Problem 1071: Greatest Common Divisor of Strings

Problem Information

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

Problem Description

For two strings

s

and

t

, we say "

t

divides

s

" if and only if

s = t + t + t + ... + t + t

(i.e.,

t

is concatenated with itself one or more times).

Given two strings

str1

and

str2

, return

the largest string

x

such that

x

divides both

str1

and

str2

.

Example 1:

Input:

str1 = "ABCABC", str2 = "ABC"

Output:

"ABC"

Example 2:

Input:

str1 = "ABABAB", str2 = "ABAB"

Output:

"AB"

Example 3:

Input:

str1 = "LEET", str2 = "CODE"

Output:

""

Constraints:

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

str1

and

str2

consist of English uppercase letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
string gcdOfStrings(string str1, string str2) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (gcd-of-strings str1 str2)
(-> string? string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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 gcdOfStrings(string str1, string str2) {

}
};
```

Java Solution:

```
/**
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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 gcdOfStrings(String str1, String str2) {

}
}
```

Python3 Solution:

```
"""
Problem: Greatest Common Divisor of Strings
Difficulty: Easy
Tags: string, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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} str1
* @param {string} str2
* @return {string}
*/
var gcdOfStrings = function(str1, str2) {

};
```

TypeScript Solution:

```
/**
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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 gcdOfStrings(str1: string, str2: string): string {
```

```
};
```

C# Solution:

```
/*
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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 GcdOfStrings(string str1, string str2) {

}
}
```

C Solution:

```
/*
* Problem: Greatest Common Divisor of Strings
* Difficulty: Easy
* Tags: string, math
*
* 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* gcdOfStrings(char* str1, char* str2) {

}
```

Go Solution:

```
// Problem: Greatest Common Divisor of Strings
// Difficulty: Easy
```

```
// Tags: string, math
//
// 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 gcdOfStrings(str1 string, str2 string) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Greatest Common Divisor of Strings
// Difficulty: Easy
// Tags: string, math
//
// 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 gcd_of_strings(str1: String, str2: String) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (gcd-of-strings str1 str2)
(-> string? string? string?)
)
```