题目地址(657. 机器人能否返回原点)

https://leetcode-cn.com/problems/robot-return-to-origin/

题目描述

在二维平面上，有一个机器人从原点 (0, 0) 开始。给出它的移动顺序，判断这个机器人在完成移动后是否在 (0, 0) 处结束。

移动顺序由字符串表示。字符 move[i] 表示其第 i 次移动。机器人的有效动作有 R（右），L（左），U（上）和 D（下）。如果机器人在完成所有动作后返回

注意：机器人“面朝”的方向无关紧要。 “R” 将始终使机器人向右移动一次，“L” 将始终向左移动等。此外，假设每次移动机器人的移动幅度相同。

示例 1:

输入: "UD"
输出: true
解释：机器人向上移动一次，然后向下移动一次。所有动作都具有相同的幅度，因此它最终回到它开始的原点。因此，我们返回 true。

示例 2:

输入: "LL"
输出: false
解释：机器人向左移动两次。它最终位于原点的左侧，距原点有两次 “移动” 的距离。我们返回 false，因为它在移动结束时没有返回原点。

前置知识

- 模拟

标签

- 模拟

难度

- 简单

入选理由

- 最简单的图题目，适合用来开头

公司

- 暂无

思路

题目比较直接，我们要做的仅仅是将题目描述翻译为代码即可。即根据题目给的**移动顺序**，模拟移动。模拟的过程中维护当前位置的坐标，并判断最终位置的坐标是否为 (0,0) 即可。

代码

- 语言支持：Python3，Java，JS，CPP

Python3 Code:

```
class Solution:
    def judgeCircle(self, moves: str) -> bool:
        x = y = 0
        for move in moves:
            if move == 'R': x += 1
            if move == 'L': x -= 1
            if move == 'U': y += 1
            if move == 'D': y -= 1

        return x == 0 and y == 0
```

Java Code:

```
class Solution {
    public boolean judgeCircle(String moves) {
        int x = 0, y = 0;
        for(char move: moves.toCharArray()) {
            if(move == 'R') {
                x++;
            } else if(move == 'L') {
                x--;
            } else if(move == 'U') {
                y++;
            } else if(move == 'D') {
                y--;
            }
```

```
        }
        return x == 0 && y == 0;
    }
}
```

JS Code:

```
var judgeCircle = function (moves) {
  let p = [0, 0];
  for (let i = 0; i < moves.length; i++) {
    if (moves[i] === "U") p[0] += 1;
    if (moves[i] === "D") p[0] -= 1;
    if (moves[i] === "L") p[1] -= 1;
    if (moves[i] === "R") p[1] += 1;
  }
  return p[0] === 0 && p[1] === 0;
};
```

CPP Code:

```
class Solution {
public:
    bool judgeCircle(string moves) {
        int x = 0, y =0;
        for (const char move : moves) {
            if (move == 'R') {
                x++;
            }
            if (move == 'L') {
                x--;
            }
            if (move == 'U') {
                y++;
            }
            if (move == 'D') {
                y--;
            }
        }
        return x==0 && y==0;
    }
};
```

复杂度分析

令 n 为字符串 moves 的长度。

- 时间复杂度：$O(n)$

- 空间复杂度：$O(1)$

> 此题解由 **力扣刷题插件** 自动生成。

力扣的小伙伴可以**关注我**，这样就会第一时间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

关注公众号力扣加加，努力用清晰直白的语言还原解题思路，并且有大量图解，手把手教你识别套路，高效刷题。

上一页 下一页