Problem 352: Data Stream as Disjoint Intervals

Problem Information

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

Problem Description

Given a data stream input of non-negative integers

a

1

, a

2

, ..., a

n

, summarize the numbers seen so far as a list of disjoint intervals.

Implement the

SummaryRanges

class:

SummaryRanges()

Initializes the object with an empty stream.

void addNum(int value)

Adds the integer

value

to the stream.

int[][] getIntervals()

Returns a summary of the integers in the stream currently as a list of disjoint intervals

[start

i

, end

i

]

. The answer should be sorted by

start

i

.

Example 1:

Input

["SummaryRanges", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals", "addNum", "getIntervals"] [[], [1], [], [3], [], [7], [], [2], [], [6], []]

Output

[null, null, [[1, 1]], null, [[1, 1], [3, 3]], null, [[1, 1], [3, 3], [7, 7]], null, [[1, 3], [7, 7]], null, [[1, 3], [6, 7]]]

Explanation

SummaryRanges summaryRanges = new SummaryRanges(); summaryRanges.addNum(1); // arr = [1] summaryRanges.getIntervals(); // return [[1, 1]] summaryRanges.addNum(3); // arr = [1, 3] summaryRanges.getIntervals(); // return [[1, 1], [3, 3]] summaryRanges.addNum(7); // arr = [1, 3, 7] summaryRanges.getIntervals(); // return [[1, 1], [3, 3], [7, 7]] summaryRanges.addNum(2); // arr = [1, 2, 3, 7] summaryRanges.getIntervals(); // return [[1, 3], [7, 7]] summaryRanges.addNum(6); // arr = [1, 2, 3, 6, 7] summaryRanges.getIntervals(); // return [[1, 3], [6, 7]]

Constraints:

0 <= value <= 10

4

At most

3 * 10

4

calls will be made to

addNum

and

getIntervals

.

At most

10

2

calls will be made to

getIntervals

.

Follow up:

What if there are lots of merges and the number of disjoint intervals is small compared to the size of the data stream?

Code Snippets

C++:

```
class SummaryRanges {
public:
SummaryRanges() {

}

void addNum(int value) {

}

vector<vector<int>> getIntervals() {

}
};

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges* obj = new SummaryRanges();
* obj->addNum(value);
* vector<vector<int>> param_2 = obj->getIntervals();
*/
```

Java:

```
class SummaryRanges {

public SummaryRanges() {

}

public void addNum(int value) {

}

public int[][] getIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = new SummaryRanges();
* obj.addNum(value);
* int[][] param_2 = obj.getIntervals();
*/
```

Python3:

```
class SummaryRanges:

def __init__(self):



def addNum(self, value: int) -> None:



def getIntervals(self) -> List[List[int]]:




# Your SummaryRanges object will be instantiated and called as such:
# obj = SummaryRanges()
# obj.addNum(value)
# param_2 = obj.getIntervals()
```

Python:

```
class SummaryRanges(object):

def __init__(self):



def addNum(self, value):
"""
:type value: int
:rtype: None
"""



def getIntervals(self):
"""
:rtype: List[List[int]]
"""




# Your SummaryRanges object will be instantiated and called as such:
# obj = SummaryRanges()
# obj.addNum(value)
# param_2 = obj.getIntervals()
```

JavaScript:

```

var SummaryRanges = function() {

};

/**
* @param {number} value
* @return {void}
*/
SummaryRanges.prototype.addNum = function(value) {

};

/**
* @return {number[][]}
*/
SummaryRanges.prototype.getIntervals = function() {
};

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = new SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

TypeScript:

```
class SummaryRanges {
constructor() {

}

addNum(value: number): void {

}

getIntervals(): number[][] {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = new SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

C#:

```
public class SummaryRanges {

public SummaryRanges() {

}

public void AddNum(int value) {
}

public int[][] GetIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = new SummaryRanges();
* obj.AddNum(value);
* int[][] param_2 = obj.GetIntervals();
*/
```

C:

```
typedef struct {

} SummaryRanges;


SummaryRanges* summaryRangesCreate() {

}

void summaryRangesAddNum(SummaryRanges* obj, int value) {

}

int** summaryRangesGetIntervals(SummaryRanges* obj, int* retSize, int**
retColSize) {

}

void summaryRangesFree(SummaryRanges* obj) {

}
```

```
/**
* Your SummaryRanges struct will be instantiated and called as such:
* SummaryRanges* obj = summaryRangesCreate();
* summaryRangesAddNum(obj, value);

* int** param_2 = summaryRangesGetIntervals(obj, retSize, retColSize);

* summaryRangesFree(obj);
*/
```

Go:

```
type SummaryRanges struct {

}


func Constructor() SummaryRanges {

}


func (this *SummaryRanges) AddNum(value int) {

}


func (this *SummaryRanges) GetIntervals() [][]int {

}


/**
* Your SummaryRanges object will be instantiated and called as such:
* obj := Constructor();
* obj.AddNum(value);
* param_2 := obj.GetIntervals();
*/
```

Kotlin:

```
class SummaryRanges() {

fun addNum(value: Int) {

}

fun getIntervals(): Array<IntArray> {

}

}

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

Swift:

```

class SummaryRanges {

init() {

}

func addNum(_ value: Int) {

}

func getIntervals() -> [[Int]] {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* let obj = SummaryRanges()
* obj.addNum(value)
* let ret_2: [[Int]] = obj.getIntervals()
*/
```

Rust:

```
struct SummaryRanges {

}



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

fn new() -> Self {

}

fn add_num(&self, value: i32) {

}

fn get_intervals(&self) -> Vec<Vec<i32>> {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* let obj = SummaryRanges::new();
* obj.add_num(value);
* let ret_2: Vec<Vec<i32>> = obj.get_intervals();
*/
```

Ruby:

```
class SummaryRanges
def initialize()

end



=begin
```

```
:type value: Integer
:rtype: Void
=end
def add_num(value)

end



=begin
:rtype: Integer[][]
=end
def get_intervals()

end



end

# Your SummaryRanges object will be instantiated and called as such:
# obj = SummaryRanges.new()
# obj.add_num(value)
# param_2 = obj.get_intervals()
```

PHP:

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

}

/**
* @param Integer $value
* @return NULL
*/
function addNum($value) {

}

/**
* @return Integer[][]
*/
function getIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* $obj = SummaryRanges();
* $obj->addNum($value);
* $ret_2 = $obj->getIntervals();
*/
```

Dart:

```
class SummaryRanges {

SummaryRanges() {

}

void addNum(int value) {

}

List<List<int>> getIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = SummaryRanges();
* obj.addNum(value);
* List<List<int>> param2 = obj.getIntervals();
*/
```

Scala:

```
class SummaryRanges() {

def addNum(value: Int): Unit = {
}

def getIntervals(): Array[Array[Int]] = {

}

}

/**
* Your SummaryRanges object will be instantiated and called as such:
* val obj = new SummaryRanges()
* obj.addNum(value)
* val param_2 = obj.getIntervals()
*/
```

Elixir:

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

end

@spec add_num(value :: integer) :: any
def add_num(value) do

end

@spec get_intervals() :: [[integer]]
def get_intervals() do

end
end

# Your functions will be called as such:
# SummaryRanges.init_()
# SummaryRanges.add_num(value)
# param_2 = SummaryRanges.get_intervals()

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

Erlang:

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

-spec summary_ranges_add_num(Value :: integer()) -> any().
summary_ranges_add_num(Value) ->
.

-spec summary_ranges_get_intervals() -> [[integer()]].
summary_ranges_get_intervals() ->
.


%% Your functions will be called as such:
%% summary_ranges_init_(),
%% summary_ranges_add_num(Value),
%% Param_2 = summary_ranges_get_intervals(),

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

Racket:

```
(define summary-ranges%
(class object%
(super-new)

(init-field)

; add-num : exact-integer? -> void?
(define/public (add-num value)
)
; get-intervals : -> (listof (listof exact-integer?))
(define/public (get-intervals)
)))

;; Your summary-ranges% object will be instantiated and called as such:
;; (define obj (new summary-ranges%))
;; (send obj add-num value)
;; (define param_2 (send obj get-intervals))
```

Solutions

C++ Solution:

```
/*
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SummaryRanges {
public:
SummaryRanges() {

}

void addNum(int value) {

}

vector<vector<int>> getIntervals() {

}
};

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges* obj = new SummaryRanges();
* obj->addNum(value);
* vector<vector<int>> param_2 = obj->getIntervals();
*/
```

Java Solution:

```
/**
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SummaryRanges {

public SummaryRanges() {

}

public void addNum(int value) {

}

public int[][] getIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = new SummaryRanges();
* obj.addNum(value);
* int[][] param_2 = obj.getIntervals();
*/
```

Python3 Solution:

```
"""
Problem: Data Stream as Disjoint Intervals
Difficulty: Hard
Tags: graph, hash, sort, search

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""
```

```
class SummaryRanges:

def __init__(self):



def addNum(self, value: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SummaryRanges(object):

def __init__(self):



def addNum(self, value):
"""
:type value: int
:rtype: None
"""



def getIntervals(self):
"""
:rtype: List[List[int]]
"""




# Your SummaryRanges object will be instantiated and called as such:
# obj = SummaryRanges()
# obj.addNum(value)
# param_2 = obj.getIntervals()
```

JavaScript Solution:

```
/**
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
```

```
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/


var SummaryRanges = function() {

};

/**
* @param {number} value
* @return {void}
*/
SummaryRanges.prototype.addNum = function(value) {

};

/**
* @return {number[][]}
*/
SummaryRanges.prototype.getIntervals = function() {

};

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = new SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

TypeScript Solution:

```
/**
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class SummaryRanges {
constructor() {

}

addNum(value: number): void {

}

getIntervals(): number[][] {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = new SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

C# Solution:

```
/*
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class SummaryRanges {

public SummaryRanges() {
}

public void AddNum(int value) {

}

public int[][] GetIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = new SummaryRanges();
* obj.AddNum(value);
* int[][] param_2 = obj.GetIntervals();
*/
```

C Solution:

```
/*
* Problem: Data Stream as Disjoint Intervals
* Difficulty: Hard
* Tags: graph, hash, sort, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {

} SummaryRanges;


SummaryRanges* summaryRangesCreate() {

}
```

```
void summaryRangesAddNum(SummaryRanges* obj, int value) {

}

int** summaryRangesGetIntervals(SummaryRanges* obj, int* retSize, int**
retColSize) {

}

void summaryRangesFree(SummaryRanges* obj) {

}

/**
* Your SummaryRanges struct will be instantiated and called as such:
* SummaryRanges* obj = summaryRangesCreate();
* summaryRangesAddNum(obj, value);

* int** param_2 = summaryRangesGetIntervals(obj, retSize, retColSize);

* summaryRangesFree(obj);
*/
```

Go Solution:

```
// Problem: Data Stream as Disjoint Intervals
// Difficulty: Hard
// Tags: graph, hash, sort, search
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type SummaryRanges struct {

}



func Constructor() SummaryRanges {
}


func (this *SummaryRanges) AddNum(value int) {

}


func (this *SummaryRanges) GetIntervals() [][]int {

}


/**
* Your SummaryRanges object will be instantiated and called as such:
* obj := Constructor();
* obj.AddNum(value);
* param_2 := obj.GetIntervals();
*/
```

Kotlin Solution:

```
class SummaryRanges() {

fun addNum(value: Int) {

}

fun getIntervals(): Array<IntArray> {

}

}

/**
* Your SummaryRanges object will be instantiated and called as such:
* var obj = SummaryRanges()
* obj.addNum(value)
* var param_2 = obj.getIntervals()
*/
```

Swift Solution:

```
class SummaryRanges {

init() {

}

func addNum(_ value: Int) {

}

func getIntervals() -> [[Int]] {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* let obj = SummaryRanges()
* obj.addNum(value)
* let ret_2: [[Int]] = obj.getIntervals()
*/
```

Rust Solution:

```
// Problem: Data Stream as Disjoint Intervals
// Difficulty: Hard
// Tags: graph, hash, sort, search
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

struct SummaryRanges {

}



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

```
*/
impl SummaryRanges {

fn new() -> Self {

}

fn add_num(&self, value: i32) {

}

fn get_intervals(&self) -> Vec<Vec<i32>> {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* let obj = SummaryRanges::new();
* obj.add_num(value);
* let ret_2: Vec<Vec<i32>> = obj.get_intervals();
*/
```

Ruby Solution:

```
class SummaryRanges
def initialize()

end


=begin
:type value: Integer
:rtype: Void
=end
def add_num(value)

end


=begin
```

```
:rtype: Integer[][]
=end
def get_intervals()

end



end

# Your SummaryRanges object will be instantiated and called as such:
# obj = SummaryRanges.new()
# obj.add_num(value)
# param_2 = obj.get_intervals()
```

PHP Solution:

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

}

/**
* @param Integer $value
* @return NULL
*/
function addNum($value) {

}

/**
* @return Integer[][]
*/
function getIntervals() {

}
}

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

```
* $obj = SummaryRanges();
* $obj->addNum($value);
* $ret_2 = $obj->getIntervals();
*/
```

Dart Solution:

```
class SummaryRanges {

SummaryRanges() {

}

void addNum(int value) {

}

List<List<int>> getIntervals() {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* SummaryRanges obj = SummaryRanges();
* obj.addNum(value);
* List<List<int>> param2 = obj.getIntervals();
*/
```

Scala Solution:

```
class SummaryRanges() {

def addNum(value: Int): Unit = {

}

def getIntervals(): Array[Array[Int]] = {

}
}

/**
* Your SummaryRanges object will be instantiated and called as such:
* val obj = new SummaryRanges()
* obj.addNum(value)
* val param_2 = obj.getIntervals()
*/
```

Elixir Solution:

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

end

@spec add_num(value :: integer) :: any
def add_num(value) do

end

@spec get_intervals() :: [[integer]]
def get_intervals() do

end
end

# Your functions will be called as such:
# SummaryRanges.init_()
# SummaryRanges.add_num(value)
# param_2 = SummaryRanges.get_intervals()

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

Erlang Solution:

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

```
-spec summary_ranges_add_num(Value :: integer()) -> any().
summary_ranges_add_num(Value) ->
.

-spec summary_ranges_get_intervals() -> [[integer()]].
summary_ranges_get_intervals() ->
.


%% Your functions will be called as such:
%% summary_ranges_init_(),
%% summary_ranges_add_num(Value),
%% Param_2 = summary_ranges_get_intervals(),

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

Racket Solution:

```
(define summary-ranges%
(class object%
(super-new)

(init-field)

; add-num : exact-integer? -> void?
(define/public (add-num value)
)
; get-intervals : -> (listof (listof exact-integer?))
(define/public (get-intervals)
)))

;; Your summary-ranges% object will be instantiated and called as such:
;; (define obj (new summary-ranges%))
;; (send obj add-num value)
;; (define param_2 (send obj get-intervals))
```