Problem 2842: Count K-Subsequences of a String With Maximum Beauty

Problem Information

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

Problem Description

You are given a string

s

and an integer

k

.

A

k-subsequence

is a

subsequence

of

s

, having length

k

, and all its characters are

unique

,

i.e

., every character occurs once.

Let

f(c)

denote the number of times the character

c

occurs in

s

.

The

beauty

of a

k-subsequence

is the

sum

of

f(c)

for every character

c

in the k-subsequence.

For example, consider

s = "abbbdd"

and

k = 2

:

f('a') = 1

,

f('b') = 3

,

f('d') = 2

Some k-subsequences of

s

are:

"

ab

bbdd"

->

"ab"

having a beauty of

f('a') + f('b') = 4

"

a

bbb

d

d"

->

"ad"

having a beauty of

f('a') + f('d') = 3

"a

b

bb

d

d"

->

"bd"

having a beauty of

f('b') + f('d') = 5

Return

an integer denoting the number of k-subsequences

whose

beauty

is the

maximum

among all

k-subsequences

. Since the answer may be too large, return it modulo

10

9

+ 7

.

A subsequence of a string is a new string formed from the original string by deleting some (possibly none) of the characters without disturbing the relative positions of the remaining characters.

Notes

f(c)

is the number of times a character

c

occurs in

s

, not a k-subsequence.

Two k-subsequences are considered different if one is formed by an index that is not present in the other. So, two k-subsequences may form the same string.

Example 1:

Input:

s = "bcca", k = 2

Output:

4

Explanation:

From s we have f('a') = 1, f('b') = 1, and f('c') = 2.

The k-subsequences of s are:

bc

ca having a beauty of f('b') + f('c') = 3

b

c

c

a having a beauty of f('b') + f('c') = 3

b

cc

a

having a beauty of f('b') + f('a') = 2 b

c

c

a

having a beauty of f('c') + f('a') = 3 bc

ca

having a beauty of f('c') + f('a') = 3 There are 4 k-subsequences that have the maximum beauty, 3. Hence, the answer is 4.

Example 2:

Input:

s = "abbcd", k = 4

Output:

2

Explanation:

From s we have f('a') = 1, f('b') = 2, f('c') = 1, and f('d') = 1. The k-subsequences of s are:

ab

b

cd

having a beauty of f('a') + f('b') + f('c') + f('d') = 5

a

b

bcd

having a beauty of f('a') + f('b') + f('c') + f('d') = 5 There are 2 k-subsequences that have the maximum beauty, 5. Hence, the answer is 2.

Constraints:

1 <= s.length <= 2 * 10

5

1 <= k <= s.length

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int countKSubsequencesWithMaxBeauty(string s, int k) {

}
};
```

Java:

```
class Solution {
public int countKSubsequencesWithMaxBeauty(String s, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countKSubsequencesWithMaxBeauty(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int CountKSubsequencesWithMaxBeauty(string s, int k) {

}
}
```

C:

```
int countKSubsequencesWithMaxBeauty(char* s, int k) {

}
```

Go:

```
func countKSubsequencesWithMaxBeauty(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun countKSubsequencesWithMaxBeauty(s: String, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int countKSubsequencesWithMaxBeauty(String s, int k) {

}
}
```

Scala:

```
object Solution {
def countKSubsequencesWithMaxBeauty(s: String, k: Int): Int = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (count-k-subsequences-with-max-beauty s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
* Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
* Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count K-Subsequences of a String With Maximum Beauty
Difficulty: Hard
Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
* Tags: string, greedy, math, 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 {number} k
* @return {number}
*/
var countKSubsequencesWithMaxBeauty = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
* Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
```

```
* Tags: string, greedy, math, 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 CountKSubsequencesWithMaxBeauty(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count K-Subsequences of a String With Maximum Beauty
* Difficulty: Hard
* Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(char* s, int k) {

}
```

Go Solution:

```
// Problem: Count K-Subsequences of a String With Maximum Beauty
// Difficulty: Hard
// Tags: string, greedy, math, 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 countKSubsequencesWithMaxBeauty(s string, k int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun countKSubsequencesWithMaxBeauty(s: String, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count K-Subsequences of a String With Maximum Beauty
// Difficulty: Hard
// Tags: string, greedy, math, 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 count_k_subsequences_with_max_beauty(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int countKSubsequencesWithMaxBeauty(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countKSubsequencesWithMaxBeauty(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-k-subsequences-with-max-beauty s k)
(-> string? exact-integer? exact-integer?)
)
```