Problem 1092: Shortest Common Supersequence

Problem Information

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

Problem Description

Given two strings

str1

and

str2

, return

the shortest string that has both

str1

and

str2

as

subsequences

. If there are multiple valid strings, return

any

of them.

A string

s

is a

subsequence

of string

t

if deleting some number of characters from

t

(possibly

0

) results in the string

s

.

Example 1:

Input:

str1 = "abac", str2 = "cab"

Output:

"cabac"

Explanation:

str1 = "abac" is a subsequence of "cabac" because we can delete the first "c". str2 = "cab" is a subsequence of "cabac" because we can delete the last "ac". The answer provided is the shortest such string that satisfies these properties.

Example 2:

Input:

str1 = "aaaaaaaa", str2 = "aaaaaaaa"

Output:

"aaaaaaaa"

Constraints:

1 <= str1.length, str2.length <= 1000

str1

and

str2

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string shortestCommonSupersequence(string str1, string str2) {

}
};
```

Java:

```
class Solution {
public String shortestCommonSupersequence(String str1, String str2) {

}
}
```

Python3:

```
class Solution:
def shortestCommonSupersequence(self, str1: str, str2: str) -> str:
```

Python:

```
class Solution(object):
def shortestCommonSupersequence(self, str1, str2):
"""
:type str1: str
:type str2: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} str1
* @param {string} str2
* @return {string}
*/
var shortestCommonSupersequence = function(str1, str2) {

};
```

TypeScript:

```
function shortestCommonSupersequence(str1: string, str2: string): string {

};
```

C#:

}

```
public class Solution {
public string ShortestCommonSupersequence(string str1, string str2) {

}
}
```

C:

```
char* shortestCommonSupersequence(char* str1, char* str2) {

}
```

Go:

```
func shortestCommonSupersequence(str1 string, str2 string) string {

}
```

Kotlin:

```
class Solution {
fun shortestCommonSupersequence(str1: String, str2: String): String {

}
}
```

Swift:

```
class Solution {
func shortestCommonSupersequence(_ str1: String, _ str2: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn shortest_common_supersequence(str1: String, str2: String) -> String {

}
}
```

Ruby:

```
# @param {String} str1
# @param {String} str2
# @return {String}
def shortest_common_supersequence(str1, str2)

end
```

PHP:

```
class Solution {

/**
* @param String $str1
* @param String $str2
* @return String
*/
function shortestCommonSupersequence($str1, $str2) {

}
}
```

Dart:

```
class Solution {
String shortestCommonSupersequence(String str1, String str2) {

}
}
```

Scala:

```
object Solution {
def shortestCommonSupersequence(str1: String, str2: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_common_supersequence(str1 :: String.t, str2 :: String.t) ::
String.t
def shortest_common_supersequence(str1, str2) do
```

```



end
end
```

Erlang:

```
-spec shortest_common_supersequence(Str1 :: unicode:unicode_binary(), Str2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
shortest_common_supersequence(Str1, Str2) ->
.
```

Racket:

```
(define/contract (shortest-common-supersequence str1 str2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
string shortestCommonSupersequence(string str1, string str2) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public String shortestCommonSupersequence(String str1, String str2) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Common Supersequence
Difficulty: Hard
Tags: string, dp

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

class Solution:
def shortestCommonSupersequence(self, str1: str, str2: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shortestCommonSupersequence(self, str1, str2):
"""
:type str1: str
:type str2: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} str1
* @param {string} str2
* @return {string}
*/
var shortestCommonSupersequence = function(str1, str2) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function shortestCommonSupersequence(str1: string, str2: string): string {

};
```

C# Solution:

```
/*
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public string ShortestCommonSupersequence(string str1, string str2) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Common Supersequence
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

char* shortestCommonSupersequence(char* str1, char* str2) {

}
```

Go Solution:

```
// Problem: Shortest Common Supersequence
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func shortestCommonSupersequence(str1 string, str2 string) string {
}
```

Kotlin Solution:

```
class Solution {
fun shortestCommonSupersequence(str1: String, str2: String): String {

}
}
```

Swift Solution:

```
class Solution {
func shortestCommonSupersequence(_ str1: String, _ str2: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Shortest Common Supersequence
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn shortest_common_supersequence(str1: String, str2: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} str1
# @param {String} str2
# @return {String}
def shortest_common_supersequence(str1, str2)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $str1
* @param String $str2
* @return String
*/
function shortestCommonSupersequence($str1, $str2) {

}
}
```

Dart Solution:

```
class Solution {
String shortestCommonSupersequence(String str1, String str2) {

}
}
```

Scala Solution:

```
object Solution {
def shortestCommonSupersequence(str1: String, str2: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_common_supersequence(str1 :: String.t, str2 :: String.t) ::
String.t
def shortest_common_supersequence(str1, str2) do

end
end
```

Erlang Solution:

```
-spec shortest_common_supersequence(Str1 :: unicode:unicode_binary(), Str2 ::
unicode:unicode_binary()) -> unicode:unicode_binary().
shortest_common_supersequence(Str1, Str2) ->
.
```

Racket Solution:

```
(define/contract (shortest-common-supersequence str1 str2)
(-> string? string? string?)
)
```