Problem 715: Range Module

Problem Information

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

Problem Description

A Range Module is a module that tracks ranges of numbers. Design a data structure to track the ranges represented as

half-open intervals

and query about them.

A

half-open interval

[left, right)

denotes all the real numbers

x

where

left <= x < right

.

Implement the

RangeModule

class:

RangeModule()

Initializes the object of the data structure.

void addRange(int left, int right)

Adds the

half-open interval

[left, right)

, tracking every real number in that interval. Adding an interval that partially overlaps with currently tracked numbers should add any numbers in the interval

[left, right)

that are not already tracked.

boolean queryRange(int left, int right)

Returns

true

if every real number in the interval

[left, right)

is currently being tracked, and

false

otherwise.

void removeRange(int left, int right)

Stops tracking every real number currently being tracked in the

half-open interval

[left, right)

.

Example 1:

Input

["RangeModule", "addRange", "removeRange", "queryRange", "queryRange", "queryRange"]
[[], [10, 20], [14, 16], [10, 14], [13, 15], [16, 17]]

Output

[null, null, null, true, false, true]

Explanation

RangeModule rangeModule = new RangeModule(); rangeModule.addRange(10, 20); rangeModule.removeRange(14, 16); rangeModule.queryRange(10, 14); // return True,(Every number in [10, 14) is being tracked) rangeModule.queryRange(13, 15); // return False,(Numbers like 14, 14.03, 14.17 in [13, 15) are not being tracked) rangeModule.queryRange(16, 17); // return True, (The number 16 in [16, 17) is still being tracked, despite the remove operation)

Constraints:

1 <= left < right <= 10

9

At most

10

4

calls will be made to

addRange

,

queryRange

, and

removeRange

.

Code Snippets

C++:

```
class RangeModule {
public:
RangeModule() {

}

void addRange(int left, int right) {

}

bool queryRange(int left, int right) {

}

void removeRange(int left, int right) {

}
};

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

```
* RangeModule* obj = new RangeModule();
* obj->addRange(left,right);
* bool param_2 = obj->queryRange(left,right);
* obj->removeRange(left,right);
*/
```

Java:

```
class RangeModule {

public RangeModule() {

}

public void addRange(int left, int right) {

}

public boolean queryRange(int left, int right) {

}

public void removeRange(int left, int right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = new RangeModule();
* obj.addRange(left,right);
* boolean param_2 = obj.queryRange(left,right);
* obj.removeRange(left,right);
*/
```

Python3:

```
class RangeModule:

def __init__(self):
```

```
def addRange(self, left: int, right: int) -> None:


def queryRange(self, left: int, right: int) -> bool:


def removeRange(self, left: int, right: int) -> None:



# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule()
# obj.addRange(left,right)
# param_2 = obj.queryRange(left,right)
# obj.removeRange(left,right)
```

Python:

```
class RangeModule(object):

def __init__(self):


def addRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: None
"""


def queryRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: bool
"""


def removeRange(self, left, right):
"""
:type left: int
```

```
:type right: int
:rtype: None
"""



# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule()
# obj.addRange(left,right)
# param_2 = obj.queryRange(left,right)
# obj.removeRange(left,right)
```

JavaScript:

```
var RangeModule = function() {

};

/**
* @param {number} left
* @param {number} right
* @return {void}
*/
RangeModule.prototype.addRange = function(left, right) {

};

/**
* @param {number} left
* @param {number} right
* @return {boolean}
*/
RangeModule.prototype.queryRange = function(left, right) {

};

/**
* @param {number} left
* @param {number} right
* @return {void}
*/
```

```
RangeModule.prototype.removeRange = function(left, right) {

};

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = new RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

TypeScript:

```
class RangeModule {
constructor() {

}

addRange(left: number, right: number): void {

}

queryRange(left: number, right: number): boolean {

}

removeRange(left: number, right: number): void {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = new RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

C#:

```
public class RangeModule {

public RangeModule() {

}

public void AddRange(int left, int right) {

}

public bool QueryRange(int left, int right) {

}

public void RemoveRange(int left, int right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = new RangeModule();
* obj.AddRange(left,right);
* bool param_2 = obj.QueryRange(left,right);
* obj.RemoveRange(left,right);
*/
```

C:

```
typedef struct {

} RangeModule;


RangeModule* rangeModuleCreate() {

}

void rangeModuleAddRange(RangeModule* obj, int left, int right) {
```

```
}

bool rangeModuleQueryRange(RangeModule* obj, int left, int right) {

}

void rangeModuleRemoveRange(RangeModule* obj, int left, int right) {

}

void rangeModuleFree(RangeModule* obj) {

}

/**
* Your RangeModule struct will be instantiated and called as such:
* RangeModule* obj = rangeModuleCreate();
* rangeModuleAddRange(obj, left, right);

* bool param_2 = rangeModuleQueryRange(obj, left, right);

* rangeModuleRemoveRange(obj, left, right);

* rangeModuleFree(obj);
*/
```

Go:

```
type RangeModule struct {

}


func Constructor() RangeModule {

}


func (this *RangeModule) AddRange(left int, right int) {

}
```

```
func (this *RangeModule) QueryRange(left int, right int) bool {

}


func (this *RangeModule) RemoveRange(left int, right int) {

}


/**
* Your RangeModule object will be instantiated and called as such:
* obj := Constructor();
* obj.AddRange(left,right);
* param_2 := obj.QueryRange(left,right);
* obj.RemoveRange(left,right);
*/
```

Kotlin:

```
class RangeModule() {

fun addRange(left: Int, right: Int) {

}

fun queryRange(left: Int, right: Int): Boolean {

}

fun removeRange(left: Int, right: Int) {

}

}

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
```

```
* obj.removeRange(left,right)
*/
```

Swift:

```
class RangeModule {

init() {

}

func addRange(_ left: Int, _ right: Int) {

}

func queryRange(_ left: Int, _ right: Int) -> Bool {

}

func removeRange(_ left: Int, _ right: Int) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* let obj = RangeModule()
* obj.addRange(left, right)
* let ret_2: Bool = obj.queryRange(left, right)
* obj.removeRange(left, right)
*/
```

Rust:

```
struct RangeModule {

}


/**
* `&self` means the method takes an immutable reference.
```

```
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl RangeModule {

fn new() -> Self {

}

fn add_range(&self, left: i32, right: i32) {

}

fn query_range(&self, left: i32, right: i32) -> bool {

}

fn remove_range(&self, left: i32, right: i32) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* let obj = RangeModule::new();
* obj.add_range(left, right);
* let ret_2: bool = obj.query_range(left, right);
* obj.remove_range(left, right);
*/
```

Ruby:

```
class RangeModule
def initialize()

end


=begin
:type left: Integer
:type right: Integer
:rtype: Void
=end
```

```
def add_range(left, right)

end


=begin
:type left: Integer
:type right: Integer
:rtype: Boolean
=end
def query_range(left, right)

end


=begin
:type left: Integer
:type right: Integer
:rtype: Void
=end
def remove_range(left, right)

end



end

# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule.new()
# obj.add_range(left, right)
# param_2 = obj.query_range(left, right)
# obj.remove_range(left, right)
```

PHP:

```
class RangeModule {
/**
*/
function __construct() {

}
```

```
/**
* @param Integer $left
* @param Integer $right
* @return NULL
*/
function addRange($left, $right) {

}

/**
* @param Integer $left
* @param Integer $right
* @return Boolean
*/
function queryRange($left, $right) {

}

/**
* @param Integer $left
* @param Integer $right
* @return NULL
*/
function removeRange($left, $right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* $obj = RangeModule();
* $obj->addRange($left, $right);
* $ret_2 = $obj->queryRange($left, $right);
* $obj->removeRange($left, $right);
*/
```

Dart:

```
class RangeModule {

RangeModule() {
```

```
}

void addRange(int left, int right) {

}

bool queryRange(int left, int right) {

}

void removeRange(int left, int right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = RangeModule();
* obj.addRange(left,right);
* bool param2 = obj.queryRange(left,right);
* obj.removeRange(left,right);
*/
```

Scala:

```
class RangeModule() {

def addRange(left: Int, right: Int): Unit = {

}

def queryRange(left: Int, right: Int): Boolean = {

}

def removeRange(left: Int, right: Int): Unit = {

}

}

/**
```

```
* Your RangeModule object will be instantiated and called as such:
* val obj = new RangeModule()
* obj.addRange(left,right)
* val param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

Elixir:

```
defmodule RangeModule do
@spec init_() :: any
def init_() do

end

@spec add_range(left :: integer, right :: integer) :: any
def add_range(left, right) do

end

@spec query_range(left :: integer, right :: integer) :: boolean
def query_range(left, right) do

end

@spec remove_range(left :: integer, right :: integer) :: any
def remove_range(left, right) do

end
end

# Your functions will be called as such:
# RangeModule.init_()
# RangeModule.add_range(left, right)
# param_2 = RangeModule.query_range(left, right)
# RangeModule.remove_range(left, right)

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

Erlang:

```
-spec range_module_init_() -> any().
range_module_init_() ->
.

-spec range_module_add_range(Left :: integer(), Right :: integer()) -> any().
range_module_add_range(Left, Right) ->
.

-spec range_module_query_range(Left :: integer(), Right :: integer()) ->
boolean().
range_module_query_range(Left, Right) ->
.

-spec range_module_remove_range(Left :: integer(), Right :: integer()) ->
any().
range_module_remove_range(Left, Right) ->
.


%% Your functions will be called as such:
%% range_module_init_(),
%% range_module_add_range(Left, Right),
%% Param_2 = range_module_query_range(Left, Right),
%% range_module_remove_range(Left, Right),

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

Racket:

```
(define range-module%
(class object%
(super-new)

(init-field)

; add-range : exact-integer? exact-integer? -> void?
(define/public (add-range left right)
)
; query-range : exact-integer? exact-integer? -> boolean?
(define/public (query-range left right)
)
; remove-range : exact-integer? exact-integer? -> void?
```

```
(define/public (remove-range left right)
)))

;; Your range-module% object will be instantiated and called as such:
;; (define obj (new range-module%))
;; (send obj add-range left right)
;; (define param_2 (send obj query-range left right))
;; (send obj remove-range left right)
```

Solutions

C++ Solution:

```
/*
* Problem: Range Module
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class RangeModule {
public:
RangeModule() {

}

void addRange(int left, int right) {

}

bool queryRange(int left, int right) {

}

void removeRange(int left, int right) {

}
```

```
};

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule* obj = new RangeModule();
* obj->addRange(left,right);
* bool param_2 = obj->queryRange(left,right);
* obj->removeRange(left,right);
*/
```

Java Solution:

```
/**
* Problem: Range Module
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class RangeModule {

public RangeModule() {

}

public void addRange(int left, int right) {

}

public boolean queryRange(int left, int right) {

}

public void removeRange(int left, int right) {

}
}
```

```
/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = new RangeModule();
* obj.addRange(left,right);
* boolean param_2 = obj.queryRange(left,right);
* obj.removeRange(left,right);
*/
```

Python3 Solution:

```
"""
Problem: Range Module
Difficulty: Hard
Tags: tree

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

class RangeModule:

def __init__(self):


def addRange(self, left: int, right: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class RangeModule(object):

def __init__(self):


def addRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: None
```

```
"""



def queryRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: bool
"""



def removeRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: None
"""




# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule()
# obj.addRange(left,right)
# param_2 = obj.queryRange(left,right)
# obj.removeRange(left,right)
```

JavaScript Solution:

```
/**
* Problem: Range Module
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/



var RangeModule = function() {
```

```
};

/**
* @param {number} left
* @param {number} right
* @return {void}
*/
RangeModule.prototype.addRange = function(left, right) {

};

/**
* @param {number} left
* @param {number} right
* @return {boolean}
*/
RangeModule.prototype.queryRange = function(left, right) {

};

/**
* @param {number} left
* @param {number} right
* @return {void}
*/
RangeModule.prototype.removeRange = function(left, right) {

};

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = new RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

TypeScript Solution:

```
/**
* Problem: Range Module
```

```
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class RangeModule {
constructor() {

}

addRange(left: number, right: number): void {

}

queryRange(left: number, right: number): boolean {

}

removeRange(left: number, right: number): void {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = new RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

C# Solution:

```
/*
* Problem: Range Module
* Difficulty: Hard
* Tags: tree
*
```

```
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class RangeModule {

public RangeModule() {

}

public void AddRange(int left, int right) {

}

public bool QueryRange(int left, int right) {

}

public void RemoveRange(int left, int right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = new RangeModule();
* obj.AddRange(left,right);
* bool param_2 = obj.QueryRange(left,right);
* obj.RemoveRange(left,right);
*/
```

C Solution:

```
/*
* Problem: Range Module
* Difficulty: Hard
* Tags: tree
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
```

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



typedef struct {

} RangeModule;


RangeModule* rangeModuleCreate() {

}

void rangeModuleAddRange(RangeModule* obj, int left, int right) {

}

bool rangeModuleQueryRange(RangeModule* obj, int left, int right) {

}

void rangeModuleRemoveRange(RangeModule* obj, int left, int right) {

}

void rangeModuleFree(RangeModule* obj) {

}

/**
* Your RangeModule struct will be instantiated and called as such:
* RangeModule* obj = rangeModuleCreate();
* rangeModuleAddRange(obj, left, right);

* bool param_2 = rangeModuleQueryRange(obj, left, right);

* rangeModuleRemoveRange(obj, left, right);

* rangeModuleFree(obj);
*/
```

Go Solution:

```
// Problem: Range Module
// Difficulty: Hard
// Tags: tree
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

type RangeModule struct {

}


func Constructor() RangeModule {

}


func (this *RangeModule) AddRange(left int, right int) {

}


func (this *RangeModule) QueryRange(left int, right int) bool {

}


func (this *RangeModule) RemoveRange(left int, right int) {

}


/**
* Your RangeModule object will be instantiated and called as such:
* obj := Constructor();
* obj.AddRange(left,right);
* param_2 := obj.QueryRange(left,right);
* obj.RemoveRange(left,right);
*/
```

Kotlin Solution:

```
class RangeModule() {

fun addRange(left: Int, right: Int) {

}

fun queryRange(left: Int, right: Int): Boolean {

}

fun removeRange(left: Int, right: Int) {

}

}

/**
* Your RangeModule object will be instantiated and called as such:
* var obj = RangeModule()
* obj.addRange(left,right)
* var param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

Swift Solution:

```
class RangeModule {

init() {

}

func addRange(_ left: Int, _ right: Int) {

}

func queryRange(_ left: Int, _ right: Int) -> Bool {

}
```

```
func removeRange(_ left: Int, _ right: Int) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* let obj = RangeModule()
* obj.addRange(left, right)
* let ret_2: Bool = obj.queryRange(left, right)
* obj.removeRange(left, right)
*/
```

Rust Solution:

```
// Problem: Range Module
// Difficulty: Hard
// Tags: tree
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

struct RangeModule {

}



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

fn new() -> Self {

}

fn add_range(&self, left: i32, right: i32) {
```

```
}

fn query_range(&self, left: i32, right: i32) -> bool {

}

fn remove_range(&self, left: i32, right: i32) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* let obj = RangeModule::new();
* obj.add_range(left, right);
* let ret_2: bool = obj.query_range(left, right);
* obj.remove_range(left, right);
*/
```

Ruby Solution:

```
class RangeModule
def initialize()

end


=begin
:type left: Integer
:type right: Integer
:rtype: Void
=end
def add_range(left, right)

end


=begin
:type left: Integer
:type right: Integer
:rtype: Boolean
```

```
=end
def query_range(left, right)

end


=begin
:type left: Integer
:type right: Integer
:rtype: Void
=end
def remove_range(left, right)

end


end

# Your RangeModule object will be instantiated and called as such:
# obj = RangeModule.new()
# obj.add_range(left, right)
# param_2 = obj.query_range(left, right)
# obj.remove_range(left, right)
```

PHP Solution:

```
class RangeModule {
/**
*/
function __construct() {

}

/**
* @param Integer $left
* @param Integer $right
* @return NULL
*/
function addRange($left, $right) {

}
```

```
/**
* @param Integer $left
* @param Integer $right
* @return Boolean
*/
function queryRange($left, $right) {

}

/**
* @param Integer $left
* @param Integer $right
* @return NULL
*/
function removeRange($left, $right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* $obj = RangeModule();
* $obj->addRange($left, $right);
* $ret_2 = $obj->queryRange($left, $right);
* $obj->removeRange($left, $right);
*/
```

Dart Solution:

```
class RangeModule {

RangeModule() {

}

void addRange(int left, int right) {

}

bool queryRange(int left, int right) {
```

```
}

void removeRange(int left, int right) {

}
}

/**
* Your RangeModule object will be instantiated and called as such:
* RangeModule obj = RangeModule();
* obj.addRange(left,right);
* bool param2 = obj.queryRange(left,right);
* obj.removeRange(left,right);
*/
```

Scala Solution:

```
class RangeModule() {

def addRange(left: Int, right: Int): Unit = {

}

def queryRange(left: Int, right: Int): Boolean = {

}

def removeRange(left: Int, right: Int): Unit = {

}

}

/**
* Your RangeModule object will be instantiated and called as such:
* val obj = new RangeModule()
* obj.addRange(left,right)
* val param_2 = obj.queryRange(left,right)
* obj.removeRange(left,right)
*/
```

Elixir Solution:

```
defmodule RangeModule do
@spec init_() :: any
def init_() do

end

@spec add_range(left :: integer, right :: integer) :: any
def add_range(left, right) do

end

@spec query_range(left :: integer, right :: integer) :: boolean
def query_range(left, right) do

end

@spec remove_range(left :: integer, right :: integer) :: any
def remove_range(left, right) do

end
end

# Your functions will be called as such:
# RangeModule.init_()
# RangeModule.add_range(left, right)
# param_2 = RangeModule.query_range(left, right)
# RangeModule.remove_range(left, right)

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

Erlang Solution:

```
-spec range_module_init_() -> any().
range_module_init_() ->
.

-spec range_module_add_range(Left :: integer(), Right :: integer()) -> any().
range_module_add_range(Left, Right) ->
.
```

```
-spec range_module_query_range(Left :: integer(), Right :: integer()) ->
boolean().
range_module_query_range(Left, Right) ->
.

-spec range_module_remove_range(Left :: integer(), Right :: integer()) ->
any().
range_module_remove_range(Left, Right) ->
.


%% Your functions will be called as such:
%% range_module_init_(),
%% range_module_add_range(Left, Right),
%% Param_2 = range_module_query_range(Left, Right),
%% range_module_remove_range(Left, Right),

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

Racket Solution:

```
(define range-module%
(class object%
(super-new)

(init-field)

; add-range : exact-integer? exact-integer? -> void?
(define/public (add-range left right)
)
; query-range : exact-integer? exact-integer? -> boolean?
(define/public (query-range left right)
)
; remove-range : exact-integer? exact-integer? -> void?
(define/public (remove-range left right)
)))

;; Your range-module% object will be instantiated and called as such:
;; (define obj (new range-module%))
```

```
;; (send obj add-range left right)
;; (define param_2 (send obj query-range left right))
;; (send obj remove-range left right)
```