Problem 567: Permutation in String

Problem Information

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

Problem Description

Given two strings

s1

and

s2

, return

true

if

s2

contains a

permutation

of

s1

, or

false

otherwise.

In other words, return

true

if one of

s1

's permutations is the substring of

s2

.

Example 1:

Input:

s1 = "ab", s2 = "eidbaooo"

Output:

true

Explanation:

s2 contains one permutation of s1 ("ba").

Example 2:

Input:

s1 = "ab", s2 = "eidboaoo"

Output:

false

Constraints:

1 <= s1.length, s2.length <= 10

4

s1

and

s2

consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-inclusion s1 s2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Permutation in String
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

C# Solution:

```
/*
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}

}
```

C Solution:

```
/*
* Problem: Permutation in String
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
```

Go Solution:

```
// Problem: Permutation in String
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func checkInclusion(s1 string, s2 string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Permutation in String
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-inclusion s1 s2)
(-> string? string? boolean?)
)
```