LeetCode   Explore (Day 26)   Problems   Mock   Contest   Discuss   Store

Description   Solution   Discuss (256)   Submissions

Quick Navigation

Average Rating: 4.18 (89 votes)   Premium

Solution

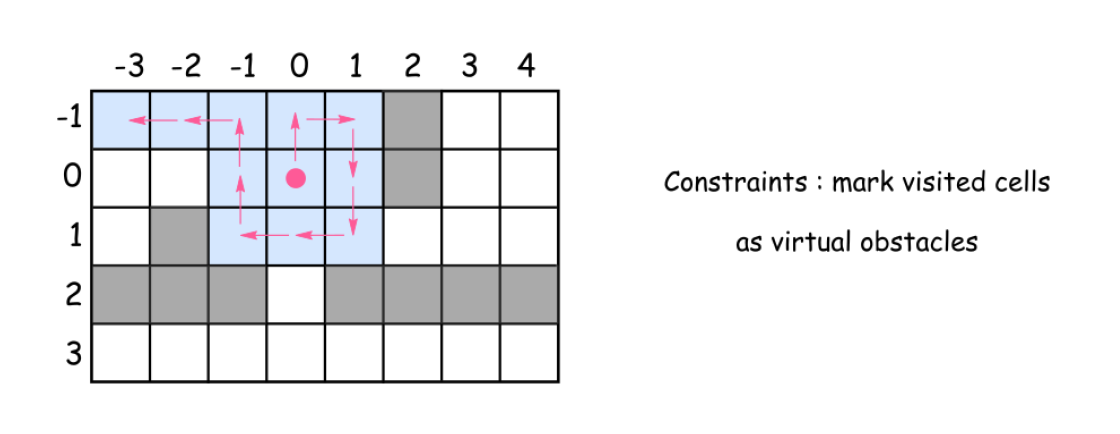
Approach 1: Spiral Backtracking

Concepts to use

Let's use here two programming concepts.

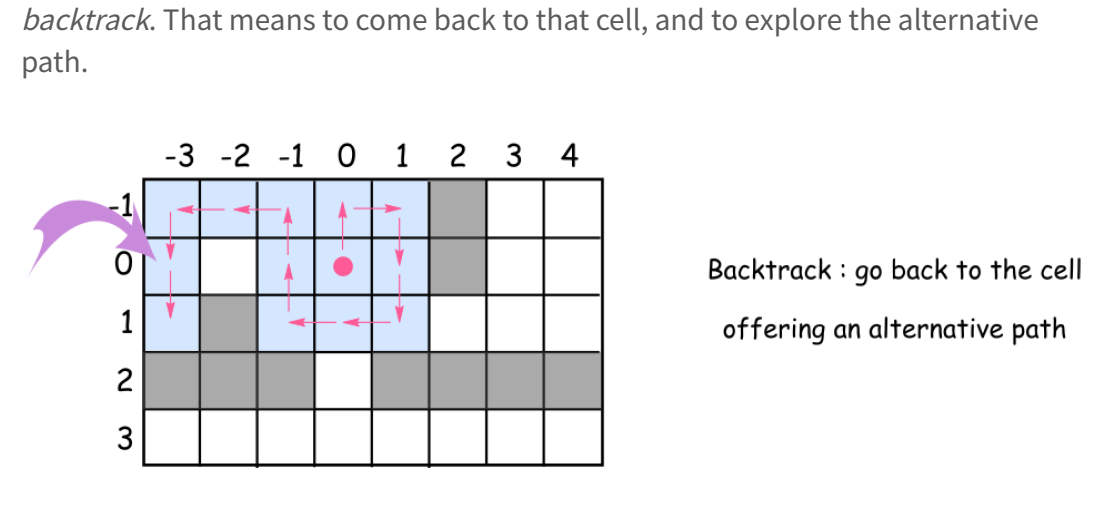
> The first one is called *constrained programming*.

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.



> 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.



Intuition

This solution is based on the same idea as maze solving algorithm called right-hand rule. 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 ?

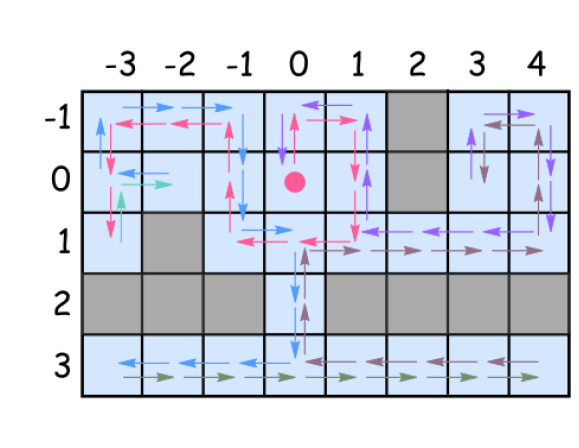
Stop when you explored all possible paths, *i.e.* all `4` directions (up, right, down, and left) for each visited cell.

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

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

        def backtrack(cell = (0, 0), d = 0):
            visited.add(cell)
            robot.clean()
            # going clockwise : 0: 'up', 1: 'right', 2: 'down', 3: 'left'
            for i in range(4):
                new_d = (d + i) % 4
                new_cell = (cell[0] + directions[new_d][0], \
                            cell[1] + directions[new_d][1])

                if not new_cell in visited and robot.move():
                    backtrack(new_cell, new_d)
                    go_back()
                # turn the robot following chosen direction :
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N - M)$, where N is a number of cells in the room and M is a number of obstacles.
  - We visit each non-obstacle cell once and only once.
  - At each visit, we will check 4 directions around the cell. Therefore, the total number of operations would be $4 \cdot (N - M)$.
- Space complexity : $\mathcal{O}(N - M)$, where N is a number of cells in the room and M is a number of obstacles.
  - We employed a hashtable to keep track of whether a non-obstacle cell is visited or not.

Comments: 69   Best   Most Votes   Newest

Type comment here... (Markdown is supported)

calvinchankf ★ 4279   April 4, 2019 10:27 AM

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

21   Show 13 replies   Reply

wangjian4814 ★ 166   August 26, 2019 3:16 AM

It is a REAL hard problem. I think the point is ask as to figure out a small proje should consider the cordinate at first. I failed...so I copy the answer.

34   Show 2 replies   Reply

zzznotsomuch ★ 296   February 14, 2020 4:36 AM

I could not understand the problem without reading the solution. Please impr description.

27   Show 3 replies   Reply

kai99 ★ 503   July 3, 2019 2:29 AM

i love this problem

28   Show 1 reply   Reply

youjiahan ★ 47   June 24, 2019 12:01 AM

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

9   Show 4 replies   Reply

hardfault ★ 462   January 7, 2020 2:47 PM

I think this question is not hard to implement. But hard to **understand** xD

7   Reply

XeQtR_Dev ★ 9   March 2, 2020 9:25 PM

The solution starts the cleaning process from (0,0) if my understanding is corr supposed to be starting from a given position?
Can someone please help me out here.

6   Show 1 reply   Reply

abilityfun ★ 35   April 7, 2019 11:10 AM

@andvary, you are wrong on the time complexity. It is M*N. The branching fact visited set. Hence, you will not visit the same square twice and complexity is l*

9   Show 3 replies   Reply

coderonin ★ 12   September 3, 2020 6:56 PM

There were a few things that were ambiguous in this question which I didn't k solution:

1. Where does the robot start? The question states that it can start anywhe coordinates passed as parameters to the cleanRoom() function. I see th as (0,0) but it is not obvious I am supposed to do that from reading the would I know if (0,0) is even a valid cell to start instead of an obstacle.
2. Speaking of being blind, it is not very clear how blind my robot should b keeping a "visited" mapping is considered cheating, since it will exist ou Only by looking at the solution can I know for sure that it is allowed.

3   Reply

jprakhar77 ★ 21   May 11, 2019 2:20 PM

It's called concepts, not conceptions, IMO.

3   Show 3 replies   Reply

< 1 2 3 4 5 6 7 >

Python3   Autocomplete

```
                robot.turnRight()
                robot.move()
                robot.turnRight()
                robot.turnRight()

        def backtrack(cell = (0, 0), d = 0):
            visited.add(cell)
            robot.clean()
            # going clockwise : 0: 'up', 1: 'right', 2: 'down', 3: 'left'
            for i in range(4):
                new_d = (d + i) % 4
                new_cell = (cell[0] + directions[new_d][0], \
                            cell[1] + directions[new_d][1])

                if not new_cell in visited and robot.move():
                    backtrack(new_cell, new_d)
                    go_back()
                # turn the robot following chosen direction : clockwise
                robot.turnRight()

        # going clockwise : 0: 'up', 1: 'right', 2: 'down', 3: 'left'
        directions = [(-1, 0), (0, 1), (1, 0), (0, -1)]
        visited = set()
        backtrack()


class Solution(object):
    def cleanRoom(self, robot):
        self.dfs(robot, 0, 0, 0, 1, set())
    def dfs(self, robot, x, y, direction_x, direction_y, visited):
        robot.clean()
        visited.add((x, y))
        for k in range(4):
            neighbor_x = x + direction_x
            neighbor_y = y + direction_y
            if (neighbor_x, neighbor_y) not in visited and robot.move():
                self.dfs(robot, neighbor_x, neighbor_y, direction_x, direction_y, visited)
                robot.turnRight()
                robot.turnRight()
                robot.move()
                robot.turnRight()
                robot.turnRight()
            robot.turnRight()
            direction_x, direction_y = direction_y, -direction_x #
```

Your previous code was restored from your local storag

Console   Contribute

Problems   Pick One   Prev   489/1739   Next   Run Code   Submit