Problem 1540: Can Convert String in K Moves

Problem Information

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

Problem Description

Given two strings

s

and

t

, your goal is to convert

s

into

t

in

k

moves or less.

During the

i

th

(

1 <= i <= k

)

move you can:

Choose any index

j

(1-indexed) from

s

, such that

1 <= j <= s.length

and

j

has not been chosen in any previous move, and shift the character at that index

i

times.

Do nothing.

Shifting a character means replacing it by the next letter in the alphabet (wrapping around so that

'z'

becomes

'a'

). Shifting a character by

i

means applying the shift operations

i

times.

Remember that any index

j

can be picked at most once.

Return

true

if it's possible to convert

s

into

t

in no more than

k

moves, otherwise return

false

.

Example 1:

Input:

s = "input", t = "ouput", k = 9

Output:

true

Explanation:

In the 6th move, we shift 'i' 6 times to get 'o'. And in the 7th move we shift 'n' to get 'u'.

Example 2:

Input:

s = "abc", t = "bcd", k = 10

Output:

false

Explanation:

We need to shift each character in s one time to convert it into t. We can shift 'a' to 'b' during the 1st move. However, there is no way to shift the other characters in the remaining moves to obtain t from s.

Example 3:

Input:

s = "aab", t = "bbb", k = 27

Output:

true

Explanation:

In the 1st move, we shift the first 'a' 1 time to get 'b'. In the 27th move, we shift the second 'a' 27 times to get 'b'.

Constraints:

1 <= s.length, t.length <= 10^5

0 <= k <= 10^9

s

,

t

contain only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def canConvertString(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 canConvertString = function(s, t, k) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

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

Racket:

```
(define/contract (can-convert-string s t k)
(-> string? string? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Can Convert String in K Moves
* 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:
bool canConvertString(string s, string t, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Can Convert String in K Moves
* 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 boolean canConvertString(String s, String t, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Can Convert String in K Moves
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 canConvertString(self, s: str, t: str, k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Can Convert String in K Moves
* 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
* @param {number} k
* @return {boolean}
*/
var canConvertString = function(s, t, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Can Convert String in K Moves
* 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 canConvertString(s: string, t: string, k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Can Convert String in K Moves
* 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 bool CanConvertString(string s, string t, int k) {

}
}
```

C Solution:

```
/*
* Problem: Can Convert String in K Moves
* 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
*/

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

}
```

Go Solution:

```
// Problem: Can Convert String in K Moves
// 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 canConvertString(s string, t string, k int) bool {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Can Convert String in K Moves
// 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 can_convert_string(s: String, t: String, k: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @param {Integer} k
# @return {Boolean}
```

```
def can_convert_string(s, t, k)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```



end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-convert-string s t k)
(-> string? string? exact-integer? boolean?)
)
```