<https://leetcode.com/problems/robot-bounded-in-circle/description/>

On an infinite plane, a robot initially stands at (0, 0) and faces north. Note that:

The **north direction** is the positive direction of the y-axis.

The **south direction** is the negative direction of the y-axis.

The **east direction** is the positive direction of the x-axis.

The **west direction** is the negative direction of the x-axis.

The robot can receive one of three instructions:

"G": go straight 1 unit.

"L": turn 90 degrees to the left (i.e., anti-clockwise direction).

"R": turn 90 degrees to the right (i.e., clockwise direction).

The robot performs the instructions given in order, and repeats them forever.

Return true if and only if there exists a circle in the plane such that the robot never leaves the circle.

**Example 1:**

**Input:** instructions = "GGLLGG"

**Output:** true

**Explanation:** The robot is initially at (0, 0) facing the north direction.

"G": move one step. Position: (0, 1). Direction: North.

"G": move one step. Position: (0, 2). Direction: North.

"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: West.

"L": turn 90 degrees anti-clockwise. Position: (0, 2). Direction: South.

"G": move one step. Position: (0, 1). Direction: South.

"G": move one step. Position: (0, 0). Direction: South.

Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1) --> (0, 2) --> (0, 1) --> (0, 0).

Based on that, we return true.

**Example 2:**

**Input:** instructions = "GG"

**Output:** false

**Explanation:** The robot is initially at (0, 0) facing the north direction.

"G": move one step. Position: (0, 1). Direction: North.

"G": move one step. Position: (0, 2). Direction: North.

Repeating the instructions, keeps advancing in the north direction and does not go into cycles.

Based on that, we return false.

**Example 3:**

**Input:** instructions = "GL"

**Output:** true

**Explanation:** The robot is initially at (0, 0) facing the north direction.

"G": move one step. Position: (0, 1). Direction: North."L": turn 90 degrees anti-clockwise. Position: (0, 1). Direction: West.

"G": move one step. Position: (-1, 1). Direction: West."L": turn 90 degrees anti-clockwise. Position: (-1, 1). Direction: South.

"G": move one step. Position: (-1, 0). Direction: South."L": turn 90 degrees anti-clockwise. Position: (-1, 0). Direction: East.

"G": move one step. Position: (0, 0). Direction: East."L": turn 90 degrees anti-clockwise. Position: (0, 0). Direction: North.

Repeating the instructions, the robot goes into the cycle: (0, 0) --> (0, 1) --> (-1, 1) --> (-1, 0) --> (0, 0).

Based on that, we return true.

**Constraints:**

1 <= instructions.length <= 100

instructions[i] is 'G', 'L' or, 'R'.

**Attempt 1: 2025-06-25**

**Solution 1: Math (10 min)**

class Solution {

    public boolean isRobotBounded(String instructions) {

        // Initial position and direction

        int x = 0;

        int y = 0;

        // Sequence: North, East, South, West

        int[] dx = new int[] {0, 1, 0, -1};

        int[] dy = new int[] {1, 0, -1, 0};

        int dir = 0; // Start facing North

        for(int i = 0; i < instructions.length(); i++) {

            char c = instructions.charAt(i);

            if(c == 'G') {

                x += dx[dir];

                y += dy[dir];

            } else if(c == 'L') {

                // Turn left (counter-clockwise)

                dir = (dir + 3) % 4;

            } else {

                // Turn right (clockwise)

                dir = (dir + 1) % 4;

            }

        }

        // The robot is bounded in a circle if:

        // 1. It returns to the origin (0,0), or

        // 2. It's not facing North after the instructions

        return (x == 0 && y == 0 || dir != 0);

    }

}

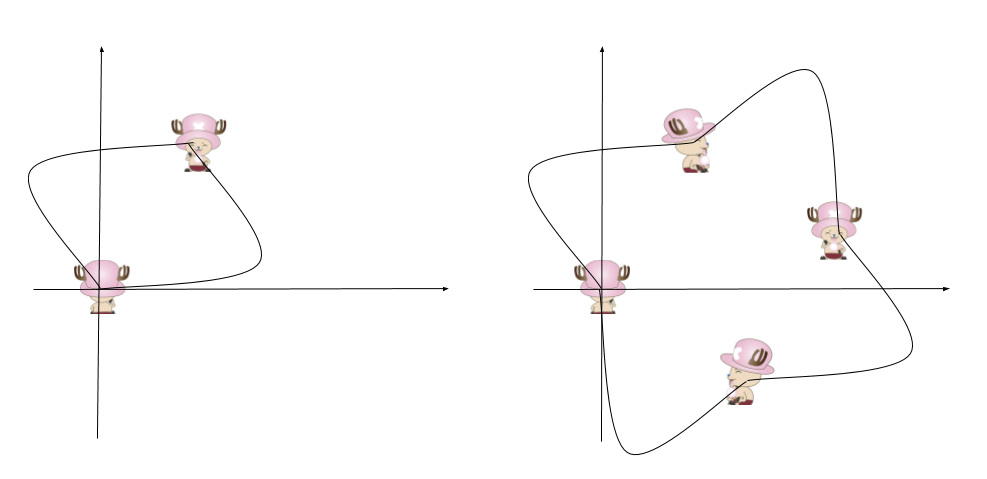
Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/robot-bounded-in-circle/solutions/290856/java-c-python-let-chopper-help-explain/>

I expect this problem to be medium problem.



**Intuition**

Let chopper help explain.

Starting at the origin and face north (0,1),

after one sequence of instructions,

if chopper return to the origin, he is obvious in an circle.

if chopper finishes with face not towards north,it will get back to the initial status in another one or three sequences.

**Explanation**

(x,y) is the location of chopper.

d[i] is the direction he is facing.

i = (i + 1) % 4 will turn right

i = (i + 3) % 4 will turn left

Check the final status after instructions.

**Complexity**

Time O(N)

Space O(1)

public boolean isRobotBounded(String ins) {

int x = 0, y = 0, i = 0, d[][] = {{0, 1}, {1, 0}, {0, -1}, { -1, 0}};

for (int j = 0; j < ins.length(); ++j)

if (ins.charAt(j) == 'R')

i = (i + 1) % 4;

else if (ins.charAt(j) == 'L')

i = (i + 3) % 4;

else {

x += d[i][0]; y += d[i][1];

}

return x == 0 && y == 0 || i > 0;

}

**Refer to Deepseek**

This problem requires determining if a robot's movement sequence will cause it to be bounded in a circle. Here's a Java solution that checks the robot's final position and direction after one cycle of instructions:

class Solution {

public boolean isRobotBounded(String instructions) {

// Initial position and direction

int x = 0, y = 0;

int[][] directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}}; // North, East, South, West

int dir = 0; // Start facing North

for (char c : instructions.toCharArray()) {

if (c == 'G') {

x += directions[dir][0];

y += directions[dir][1];

} else if (c == 'L') {

dir = (dir + 3) % 4; // Turn left (counter-clockwise)

} else if (c == 'R') {

dir = (dir + 1) % 4; // Turn right (clockwise)

}

}

// The robot is bounded in a circle if:

// 1. It returns to the origin (0,0), or

// 2. It's not facing North after the instructions

return (x == 0 && y == 0) || (dir != 0);

}

}

**Explanation:**

**Initialization**: We start at position (0, 0) facing North (direction index 0).

**Direction Handling**:

The directions array represents the four possible directions: North, East, South, West.

L (left) and R (right) commands change the direction index accordingly:

Left turn: (current + 3) % 4 (equivalent to -1 mod 4)

Right turn: (current + 1) % 4

**Movement Handling**:

G (go) command moves the robot one unit in its current direction by updating x and y coordinates.

**Bounded Check:**

**If after one cycle of instructions the robot returns to (0, 0), it will clearly stay in a circle.**

**If the robot isn't facing North after the cycle, it will eventually return to the origin after repeating the instructions 1-3 more times (due to the direction change causing a rotation).**

This solution runs in O(n) time where n is the length of the instructions string, with O(1) space complexity.

**Why Does Facing Non-North After One Cycle Imply a Bounded Circle?**

The key observation here is that **if the robot is not facing North after completing one cycle of instructions, repeating the same sequence of moves will cause it to rotate and eventually return to the origin**, forming a closed loop (circle).

**Breaking It Down:**

**Initial Direction (North) vs. Final Direction**

If the robot ends up facing **North (dir = 0)** after one cycle but is not at (0, 0), it will keep moving away from the origin indefinitely (unbounded).

If it ends up facing **any other direction (dir ≠ 0)**, repeating the same sequence will cause the robot to **rotate** rather than move away indefinitely.

**How Direction Changes Affect Movement Over Multiple Cycles**

After **1 full cycle**:

If dir = 0 (North): The robot moves in a straight line (unbounded unless already at origin).

If dir = 1 (East): After 4 cycles, it completes a full rotation (360°) and returns to North.

If dir = 2 (South): After 2 cycles, it faces North again.

If dir = 3 (West): After 4 cycles, it completes a full rotation.

**Why Does Non-North Direction Guarantee a Circle?**

If the robot is **not facing North**, repeating the same instructions will **change its net displacement**:

After **1 cycle**: Moves in some direction (e.g., East).

After **2 cycles**: Turns again (e.g., South).

After **3 or 4 cycles**: The turns cancel out the initial movement, bringing it back to the origin.

**Example**:

Suppose after one cycle, the robot is at (1, 1) facing **East (dir = 1)**.

After the **second cycle**, it turns **90° right again (now South)** and moves differently.

After the **third cycle**, it turns **right again (West)**.

After the **fourth cycle**, it completes a full rotation and returns to the origin.

**Mathematical Explanation**

The robot's movement can be modeled as a **vector sum**.

If the final direction is not North, the **cumulative effect of repeated cycles** is a **closed loop** (due to rotation).

If it were still facing North, the movement would just keep adding in the same direction (unbounded).

**Conclusion**

**(x == 0 && y == 0)**: The robot is already at the origin after one cycle (obviously bounded).

**(dir != 0)**: The robot is not facing North, meaning repeating the instructions will cause it to **rotate and eventually return to the origin**, keeping it bounded in a circle.

This is why the condition (x == 0 && y == 0) || (dir != 0) correctly identifies if the robot's path is bounded. 🚀