Problem 1397: Find All Good Strings

Problem Information

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

Problem Description

Given the strings

s1

and

s2

of size

n

and the string

evil

, return

the number of

good

strings

.

A

good

string has size

n

, it is alphabetically greater than or equal to

s1

, it is alphabetically smaller than or equal to

s2

, and it does not contain the string

evil

as a substring. Since the answer can be a huge number, return this

modulo

10

9

+ 7

.

Example 1:

Input:

n = 2, s1 = "aa", s2 = "da", evil = "b"

Output:

51

Explanation:

There are 25 good strings starting with 'a': "aa","ac","ad",...,"az". Then there are 25 good strings starting with 'c': "ca","cc","cd",...,"cz" and finally there is one good string starting with 'd': "da".

Example 2:

Input:

n = 8, s1 = "leetcode", s2 = "leetgoes", evil = "leet"

Output:

0

Explanation:

All strings greater than or equal to s1 and smaller than or equal to s2 start with the prefix "leet", therefore, there is not any good string.

Example 3:

Input:

n = 2, s1 = "gx", s2 = "gz", evil = "x"

Output:

2

Constraints:

s1.length == n

s2.length == n

s1 <= s2

1 <= n <= 500

1 <= evil.length <= 50

All strings consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int findGoodStrings(int n, string s1, string s2, string evil) {

}
};
```

Java:

```
class Solution {
public int findGoodStrings(int n, String s1, String s2, String evil) {

}
}
```

Python3:

```
class Solution:
def findGoodStrings(self, n: int, s1: str, s2: str, evil: str) -> int:
```

Python:

```
class Solution(object):
def findGoodStrings(self, n, s1, s2, evil):
"""
:type n: int
:type s1: str
:type s2: str
:type evil: str
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {string} s1
* @param {string} s2
* @param {string} evil
* @return {number}
*/
var findGoodStrings = function(n, s1, s2, evil) {

};
```

TypeScript:

```
function findGoodStrings(n: number, s1: string, s2: string, evil: string):
number {

};
```

C#:

```
public class Solution {
public int FindGoodStrings(int n, string s1, string s2, string evil) {

}
}
```

C:

```
int findGoodStrings(int n, char* s1, char* s2, char* evil) {

}
```

Go:

```
func findGoodStrings(n int, s1 string, s2 string, evil string) int {

}
```

Kotlin:

```
class Solution {
fun findGoodStrings(n: Int, s1: String, s2: String, evil: String): Int {

}
}
```

Swift:

```
class Solution {
func findGoodStrings(_ n: Int, _ s1: String, _ s2: String, _ evil: String) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_good_strings(n: i32, s1: String, s2: String, evil: String) -> i32
{

}
}
```

Ruby:

```
# @param {Integer} n
# @param {String} s1
# @param {String} s2
# @param {String} evil
# @return {Integer}
def find_good_strings(n, s1, s2, evil)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param String $s1
* @param String $s2
* @param String $evil
* @return Integer
*/
function findGoodStrings($n, $s1, $s2, $evil) {

}
}
```

Dart:

```
class Solution {
int findGoodStrings(int n, String s1, String s2, String evil) {

}
}
```

Scala:

```
object Solution {
def findGoodStrings(n: Int, s1: String, s2: String, evil: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_good_strings(n :: integer, s1 :: String.t, s2 :: String.t, evil ::
String.t) :: integer
def find_good_strings(n, s1, s2, evil) do

end
end
```

Erlang:

```
-spec find_good_strings(N :: integer(), S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), Evil :: unicode:unicode_binary()) -> integer().
find_good_strings(N, S1, S2, Evil) ->
.
```

Racket:

```
(define/contract (find-good-strings n s1 s2 evil)
(-> exact-integer? string? string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 findGoodStrings(int n, string s1, string s2, string evil) {

}
};
```

Java Solution:

```
/**
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 findGoodStrings(int n, String s1, String s2, String evil) {
}
}
```

Python3 Solution:

```
"""
Problem: Find All Good Strings
Difficulty: Hard
Tags: string, tree, dp

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 findGoodStrings(self, n: int, s1: str, s2: str, evil: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findGoodStrings(self, n, s1, s2, evil):
"""
:type n: int
:type s1: str
:type s2: str
:type evil: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 {number} n
* @param {string} s1
* @param {string} s2
* @param {string} evil
* @return {number}
*/
var findGoodStrings = function(n, s1, s2, evil) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 findGoodStrings(n: number, s1: string, s2: string, evil: string):
number {

};
```

C# Solution:

```
/*
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 FindGoodStrings(int n, string s1, string s2, string evil) {

}
}
```

C Solution:

```
/*
* Problem: Find All Good Strings
* Difficulty: Hard
* Tags: string, tree, dp
*
* 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 findGoodStrings(int n, char* s1, char* s2, char* evil) {

}
```

Go Solution:

```
// Problem: Find All Good Strings
// Difficulty: Hard
// Tags: string, tree, dp
//
// 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 findGoodStrings(n int, s1 string, s2 string, evil string) int {

}
```

Kotlin Solution:

```
class Solution {
fun findGoodStrings(n: Int, s1: String, s2: String, evil: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func findGoodStrings(_ n: Int, _ s1: String, _ s2: String, _ evil: String) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Find All Good Strings
// Difficulty: Hard
// Tags: string, tree, dp
//
// 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 find_good_strings(n: i32, s1: String, s2: String, evil: String) -> i32
{

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {String} s1
# @param {String} s2
# @param {String} evil
# @return {Integer}
def find_good_strings(n, s1, s2, evil)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param String $s1
* @param String $s2
* @param String $evil
* @return Integer
*/
function findGoodStrings($n, $s1, $s2, $evil) {

}
}
```

Dart Solution:

```
class Solution {
int findGoodStrings(int n, String s1, String s2, String evil) {

}
}
```

Scala Solution:

```
object Solution {
def findGoodStrings(n: Int, s1: String, s2: String, evil: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_good_strings(n :: integer, s1 :: String.t, s2 :: String.t, evil :: String.t) :: integer
def find_good_strings(n, s1, s2, evil) do

end
end
```

Erlang Solution:

```
-spec find_good_strings(N :: integer(), S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), Evil :: unicode:unicode_binary()) -> integer().
find_good_strings(N, S1, S2, Evil) ->
.
```

Racket Solution:

```
(define/contract (find-good-strings n s1 s2 evil)
(-> exact-integer? string? string? string? exact-integer?)
)
```