Problem 1813: Sentence Similarity III

Problem Information

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

Problem Description

You are given two strings

sentence1

and

sentence2

, each representing a

sentence

composed of words. A sentence is a list of

words

that are separated by a

single

space with no leading or trailing spaces. Each word consists of only uppercase and lowercase English characters.

Two sentences

s1

and

s2

are considered

similar

if it is possible to insert an arbitrary sentence (

possibly empty

) inside one of these sentences such that the two sentences become equal.

Note

that the inserted sentence must be separated from existing words by spaces.

For example,

s1 = "Hello Jane"

and

s2 = "Hello my name is Jane"

can be made equal by inserting

"my name is"

between

"Hello"

and

"Jane"

in s1.

s1 = "Frog cool"

and

s2 = "Frogs are cool"

are

not

similar, since although there is a sentence

"s are"

inserted into

s1

, it is not separated from

"Frog"

by a space.

Given two sentences

sentence1

and

sentence2

, return

true

if

sentence1

and

sentence2

are

similar

. Otherwise, return

false

.

Example 1:

Input:

sentence1 = "My name is Haley", sentence2 = "My Haley"

Output:

true

Explanation:

sentence2

can be turned to

sentence1

by inserting "name is" between "My" and "Haley".

Example 2:

Input:

sentence1 = "of", sentence2 = "A lot of words"

Output:

false

Explanation:

No single sentence can be inserted inside one of the sentences to make it equal to the other.

Example 3:

Input:

sentence1 = "Eating right now", sentence2 = "Eating"

Output:

true

Explanation:

sentence2

can be turned to

sentence1

by inserting "right now" at the end of the sentence.

Constraints:

1 <= sentence1.length, sentence2.length <= 100

sentence1

and

sentence2

consist of lowercase and uppercase English letters and spaces.

The words in

sentence1

and

sentence2

are separated by a single space.

Code Snippets

C++:

```
class Solution {
public:
bool areSentencesSimilar(string sentence1, string sentence2) {

}
};
```

Java:

```
class Solution {
public boolean areSentencesSimilar(String sentence1, String sentence2) {

}
}
```

Python3:

```
class Solution:
def areSentencesSimilar(self, sentence1: str, sentence2: str) -> bool:
```

Python:

```
class Solution(object):
def areSentencesSimilar(self, sentence1, sentence2):
"""
:type sentence1: str
:type sentence2: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} sentence1
* @param {string} sentence2
* @return {boolean}
*/
var areSentencesSimilar = function(sentence1, sentence2) {

};
```

TypeScript:

```
function areSentencesSimilar(sentence1: string, sentence2: string): boolean {

};
```

C#:

```
public class Solution {
public bool AreSentencesSimilar(string sentence1, string sentence2) {

}
}
```

C:

```
bool areSentencesSimilar(char* sentence1, char* sentence2) {

}
```

Go:

```
func areSentencesSimilar(sentence1 string, sentence2 string) bool {
}
```

Kotlin:

```
class Solution {
fun areSentencesSimilar(sentence1: String, sentence2: String): Boolean {

}
}
```

Swift:

```
class Solution {
func areSentencesSimilar(_ sentence1: String, _ sentence2: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn are_sentences_similar(sentence1: String, sentence2: String) -> bool {

}
}
```

Ruby:

```
# @param {String} sentence1
# @param {String} sentence2
# @return {Boolean}
def are_sentences_similar(sentence1, sentence2)

end
```

PHP:

```
class Solution {

/**
* @param String $sentence1
* @param String $sentence2
* @return Boolean
*/
function areSentencesSimilar($sentence1, $sentence2) {

}
}
```

Dart:

```
class Solution {
bool areSentencesSimilar(String sentence1, String sentence2) {

}
}
```

Scala:

```
object Solution {
def areSentencesSimilar(sentence1: String, sentence2: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec are_sentences_similar(sentence1 :: String.t, sentence2 :: String.t) ::
boolean
def are_sentences_similar(sentence1, sentence2) do

end
end
```

Erlang:

```
-spec are_sentences_similar(Sentence1 :: unicode:unicode_binary(), Sentence2
:: unicode:unicode_binary()) -> boolean().
are_sentences_similar(Sentence1, Sentence2) ->
.
```

Racket:

```
(define/contract (are-sentences-similar sentence1 sentence2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sentence Similarity III
* 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 areSentencesSimilar(string sentence1, string sentence2) {

}
};
```

Java Solution:

```
/**
* Problem: Sentence Similarity III
* 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 areSentencesSimilar(String sentence1, String sentence2) {

}
}
```

Python3 Solution:

```
"""
Problem: Sentence Similarity III
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 areSentencesSimilar(self, sentence1: str, sentence2: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def areSentencesSimilar(self, sentence1, sentence2):
"""
:type sentence1: str
:type sentence2: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Sentence Similarity III
* 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} sentence1
* @param {string} sentence2
* @return {boolean}
*/
var areSentencesSimilar = function(sentence1, sentence2) {

};
```

TypeScript Solution:

```
/**
* Problem: Sentence Similarity III
* 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 areSentencesSimilar(sentence1: string, sentence2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Sentence Similarity III
* 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 AreSentencesSimilar(string sentence1, string sentence2) {

}
}
```

C Solution:

```
/*
* Problem: Sentence Similarity III
* 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 areSentencesSimilar(char* sentence1, char* sentence2) {

}
```

Go Solution:

```
// Problem: Sentence Similarity III
// 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 areSentencesSimilar(sentence1 string, sentence2 string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun areSentencesSimilar(sentence1: String, sentence2: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func areSentencesSimilar(_ sentence1: String, _ sentence2: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Sentence Similarity III
// 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 are_sentences_similar(sentence1: String, sentence2: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} sentence1
# @param {String} sentence2
# @return {Boolean}
def are_sentences_similar(sentence1, sentence2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $sentence1
* @param String $sentence2
* @return Boolean
*/
function areSentencesSimilar($sentence1, $sentence2) {

}
}
```

Dart Solution:

```
class Solution {
bool areSentencesSimilar(String sentence1, String sentence2) {

}
}
```

Scala Solution:

```
object Solution {
def areSentencesSimilar(sentence1: String, sentence2: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec are_sentences_similar(sentence1 :: String.t, sentence2 :: String.t) ::
boolean
def are_sentences_similar(sentence1, sentence2) do

end
end
```

Erlang Solution:

```
-spec are_sentences_similar(Sentence1 :: unicode:unicode_binary(), Sentence2
:: unicode:unicode_binary()) -> boolean().
are_sentences_similar(Sentence1, Sentence2) ->
.
```

Racket Solution:

```
(define/contract (are-sentences-similar sentence1 sentence2)
(-> string? string? boolean?)
)
```