Problem 3365: Rearrange K Substrings to Form Target String

Problem Information

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

Problem Description

You are given two strings

s

and

t

, both of which are anagrams of each other, and an integer

k

.

Your task is to determine whether it is possible to split the string

s

into

k

equal-sized substrings, rearrange the substrings, and concatenate them in

any order

to create a new string that matches the given string

t

.

Return

true

if this is possible, otherwise, return

false

.

An

anagram

is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once.

A

substring

is a contiguous

non-empty

sequence of characters within a string.

Example 1:

Input:

s = "abcd", t = "cdab", k = 2

Output:

true

Explanation:

Split

s

into 2 substrings of length 2:

["ab", "cd"]

.

Rearranging these substrings as

["cd", "ab"]

, and then concatenating them results in

"cdab"

, which matches

t

.

Example 2:

Input:

s = "aabbcc", t = "bbaacc", k = 3

Output:

true

Explanation:

Split

s

into 3 substrings of length 2:

["aa", "bb", "cc"]

.

Rearranging these substrings as

["bb", "aa", "cc"]

, and then concatenating them results in

"bbaacc"

, which matches

t

.

Example 3:

Input:

s = "aabbcc", t = "bbaacc", k = 2

Output:

false

Explanation:

Split

s

into 2 substrings of length 3:

["aab", "bcc"]

.

These substrings cannot be rearranged to form

t = "bbaacc"

, so the output is

false

.

Constraints:

1 <= s.length == t.length <= 2 * 10

5

1 <= k <= s.length

s.length

is divisible by

k

.

s

and

t

consist only of lowercase English letters.

The input is generated such that

s

and

t

are anagrams of each other.

Code Snippets

C++:

```
class Solution {
public:
bool isPossibleToRearrange(string s, string t, int k) {

}
};
```

Java:

```
class Solution {
public boolean isPossibleToRearrange(String s, String t, int k) {

}
}
```

Python3:

```
class Solution:
def isPossibleToRearrange(self, s: str, t: str, k: int) -> bool:
```

Python:

```
class Solution(object):
def isPossibleToRearrange(self, s, t, k):
"""
:type s: str
:type t: str
:type k: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} t
* @param {number} k
* @return {boolean}
*/
var isPossibleToRearrange = function(s, t, k) {

};
```

TypeScript:

```
function isPossibleToRearrange(s: string, t: string, k: number): boolean {

};
```

C#:

```
public class Solution {
public bool IsPossibleToRearrange(string s, string t, int k) {

}
}
```

C:

```
bool isPossibleToRearrange(char* s, char* t, int k) {

}
```

Go:

```
func isPossibleToRearrange(s string, t string, k int) bool {

}
```

Kotlin:

```
class Solution {
fun isPossibleToRearrange(s: String, t: String, k: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func isPossibleToRearrange(_ s: String, _ t: String, _ k: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_possible_to_rearrange(s: String, t: String, k: i32) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @param {Integer} k
# @return {Boolean}
def is_possible_to_rearrange(s, t, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @param Integer $k
* @return Boolean
*/
function isPossibleToRearrange($s, $t, $k) {

}
}
```

Dart:

```
class Solution {
bool isPossibleToRearrange(String s, String t, int k) {

}
}
```

Scala:

```
object Solution {
def isPossibleToRearrange(s: String, t: String, k: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_possible_to_rearrange(s :: String.t, t :: String.t, k :: integer) ::
boolean
def is_possible_to_rearrange(s, t, k) do

end
end
```

Erlang:

```
-spec is_possible_to_rearrange(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), K :: integer()) -> boolean().
```

```
is_possible_to_rearrange(S, T, K) ->
.
```

Racket:

```
(define/contract (is-possible-to-rearrange s t k)
(-> string? string? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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:
bool isPossibleToRearrange(string s, string t, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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 boolean isPossibleToRearrange(String s, String t, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Rearrange K Substrings to Form Target String
Difficulty: Medium
Tags: string, tree, hash, sort

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 isPossibleToRearrange(self, s: str, t: str, k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isPossibleToRearrange(self, s, t, k):
"""
:type s: str
:type t: str
:type k: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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} s
* @param {string} t
* @param {number} k
* @return {boolean}
*/
var isPossibleToRearrange = function(s, t, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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 isPossibleToRearrange(s: string, t: string, k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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 bool IsPossibleToRearrange(string s, string t, int k) {

}
}
```

C Solution:

```
/*
* Problem: Rearrange K Substrings to Form Target String
* Difficulty: Medium
* Tags: string, tree, hash, sort
*
* 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
*/

bool isPossibleToRearrange(char* s, char* t, int k) {

}
```

Go Solution:

```
// Problem: Rearrange K Substrings to Form Target String
// Difficulty: Medium
// Tags: string, tree, hash, sort
//
// 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 isPossibleToRearrange(s string, t string, k int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPossibleToRearrange(s: String, t: String, k: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isPossibleToRearrange(_ s: String, _ t: String, _ k: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Rearrange K Substrings to Form Target String
// Difficulty: Medium
// Tags: string, tree, hash, sort
//
// 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 is_possible_to_rearrange(s: String, t: String, k: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @param {Integer} k
# @return {Boolean}
def is_possible_to_rearrange(s, t, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @param Integer $k
* @return Boolean
*/
function isPossibleToRearrange($s, $t, $k) {

}
}
```

Dart Solution:

```
class Solution {
bool isPossibleToRearrange(String s, String t, int k) {

}
}
```

Scala Solution:

```
object Solution {
def isPossibleToRearrange(s: String, t: String, k: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_possible_to_rearrange(s :: String.t, t :: String.t, k :: integer) ::
boolean
def is_possible_to_rearrange(s, t, k) do

end
end
```

Erlang Solution:

```
-spec is_possible_to_rearrange(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary(), K :: integer()) -> boolean().
is_possible_to_rearrange(S, T, K) ->
.
```

Racket Solution:

```
(define/contract (is-possible-to-rearrange s t k)
(-> string? string? exact-integer? boolean?)
)
```