Problem 1041: Robot Bounded In Circle

Problem Information

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

Problem 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'

.

Code Snippets

C++:

```
class Solution {
public:
bool isRobotBounded(string instructions) {

}
};
```

Java:

```
class Solution {
public boolean isRobotBounded(String instructions) {

}
}
```

Python3:

```
class Solution:
def isRobotBounded(self, instructions: str) -> bool:
```

Python:

```
class Solution(object):
def isRobotBounded(self, instructions):
"""
:type instructions: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} instructions
* @return {boolean}
*/
var isRobotBounded = function(instructions) {

};
```

TypeScript:

```
function isRobotBounded(instructions: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsRobotBounded(string instructions) {

}
}
```

C:

```
bool isRobotBounded(char* instructions) {

}
```

Go:

```
func isRobotBounded(instructions string) bool {

}
```

Kotlin:

```
class Solution {
fun isRobotBounded(instructions: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isRobotBounded(_ instructions: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_robot_bounded(instructions: String) -> bool {

}
}
```

Ruby:

```
# @param {String} instructions
# @return {Boolean}
def is_robot_bounded(instructions)

end
```

PHP:

```
class Solution {

/**
* @param String $instructions
* @return Boolean
*/
function isRobotBounded($instructions) {

}
}
```

Dart:

```
class Solution {
bool isRobotBounded(String instructions) {

}
}
```

Scala:

```
object Solution {
def isRobotBounded(instructions: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_robot_bounded(instructions :: String.t) :: boolean
def is_robot_bounded(instructions) do

end
end
```

Erlang:

```
-spec is_robot_bounded(Instructions :: unicode:unicode_binary()) ->
boolean().
is_robot_bounded(Instructions) ->
.
```

Racket:

```
(define/contract (is-robot-bounded instructions)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool isRobotBounded(string instructions) {

}
};
```

Java Solution:

```
/**
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean isRobotBounded(String instructions) {

}
}
```

Python3 Solution:

```
"""
Problem: Robot Bounded In Circle
Difficulty: Medium
Tags: string, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def isRobotBounded(self, instructions: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isRobotBounded(self, instructions):
"""
:type instructions: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} instructions
* @return {boolean}
*/
var isRobotBounded = function(instructions) {

};
```

TypeScript Solution:

```
/**
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function isRobotBounded(instructions: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool IsRobotBounded(string instructions) {

}
}
```

C Solution:

```
/*
* Problem: Robot Bounded In Circle
* Difficulty: Medium
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool isRobotBounded(char* instructions) {

}
```

Go Solution:

```
// Problem: Robot Bounded In Circle
// Difficulty: Medium
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func isRobotBounded(instructions string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isRobotBounded(instructions: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isRobotBounded(_ instructions: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Robot Bounded In Circle
// Difficulty: Medium
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn is_robot_bounded(instructions: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} instructions
# @return {Boolean}
def is_robot_bounded(instructions)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $instructions
* @return Boolean
*/
function isRobotBounded($instructions) {

}
}
```

Dart Solution:

```
class Solution {
bool isRobotBounded(String instructions) {

}
}
```

Scala Solution:

```
object Solution {
def isRobotBounded(instructions: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_robot_bounded(instructions :: String.t) :: boolean
def is_robot_bounded(instructions) do

end
end
```

Erlang Solution:

```
-spec is_robot_bounded(Instructions :: unicode:unicode_binary()) ->
boolean().
is_robot_bounded(Instructions) ->
.
```

Racket Solution:

```
(define/contract (is-robot-bounded instructions)
(-> string? boolean?)
)
```