Problem 2213: Longest Substring of One Repeating Character

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

. You are also given a

0-indexed

string

queryCharacters

of length

k

and a

0-indexed

array of integer

indices

queryIndices

of length

k

, both of which are used to describe

k

queries.

The

i

th

query updates the character in

s

at index

queryIndices[i]

to the character

queryCharacters[i]

.

Return

an array

lengths

of length

k

where

lengths[i]

is the

length

of the

longest substring

of

s

consisting of

only one repeating

character

after

the

i

th

query

is performed.

Example 1:

Input:

s = "babacc", queryCharacters = "bcb", queryIndices = [1,3,3]

Output:

[3,3,4]

Explanation:

- 1

st

query updates s = "

b

b

b

acc". The longest substring consisting of one repeating character is "bbb" with length 3. - 2

nd

query updates s = "bbb

c

cc

". The longest substring consisting of one repeating character can be "bbb" or "ccc" with length 3. - 3

rd

query updates s = "

bbb

b

cc". The longest substring consisting of one repeating character is "bbbb" with length 4. Thus, we return [3,3,4].

Example 2:

Input:

s = "abyzz", queryCharacters = "aa", queryIndices = [2,1]

Output:

[2,3]

Explanation:

- 1

st

query updates s = "ab

a

zz

". The longest substring consisting of one repeating character is "zz" with length 2. - 2

nd

query updates s = "

a

a

a

zz". The longest substring consisting of one repeating character is "aaa" with length 3. Thus, we return [2,3].

Constraints:

1 <= s.length <= 10

5

s

consists of lowercase English letters.

k == queryCharacters.length == queryIndices.length

1 <= k <= 10

5

queryCharacters

consists of lowercase English letters.

0 <= queryIndices[i] < s.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> longestRepeating(string s, string queryCharacters, vector<int>&
queryIndices) {
}
};
```

Java:

```
class Solution {
public int[] longestRepeating(String s, String queryCharacters, int[]
queryIndices) {

}
}
```

Python3:

```
class Solution:
def longestRepeating(self, s: str, queryCharacters: str, queryIndices:
List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def longestRepeating(self, s, queryCharacters, queryIndices):
"""
:type s: str
:type queryCharacters: str
:type queryIndices: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} queryCharacters
* @param {number[]} queryIndices
* @return {number[]}
*/
var longestRepeating = function(s, queryCharacters, queryIndices) {

};
```

TypeScript:

}

```
function longestRepeating(s: string, queryCharacters: string, queryIndices:
number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] LongestRepeating(string s, string queryCharacters, int[]
queryIndices) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestRepeating(char* s, char* queryCharacters, int* queryIndices, int
queryIndicesSize, int* returnSize) {

}
```

Go:

```
func longestRepeating(s string, queryCharacters string, queryIndices []int)
[]int {

}
```

Kotlin:

```
class Solution {
fun longestRepeating(s: String, queryCharacters: String, queryIndices:
IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func longestRepeating(_ s: String, _ queryCharacters: String, _ queryIndices:
[Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn longest_repeating(s: String, query_characters: String, query_indices:
Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} query_characters
# @param {Integer[]} query_indices
# @return {Integer[]}
def longest_repeating(s, query_characters, query_indices)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $queryCharacters
* @param Integer[] $queryIndices
* @return Integer[]
*/
function longestRepeating($s, $queryCharacters, $queryIndices) {

}
}
```

Dart:

```
class Solution {
List<int> longestRepeating(String s, String queryCharacters, List<int>
queryIndices) {

}
}
```

Scala:

```
object Solution {
def longestRepeating(s: String, queryCharacters: String, queryIndices:
Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_repeating(s :: String.t, query_characters :: String.t,
query_indices :: [integer]) :: [integer]
def longest_repeating(s, query_characters, query_indices) do

end
end
```

Erlang:

```
-spec longest_repeating(S :: unicode:unicode_binary(), QueryCharacters ::
unicode:unicode_binary(), QueryIndices :: [integer()]) -> [integer()].
longest_repeating(S, QueryCharacters, QueryIndices) ->
.
```

Racket:

```
(define/contract (longest-repeating s queryCharacters queryIndices)
(-> string? string? (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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:
vector<int> longestRepeating(string s, string queryCharacters, vector<int>&
queryIndices) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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 int[] longestRepeating(String s, String queryCharacters, int[]
queryIndices) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Substring of One Repeating Character
```

```
Difficulty: Hard
Tags: array, string, tree

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 longestRepeating(self, s: str, queryCharacters: str, queryIndices:
List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestRepeating(self, s, queryCharacters, queryIndices):
"""
:type s: str
:type queryCharacters: str
:type queryIndices: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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 {string} queryCharacters
* @param {number[]} queryIndices
* @return {number[]}
*/
var longestRepeating = function(s, queryCharacters, queryIndices) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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 longestRepeating(s: string, queryCharacters: string, queryIndices:
number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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 int[] LongestRepeating(string s, string queryCharacters, int[]
queryIndices) {

}
}
```

C Solution:

```
/*
* Problem: Longest Substring of One Repeating Character
* Difficulty: Hard
* Tags: array, string, tree
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* longestRepeating(char* s, char* queryCharacters, int* queryIndices, int
queryIndicesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Substring of One Repeating Character
// Difficulty: Hard
// Tags: array, string, tree
//
// 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 longestRepeating(s string, queryCharacters string, queryIndices []int)
[]int {

}
```

Kotlin Solution:

```
class Solution {
fun longestRepeating(s: String, queryCharacters: String, queryIndices:
IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func longestRepeating(_ s: String, _ queryCharacters: String, _ queryIndices:
[Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Longest Substring of One Repeating Character
// Difficulty: Hard
// Tags: array, string, tree
//
// 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 longest_repeating(s: String, query_characters: String, query_indices:
Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} query_characters
# @param {Integer[]} query_indices
# @return {Integer[]}
def longest_repeating(s, query_characters, query_indices)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $queryCharacters
* @param Integer[] $queryIndices
* @return Integer[]
*/
function longestRepeating($s, $queryCharacters, $queryIndices) {

}
}
```

Dart Solution:

```
class Solution {
List<int> longestRepeating(String s, String queryCharacters, List<int>
queryIndices) {

}
}
```

Scala Solution:

```
object Solution {
def longestRepeating(s: String, queryCharacters: String, queryIndices:
Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_repeating(s :: String.t, query_characters :: String.t,
query_indices :: [integer]) :: [integer]
def longest_repeating(s, query_characters, query_indices) do

end
end
```

Erlang Solution:

```
-spec longest_repeating(S :: unicode:unicode_binary(), QueryCharacters ::
unicode:unicode_binary(), QueryIndices :: [integer()]) -> [integer()].
longest_repeating(S, QueryCharacters, QueryIndices) ->
.
```

Racket Solution:

```
(define/contract (longest-repeating s queryCharacters queryIndices)
(-> string? string? (listof exact-integer?) (listof exact-integer?))
)
```