Problem 1061: Lexicographically Smallest Equivalent String

Problem Information

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

Problem Description

You are given two strings of the same length

s1

and

s2

and a string

baseStr

.

We say

s1[i]

and

s2[i]

are equivalent characters.

For example, if

s1 = "abc"

and

s2 = "cde"

, then we have

'a' == 'c'

,

'b' == 'd'

, and

'c' == 'e'

.

Equivalent characters follow the usual rules of any equivalence relation:

Reflexivity:

'a' == 'a'

.

Symmetry:

'a' == 'b'

implies

'b' == 'a'

.

Transitivity:

'a' == 'b'

and

'b' == 'c'

implies

'a' == 'c'

.

For example, given the equivalency information from

s1 = "abc"

and

s2 = "cde"

,

"acd"

and

"aab"

are equivalent strings of

baseStr = "eed"

, and

"aab"

is the lexicographically smallest equivalent string of baseStr

.

Return

the lexicographically smallest equivalent string of

baseStr

by using the equivalency information from

s1

and

s2

.

Example 1:

Input:

s1 = "parker", s2 = "morris", baseStr = "parser"

Output:

"makkek"

Explanation:

Based on the equivalency information in s1 and s2, we can group their characters as [m,p], [a,o], [k,r,s], [e,i]. The characters in each group are equivalent and sorted in lexicographical order. So the answer is "makkek".

Example 2:

Input:

s1 = "hello", s2 = "world", baseStr = "hold"

Output:

"hdld"

Explanation:

Based on the equivalency information in s1 and s2, we can group their characters as [h,w], [d,e,o], [l,r]. So only the second letter 'o' in baseStr is changed to 'd', the answer is "hdld".

Example 3:

Input:

s1 = "leetcode", s2 = "programs", baseStr = "sourcecode"

Output:

"aauaaaaada"

Explanation:

We group the equivalent characters in s1 and s2 as [a,o,e,r,s,c], [l,p], [g,t] and [d,m], thus all letters in baseStr except 'u' and 'd' are transformed to 'a', the answer is "aauaaaaada".

Constraints:

1 <= s1.length, s2.length, baseStr <= 1000

s1.length == s2.length

s1

,

s2

, and

baseStr

consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String smallestEquivalentString(String s1, String s2, String baseStr)
{

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function smallestEquivalentString(s1: string, s2: string, baseStr: string):
string {

};
```

C#:

```
public class Solution {
public string SmallestEquivalentString(string s1, string s2, string baseStr)
{

}
}
```

C:

```
char* smallestEquivalentString(char* s1, char* s2, char* baseStr) {

}
```

Go:

```
func smallestEquivalentString(s1 string, s2 string, baseStr string) string {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String smallestEquivalentString(String s1, String s2, String baseStr) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (smallest-equivalent-string s1 s2 baseStr)
(-> string? string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string smallestEquivalentString(string s1, string s2, string baseStr) {

}
};
```

Java Solution:

```
/**
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String smallestEquivalentString(String s1, String s2, String baseStr)
{

}
}
```

Python3 Solution:

```
"""
Problem: Lexicographically Smallest Equivalent String
Difficulty: Medium
Tags: string, graph, sort

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function smallestEquivalentString(s1: string, s2: string, baseStr: string):
string {

};
```

C# Solution:

```
/*
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public string SmallestEquivalentString(string s1, string s2, string baseStr)
{

}
}
```

C Solution:

```
/*
* Problem: Lexicographically Smallest Equivalent String
* Difficulty: Medium
* Tags: string, graph, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* smallestEquivalentString(char* s1, char* s2, char* baseStr) {

}
```

Go Solution:

```
// Problem: Lexicographically Smallest Equivalent String
// Difficulty: Medium
// Tags: string, graph, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func smallestEquivalentString(s1 string, s2 string, baseStr string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Lexicographically Smallest Equivalent String
// Difficulty: Medium
// Tags: string, graph, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String smallestEquivalentString(String s1, String s2, String baseStr) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```erlang
-spec smallest_equivalent_string(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), BaseStr :: unicode:unicode_binary()) ->
unicode:unicode_binary().
smallest_equivalent_string(S1, S2, BaseStr) ->
.
```

Racket Solution:

```racket
(define/contract (smallest-equivalent-string s1 s2 baseStr)
(-> string? string? string? string?)
)
```