(Robot robot) {

}
}
```

Go Solution:

```
// Problem: Robot Room Cleaner
// Difficulty: Hard
// Tags: tree
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* type Robot struct {
* }
```

```
*
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* func (robot *Robot) Move() bool {}
*
* // Robot will stay in the same cell after calling TurnLeft/TurnRight.
* // Each turn will be 90 degrees.
* func (robot *Robot) TurnLeft() {}
* func (robot *Robot) TurnRight() {}
*
* // Clean the current cell.
* func (robot *Robot) Clean() {}
*/

func cleanRoom(robot *Robot) {

}
```

Kotlin Solution:

```
/**
* // This is the Robot's API interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* fun move(): Boolean {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* fun turnLeft() {}
* fun turnRight() {}
*
* // Clean the current cell.
* fun clean() {}
* }
*/

class Solution {
```

```
fun cleanRoom(robot: Robot) {

}
}
```

Swift Solution:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* public class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* public func move() -> Bool {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* public func turnLeft() {}
* public func turnRight() {}
*
* // Clean the current cell.
* public func clean() {}
* }
*/

class Solution {
func cleanRoom(_ robot: Robot) {

}
}
```

Ruby Solution:

```
# This is the robot's control interface.
# You should not implement it, or speculate about its implementation
# class Robot
# def move():
# Returns true if the cell in front is open and robot moves into the cell.
# Returns false if the cell in front is blocked and robot stays in the
current cell.
```

```
# end
#
# def turnLeft():
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# end
#
# def turnRight():
# Robot will stay in the same cell after calling turnLeft/turnRight.
# Each turn will be 90 degrees.
# end
#
# def clean():
# Clean the current cell.
# end
# end

# @param {Robot} robot
# @return {}
def cleanRoom(robot)

end
```

PHP Solution:

```
/**
* // This is the Robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* function move() {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* function turnLeft() {}
* function turnRight() {}
*
* // Clean the current cell.
* function clean() {}
* }
*/

class Solution {
/**
* @param Robot $robot
* @return
*/
function cleanRoom($robot) {

}
}
```

Scala Solution:

```
/**
* // This is the robot's control interface.
* // You should not implement it, or speculate about its implementation
* class Robot {
* // Returns true if the cell in front is open and robot moves into the cell.
* // Returns false if the cell in front is blocked and robot stays in the
current cell.
* def move(): Boolean = {}
*
* // Robot will stay in the same cell after calling turnLeft/turnRight.
* // Each turn will be 90 degrees.
* def turnLeft(): Unit = {}
* def turnRight(): Unit = {}
*
* // Clean the current cell.
* def clean(): Unit = {}
* }
*/

object Solution {
def cleanRoom(robot: Robot): Unit = {

}
}
```