Problem 777: Swap Adjacent in LR String

Problem Information

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

Problem Description

In a string composed of

'L'

,

'R'

, and

'X'

characters, like

"RXXLRXRXL"

, a move consists of either replacing one occurrence of

"XL"

with

"LX"

, or replacing one occurrence of

"RX"

with

"XR"

. Given the starting string

start

and the ending string

result

, return

True

if and only if there exists a sequence of moves to transform

start

to

result

.

Example 1:

Input:

start = "RXXLRXRXL", result = "XRLXXRRLX"

Output:

true

Explanation:

We can transform start to result following these steps: RXXLRXRXL -> XRXLRXRXL -> XRLXRXRXL -> XRLXXRRXL -> XRLXXRRLX

Example 2:

Input:

start = "X", result = "L"

Output:

false

Constraints:

1 <= start.length <= 10

4

start.length == result.length

Both

start

and

result

will only consist of characters in

'L'

,

'R'

, and

'X'

.

Code Snippets

C++:

```
class Solution {
public:
bool canTransform(string start, string result) {

}
};
```

Java:

```
class Solution {
public boolean canTransform(String start, String result) {

}
}
```

Python3:

```
class Solution:
def canTransform(self, start: str, result: str) -> bool:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function canTransform(start: string, result: string): boolean {

};
```

C#:

```
public class Solution {
public bool CanTransform(string start, string result) {

}
}
```

C:

```
bool canTransform(char* start, char* result) {

}
```

Go:

```
func canTransform(start string, result string) bool {

}
```

Kotlin:

```
class Solution {
fun canTransform(start: String, result: String): Boolean {

}
}
```

Swift:

```
class Solution {
func canTransform(_ start: String, _ result: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_transform(start: String, result: String) -> bool {

}
}
```

Ruby:

```
# @param {String} start
# @param {String} result
# @return {Boolean}
def can_transform(start, result)

end
```

PHP:

```
class Solution {

/**
* @param String $start
* @param String $result
* @return Boolean
*/
function canTransform($start, $result) {

}
}
```

Dart:

```
class Solution {
bool canTransform(String start, String result) {
```

```
}
}
```

Scala:

```
object Solution {
def canTransform(start: String, result: String): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec can_transform(Start :: unicode:unicode_binary(), Result ::
unicode:unicode_binary()) -> boolean().
can_transform(Start, Result) ->
.
```

Racket:

```
(define/contract (can-transform start result)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Swap Adjacent in LR 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 canTransform(string start, string result) {

}
};
```

Java Solution:

```
/**
* Problem: Swap Adjacent in LR 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 canTransform(String start, String result) {

}
}
```

Python3 Solution:

```
"""
Problem: Swap Adjacent in LR 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 canTransform(self, start: str, result: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Swap Adjacent in LR 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} result
* @return {boolean}
*/
var canTransform = function(start, result) {

};
```

TypeScript Solution:

```
/**
* Problem: Swap Adjacent in LR 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 canTransform(start: string, result: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Swap Adjacent in LR 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 CanTransform(string start, string result) {

}
}
```

C Solution:

```
/*
* Problem: Swap Adjacent in LR 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 canTransform(char* start, char* result) {

}
```

Go Solution:

```
// Problem: Swap Adjacent in LR 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 canTransform(start string, result string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canTransform(start: String, result: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canTransform(_ start: String, _ result: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Swap Adjacent in LR 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_transform(start: String, result: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} start
# @param {String} result
# @return {Boolean}
def can_transform(start, result)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $start
* @param String $result
* @return Boolean
*/
function canTransform($start, $result) {

}
}
```

Dart Solution:

```
class Solution {
bool canTransform(String start, String result) {

}
}
```

Scala Solution:

```
object Solution {
def canTransform(start: String, result: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec can_transform(Start :: unicode:unicode_binary(), Result ::
unicode:unicode_binary()) -> boolean().
can_transform(Start, Result) ->
.
```

Racket Solution:

```
(define/contract (can-transform start result)
(-> string? string? boolean?)
)
```