Problem 3571: Find the Shortest Superstring II

Problem Information

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

Problem Description

You are given

two

strings,

s1

and

s2

. Return the

shortest

possible

string that contains both

s1

and

s2

as substrings. If there are multiple valid answers, return

any

one of them.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s1 = "aba", s2 = "bab"

Output:

"abab"

Explanation:

"abab"

is the shortest string that contains both

"aba"

and

"bab"

as substrings.

Example 2:

Input:

s1 = "aa", s2 = "aaa"

Output:

"aaa"

Explanation:

"aa"

is already contained within

"aaa"

, so the shortest superstring is

"aaa"

.

Constraints:

1 <= s1.length <= 100

1 <= s2.length <= 100

s1

and

s2

consist of lowercase English letters only.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String shortestSuperstring(String s1, String s2) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestSuperstring(s1: string, s2: string): string {

};
```

C#:

```
public class Solution {
public string ShortestSuperstring(string s1, string s2) {

}
}
```

C:

```
char* shortestSuperstring(char* s1, char* s2) {

}
```

Go:

```
func shortestSuperstring(s1 string, s2 string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String shortestSuperstring(String s1, String s2) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-superstring s1 s2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
```

```
string shortestSuperstring(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String shortestSuperstring(String s1, String s2) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Shortest Superstring II
Difficulty: Easy
Tags: string, tree

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* 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 {string}
*/
var shortestSuperstring = function(s1, s2) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function shortestSuperstring(s1: string, s2: string): string {
};
```

C# Solution:

```
/*
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string ShortestSuperstring(string s1, string s2) {

}
}
```

C Solution:

```
/*
* Problem: Find the Shortest Superstring II
* Difficulty: Easy
* Tags: string, tree
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* shortestSuperstring(char* s1, char* s2) {

}
```

Go Solution:

```
// Problem: Find the Shortest Superstring II
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func shortestSuperstring(s1 string, s2 string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Shortest Superstring II
// Difficulty: Easy
// Tags: string, tree
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String shortestSuperstring(String s1, String s2) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-superstring s1 s2)
(-> string? string? string?)
)
```