Problem 1347: Minimum Number of Steps to Make Two Strings Anagram

Problem Information

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

Problem Description

You are given two strings of the same length

s

and

t

. In one step you can choose

any character

of

t

and replace it with

another character

.

Return

the minimum number of steps

to make

t

an anagram of

s

.

An

Anagram

of a string is a string that contains the same characters with a different (or the same) ordering.

Example 1:

Input:

s = "bab", t = "aba"

Output:

1

Explanation:

Replace the first 'a' in t with b, t = "bba" which is anagram of s.

Example 2:

Input:

s = "leetcode", t = "practice"

Output:

5

Explanation:

Replace 'p', 'r', 'a', 'i' and 'c' from t with proper characters to make t anagram of s.

Example 3:

Input:

s = "anagram", t = "mangaar"

Output:

0

Explanation:

"anagram" and "mangaar" are anagrams.

Constraints:

1 <= s.length <= 5 * 10

4

s.length == t.length

s

and

t

consist of lowercase English letters only.

Code Snippets

C++:

```
class Solution {
public:
int minSteps(string s, string t) {

}
};
```

Java:

```
class Solution {
public int minSteps(String s, String t) {

}
}
```

Python3:

```
class Solution:
def minSteps(self, s: str, t: str) -> int:
```

Python:

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

JavaScript:

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

};
```

}

TypeScript:

```
function minSteps(s: string, t: string): number {

};
```

C#:

```
public class Solution {
public int MinSteps(string s, string t) {

}
}
```

C:

```
int minSteps(char* s, char* t) {

}
```

Go:

```
func minSteps(s string, t string) int {

}
```

Kotlin:

```
class Solution {
fun minSteps(s: String, t: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def min_steps(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function minSteps($s, $t) {

}
}
```

Dart:

```
class Solution {
int minSteps(String s, String t) {

}
}
```

Scala:

```
object Solution {
def minSteps(s: String, t: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec min_steps(S :: unicode:unicode_binary(), T :: unicode:unicode_binary())
-> integer().
min_steps(S, T) ->
.
```

Racket:

```
(define/contract (min-steps s t)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
```

```
int minSteps(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int minSteps(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Steps to Make Two Strings Anagram
Difficulty: Medium
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def minSteps(self, s: str, t: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @param {string} t
* @return {number}
*/
var minSteps = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function minSteps(s: string, t: string): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MinSteps(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Steps to Make Two Strings Anagram
* Difficulty: Medium
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minSteps(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Minimum Number of Steps to Make Two Strings Anagram
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func minSteps(s string, t string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSteps(s: String, t: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Steps to Make Two Strings Anagram
// Difficulty: Medium
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn min_steps(s: String, t: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def min_steps(s, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function minSteps($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
int minSteps(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def minSteps(s: String, t: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec min_steps(S :: unicode:unicode_binary(), T :: unicode:unicode_binary())
-> integer().
min_steps(S, T) ->
.
```

Racket Solution:

```
(define/contract (min-steps s t)
(-> string? string? exact-integer?)
)
```