Problem 1433: Check If a String Can Break Another String

Problem Information

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

Problem Description

Given two strings:

s1

and

s2

with the same size, check if some permutation of string

s1

can break some permutation of string

s2

or vice-versa. In other words

s2

can break

s1

or vice-versa.

A string

x

can break string

y

(both of size

n

) if

x[i] >= y[i]

(in alphabetical order) for all

i

between

0

and

n-1

.

Example 1:

Input:

s1 = "abc", s2 = "xya"

Output:

true

Explanation:

"ayx" is a permutation of s2="xya" which can break to string "abc" which is a permutation of s1="abc".

Example 2:

Input:

s1 = "abe", s2 = "acd"

Output:

false

Explanation:

All permutations for s1="abe" are: "abe", "aeb", "bae", "bea", "eab" and "eba" and all permutation for s2="acd" are: "acd", "adc", "cad", "cda", "dac" and "dca". However, there is not any permutation from s1 which can break some permutation from s2 and vice-versa.

Example 3:

Input:

s1 = "leetcodee", s2 = "interview"

Output:

true

Constraints:

s1.length == n

s2.length == n

1 <= n <= 10^5

All strings consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-if-can-break s1 s2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If a String Can Break Another String
```

```
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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 checkIfCanBreak(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: Check If a String Can Break Another String
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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 checkIfCanBreak(String s1, String s2) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If a String Can Break Another String
Difficulty: Medium
Tags: string, greedy, sort

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 checkIfCanBreak(self, s1: str, s2: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check If a String Can Break Another String
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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} s1
* @param {string} s2
* @return {boolean}
*/
var checkIfCanBreak = function(s1, s2) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If a String Can Break Another String
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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 checkIfCanBreak(s1: string, s2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check If a String Can Break Another String
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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 CheckIfCanBreak(string s1, string s2) {

}
}
```

C Solution:

```
/*
* Problem: Check If a String Can Break Another String
* Difficulty: Medium
* Tags: string, greedy, sort
*
* 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 checkIfCanBreak(char* s1, char* s2) {

}
```

Go Solution:

```
// Problem: Check If a String Can Break Another String
// Difficulty: Medium
// Tags: string, greedy, sort
//
// 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 checkIfCanBreak(s1 string, s2 string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check If a String Can Break Another String
// Difficulty: Medium
// Tags: string, greedy, sort
```

```
//
// 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 check_if_can_break(s1: String, s2: String) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-if-can-break s1 s2)
(-> string? string? boolean?)
)
```