Problem 3474: Lexicographically Smallest Generated String

Problem Information

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

Problem Description

You are given two strings,

str1

and

str2

, of lengths

n

and

m

, respectively.

A string

word

of length

n + m - 1

is defined to be

generated

by

str1

and

str2

if it satisfies the following conditions for

each

index

0 <= i <= n - 1

:

If

str1[i] == 'T'

, the

substring

of

word

with size

m

starting at index

i

is

equal

to

str2

, i.e.,

word[i..(i + m - 1)] == str2

.

If

str1[i] == 'F'

, the

substring

of

word

with size

m

starting at index

i

is

not equal

to

str2

, i.e.,

word[i..(i + m - 1)] != str2

.

Return the

lexicographically smallest

possible string that can be

generated

by

str1

and

str2

. If no string can be generated, return an empty string

""

.

Example 1:

Input:

str1 = "TFTF", str2 = "ab"

Output:

"ababa"

Explanation:

The table below represents the string

"ababa"

Index

T/F

Substring of length

m

0

'T'

"ab"

1

'F'

"ba"

2

'T'

"ab"

3

'F'

"ba"

The strings

"ababa"

and

"ababb"

can be generated by

str1

and

str2

.

Return

"ababa"

since it is the lexicographically smaller string.

Example 2:

Input:

str1 = "TFTF", str2 = "abc"

Output:

""

Explanation:

No string that satisfies the conditions can be generated.

Example 3:

Input:

str1 = "F", str2 = "d"

Output:

"a"

Constraints:

1 <= n == str1.length <= 10

4

1 <= m == str2.length <= 500

str1

consists only of

'T'

or

'F'

.

str2

consists only of lowercase English characters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (generate-string str1 str2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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 generateString(string str1, string str2) {

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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 generateString(String str1, String str2) {

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest Generated String
Difficulty: Hard
Tags: string, tree, graph, greedy

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 generateString(self, str1: str, str2: str) -> str:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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} str1
* @param {string} str2
* @return {string}
*/
var generateString = function(str1, str2) {

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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 generateString(str1: string, str2: string): string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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 GenerateString(string str1, string str2) {

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest Generated String
* Difficulty: Hard
* Tags: string, tree, graph, greedy
*
* 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* generateString(char* str1, char* str2) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest Generated String
// Difficulty: Hard
// Tags: string, tree, graph, greedy
//
// 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 generateString(str1 string, str2 string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest Generated String
// Difficulty: Hard
// Tags: string, tree, graph, greedy
//
// 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 generate_string(str1: String, str2: String) -> String {
```

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (generate-string str1 str2)
(-> string? string? string?)
)
```