Problem 657: Robot Return to Origin

Problem Information

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

Problem Description

There is a robot starting at the position

(0, 0)

, the origin, on a 2D plane. Given a sequence of its moves, judge if this robot

ends up at

(0, 0)

after it completes its moves.

You are given a string

moves

that represents the move sequence of the robot where

moves[i]

represents its

i

th

move. Valid moves are

'R'

(right),

'L'

(left),

'U'

(up), and

'D'

(down).

Return

true

if the robot returns to the origin after it finishes all of its moves, or

false

otherwise

.

Note

: The way that the robot is "facing" is irrelevant.

'R'

will always make the robot move to the right once,

'L'

will always make it move left, etc. Also, assume that the magnitude of the robot's movement is the same for each move.

Example 1:

Input:

moves = "UD"

Output:

true

Explanation

: The robot moves up once, and then down once. All moves have the same magnitude, so it ended up at the origin where it started. Therefore, we return true.

Example 2:

Input:

moves = "LL"

Output:

false

Explanation

: The robot moves left twice. It ends up two "moves" to the left of the origin. We return false because it is not at the origin at the end of its moves.

Constraints:

1 <= moves.length <= 2 * 10

moves

only contains the characters

'U'

,

'D'

,

'L'

and

'R'

.

Code Snippets

C++:

```
class Solution {
public:
bool judgeCircle(string moves) {

}
};
```

Java:

```
class Solution {
public boolean judgeCircle(String moves) {

}
}
```

Python3:

```
class Solution:
def judgeCircle(self, moves: str) -> bool:
```

Python:

```
class Solution(object):
def judgeCircle(self, moves):
"""
:type moves: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} moves
* @return {boolean}
*/
var judgeCircle = function(moves) {

};
```

TypeScript:

```
function judgeCircle(moves: string): boolean {

};
```

C#:

```
public class Solution {
public bool JudgeCircle(string moves) {

}
}
```

C:

```
bool judgeCircle(char* moves) {

}
```

Go:

```
func judgeCircle(moves string) bool {

}
```

Kotlin:

```
class Solution {
fun judgeCircle(moves: String): Boolean {

}
}
```

Swift:

```
class Solution {
func judgeCircle(_ moves: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn judge_circle(moves: String) -> bool {

}
}
```

Ruby:

```
# @param {String} moves
# @return {Boolean}
def judge_circle(moves)

end
```

PHP:

```
class Solution {

/**
```

```
* @param String $moves
* @return Boolean
*/
function judgeCircle($moves) {

}
}
```

Dart:

```
class Solution {
bool judgeCircle(String moves) {

}
}
```

Scala:

```
object Solution {
def judgeCircle(moves: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec judge_circle(moves :: String.t) :: boolean
def judge_circle(moves) do

end
end
```

Erlang:

```
-spec judge_circle(Moves :: unicode:unicode_binary()) -> boolean().
judge_circle(Moves) ->
.
```

Racket:

```
(define/contract (judge-circle moves)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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 judgeCircle(string moves) {

}
};
```

Java Solution:

```
/**
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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 judgeCircle(String moves) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Robot Return to Origin
Difficulty: Easy
Tags: string

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 judgeCircle(self, moves: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def judgeCircle(self, moves):
"""
:type moves: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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} moves
* @return {boolean}
*/
var judgeCircle = function(moves) {

};
```

TypeScript Solution:

```
/**
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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 judgeCircle(moves: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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 JudgeCircle(string moves) {

}
}
```

C Solution:

```
/*
* Problem: Robot Return to Origin
* Difficulty: Easy
* Tags: string
*
* 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 judgeCircle(char* moves) {

}
```

Go Solution:

```
// Problem: Robot Return to Origin
// Difficulty: Easy
// Tags: string
//
// 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 judgeCircle(moves string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun judgeCircle(moves: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func judgeCircle(_ moves: String) -> Bool {
```

```
}
}
```

Rust Solution:

```
// Problem: Robot Return to Origin
// Difficulty: Easy
// Tags: string
//
// 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 judge_circle(moves: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} moves
# @return {Boolean}
def judge_circle(moves)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $moves
* @return Boolean
*/
function judgeCircle($moves) {

}
}
```

Dart Solution:

```
class Solution {
bool judgeCircle(String moves) {

}
}
```

Scala Solution:

```
object Solution {
def judgeCircle(moves: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec judge_circle(moves :: String.t) :: boolean
def judge_circle(moves) do

end
end
```

Erlang Solution:

```
-spec judge_circle(Moves :: unicode:unicode_binary()) -> boolean().
judge_circle(Moves) ->
.
```

Racket Solution:

```
(define/contract (judge-circle moves)
(-> string? boolean?)
)
```