Problem 2337: Move Pieces to Obtain a String

Problem Information

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

Problem Description

You are given two strings

start

and

target

, both of length

n

. Each string consists

only

of the characters

'L'

,

'R'

, and

'_'

where:

The characters

'L'

and

'R'

represent pieces, where a piece

'L'

can move to the

left

only if there is a

blank

space directly to its left, and a piece

'R'

can move to the

right

only if there is a

blank

space directly to its right.

The character

'_'

represents a blank space that can be occupied by

any

of the

'L'

or

'R'

pieces.

Return

true

if it is possible to obtain the string

target

by moving the pieces of the string

start

any

number of times

. Otherwise, return

false

.

Example 1:

Input:

start = "_L__R__R_", target = "L______RR"

Output:

true

Explanation:

We can obtain the string target from start by doing the following moves: - Move the first piece one step to the left, start becomes equal to "

L

___R__R_". - Move the last piece one step to the right, start becomes equal to "L___R___

R

". - Move the second piece three steps to the right, start becomes equal to "L______

R

R". Since it is possible to get the string target from start, we return true.

Example 2:

Input:

start = "R_L_", target = "__LR"

Output:

false

Explanation:

The 'R' piece in the string start can move one step to the right to obtain "_

R

L_". After that, no pieces can move anymore, so it is impossible to obtain the string target from start.

Example 3:

Input:

start = "_R", target = "R_"

Output:

false

Explanation:

The piece in the string start can move only to the right, so it is impossible to obtain the string target from start.

Constraints:

n == start.length == target.length

1 <= n <= 10

5

start

and

target

consist of the characters

'L'

,

'R'

, and

'_'

.

Code Snippets

C++:

```
class Solution {
public:
bool canChange(string start, string target) {

}
};
```

Java:

```
class Solution {
public boolean canChange(String start, String target) {

}
}
```

Python3:

```
class Solution:
def canChange(self, start: str, target: str) -> bool:
```

Python:

```
class Solution(object):
def canChange(self, start, target):
"""
:type start: str
```

```
:type target: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} start
* @param {string} target
* @return {boolean}
*/
var canChange = function(start, target) {

};
```

TypeScript:

```
function canChange(start: string, target: string): boolean {

};
```

C#:

```
public class Solution {
public bool CanChange(string start, string target) {

}
}
```

C:

```
bool canChange(char* start, char* target) {

}
```

Go:

```
func canChange(start string, target string) bool {

}
```

Kotlin:

```
class Solution {
fun canChange(start: String, target: String): Boolean {

}
}
```

Swift:

```
class Solution {
func canChange(_ start: String, _ target: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_change(start: String, target: String) -> bool {

}
}
```

Ruby:

```
# @param {String} start
# @param {String} target
# @return {Boolean}
def can_change(start, target)

end
```

PHP:

```
class Solution {

/**
* @param String $start
* @param String $target
* @return Boolean
*/
function canChange($start, $target) {

}
```

```
}
```

Dart:

```
class Solution {
bool canChange(String start, String target) {

}
}
```

Scala:

```
object Solution {
def canChange(start: String, target: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_change(start :: String.t, target :: String.t) :: boolean
def can_change(start, target) do

end
end
```

Erlang:

```
-spec can_change(Start :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> boolean().
can_change(Start, Target) ->
.
```

Racket:

```
(define/contract (can-change start target)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Move Pieces to Obtain a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool canChange(string start, string target) {

}
};
```

Java Solution:

```
/**
* Problem: Move Pieces to Obtain a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean canChange(String start, String target) {

}
}
```

Python3 Solution:

```
"""
Problem: Move Pieces to Obtain a String
```

```
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def canChange(self, start: str, target: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canChange(self, start, target):
"""
:type start: str
:type target: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Move Pieces to Obtain a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} start
* @param {string} target
* @return {boolean}
*/
var canChange = function(start, target) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Move Pieces to Obtain a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function canChange(start: string, target: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Move Pieces to Obtain a String
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CanChange(string start, string target) {

}
}
```

C Solution:

```
/*
* Problem: Move Pieces to Obtain a String
```

```
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool canChange(char* start, char* target) {

}
```

Go Solution:

```
// Problem: Move Pieces to Obtain a String
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func canChange(start string, target string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canChange(start: String, target: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canChange(_ start: String, _ target: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Move Pieces to Obtain a String
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn can_change(start: String, target: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} start
# @param {String} target
# @return {Boolean}
def can_change(start, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $start
* @param String $target
* @return Boolean
*/
function canChange($start, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool canChange(String start, String target) {

}
}
```

Scala Solution:

```
object Solution {
def canChange(start: String, target: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_change(start :: String.t, target :: String.t) :: boolean
def can_change(start, target) do

end
end
```

Erlang Solution:

```
-spec can_change(Start :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> boolean().
can_change(Start, Target) ->
.
```

Racket Solution:

```
(define/contract (can-change start target)
(-> string? string? boolean?)
)
```