Problem 3271: Hash Divided String

Problem Information

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

Problem Description

You are given a string

s

of length

n

and an integer

k

, where

n

is a

multiple

of

k

. Your task is to hash the string

s

into a new string called

result

, which has a length of

n / k

.

First, divide

s

into

n / k

substrings

, each with a length of

k

. Then, initialize

result

as an

empty

string.

For each

substring

in order from the beginning:

The

hash value

of a character is the index of that characte

r in the

English alphabet

(e.g.,

'a' $\rightarrow$

0

,

'b' $\rightarrow$

1

, ...,

'z' $\rightarrow$

25

).

Calculate the

sum

of all the

hash values

of the characters in the substring.

Find the remainder of this sum when divided by 26, which is called

hashedChar

.

Identify the character in the English lowercase alphabet that corresponds to

hashedChar

.

Append that character to the end of

result

.

Return

result

.

Example 1:

Input:

s = "abcd", k = 2

Output:

"bf"

Explanation:

First substring:

"ab"

,

0 + 1 = 1

,

1 % 26 = 1

,

result[0] = 'b'

.

Second substring:

"cd"

,

2 + 3 = 5

,

5 % 26 = 5

,

result[1] = 'f'

.

Example 2:

Input:

s = "mxz", k = 3

Output:

"i"

Explanation:

The only substring:

"mxz"

,

12 + 23 + 25 = 60

,

60 % 26 = 8

,

result[0] = 'i'

.

Constraints:

1 <= k <= 100

k <= s.length <= 1000

s.length

is divisible by

k

.

s

consists only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (string-hash s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String stringHash(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Hash Divided String
Difficulty: Medium
Tags: string, tree, hash
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function stringHash(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public string StringHash(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Hash Divided String
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(h) for recursion stack where h is height
*/

char* stringHash(char* s, int k) {

}
```

Go Solution:

```
// Problem: Hash Divided String
// Difficulty: Medium
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func stringHash(s string, k int) string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Hash Divided String
// Difficulty: Medium
```

```
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (string-hash s k)
(-> string? exact-integer? string?)
)
```