Problem 2671: Frequency Tracker

Problem Information

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

Problem Description

Design a data structure that keeps track of the values in it and answers some queries regarding their frequencies.

Implement the

FrequencyTracker

class.

FrequencyTracker()

: Initializes the

FrequencyTracker

object with an empty array initially.

void add(int number)

: Adds

number

to the data structure.

void deleteOne(int number)

: Deletes

one

occurrence of

number

from the data structure. The data structure

may not contain

number

, and in this case nothing is deleted.

bool hasFrequency(int frequency)

: Returns

true

if there is a number in the data structure that occurs

frequency

number of times, otherwise, it returns

false

.

Example 1:

Input

["FrequencyTracker", "add", "add", "hasFrequency"] [[], [3], [3], [2]] Output

[null, null, null, true]

Explanation

FrequencyTracker frequencyTracker = new FrequencyTracker(); frequencyTracker.add(3); // The data structure now contains [3] frequencyTracker.add(3); // The data structure now contains [3, 3] frequencyTracker.hasFrequency(2); // Returns true, because 3 occurs twice

Example 2:

Input

["FrequencyTracker", "add", "deleteOne", "hasFrequency"] [[], [1], [1], [1]]

Output

[null, null, null, false]

Explanation

FrequencyTracker frequencyTracker = new FrequencyTracker(); frequencyTracker.add(1); // The data structure now contains [1] frequencyTracker.deleteOne(1); // The data structure becomes empty [] frequencyTracker.hasFrequency(1); // Returns false, because the data structure is empty

Example 3:

Input

["FrequencyTracker", "hasFrequency", "add", "hasFrequency"] [[], [2], [3], [1]]

Output

[null, false, null, true]

Explanation

```
FrequencyTracker frequencyTracker = new FrequencyTracker();
frequencyTracker.hasFrequency(2); // Returns false, because the data structure is empty
frequencyTracker.add(3); // The data structure now contains [3]
frequencyTracker.hasFrequency(1); // Returns true, because 3 occurs once
```

Constraints:

1 <= number <= 10

5

1 <= frequency <= 10

5

At most,

2 * 10

5

calls will be made to

add

,

deleteOne

, and

hasFrequency

in

total

.

```
}
```

Code Snippets

C++:

```
class FrequencyTracker {
public:
FrequencyTracker() {

}

void add(int number) {

}

void deleteOne(int number) {

}

bool hasFrequency(int frequency) {

}
};

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker* obj = new FrequencyTracker();
* obj->add(number);
* obj->deleteOne(number);
* bool param_3 = obj->hasFrequency(frequency);
*/
```

Java:

```
class FrequencyTracker {

public FrequencyTracker() {

}

public void add(int number) {
}

public void deleteOne(int number) {

}

public boolean hasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = new FrequencyTracker();
* obj.add(number);
* obj.deleteOne(number);
* boolean param_3 = obj.hasFrequency(frequency);
*/
```

Python3:

```
class FrequencyTracker:

def __init__(self):


def add(self, number: int) -> None:


def deleteOne(self, number: int) -> None:


def hasFrequency(self, frequency: int) -> bool:



# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker()
# obj.add(number)
# obj.deleteOne(number)
# param_3 = obj.hasFrequency(frequency)
```

Python:

```
class FrequencyTracker(object):

def __init__(self):



def add(self, number):
"""
:type number: int
:rtype: None
"""



def deleteOne(self, number):
"""
:type number: int
:rtype: None
"""



def hasFrequency(self, frequency):
"""
:type frequency: int
:rtype: bool
"""




# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker()
# obj.add(number)
# obj.deleteOne(number)
# param_3 = obj.hasFrequency(frequency)
```

JavaScript:

```


var FrequencyTracker = function() {


};
```

```
/**
* @param {number} number
* @return {void}
*/
FrequencyTracker.prototype.add = function(number) {

};

/**
* @param {number} number
* @return {void}
*/
FrequencyTracker.prototype.deleteOne = function(number) {

};

/**
* @param {number} frequency
* @return {boolean}
*/
FrequencyTracker.prototype.hasFrequency = function(frequency) {

};

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

TypeScript:

```
class FrequencyTracker {
constructor() {

}

add(number: number): void {

}
deleteOne(number: number): void {

}

hasFrequency(frequency: number): boolean {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

C#:

```
public class FrequencyTracker {

public FrequencyTracker() {

}

public void Add(int number) {

}

public void DeleteOne(int number) {

}

public bool HasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = new FrequencyTracker();
* obj.Add(number);
* obj.DeleteOne(number);
* bool param_3 = obj.HasFrequency(frequency);
*/
```

C:

```



typedef struct {

} FrequencyTracker;


FrequencyTracker* frequencyTrackerCreate() {

}

void frequencyTrackerAdd(FrequencyTracker* obj, int number) {

}

void frequencyTrackerDeleteOne(FrequencyTracker* obj, int number) {

}

bool frequencyTrackerHasFrequency(FrequencyTracker* obj, int frequency) {

}

void frequencyTrackerFree(FrequencyTracker* obj) {

}

/**
* Your FrequencyTracker struct will be instantiated and called as such:
* FrequencyTracker* obj = frequencyTrackerCreate();
* frequencyTrackerAdd(obj, number);

* frequencyTrackerDeleteOne(obj, number);
* bool param_3 = frequencyTrackerHasFrequency(obj, frequency);

* frequencyTrackerFree(obj);
*/
```

Go:

```
type FrequencyTracker struct {

}


func Constructor() FrequencyTracker {

}


func (this *FrequencyTracker) Add(number int) {

}


func (this *FrequencyTracker) DeleteOne(number int) {

}


func (this *FrequencyTracker) HasFrequency(frequency int) bool {

}


/**
* Your FrequencyTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(number);
* obj.DeleteOne(number);
* param_3 := obj.HasFrequency(frequency);
*/
```

Kotlin:

```
class FrequencyTracker() {

fun add(number: Int) {

}

fun deleteOne(number: Int) {

}

fun hasFrequency(frequency: Int): Boolean {

}

}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

Swift:

```

class FrequencyTracker {

init() {

}

func add(_ number: Int) {

}

func deleteOne(_ number: Int) {

}

func hasFrequency(_ frequency: Int) -> Bool {
}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* let obj = FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* let ret_3: Bool = obj.hasFrequency(frequency)
*/
```

Rust:

```
struct FrequencyTracker {

}



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

fn new() -> Self {

}

fn add(&self, number: i32) {

}

fn delete_one(&self, number: i32) {

}

fn has_frequency(&self, frequency: i32) -> bool {

}
}

/**
```

```
* Your FrequencyTracker object will be instantiated and called as such:
* let obj = FrequencyTracker::new();
* obj.add(number);
* obj.delete_one(number);
* let ret_3: bool = obj.has_frequency(frequency);
*/
```

Ruby:

```
class FrequencyTracker
def initialize()

end



=begin
:type number: Integer
:rtype: Void
=end
def add(number)

end



=begin
:type number: Integer
:rtype: Void
=end
def delete_one(number)

end



=begin
:type frequency: Integer
:rtype: Boolean
=end
def has_frequency(frequency)

end
```

```

end

# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker.new()
# obj.add(number)
# obj.delete_one(number)
# param_3 = obj.has_frequency(frequency)
```

PHP:

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

}

/**
* @param Integer $number
* @return NULL
*/
function add($number) {

}

/**
* @param Integer $number
* @return NULL
*/
function deleteOne($number) {

}

/**
* @param Integer $frequency
* @return Boolean
*/
function hasFrequency($frequency) {

}
}
```

```
/**
* Your FrequencyTracker object will be instantiated and called as such:
* $obj = FrequencyTracker();
* $obj->add($number);
* $obj->deleteOne($number);
* $ret_3 = $obj->hasFrequency($frequency);
*/
```

Dart:

```
class FrequencyTracker {

FrequencyTracker() {

}

void add(int number) {

}

void deleteOne(int number) {

}

bool hasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = FrequencyTracker();
* obj.add(number);
* obj.deleteOne(number);
* bool param3 = obj.hasFrequency(frequency);
*/
```

Scala:

```
class FrequencyTracker() {

def add(number: Int): Unit = {
}

def deleteOne(number: Int): Unit = {

}

def hasFrequency(frequency: Int): Boolean = {

}

}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* val obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* val param_3 = obj.hasFrequency(frequency)
*/
```

Elixir:

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

end

@spec add(number :: integer) :: any
def add(number) do

end

@spec delete_one(number :: integer) :: any
def delete_one(number) do

end

@spec has_frequency(frequency :: integer) :: boolean
def has_frequency(frequency) do
```

```
end
end

# Your functions will be called as such:
# FrequencyTracker.init_()
# FrequencyTracker.add(number)
# FrequencyTracker.delete_one(number)
# param_3 = FrequencyTracker.has_frequency(frequency)

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

Erlang:

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

-spec frequency_tracker_add(Number :: integer()) -> any().
frequency_tracker_add(Number) ->
.

-spec frequency_tracker_delete_one(Number :: integer()) -> any().
frequency_tracker_delete_one(Number) ->
.

-spec frequency_tracker_has_frequency(Frequency :: integer()) -> boolean().
frequency_tracker_has_frequency(Frequency) ->
.


%% Your functions will be called as such:
%% frequency_tracker_init_(),
%% frequency_tracker_add(Number),
%% frequency_tracker_delete_one(Number),
%% Param_3 = frequency_tracker_has_frequency(Frequency),

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

Racket:

```
(define frequency-tracker%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add number)
)
; delete-one : exact-integer? -> void?
(define/public (delete-one number)
)
; has-frequency : exact-integer? -> boolean?
(define/public (has-frequency frequency)
)))

;; Your frequency-tracker% object will be instantiated and called as such:
;; (define obj (new frequency-tracker%))
;; (send obj add number)
;; (send obj delete-one number)
;; (define param_3 (send obj has-frequency frequency))
```

Solutions

C++ Solution:

```
/*
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class FrequencyTracker {
public:
FrequencyTracker() {

}
void add(int number) {

}

void deleteOne(int number) {

}

bool hasFrequency(int frequency) {

}
};

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker* obj = new FrequencyTracker();
* obj->add(number);
* obj->deleteOne(number);
* bool param_3 = obj->hasFrequency(frequency);
*/
```

Java Solution:

```
/**
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class FrequencyTracker {

public FrequencyTracker() {

}

public void add(int number) {
}

public void deleteOne(int number) {

}

public boolean hasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = new FrequencyTracker();
* obj.add(number);
* obj.deleteOne(number);
* boolean param_3 = obj.hasFrequency(frequency);
*/
```

Python3 Solution:

```
"""
Problem: Frequency Tracker
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class FrequencyTracker:

def __init__(self):



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

Python Solution:

```
class FrequencyTracker(object):

def __init__(self):



def add(self, number):
"""
:type number: int
:rtype: None
"""



def deleteOne(self, number):
"""
:type number: int
:rtype: None
"""



def hasFrequency(self, frequency):
"""
:type frequency: int
:rtype: bool
"""




# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker()
# obj.add(number)
# obj.deleteOne(number)
# param_3 = obj.hasFrequency(frequency)
```

JavaScript Solution:

```
/**
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/


var FrequencyTracker = function() {

};

/**
* @param {number} number
* @return {void}
*/
FrequencyTracker.prototype.add = function(number) {

};

/**
* @param {number} number
* @return {void}
*/
FrequencyTracker.prototype.deleteOne = function(number) {

};

/**
* @param {number} frequency
* @return {boolean}
*/
FrequencyTracker.prototype.hasFrequency = function(frequency) {

};

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

TypeScript Solution:

```
/**
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class FrequencyTracker {
constructor() {

}

add(number: number): void {

}

deleteOne(number: number): void {

}

hasFrequency(frequency: number): boolean {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

C# Solution:

```
/*
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class FrequencyTracker {

public FrequencyTracker() {

}

public void Add(int number) {

}

public void DeleteOne(int number) {

}

public bool HasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = new FrequencyTracker();
* obj.Add(number);
* obj.DeleteOne(number);
* bool param_3 = obj.HasFrequency(frequency);
*/
```

C Solution:

```
/*
* Problem: Frequency Tracker
* Difficulty: Medium
* Tags: array, hash
*
```

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




typedef struct {

} FrequencyTracker;


FrequencyTracker* frequencyTrackerCreate() {

}

void frequencyTrackerAdd(FrequencyTracker* obj, int number) {

}

void frequencyTrackerDeleteOne(FrequencyTracker* obj, int number) {

}

bool frequencyTrackerHasFrequency(FrequencyTracker* obj, int frequency) {

}

void frequencyTrackerFree(FrequencyTracker* obj) {

}

/**
* Your FrequencyTracker struct will be instantiated and called as such:
* FrequencyTracker* obj = frequencyTrackerCreate();
* frequencyTrackerAdd(obj, number);

* frequencyTrackerDeleteOne(obj, number);

* bool param_3 = frequencyTrackerHasFrequency(obj, frequency);
* frequencyTrackerFree(obj);
*/
```

Go Solution:

```
// Problem: Frequency Tracker
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type FrequencyTracker struct {

}


func Constructor() FrequencyTracker {

}


func (this *FrequencyTracker) Add(number int) {

}


func (this *FrequencyTracker) DeleteOne(number int) {

}


func (this *FrequencyTracker) HasFrequency(frequency int) bool {

}


/**
* Your FrequencyTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(number);
* obj.DeleteOne(number);
* param_3 := obj.HasFrequency(frequency);
*/
```

Kotlin Solution:

```
class FrequencyTracker() {

fun add(number: Int) {

}

fun deleteOne(number: Int) {

}

fun hasFrequency(frequency: Int): Boolean {

}

}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* var obj = FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* var param_3 = obj.hasFrequency(frequency)
*/
```

Swift Solution:

```
class FrequencyTracker {

init() {

}

func add(_ number: Int) {
}

func deleteOne(_ number: Int) {

}

func hasFrequency(_ frequency: Int) -> Bool {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* let obj = FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* let ret_3: Bool = obj.hasFrequency(frequency)
*/
```

Rust Solution:

```
// Problem: Frequency Tracker
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct FrequencyTracker {

}



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

```
fn new() -> Self {

}

fn add(&self, number: i32) {

}

fn delete_one(&self, number: i32) {

}

fn has_frequency(&self, frequency: i32) -> bool {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* let obj = FrequencyTracker::new();
* obj.add(number);
* obj.delete_one(number);
* let ret_3: bool = obj.has_frequency(frequency);
*/
```

Ruby Solution:

```
class FrequencyTracker
def initialize()

end



=begin
:type number: Integer
:rtype: Void
=end
def add(number)

end
```

```


=begin
:type number: Integer
:rtype: Void
=end
def delete_one(number)

end



=begin
:type frequency: Integer
:rtype: Boolean
=end
def has_frequency(frequency)

end



end

# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker.new()
# obj.add(number)
# obj.delete_one(number)
# param_3 = obj.has_frequency(frequency)
```

PHP Solution:

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

}

/**
* @param Integer $number
* @return NULL
*/
function add($number) {
}

/**
* @param Integer $number
* @return NULL
*/
function deleteOne($number) {

}

/**
* @param Integer $frequency
* @return Boolean
*/
function hasFrequency($frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* $obj = FrequencyTracker();
* $obj->add($number);
* $obj->deleteOne($number);
* $ret_3 = $obj->hasFrequency($frequency);
*/
```

Dart Solution:

```
class FrequencyTracker {

FrequencyTracker() {

}

void add(int number) {

}

void deleteOne(int number) {
}

bool hasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = FrequencyTracker();
* obj.add(number);
* obj.deleteOne(number);
* bool param3 = obj.hasFrequency(frequency);
*/
```

Scala Solution:

```
class FrequencyTracker() {

def add(number: Int): Unit = {

}

def deleteOne(number: Int): Unit = {

}

def hasFrequency(frequency: Int): Boolean = {

}

}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* val obj = new FrequencyTracker()
* obj.add(number)
* obj.deleteOne(number)
* val param_3 = obj.hasFrequency(frequency)
*/
```

Elixir Solution:

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

end

@spec add(number :: integer) :: any
def add(number) do

end

@spec delete_one(number :: integer) :: any
def delete_one(number) do

end

@spec has_frequency(frequency :: integer) :: boolean
def has_frequency(frequency) do

end
end

# Your functions will be called as such:
# FrequencyTracker.init_()
# FrequencyTracker.add(number)
# FrequencyTracker.delete_one(number)
# param_3 = FrequencyTracker.has_frequency(frequency)

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

Erlang Solution:

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

-spec frequency_tracker_add(Number :: integer()) -> any().
frequency_tracker_add(Number) ->
.
```

```
-spec frequency_tracker_delete_one(Number :: integer()) -> any().
frequency_tracker_delete_one(Number) ->
.

-spec frequency_tracker_has_frequency(Frequency :: integer()) -> boolean().
frequency_tracker_has_frequency(Frequency) ->
.


%% Your functions will be called as such:
%% frequency_tracker_init_(),
%% frequency_tracker_add(Number),
%% frequency_tracker_delete_one(Number),
%% Param_3 = frequency_tracker_has_frequency(Frequency),

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

Racket Solution:

```
(define frequency-tracker%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add number)
)
; delete-one : exact-integer? -> void?
(define/public (delete-one number)
)
; has-frequency : exact-integer? -> boolean?
(define/public (has-frequency frequency)
)))

;; Your frequency-tracker% object will be instantiated and called as such:
;; (define obj (new frequency-tracker%))
;; (send obj add number)
;; (send obj delete-one number)
;; (define param_3 (send obj has-frequency frequency))
```