Problem 972: Equal Rational Numbers

Problem Information

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

Problem Description

Given two strings

s

and

t

, each of which represents a non-negative rational number, return

true

if and only if they represent the same number. The strings may use parentheses to denote the repeating part of the rational number.

A

rational number

can be represented using up to three parts:

<IntegerPart>

,

<NonRepeatingPart>

, and a

<RepeatingPart>

. The number will be represented in one of the following three ways:

<IntegerPart>

For example,

12

,

0

, and

123

.

<IntegerPart>

<.>

<NonRepeatingPart>

For example,

0.5

,

1.

,

2.12

, and

123.0001

.

<IntegerPart>

<.>

<NonRepeatingPart>

<(>

<RepeatingPart>

<)>

For example,

0.1(6)

,

1.(9)

,

123.00(1212)

.

The repeating portion of a decimal expansion is conventionally denoted within a pair of round brackets. For example:

1/6 = 0.16666666... = 0.1(6) = 0.1666(6) = 0.166(66)

.

Example 1:

Input:

s = "0.(52)", t = "0.5(25)"

Output:

true

Explanation:

Because "0.(52)" represents 0.52525252..., and "0.5(25)" represents 0.52525252525..... , the strings represent the same number.

Example 2:

Input:

s = "0.1666(6)", t = "0.166(66)"

Output:

true

Example 3:

Input:

s = "0.9(9)", t = "1."

Output:

true

Explanation:

"0.9(9)" represents 0.999999999... repeated forever, which equals 1. [

See this link for an explanation.

] "1." represents the number 1, which is formed correctly: (IntegerPart) = "1" and (NonRepeatingPart) = "".

Constraints:

Each part consists only of digits.

The

<IntegerPart>

does not have leading zeros (except for the zero itself).

1 <= <IntegerPart>.length <= 4

0 <= <NonRepeatingPart>.length <= 4

1 <= <RepeatingPart>.length <= 4

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Equal Rational Numbers
* Difficulty: Hard
* 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 isRationalEqual(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Equal Rational Numbers
* Difficulty: Hard
* 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 isRationalEqual(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Equal Rational Numbers
Difficulty: Hard
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 isRationalEqual(self, s: str, t: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Equal Rational Numbers
* Difficulty: Hard
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Equal Rational Numbers
* Difficulty: Hard
* 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 isRationalEqual(s: string, t: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Equal Rational Numbers
* Difficulty: Hard
* 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 IsRationalEqual(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Equal Rational Numbers
* Difficulty: Hard
* 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 isRationalEqual(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Equal Rational Numbers
// Difficulty: Hard
// 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 isRationalEqual(s string, t string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Equal Rational Numbers
// Difficulty: Hard
// 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_rational_equal(s: String, t: String) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

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