Problem 884: Uncommon Words from Two Sentences

Problem Information

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

Problem Description

A

sentence

is a string of single-space separated words where each word consists only of lowercase letters.

A word is

uncommon

if it appears exactly once in one of the sentences, and

does not appear

in the other sentence.

Given two

sentences

s1

and

s2

, return

a list of all the

uncommon words

. You may return the answer in

any order

.

Example 1:

Input:

s1 = "this apple is sweet", s2 = "this apple is sour"

Output:

["sweet","sour"]

Explanation:

The word

"sweet"

appears only in

s1

, while the word

"sour"

appears only in

s2

.

Example 2:

Input:

s1 = "apple apple", s2 = "banana"

Output:

["banana"]

Constraints:

1 <= s1.length, s2.length <= 200

s1

and

s2

consist of lowercase English letters and spaces.

s1

and

s2

do not have leading or trailing spaces.

All the words in

s1

and

s2

are separated by a single space.

Code Snippets

C++:

```
class Solution {
public:
vector<string> uncommonFromSentences(string s1, string s2) {

}
};
```

Java:

```
class Solution {
public String[] uncommonFromSentences(String s1, String s2) {

}
}
```

Python3:

```
class Solution:
def uncommonFromSentences(self, s1: str, s2: str) -> List[str]:
```

Python:

```
class Solution(object):
def uncommonFromSentences(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {string} s2
* @return {string[]}
*/
var uncommonFromSentences = function(s1, s2) {

};
```

TypeScript:

```
function uncommonFromSentences(s1: string, s2: string): string[] {

};
```

C#:

```
public class Solution {
public string[] UncommonFromSentences(string s1, string s2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** uncommonFromSentences(char* s1, char* s2, int* returnSize) {

}
```

Go:

```
func uncommonFromSentences(s1 string, s2 string) []string {

}
```

Kotlin:

```
class Solution {
fun uncommonFromSentences(s1: String, s2: String): Array<String> {
}
}
```

Swift:

```
class Solution {
func uncommonFromSentences(_ s1: String, _ s2: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn uncommon_from_sentences(s1: String, s2: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @return {String[]}
def uncommon_from_sentences(s1, s2)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return String[]
*/
function uncommonFromSentences($s1, $s2) {

}
}
```

Dart:

```
class Solution {
List<String> uncommonFromSentences(String s1, String s2) {

}
}
```

Scala:

```
object Solution {
def uncommonFromSentences(s1: String, s2: String): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec uncommon_from_sentences(s1 :: String.t, s2 :: String.t) :: [String.t]
def uncommon_from_sentences(s1, s2) do

end
end
```

Erlang:

```
-spec uncommon_from_sentences(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
uncommon_from_sentences(S1, S2) ->
.
```

Racket:

```
(define/contract (uncommon-from-sentences s1 s2)
(-> string? string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<string> uncommonFromSentences(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String[] uncommonFromSentences(String s1, String s2) {

}
}
```

Python3 Solution:

```
"""
Problem: Uncommon Words from Two Sentences
Difficulty: Easy
Tags: string, hash
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def uncommonFromSentences(self, s1: str, s2: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def uncommonFromSentences(self, s1, s2):
"""
:type s1: str
:type s2: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s1
* @param {string} s2
* @return {string[]}
*/
var uncommonFromSentences = function(s1, s2) {

};
```

TypeScript Solution:

```
/**
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function uncommonFromSentences(s1: string, s2: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string[] UncommonFromSentences(string s1, string s2) {

}
}
```

C Solution:

```
/*
* Problem: Uncommon Words from Two Sentences
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** uncommonFromSentences(char* s1, char* s2, int* returnSize) {

}
```

Go Solution:

```
// Problem: Uncommon Words from Two Sentences
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func uncommonFromSentences(s1 string, s2 string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun uncommonFromSentences(s1: String, s2: String): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func uncommonFromSentences(_ s1: String, _ s2: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Uncommon Words from Two Sentences
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn uncommon_from_sentences(s1: String, s2: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @return {String[]}
def uncommon_from_sentences(s1, s2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return String[]
*/
function uncommonFromSentences($s1, $s2) {

}
}
```

Dart Solution:

```
class Solution {
List<String> uncommonFromSentences(String s1, String s2) {

}
}
```

Scala Solution:

```scala
object Solution {
def uncommonFromSentences(s1: String, s2: String): Array[String] = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec uncommon_from_sentences(s1 :: String.t, s2 :: String.t) :: [String.t]
def uncommon_from_sentences(s1, s2) do

end
end
```

Erlang Solution:

```erlang
-spec uncommon_from_sentences(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> [unicode:unicode_binary()].
uncommon_from_sentences(S1, S2) ->
.
```

Racket Solution:

```racket
(define/contract (uncommon-from-sentences s1 s2)
(-> string? string? (listof string?))
)
```