Problem 1585: Check If String Is Transformable With Substring Sort Operations

Problem Information

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

Problem Description

Given two strings

s

and

t

, transform string

s

into string

t

using the following operation any number of times:

Choose a

non-empty

substring in

s

and sort it in place so the characters are in

ascending order

.

For example, applying the operation on the underlined substring in

"1

4234

"

results in

"1

2344

"

.

Return

true

if

it is possible to transform

s

into

t

. Otherwise, return

false

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "84532", t = "34852"

Output:

true

Explanation:

You can transform s into t using the following sort operations: "84

53

2" (from index 2 to 3) -> "84

35

2" "

843

52" (from index 0 to 2) -> "

348

52"

Example 2:

Input:

s = "34521", t = "23415"

Output:

true

Explanation:

You can transform s into t using the following sort operations: "

3452

1" -> "

2345

1" "234

51

" -> "234

15

"

Example 3:

Input:

s = "12345", t = "12435"

Output:

false

Constraints:

s.length == t.length

1 <= s.length <= 10

5

s

and

t

consist of only digits.

Code Snippets

C++:

```
class Solution {
public:
bool isTransformable(string s, string t) {

}
};
```

Java:

```
class Solution {
public boolean isTransformable(String s, String t) {

}
}
```

Python3:

```
class Solution:
def isTransformable(self, s: str, t: str) -> bool:
```

Python:

```
class Solution(object):
def isTransformable(self, s, t):
"""
:type s: str
:type t: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} t
* @return {boolean}
*/
var isTransformable = function(s, t) {

};
```

TypeScript:

```
function isTransformable(s: string, t: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsTransformable(string s, string t) {

}
}
```

C:

```
bool isTransformable(char* s, char* t) {

}
```

Go:

```
func isTransformable(s string, t string) bool {

}
```

Kotlin:

```
class Solution {
fun isTransformable(s: String, t: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isTransformable(_ s: String, _ t: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_transformable(s: String, t: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def is_transformable(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function isTransformable($s, $t) {

}
}
```

Dart:

```
class Solution {
bool isTransformable(String s, String t) {

}
}
```

Scala:

```
object Solution {
def isTransformable(s: String, t: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_transformable(s :: String.t, t :: String.t) :: boolean
def is_transformable(s, t) do

end
end
```

Erlang:

```
-spec is_transformable(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
```

```
is_transformable(S, T) ->
.
```

Racket:

```
(define/contract (is-transformable s t)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
bool isTransformable(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

class Solution {
public boolean isTransformable(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If String Is Transformable With Substring Sort Operations
Difficulty: Hard
Tags: string, tree, greedy, sort

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def isTransformable(self, s: str, t: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isTransformable(self, s, t):
"""
:type s: str
:type t: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @param {string} t
* @return {boolean}
*/
var isTransformable = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function isTransformable(s: string, t: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

public class Solution {
public bool IsTransformable(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Check If String Is Transformable With Substring Sort Operations
* Difficulty: Hard
* Tags: string, tree, greedy, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

bool isTransformable(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Check If String Is Transformable With Substring Sort Operations
// Difficulty: Hard
// Tags: string, tree, greedy, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func isTransformable(s string, t string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isTransformable(s: String, t: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isTransformable(_ s: String, _ t: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check If String Is Transformable With Substring Sort Operations
// Difficulty: Hard
// Tags: string, tree, greedy, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn is_transformable(s: String, t: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Boolean}
def is_transformable(s, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Boolean
*/
function isTransformable($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
bool isTransformable(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def isTransformable(s: String, t: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_transformable(s :: String.t, t :: String.t) :: boolean
def is_transformable(s, t) do

end
end
```

Erlang Solution:

```
-spec is_transformable(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> boolean().
is_transformable(S, T) ->
```

```
.
```

Racket Solution:

```
(define/contract (is-transformable s t)
(-> string? string? boolean?)
)
```