Problem 2080: Range Frequency Queries

Problem Information

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

Problem Description

Design a data structure to find the

frequency

of a given value in a given subarray.

The

frequency

of a value in a subarray is the number of occurrences of that value in the subarray.

Implement the

RangeFreqQuery

class:

RangeFreqQuery(int[] arr)

Constructs an instance of the class with the given

0-indexed

integer array

arr

.

int query(int left, int right, int value)

Returns the

frequency

of

value

in the subarray

arr[left...right]

.

A

subarray

is a contiguous sequence of elements within an array.

arr[left...right]

denotes the subarray that contains the elements of

nums

between indices

left

and

right

(

inclusive

).

Example 1:

Input

["RangeFreqQuery", "query", "query"] [[[12, 33, 4, 56, 22, 2, 34, 33, 22, 12, 34, 56]], [1, 2, 4], [0, 11, 33]]

Output

[null, 1, 2]

Explanation

RangeFreqQuery rangeFreqQuery = new RangeFreqQuery([12, 33, 4, 56, 22, 2, 34, 33, 22, 12, 34, 56]); rangeFreqQuery.query(1, 2, 4); // return 1. The value 4 occurs 1 time in the subarray [33, 4] rangeFreqQuery.query(0, 11, 33); // return 2. The value 33 occurs 2 times in the whole array.

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i], value <= 10

4

0 <= left <= right < arr.length

At most

10

5

calls will be made to

query

Code Snippets

C++:

```
class RangeFreqQuery {
public:
RangeFreqQuery(vector<int>& arr) {

}

int query(int left, int right, int value) {

}
};

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery* obj = new RangeFreqQuery(arr);
* int param_1 = obj->query(left,right,value);
*/
```

Java:

```
class RangeFreqQuery {

public RangeFreqQuery(int[] arr) {

}

public int query(int left, int right, int value) {

}
}
```

```
/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = new RangeFreqQuery(arr);
* int param_1 = obj.query(left,right,value);
*/
```

Python3:

```
class RangeFreqQuery:

def __init__(self, arr: List[int]):



def query(self, left: int, right: int, value: int) -> int:




# Your RangeFreqQuery object will be instantiated and called as such:
# obj = RangeFreqQuery(arr)
# param_1 = obj.query(left,right,value)
```

Python:

```
class RangeFreqQuery(object):

def __init__(self, arr):
"""
:type arr: List[int]
"""



def query(self, left, right, value):
"""
:type left: int
:type right: int
:type value: int
:rtype: int
"""




# Your RangeFreqQuery object will be instantiated and called as such:
```

```
# obj = RangeFreqQuery(arr)
# param_1 = obj.query(left,right,value)
```

JavaScript:

```
/**
* @param {number[]} arr
*/
var RangeFreqQuery = function(arr) {

};

/**
* @param {number} left
* @param {number} right
* @param {number} value
* @return {number}
*/
RangeFreqQuery.prototype.query = function(left, right, value) {

};

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* var obj = new RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

TypeScript:

```
class RangeFreqQuery {
constructor(arr: number[]) {

}

query(left: number, right: number, value: number): number {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
```

```
* var obj = new RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

C#:

```
public class RangeFreqQuery {

public RangeFreqQuery(int[] arr) {

}

public int Query(int left, int right, int value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = new RangeFreqQuery(arr);
* int param_1 = obj.Query(left,right,value);
*/
```

C:

```



typedef struct {

} RangeFreqQuery;


RangeFreqQuery* rangeFreqQueryCreate(int* arr, int arrSize) {

}

int rangeFreqQueryQuery(RangeFreqQuery* obj, int left, int right, int value)
{

}
```

```
void rangeFreqQueryFree(RangeFreqQuery* obj) {

}

/**
* Your RangeFreqQuery struct will be instantiated and called as such:
* RangeFreqQuery* obj = rangeFreqQueryCreate(arr, arrSize);
* int param_1 = rangeFreqQueryQuery(obj, left, right, value);

* rangeFreqQueryFree(obj);
*/
```

Go:

```
type RangeFreqQuery struct {

}


func Constructor(arr []int) RangeFreqQuery {

}


func (this *RangeFreqQuery) Query(left int, right int, value int) int {

}


/**
* Your RangeFreqQuery object will be instantiated and called as such:
* obj := Constructor(arr);
* param_1 := obj.Query(left,right,value);
*/
```

Kotlin:

```
class RangeFreqQuery(arr: IntArray) {

fun query(left: Int, right: Int, value: Int): Int {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* var obj = RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

Swift:

```
class RangeFreqQuery {

init(_ arr: [Int]) {

}

func query(_ left: Int, _ right: Int, _ value: Int) -> Int {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* let obj = RangeFreqQuery(arr)
* let ret_1: Int = obj.query(left, right, value)
*/
```

Rust:

```
struct RangeFreqQuery {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl RangeFreqQuery {
```

```
fn new(arr: Vec<i32>) -> Self {

}

fn query(&self, left: i32, right: i32, value: i32) -> i32 {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* let obj = RangeFreqQuery::new(arr);
* let ret_1: i32 = obj.query(left, right, value);
*/
```

Ruby:

```
class RangeFreqQuery

=begin
:type arr: Integer[]
=end
def initialize(arr)

end


=begin
:type left: Integer
:type right: Integer
:type value: Integer
:rtype: Integer
=end
def query(left, right, value)

end


end

# Your RangeFreqQuery object will be instantiated and called as such:
# obj = RangeFreqQuery.new(arr)
# param_1 = obj.query(left, right, value)
```

PHP:

```
class RangeFreqQuery {
/**
* @param Integer[] $arr
*/
function __construct($arr) {

}

/**
* @param Integer $left
* @param Integer $right
* @param Integer $value
* @return Integer
*/
function query($left, $right, $value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* $obj = RangeFreqQuery($arr);
* $ret_1 = $obj->query($left, $right, $value);
*/
```

Dart:

```
class RangeFreqQuery {

RangeFreqQuery(List<int> arr) {

}

int query(int left, int right, int value) {

}
}
```

```
/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = RangeFreqQuery(arr);
* int param1 = obj.query(left,right,value);
*/
```

Scala:

```
class RangeFreqQuery(_arr: Array[Int]) {

def query(left: Int, right: Int, value: Int): Int = {

}

}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* val obj = new RangeFreqQuery(arr)
* val param_1 = obj.query(left,right,value)
*/
```

Elixir:

```
defmodule RangeFreqQuery do
@spec init_(arr :: [integer]) :: any
def init_(arr) do

end

@spec query(left :: integer, right :: integer, value :: integer) :: integer
def query(left, right, value) do

end
end

# Your functions will be called as such:
# RangeFreqQuery.init_(arr)
# param_1 = RangeFreqQuery.query(left, right, value)

# RangeFreqQuery.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang:

```
-spec range_freq_query_init_(Arr :: [integer()]) -> any().
range_freq_query_init_(Arr) ->
.

-spec range_freq_query_query(Left :: integer(), Right :: integer(), Value ::
integer()) -> integer().
range_freq_query_query(Left, Right, Value) ->
.



%% Your functions will be called as such:
%% range_freq_query_init_(Arr),
%% Param_1 = range_freq_query_query(Left, Right, Value),

%% range_freq_query_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket:

```
(define range-freq-query%
(class object%
(super-new)

; arr : (listof exact-integer?)
(init-field
arr)

; query : exact-integer? exact-integer? exact-integer? -> exact-integer?
(define/public (query left right value)
)))

;; Your range-freq-query% object will be instantiated and called as such:
;; (define obj (new range-freq-query% [arr arr]))
;; (define param_1 (send obj query left right value))
```

Solutions

C++ Solution:

```
/*
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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 RangeFreqQuery {
public:
RangeFreqQuery(vector<int>& arr) {

}

int query(int left, int right, int value) {

}
};

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery* obj = new RangeFreqQuery(arr);
* int param_1 = obj->query(left,right,value);
*/
```

Java Solution:

```
/**
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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 RangeFreqQuery {

public RangeFreqQuery(int[] arr) {
}

public int query(int left, int right, int value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = new RangeFreqQuery(arr);
* int param_1 = obj.query(left,right,value);
*/
```

Python3 Solution:

```
"""
Problem: Range Frequency Queries
Difficulty: Medium
Tags: array, tree, hash, search

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 RangeFreqQuery:

def __init__(self, arr: List[int]):



def query(self, left: int, right: int, value: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class RangeFreqQuery(object):

def __init__(self, arr):
"""
```

```
:type arr: List[int]
"""


def query(self, left, right, value):
"""
:type left: int
:type right: int
:type value: int
:rtype: int
"""




# Your RangeFreqQuery object will be instantiated and called as such:
# obj = RangeFreqQuery(arr)
# param_1 = obj.query(left,right,value)
```

JavaScript Solution:

```
/**
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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 {number[]} arr
*/
var RangeFreqQuery = function(arr) {

};

/**
* @param {number} left
* @param {number} right
* @param {number} value
* @return {number}
*/
RangeFreqQuery.prototype.query = function(left, right, value) {

};

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* var obj = new RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

TypeScript Solution:

```
/**
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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 RangeFreqQuery {
constructor(arr: number[]) {

}

query(left: number, right: number, value: number): number {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* var obj = new RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

C# Solution:

```
/*
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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 RangeFreqQuery {

public RangeFreqQuery(int[] arr) {

}

public int Query(int left, int right, int value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = new RangeFreqQuery(arr);
* int param_1 = obj.Query(left,right,value);
*/
```

C Solution:

```
/*
* Problem: Range Frequency Queries
* Difficulty: Medium
* Tags: array, tree, hash, search
*
* 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
*/
```

```c
typedef struct {

} RangeFreqQuery;


RangeFreqQuery* rangeFreqQueryCreate(int* arr, int arrSize) {

}

int rangeFreqQueryQuery(RangeFreqQuery* obj, int left, int right, int value)
{

}

void rangeFreqQueryFree(RangeFreqQuery* obj) {

}

/**
* Your RangeFreqQuery struct will be instantiated and called as such:
* RangeFreqQuery* obj = rangeFreqQueryCreate(arr, arrSize);
* int param_1 = rangeFreqQueryQuery(obj, left, right, value);

* rangeFreqQueryFree(obj);
*/
```

Go Solution:

```go
// Problem: Range Frequency Queries
// Difficulty: Medium
// Tags: array, tree, hash, search
//
// 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

type RangeFreqQuery struct {

}
```

```
func Constructor(arr []int) RangeFreqQuery {

}


func (this *RangeFreqQuery) Query(left int, right int, value int) int {

}


/**
* Your RangeFreqQuery object will be instantiated and called as such:
* obj := Constructor(arr);
* param_1 := obj.Query(left,right,value);
*/
```

Kotlin Solution:

```
class RangeFreqQuery(arr: IntArray) {

fun query(left: Int, right: Int, value: Int): Int {

}

}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* var obj = RangeFreqQuery(arr)
* var param_1 = obj.query(left,right,value)
*/
```

Swift Solution:

```
class RangeFreqQuery {

init(_ arr: [Int]) {

}
```

```swift
func query(_ left: Int, _ right: Int, _ value: Int) -> Int {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* let obj = RangeFreqQuery(arr)
* let ret_1: Int = obj.query(left, right, value)
*/
```

Rust Solution:

```rust
// Problem: Range Frequency Queries
// Difficulty: Medium
// Tags: array, tree, hash, search
//
// 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

struct RangeFreqQuery {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl RangeFreqQuery {

fn new(arr: Vec<i32>) -> Self {

}

fn query(&self, left: i32, right: i32, value: i32) -> i32 {

}
}
```

```
/**
* Your RangeFreqQuery object will be instantiated and called as such:
* let obj = RangeFreqQuery::new(arr);
* let ret_1: i32 = obj.query(left, right, value);
*/
```

Ruby Solution:

```
class RangeFreqQuery

=begin
:type arr: Integer[]
=end
def initialize(arr)

end



=begin
:type left: Integer
:type right: Integer
:type value: Integer
:rtype: Integer
=end
def query(left, right, value)

end



end

# Your RangeFreqQuery object will be instantiated and called as such:
# obj = RangeFreqQuery.new(arr)
# param_1 = obj.query(left, right, value)
```

PHP Solution:

```
class RangeFreqQuery {
/**
* @param Integer[] $arr
*/
function __construct($arr) {

}

/**
* @param Integer $left
* @param Integer $right
* @param Integer $value
* @return Integer
*/
function query($left, $right, $value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* $obj = RangeFreqQuery($arr);
* $ret_1 = $obj->query($left, $right, $value);
*/
```

Dart Solution:

```
class RangeFreqQuery {

RangeFreqQuery(List<int> arr) {

}

int query(int left, int right, int value) {

}
}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* RangeFreqQuery obj = RangeFreqQuery(arr);
* int param1 = obj.query(left,right,value);
*/
```

Scala Solution:

```
class RangeFreqQuery(_arr: Array[Int]) {

def query(left: Int, right: Int, value: Int): Int = {

}

}

/**
* Your RangeFreqQuery object will be instantiated and called as such:
* val obj = new RangeFreqQuery(arr)
* val param_1 = obj.query(left,right,value)
*/
```

Elixir Solution:

```
defmodule RangeFreqQuery do
@spec init_(arr :: [integer]) :: any
def init_(arr) do

end

@spec query(left :: integer, right :: integer, value :: integer) :: integer
def query(left, right, value) do

end
end

# Your functions will be called as such:
# RangeFreqQuery.init_(arr)
# param_1 = RangeFreqQuery.query(left, right, value)

# RangeFreqQuery.init_ will be called before every test case, in which you
can do some necessary initializations.
```

Erlang Solution:

```
-spec range_freq_query_init_(Arr :: [integer()]) -> any().
range_freq_query_init_(Arr) ->
.

-spec range_freq_query_query(Left :: integer(), Right :: integer(), Value ::
```

```
integer()) -> integer().
range_freq_query_query(Left, Right, Value) ->
.



%% Your functions will be called as such:
%% range_freq_query_init_(Arr),
%% Param_1 = range_freq_query_query(Left, Right, Value),

%% range_freq_query_init_ will be called before every test case, in which you
can do some necessary initializations.
```

Racket Solution:

```
(define range-freq-query%
(class object%
(super-new)

; arr : (listof exact-integer?)
(init-field
arr)

; query : exact-integer? exact-integer? exact-integer? -> exact-integer?
(define/public (query left right value)
)))

;; Your range-freq-query% object will be instantiated and called as such:
;; (define obj (new range-freq-query% [arr arr]))
;; (define param_1 (send obj query left right value))
```