Problem 2800: Shortest String That Contains Three Strings

Problem Information

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

Problem Description

Given three strings

a

,

b

, and

c

, your task is to find a string that has the

minimum

length and contains all three strings as

substrings

.

If there are multiple such strings, return the

lexicographically

smallest

one.

Return

a string denoting the answer to the problem.

Notes

A string

a

is

lexicographically smaller

than a string

b

(of the same length) if in the first position where

a

and

b

differ, string

a

has a letter that appears

earlier

in the alphabet than the corresponding letter in

b

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

a = "abc", b = "bca", c = "aaa"

Output:

"aaabca"

Explanation:

We show that "aaabca" contains all the given strings: a = ans[2...4], b = ans[3..5], c = ans[0..2]. It can be shown that the length of the resulting string would be at least 6 and "aaabca" is the lexicographically smallest one.

Example 2:

Input:

a = "ab", b = "ba", c = "aba"

Output:

"aba"

Explanation:

We show that the string "aba" contains all the given strings: a = ans[0..1], b = ans[1..2], c = ans[0..2]. Since the length of c is 3, the length of the resulting string would be at least 3. It can be shown that "aba" is the lexicographically smallest one.

Constraints:

1 <= a.length, b.length, c.length <= 100

a

,

b

,

c

consist only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string minimumString(string a, string b, string c) {

}
};
```

Java:

```
class Solution {
public String minimumString(String a, String b, String c) {

}
}
```

Python3:

```
class Solution:
def minimumString(self, a: str, b: str, c: str) -> str:
```

Python:

```
class Solution(object):
def minimumString(self, a, b, c):
"""
:type a: str
:type b: str
:type c: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} a
* @param {string} b
* @param {string} c
* @return {string}
*/
var minimumString = function(a, b, c) {

};
```

TypeScript:

```
function minimumString(a: string, b: string, c: string): string {

};
```

C#:

```
public class Solution {
public string MinimumString(string a, string b, string c) {

}
}
```

C:

```
char* minimumString(char* a, char* b, char* c) {

}
```

Go:

```
func minimumString(a string, b string, c string) string {

}
```

Kotlin:

```
class Solution {
fun minimumString(a: String, b: String, c: String): String {

}
}
```

Swift:

```
class Solution {
func minimumString(_ a: String, _ b: String, _ c: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_string(a: String, b: String, c: String) -> String {

}
}
```

Ruby:

```
# @param {String} a
# @param {String} b
# @param {String} c
# @return {String}
def minimum_string(a, b, c)

end
```

PHP:

```
class Solution {

/**
* @param String $a
* @param String $b
* @param String $c
* @return String
*/
function minimumString($a, $b, $c) {

}
}
```

Dart:

```
class Solution {
String minimumString(String a, String b, String c) {

}
}
```

Scala:

```
object Solution {
def minimumString(a: String, b: String, c: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_string(a :: String.t, b :: String.t, c :: String.t) :: String.t
def minimum_string(a, b, c) do

end
end
```

Erlang:

```
-spec minimum_string(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary(), C :: unicode:unicode_binary()) ->
unicode:unicode_binary().
minimum_string(A, B, C) ->
.
```

Racket:

```
(define/contract (minimum-string a b c)
(-> string? string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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 minimumString(string a, string b, string c) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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 minimumString(String a, String b, String c) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest String That Contains Three Strings
Difficulty: Medium
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 minimumString(self, a: str, b: str, c: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumString(self, a, b, c):
"""
:type a: str
:type b: str
:type c: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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} a
* @param {string} b
* @param {string} c
* @return {string}
*/
var minimumString = function(a, b, c) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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 minimumString(a: string, b: string, c: string): string {

};
```

C# Solution:

```
/*
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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 MinimumString(string a, string b, string c) {

}
}
```

C Solution:

```
/*
* Problem: Shortest String That Contains Three Strings
* Difficulty: Medium
* 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* minimumString(char* a, char* b, char* c) {

}
```

Go Solution:

```
// Problem: Shortest String That Contains Three Strings
// Difficulty: Medium
// 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 minimumString(a string, b string, c string) string {

}
```

Kotlin Solution:

```
class Solution {
fun minimumString(a: String, b: String, c: String): String {

}
}
```

Swift Solution:

```
class Solution {
func minimumString(_ a: String, _ b: String, _ c: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Shortest String That Contains Three Strings
// Difficulty: Medium
// 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 minimum_string(a: String, b: String, c: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} a
# @param {String} b
# @param {String} c
# @return {String}
def minimum_string(a, b, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $a
* @param String $b
* @param String $c
* @return String
*/
function minimumString($a, $b, $c) {

}
}
```

Dart Solution:

```
class Solution {
String minimumString(String a, String b, String c) {

}
}
```

Scala Solution:

```
object Solution {
def minimumString(a: String, b: String, c: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_string(a :: String.t, b :: String.t, c :: String.t) :: String.t
def minimum_string(a, b, c) do

end
end
```

Erlang Solution:

```
-spec minimum_string(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary(), C :: unicode:unicode_binary()) ->
unicode:unicode_binary().
minimum_string(A, B, C) ->
.
```

Racket Solution:

```
(define/contract (minimum-string a b c)
(-> string? string? string? string?)
)
```