Problem 2564: Substring XOR Queries

Problem Information

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

Problem Description

You are given a

binary string

s

, and a

2D

integer array

queries

where

queries[i] = [first

i

, second

i

]

.

For the

i

th

query, find the

shortest substring

of

s

whose

decimal value

,

val

, yields

second

i

when

bitwise XORed

with

first

i

. In other words,

val ^ first

i

== second

i

.

The answer to the

i

th

query is the endpoints (

0-indexed

) of the substring

[left

i

, right

i

]

or

[-1, -1]

if no such substring exists. If there are multiple answers, choose the one with the

minimum

left

i

.

Return an array

ans

where

ans[i] = [left

i

, right

i

]

is the answer to the

i

th

query.

A

substring

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

Example 1:

Input:

s = "101101", queries = [[0,5],[1,2]]

Output:

[[0,2],[2,3]]

Explanation:

For the first query the substring in range

[0,2]

is

"101"

which has a decimal value of

5

, and

5 ^ 0 = 5

, hence the answer to the first query is

[0,2]

. In the second query, the substring in range

[2,3]

is

"11",

and has a decimal value of

3

, and

3

^ 1 = 2

. So,

[2,3]

is returned for the second query.

Example 2:

Input:

s = "0101", queries = [[12,8]]

Output:

[[-1,-1]]

Explanation:

In this example there is no substring that answers the query, hence

[-1,-1] is returned

.

Example 3:

Input:

s = "1", queries = [[4,5]]

Output:

[[0,0]]

Explanation:

For this example, the substring in range

[0,0]

has a decimal value of

1

, and

1 ^ 4 = 5

. So, the answer is

[0,0]

.

Constraints:

1 <= s.length <= 10

4

s[i]

is either

'0'

or

'1'

.

1 <= queries.length <= 10

5

0 <= first

i

, second

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> substringXorQueries(string s, vector<vector<int>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[][] substringXorQueries(String s, int[][] queries) {
```

```
}
}
```

Python3:

```
class Solution:
def substringXorQueries(self, s: str, queries: List[List[int]]) ->
List[List[int]]:
```

Python:

```
class Solution(object):
def substringXorQueries(self, s, queries):
"""
:type s: str
:type queries: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[][]} queries
* @return {number[][]}
*/
var substringXorQueries = function(s, queries) {

};
```

TypeScript:

```
function substringXorQueries(s: string, queries: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SubstringXorQueries(string s, int[][] queries) {

}
```

```
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** substringXorQueries(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func substringXorQueries(s string, queries [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun substringXorQueries(s: String, queries: Array<IntArray>): Array<IntArray>
{

}
}
```

Swift:

```
class Solution {
func substringXorQueries(_ s: String, _ queries: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn substring_xor_queries(s: String, queries: Vec<Vec<i32>>) ->
```

```
Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Integer[][]}
def substring_xor_queries(s, queries)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $queries
* @return Integer[][]
*/
function substringXorQueries($s, $queries) {

}
}
```

Dart:

```
class Solution {
List<List<int>> substringXorQueries(String s, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def substringXorQueries(s: String, queries: Array[Array[Int]]):
Array[Array[Int]] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec substring_xor_queries(s :: String.t, queries :: [[integer]]) ::
[[integer]]
def substring_xor_queries(s, queries) do

end
end
```

Erlang:

```
-spec substring_xor_queries(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [[integer()]].
substring_xor_queries(S, Queries) ->
.
```

Racket:

```
(define/contract (substring-xor-queries s queries)
(-> string? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
vector<vector<int>> substringXorQueries(string s, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int[][] substringXorQueries(String s, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Substring XOR Queries
Difficulty: Medium
Tags: array, string, tree, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def substringXorQueries(self, s: str, queries: List[List[int]]) ->
List[List[int]]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def substringXorQueries(self, s, queries):
"""
:type s: str
:type queries: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {number[][]} queries
* @return {number[][]}
*/
var substringXorQueries = function(s, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function substringXorQueries(s: string, queries: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* 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[][] SubstringXorQueries(string s, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Substring XOR Queries
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Return an array of arrays of size *returnSize.
```

```
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** substringXorQueries(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Substring XOR Queries
// Difficulty: Medium
// Tags: array, string, tree, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func substringXorQueries(s string, queries [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun substringXorQueries(s: String, queries: Array<IntArray>): Array<IntArray>
{

}
}
```

Swift Solution:

```
class Solution {
func substringXorQueries(_ s: String, _ queries: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Substring XOR Queries
// Difficulty: Medium
// Tags: array, string, tree, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn substring_xor_queries(s: String, queries: Vec<Vec<i32>>) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Integer[][]}
def substring_xor_queries(s, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $queries
* @return Integer[][]
*/
function substringXorQueries($s, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> substringXorQueries(String s, List<List<int>> queries) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def substringXorQueries(s: String, queries: Array[Array[Int]]):
Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec substring_xor_queries(s :: String.t, queries :: [[integer]]) ::
[[integer]]
def substring_xor_queries(s, queries) do

end
end
```

Erlang Solution:

```
-spec substring_xor_queries(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [[integer()]].
substring_xor_queries(S, Queries) ->
.
```

Racket Solution:

```
(define/contract (substring-xor-queries s queries)
(-> string? (listof (listof exact-integer?)) (listof (listof
exact-integer?)))
)
```