Articles > 489. Robot Room Cleaner

Previous (/articles/maximal-rectangle/) Next (/articles/n-queens/)

489. Robot Room Cleaner (/problems/robot-room-cleaner/)

March 31, 2019 | 20.9K views

Average Rating: 4.31 (35 votes)

Given a robot cleaner in a room modeled as a grid.

Each cell in the grid can be empty or blocked.

The robot cleaner with 4 given APIs can move forward, turn left or turn right. Each turn it made is 90 degrees.

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

Design an algorithm to clean the entire room using only the 4 given APIs shown below.

```
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();
}
```

Example:

```
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

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.
```

Notes:

1. 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 mentioned 4 APIs, without knowing the room layout and the initial robot's position.
2. The robot's initial position will always be in an accessible cell.
3. The initial direction of the robot will be facing up.
4. All accessible cells are connected, which means the all cells marked as 1 will be accessible by the robot.
5. Assume all four edges of the grid are all surrounded by wall.

Solution

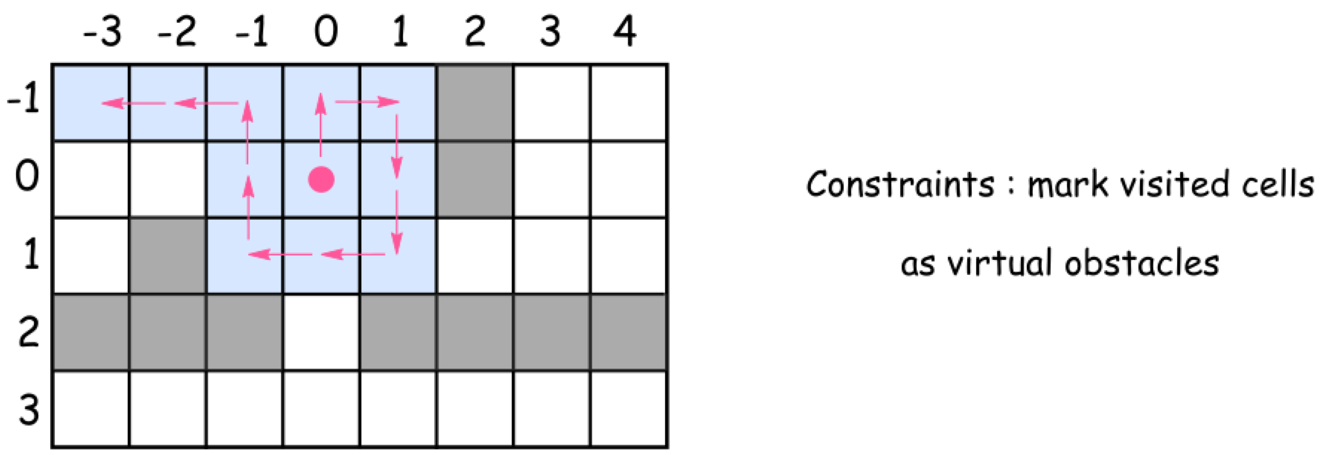
Approach 1: Spiral Backtracking

Concepts to use

Let's use here two programming concepts.

> The first one is called *constrained programming*.

That basically means to put restrictions after each robot move. Robot moves, and the cell is marked as `visited`. That propagates *constraints* and helps to reduce the number of combinations to consider.

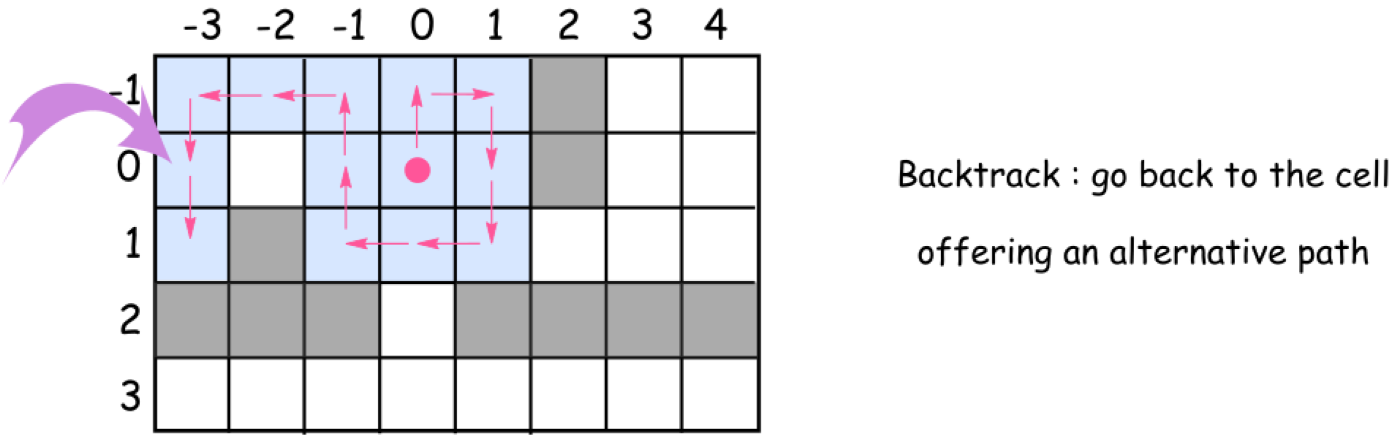


Constraints : mark visited cells

as virtual obstacles

> The second one called *backtracking*.

Let's imagine that after several moves the robot is surrounded by the visited cells. But several steps before there was a cell which proposed an alternative path to go. That path wasn't used and hence the room is not yet cleaned up. What to do? *To backtrack*. That means to come back to that cell, and to explore the alternative path.



Backtrack : go back to the cell

offering an alternative path

Intuition

This solution is based on the same idea as maze solving algorithm called right-hand rule (https://en.wikipedia.org/wiki/Maze_solving_algorithm#Wall_follower). Go forward, cleaning and marking all the cells on the way as visited. At the obstacle *turn right*, again go forward, *etc*. Always *turn right* at the obstacles and then go forward. Consider already visited cells as virtual obstacles.

> What do do if after the right turn there is an obstacle just in front ?

Turn right again.

> How to explore the alternative paths from the cell ?

Go back to that cell and then *turn right* from your last explored direction.

> When to stop ?

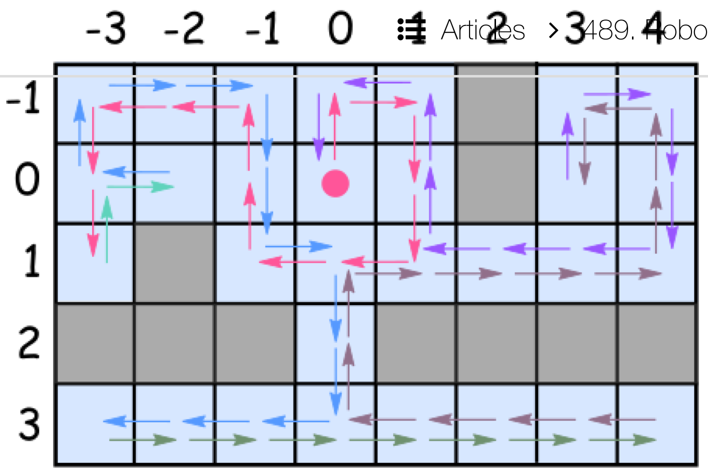
Stop when you explored all possible paths, *i.e.* all `4` directions (up, right, down, and left) for each visited cell.

Algorithm

Time to write down the algorithm for the backtrack function `backtrack(cell = (0, 0), direction = 0)`.

- Mark the cell as visited and clean it up.
- Explore `4` directions : `up`, `right`, `down`, and `left` (the order is important since the idea is always to turn right) :
    - Check the next cell in the chosen direction :
        - If it's not visited yet and there is no obtacles :
            - Move forward.
            - Explore next cells `backtrack(new_cell, new_direction)`.
            - Backtrack, *i.e.* go back to the previous cell.
        - Turn right because now there is an obstacle (or a virtual obstacle) just in front.

Implementation



Java Python Copy

```java
public class Pair<F, S> {
  public F first;
  public S second;

  public Pair(F first, S second) {
    this.first = first;
    this.second = second;
  }

  @Override
  public boolean equals(Object o) {
    Pair<F, S> p = (Pair<F, S>) o;
    return Objects.equals(p.first, first) && Objects.equals(p.second, second);
  }

  @Override
  public int hashCode() {
    return first.hashCode() ^ second.hashCode();
  }
}

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;

```

Complexity Analysis

- Time complexity : $\mathcal{O}(4^{N-M})$, where N is a number of cells in the room and M is a number of obstacles, because for each cell the algorithm checks 4 directions.

- Space complexity : $\mathcal{O}(N - M)$, where N is a number of cells in the room and M is a number of obstacles, to track visited cells.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/maximal-rectangle/)

Next (/articles/n-queens/)

Comments: 21

Sort By ▾

Type comment here... (Markdown is supported)

Preview | Post

park29 (park29) ★ 126 April 9, 2019 3:30 PM

This problem is so weird

44 ⌃ ⌄ Share Reply

sebcoe (sebcoe) ★ 19 April 21, 2019 4:51 PM

Are we sure about the time complexity? Assume there are no obstacles.

- If there is only one cell we would get 1 call to `backtrack`.
- If there are two cells we would get 2 calls to `backtrack`.
- If there are three cells we would get 3 calls to `backtrack`

Read More

11 ⌃ ⌄ Share Reply

SHOW 1 REPLY

abilityfun (abilityfun) ★ 25 April 6, 2019 8:10 PM

@andvary (https://leetcode.com/andvary), you are wrong on the time complexity. It is M*N. The branching factor is 4 but we maintain a visited set. Hence, you will not visit the same square twice and complexity is limited to M*N.

8 ⌃ ⌄ Share Reply

SHOW 3 REPLIES

calvinchankf (calvinchankf) ★ 1453 April 3, 2019 7:27 PM

i am confused, why we need `new_d = (d + i) % 4` ?

5 ⌃ ⌄ Share Reply

Articles › 489. Robot Room Cleaner ▾

SHOW 6 REPLIES

youjiahan (youjiahan) ★ 10 June 23, 2019 9:01 AM

Each node is visited only once why the time complexity O(4^N-M)

4 ⌃ ⌄ Share Reply

SHOW 1 REPLY

kai99 (kai99) ★ 268 July 2, 2019 11:29 AM

i love this problem

3 ⌃ ⌄ Share Reply

SHOW 1 REPLY

wangjian4814 (wangjian4814) ★ 31 August 25, 2019 12:16 PM

It is a REAL hard problem. I think the point is ask as to figure out a small project, not a problem. We should consider the cordinate at first. I failed...so I copy the answer.

2 ⌃ ⌄ Share Reply

SHOW 1 REPLY

jprakhar77 (jprakhar77) ★ 5 May 10, 2019 11:20 PM

It's called concepts, not conceptions, IMO.

2 ⌃ ⌄ Share Reply

SHOW 3 REPLIES

Sithis (sithis) ★ 6360 April 6, 2019 3:07 PM

I would avoid using classes from `javafx` package since it was removed from jdk 11 (https://www.infoworld.com/article/3305073/removed-from-jdk-11-javafx-11-arrives-as-a-standalone-module.html). It would be better to either create your own `Pair` class or just use a string.

2 ⌃ ⌄ Share Reply

SHOW 1 REPLY

je390 (je390) ★ 47 April 1, 2019 10:24 AM

thanks !

1 ⌃ ⌄ Share Reply

‹ 1 2 3 ›

☰ Articles › 489. Robot Room Cleaner ▾