Problem 2851: String Transformation

Problem Information

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

Problem Description

You are given two strings

s

and

t

of equal length

n

. You can perform the following operation on the string

s

:

Remove a

suffix

of

s

of length

l

where

$0 < l < n$

and append it at the start of

s

.

For example, let

$s = 'abcd'$

then in one operation you can remove the suffix

'cd'

and append it in front of

s

making

$s = 'cdab'$

.

You are also given an integer

k

. Return

the number of ways in which

s

can be transformed into

t

in

exactly

k

operations.

Since the answer can be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

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

Output:

2

Explanation:

First way: In first operation, choose suffix from index = 3, so resulting s = "dabc". In second operation, choose suffix from index = 3, so resulting s = "cdab".

Second way: In first operation, choose suffix from index = 1, so resulting s = "bcda". In second operation, choose suffix from index = 1, so resulting s = "cdab".

Example 2:

Input:

s = "ababab", t = "ababab", k = 1

Output:

2

Explanation:

First way: Choose suffix from index = 2, so resulting s = "ababab".

Second way: Choose suffix from index = 4, so resulting s = "ababab".

Constraints:

2 <= s.length <= 5 * 10

5

1 <= k <= 10

15

s.length == t.length

s

and

t

consist of only lowercase English alphabets.

Code Snippets

C++:

```
class Solution {
public:
int numberOfWays(string s, string t, long long k) {

}
};
```

Java:

```
class Solution {
public int numberOfWays(String s, String t, long k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
```

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int NumberOfWays(string s, string t, long k) {

}
}
```

C:

```
int numberOfWays(char* s, char* t, long long k) {

}
```

Go:

```
func numberOfWays(s string, t string, k int64) int {

}
```

Kotlin:

```
class Solution {
fun numberOfWays(s: String, t: String, k: Long): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn number_of_ways(s: String, t: String, k: i64) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def numberOfWays(s: String, t: String, k: Long): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-of-ways s t k)
(-> string? string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: String Transformation
* Difficulty: Hard
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numberOfWays(string s, string t, long long k) {

}
};
```

Java Solution:

```
/**
* Problem: String Transformation
* Difficulty: Hard
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberOfWays(String s, String t, long k) {

}
}
```

Python3 Solution:

```
"""
Problem: String Transformation
Difficulty: Hard
Tags: string, dp, math
```

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

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: String Transformation
* Difficulty: Hard
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: String Transformation
* Difficulty: Hard
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfWays(s: string, t: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: String Transformation
* Difficulty: Hard
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumberOfWays(string s, string t, long k) {

}
}
```

C Solution:

```
/*
* Problem: String Transformation
* Difficulty: Hard
```

```
* Tags: string, dp, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numberOfWays(char* s, char* t, long long k) {

}
```

Go Solution:

```
// Problem: String Transformation
// Difficulty: Hard
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfWays(s string, t string, k int64) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfWays(s: String, t: String, k: Long): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: String Transformation
// Difficulty: Hard
// Tags: string, dp, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn number_of_ways(s: String, t: String, k: i64) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def numberOfWays(s: String, t: String, k: Long): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-of-ways s t k)
(-> string? string? exact-integer? exact-integer?)
)
```