Problem 734: Sentence Similarity

Problem Information

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

Problem Description

We can represent a sentence as an array of words, for example, the sentence

"I am happy with leetcode"

can be represented as

arr = ["I","am",happy","with","leetcode"]

.

Given two sentences

sentence1

and

sentence2

each represented as a string array and given an array of string pairs

similarPairs

where

similarPairs[i] = [x

i

, y

i

]

indicates that the two words

x

i

and

y

i

are similar.

Return

true

if

sentence1

and

sentence2

are similar, or

false

if they are not similar

.

Two sentences are similar if:

They have

the same length

(i.e., the same number of words)

sentence1[i]

and

sentence2[i]

are similar.

Notice that a word is always similar to itself, also notice that the similarity relation is not transitive. For example, if the words

a

and

b

are similar, and the words

b

and

c

are similar,

a

and

c

are

not necessarily similar

.

Example 1:

Input:

sentence1 = ["great","acting","skills"], sentence2 = ["fine","drama","talent"], similarPairs = [["great","fine"],["drama","acting"],["skills","talent"]]

Output:

true

Explanation:

The two sentences have the same length and each word i of sentence1 is also similar to the corresponding word in sentence2.

Example 2:

Input:

sentence1 = ["great"], sentence2 = ["great"], similarPairs = []

Output:

true

Explanation:

A word is similar to itself.

Example 3:

Input:

sentence1 = ["great"], sentence2 = ["doubleplus","good"], similarPairs = [["great","doubleplus"]]

Output:

false

Explanation:

As they don't have the same length, we return false.

Constraints:

1 <= sentence1.length, sentence2.length <= 1000

1 <= sentence1[i].length, sentence2[i].length <= 20

sentence1[i]

and

sentence2[i]

consist of English letters.

0 <= similarPairs.length <= 1000

similarPairs[i].length == 2

1 <= x

i

.length, y

i

.length <= 20

x

i

and

y

i

consist of lower-case and upper-case English letters.

All the pairs

(x

i

,

y

i

)

are

distinct

.

Code Snippets

C++:

```
class Solution {
public:
bool areSentencesSimilar(vector<string>& sentence1, vector<string>&
sentence2, vector<vector<string>>& similarPairs) {

}
};
```

Java:

```
class Solution {
public boolean areSentencesSimilar(String[] sentence1, String[] sentence2,
List<List<String>> similarPairs) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} sentence1
* @param {string[]} sentence2
* @param {string[][]} similarPairs
* @return {boolean}
```

```
*/
var areSentencesSimilar = function(sentence1, sentence2, similarPairs) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public bool AreSentencesSimilar(string[] sentence1, string[] sentence2, IList<IList<string>> similarPairs) {

}
}
```

C:

```
bool areSentencesSimilar(char** sentence1, int sentence1Size, char** sentence2, int sentence2Size, char*** similarPairs, int similarPairsSize, int* similarPairsColSize) {

}
```

Go:

```
func areSentencesSimilar(sentence1 []string, sentence2 []string, similarPairs [][]string) bool {

}
```

Kotlin:

```
class Solution {
fun areSentencesSimilar(sentence1: Array<String>, sentence2: Array<String>, similarPairs: List<List<String>>): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool areSentencesSimilar(List<String> sentence1, List<String> sentence2,
List<List<String>> similarPairs) {

}
}
```

Scala:

```
object Solution {
def areSentencesSimilar(sentence1: Array[String], sentence2: Array[String],
similarPairs: List[List[String]]): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool areSentencesSimilar(vector<string>& sentence1, vector<string>&
sentence2, vector<vector<string>>& similarPairs) {

}
};
```

Java Solution:

```
/**
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean areSentencesSimilar(String[] sentence1, String[] sentence2,
List<List<String>> similarPairs) {
}
}
```

Python3 Solution:

```
"""
Problem: Sentence Similarity
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def areSentencesSimilar(self, sentence1: List[str], sentence2: List[str],
similarPairs: List[List[str]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function areSentencesSimilar(sentence1: string[], sentence2: string[],
similarPairs: string[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public bool AreSentencesSimilar(string[] sentence1, string[] sentence2, IList<IList<string>> similarPairs) {

}
}
```

C Solution:

```
/*
* Problem: Sentence Similarity
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool areSentencesSimilar(char** sentence1, int sentence1Size, char** sentence2, int sentence2Size, char*** similarPairs, int similarPairsSize, int* similarPairsColSize) {

}
```

Go Solution:

```
// Problem: Sentence Similarity
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func areSentencesSimilar(sentence1 []string, sentence2 []string, similarPairs [][]string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun areSentencesSimilar(sentence1: Array<String>, sentence2: Array<String>,
similarPairs: List<List<String>>): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sentence Similarity
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

```


end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool areSentencesSimilar(List<String> sentence1, List<String> sentence2,
List<List<String>> similarPairs) {

}
}
```

Scala Solution:

```
object Solution {
def areSentencesSimilar(sentence1: Array[String], sentence2: Array[String],
similarPairs: List[List[String]]): Boolean = {

}
}
```

Elixir Solution:

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

```
def are_sentences_similar(sentence1, sentence2, similar_pairs) do

end
end
```

Erlang Solution:

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

Racket Solution:

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