Problem 2156: Find Substring With Given Hash Value

Problem Information

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

Problem Description

The hash of a

0-indexed

string

s

of length

k

, given integers

p

and

m

, is computed using the following function:

hash(s, p, m) = (val(s[0]) * p

0

+ val(s[1]) * p

1

+ ... + val(s[k-1]) * p

k-1

) mod m

.

Where

val(s[i])

represents the index of

s[i]

in the alphabet from

val('a') = 1

to

val('z') = 26

.

You are given a string

s

and the integers

power

,

modulo

,

k

, and

hashValue.

Return

sub

,

the

first

substring

of

s

of length

k

such that

hash(sub, power, modulo) == hashValue

.

The test cases will be generated such that an answer always exists

.

A

substring

is a contiguous non-empty sequence of characters within a string.

Example 1:

Input:

s = "leetcode", power = 7, modulo = 20, k = 2, hashValue = 0

Output:

"ee"

Explanation:

The hash of "ee" can be computed to be hash("ee", 7, 20) = (5 * 1 + 5 * 7) mod 20 = 40 mod 20 = 0. "ee" is the first substring of length 2 with hashValue 0. Hence, we return "ee".

Example 2:

Input:

s = "fbxzaad", power = 31, modulo = 100, k = 3, hashValue = 32

Output:

"fbx"

Explanation:

The hash of "fbx" can be computed to be hash("fbx", 31, 100) = (6 * 1 + 2 * 31 + 24 * 31

2

) mod 100 = 23132 mod 100 = 32. The hash of "bxz" can be computed to be hash("bxz", 31, 100) = (2 * 1 + 24 * 31 + 26 * 31

2

) mod 100 = 25732 mod 100 = 32. "fbx" is the first substring of length 3 with hashValue 32. Hence, we return "fbx". Note that "bxz" also has a hash of 32 but it appears later than "fbx".

Constraints:

1 <= k <= s.length <= 2 * 10

4

1 <= power, modulo <= 10

9

0 <= hashValue < modulo

s

consists of lowercase English letters only.

The test cases are generated such that an answer always

exists

.

Code Snippets

C++:

```
class Solution {
public:
```

```
string subStrHash(string s, int power, int modulo, int k, int hashValue) {

}
};
```

Java:

```
class Solution {
public String subStrHash(String s, int power, int modulo, int k, int
hashValue) {

}
}
```

Python3:

```
class Solution:
def subStrHash(self, s: str, power: int, modulo: int, k: int, hashValue: int)
-> str:
```

Python:

```
class Solution(object):
def subStrHash(self, s, power, modulo, k, hashValue):
"""
:type s: str
:type power: int
:type modulo: int
:type k: int
:type hashValue: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} power
* @param {number} modulo
* @param {number} k
* @param {number} hashValue
* @return {string}
```

```
*/
var subStrHash = function(s, power, modulo, k, hashValue) {

};
```

TypeScript:

```
function subStrHash(s: string, power: number, modulo: number, k: number,
hashValue: number): string {

};
```

C#:

```
public class Solution {
public string SubStrHash(string s, int power, int modulo, int k, int
hashValue) {

}
}
```

C:

```
char* subStrHash(char* s, int power, int modulo, int k, int hashValue) {

}
```

Go:

```
func subStrHash(s string, power int, modulo int, k int, hashValue int) string
{

}
```

Kotlin:

```
class Solution {
fun subStrHash(s: String, power: Int, modulo: Int, k: Int, hashValue: Int):
String {

}
}
```

Swift:

```
class Solution {
func subStrHash(_ s: String, _ power: Int, _ modulo: Int, _ k: Int, _
hashValue: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn sub_str_hash(s: String, power: i32, modulo: i32, k: i32, hash_value:
i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} power
# @param {Integer} modulo
# @param {Integer} k
# @param {Integer} hash_value
# @return {String}
def sub_str_hash(s, power, modulo, k, hash_value)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $power
* @param Integer $modulo
* @param Integer $k
* @param Integer $hashValue
* @return String
*/
function subStrHash($s, $power, $modulo, $k, $hashValue) {

}
}
```

Dart:

```
class Solution {
String subStrHash(String s, int power, int modulo, int k, int hashValue) {

}
}
```

Scala:

```
object Solution {
def subStrHash(s: String, power: Int, modulo: Int, k: Int, hashValue: Int):
String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sub_str_hash(s :: String.t, power :: integer, modulo :: integer, k ::
integer, hash_value :: integer) :: String.t
def sub_str_hash(s, power, modulo, k, hash_value) do

end
end
```

Erlang:

```
-spec sub_str_hash(S :: unicode:unicode_binary(), Power :: integer(), Modulo
:: integer(), K :: integer(), HashValue :: integer()) ->
unicode:unicode_binary().
sub_str_hash(S, Power, Modulo, K, HashValue) ->
.
```

Racket:

```
(define/contract (sub-str-hash s power modulo k hashValue)
(-> string? exact-integer? exact-integer? exact-integer? exact-integer?
string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
string subStrHash(string s, int power, int modulo, int k, int hashValue) {

}
};
```

Java Solution:

```
/**
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public String subStrHash(String s, int power, int modulo, int k, int
hashValue) {
}
}
```

Python3 Solution:

```
"""
Problem: Find Substring With Given Hash Value
Difficulty: Hard
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def subStrHash(self, s, power, modulo, k, hashValue):
"""
:type s: str
:type power: int
:type modulo: int
:type k: int
:type hashValue: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* 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} power
* @param {number} modulo
* @param {number} k
* @param {number} hashValue
* @return {string}
*/
var subStrHash = function(s, power, modulo, k, hashValue) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function subStrHash(s: string, power: number, modulo: number, k: number,
hashValue: number): string {

};
```

C# Solution:

```
/*
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* 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 SubStrHash(string s, int power, int modulo, int k, int
hashValue) {

}
}
```

C Solution:

```
/*
* Problem: Find Substring With Given Hash Value
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

char* subStrHash(char* s, int power, int modulo, int k, int hashValue) {

}
```

Go Solution:

```
// Problem: Find Substring With Given Hash Value
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func subStrHash(s string, power int, modulo int, k int, hashValue int) string
{


}
```

Kotlin Solution:

```
class Solution {
fun subStrHash(s: String, power: Int, modulo: Int, k: Int, hashValue: Int):
String {

}
}
```

Swift Solution:

```
class Solution {
func subStrHash(_ s: String, _ power: Int, _ modulo: Int, _ k: Int, _
hashValue: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Find Substring With Given Hash Value
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn sub_str_hash(s: String, power: i32, modulo: i32, k: i32, hash_value:
i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} power
# @param {Integer} modulo
# @param {Integer} k
# @param {Integer} hash_value
# @return {String}
def sub_str_hash(s, power, modulo, k, hash_value)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $power
* @param Integer $modulo
* @param Integer $k
* @param Integer $hashValue
* @return String
*/
function subStrHash($s, $power, $modulo, $k, $hashValue) {

}
}
```

Dart Solution:

```
class Solution {
String subStrHash(String s, int power, int modulo, int k, int hashValue) {

}
}
```

Scala Solution:

```
object Solution {
def subStrHash(s: String, power: Int, modulo: Int, k: Int, hashValue: Int):
String = {

}


}
```

Elixir Solution:

```
defmodule Solution do
@spec sub_str_hash(s :: String.t, power :: integer, modulo :: integer, k ::
integer, hash_value :: integer) :: String.t
def sub_str_hash(s, power, modulo, k, hash_value) do

end
end
```

Erlang Solution:

```
-spec sub_str_hash(S :: unicode:unicode_binary(), Power :: integer(), Modulo
:: integer(), K :: integer(), HashValue :: integer()) ->
unicode:unicode_binary().
sub_str_hash(S, Power, Modulo, K, HashValue) ->
.
```

Racket Solution:

```
(define/contract (sub-str-hash s power modulo k hashValue)
(-> string? exact-integer? exact-integer? exact-integer? exact-integer?
string?)
)
```