Problem 2060: Check if an Original String Exists Given Two Encoded Strings

Problem Information

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

Problem Description

An original string, consisting of lowercase English letters, can be encoded by the following steps:

Arbitrarily

split

it into a

sequence

of some number of

non-empty

substrings.

Arbitrarily choose some elements (possibly none) of the sequence, and

replace

each with

its length

(as a numeric string).

Concatenate

the sequence as the encoded string.

For example,

one way

to encode an original string

"abcdefghijklmnop"

might be:

Split it as a sequence:

["ab", "cdefghijklmn", "o", "p"]

.

Choose the second and third elements to be replaced by their lengths, respectively. The sequence becomes

["ab", "12", "1", "p"]

.

Concatenate the elements of the sequence to get the encoded string:

"ab121p"

.

Given two encoded strings

s1

and

s2

, consisting of lowercase English letters and digits

1-9

(inclusive), return

true

if there exists an original string that could be encoded as

both

s1

and

s2

. Otherwise, return

false

.

Note

: The test cases are generated such that the number of consecutive digits in

s1

and

s2

does not exceed

3

.

Example 1:

Input:

s1 = "internationalization", s2 = "i18n"

Output:

true

Explanation:

It is possible that "internationalization" was the original string. - "internationalization" -> Split: ["internationalization"] -> Do not replace any element -> Concatenate: "internationalization", which is s1. - "internationalization" -> Split: ["i", "nternationalizatio", "n"] -> Replace: ["i", "18", "n"] -> Concatenate: "i18n", which is s2

Example 2:

Input:

s1 = "l123e", s2 = "44"

Output:

true

Explanation:

It is possible that "leetcode" was the original string. - "leetcode" -> Split: ["l", "e", "et", "cod", "e"] -> Replace: ["l", "1", "2", "3", "e"] -> Concatenate: "l123e", which is s1. - "leetcode" -> Split: ["leet", "code"] -> Replace: ["4", "4"] -> Concatenate: "44", which is s2.

Example 3:

Input:

s1 = "a5b", s2 = "c5b"

Output:

false

Explanation:

It is impossible. - The original string encoded as s1 must start with the letter 'a'. - The original string encoded as s2 must start with the letter 'c'.

Constraints:

1 <= s1.length, s2.length <= 40

s1

and

s2

consist of digits

1-9

(inclusive), and lowercase English letters only.

The number of consecutive digits in

s1

and

s2

does not exceed

3

.

Code Snippets

C++:

```
class Solution {
public:
bool possiblyEquals(string s1, string s2) {

}
};
```

Java:

```
class Solution {
public boolean possiblyEquals(String s1, String s2) {

}
}
```

Python3:

```
class Solution:
def possiblyEquals(self, s1: str, s2: str) -> bool:
```

Python:

```
class Solution(object):
def possiblyEquals(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {string} s2
* @return {boolean}
*/
var possiblyEquals = function(s1, s2) {

};
```

TypeScript:

```
function possiblyEquals(s1: string, s2: string): boolean {

};
```

C#:

```
public class Solution {
public bool PossiblyEquals(string s1, string s2) {

}
}
```

C:

```
bool possiblyEquals(char* s1, char* s2) {

}
```

Go:

```
func possiblyEquals(s1 string, s2 string) bool {

}
```

Kotlin:

```
class Solution {
fun possiblyEquals(s1: String, s2: String): Boolean {

}
}
```

Swift:

```
class Solution {
func possiblyEquals(_ s1: String, _ s2: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn possibly_equals(s1: String, s2: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @return {Boolean}
def possibly_equals(s1, s2)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return Boolean
*/
function possiblyEquals($s1, $s2) {

}
}
```

Dart:

```
class Solution {
bool possiblyEquals(String s1, String s2) {
```

```
}
}
```

Scala:

```
object Solution {
def possiblyEquals(s1: String, s2: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec possibly_equals(s1 :: String.t, s2 :: String.t) :: boolean
def possibly_equals(s1, s2) do

end
end
```

Erlang:

```
-spec possibly_equals(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> boolean().
possibly_equals(S1, S2) ->
.
```

Racket:

```
(define/contract (possibly-equals s1 s2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if an Original String Exists Given Two Encoded Strings
```

```
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool possiblyEquals(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: Check if an Original String Exists Given Two Encoded Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean possiblyEquals(String s1, String s2) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if an Original String Exists Given Two Encoded Strings
Difficulty: Hard
Tags: string, tree, dp

Approach: String manipulation with hash map or two pointers
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def possiblyEquals(self, s1: str, s2: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def possiblyEquals(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check if an Original String Exists Given Two Encoded Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s1
* @param {string} s2
* @return {boolean}
*/
var possiblyEquals = function(s1, s2) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if an Original String Exists Given Two Encoded Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function possiblyEquals(s1: string, s2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if an Original String Exists Given Two Encoded Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool PossiblyEquals(string s1, string s2) {

}
}
```

C Solution:

```
/*
* Problem: Check if an Original String Exists Given Two Encoded Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

bool possiblyEquals(char* s1, char* s2) {

}
```

Go Solution:

```
// Problem: Check if an Original String Exists Given Two Encoded Strings
// Difficulty: Hard
// Tags: string, tree, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func possiblyEquals(s1 string, s2 string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun possiblyEquals(s1: String, s2: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func possiblyEquals(_ s1: String, _ s2: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check if an Original String Exists Given Two Encoded Strings
// Difficulty: Hard
// Tags: string, tree, dp
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn possibly_equals(s1: String, s2: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @return {Boolean}
def possibly_equals(s1, s2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return Boolean
*/
function possiblyEquals($s1, $s2) {

}
}
```

Dart Solution:

```
class Solution {
bool possiblyEquals(String s1, String s2) {

}
}
```

Scala Solution:

```
object Solution {
def possiblyEquals(s1: String, s2: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec possibly_equals(s1 :: String.t, s2 :: String.t) :: boolean
def possibly_equals(s1, s2) do

end
end
```

Erlang Solution:

```
-spec possibly_equals(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> boolean().
possibly_equals(S1, S2) ->
.
```

Racket Solution:

```
(define/contract (possibly-equals s1 s2)
(-> string? string? boolean?)
)
```