Problem 1429: First Unique Number

Problem Information

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

Problem Description

You have a queue of integers, you need to retrieve the first unique integer in the queue.

Implement the

FirstUnique

class:

FirstUnique(int[] nums)

Initializes the object with the numbers in the queue.

int showFirstUnique()

returns the value of

the first unique

integer of the queue, and returns

-1

if there is no such integer.

void add(int value)

insert value to the queue.

Example 1:

Input:

["FirstUnique","showFirstUnique","add","showFirstUnique","add","showFirstUnique","add","showFirstUnique"] [[[2,3,5]],[],[5],[],[2],[],[3],[]]

Output:

[null,2,null,2,null,3,null,-1]

Explanation:

FirstUnique firstUnique = new FirstUnique([2,3,5]); firstUnique.showFirstUnique(); // return 2 firstUnique.add(5); // the queue is now [2,3,5,5] firstUnique.showFirstUnique(); // return 2 firstUnique.add(2);   // the queue is now [2,3,5,5,2] firstUnique.showFirstUnique(); // return 3 firstUnique.add(3);   // the queue is now [2,3,5,5,2,3] firstUnique.showFirstUnique(); // return -1

Example 2:

Input:

["FirstUnique","showFirstUnique","add","add","add","add","add","showFirstUnique"] [[[7,7,7,7,7,7]],[],[7],[3],[3],[7],[17],[]]

Output:

[null,-1,null,null,null,null,null,17]

Explanation:

FirstUnique firstUnique = new FirstUnique([7,7,7,7,7,7]); firstUnique.showFirstUnique(); // return -1 firstUnique.add(7); // the queue is now [7,7,7,7,7,7,7] firstUnique.add(3);   // the queue is now [7,7,7,7,7,7,7,3] firstUnique.add(3);   // the queue is now [7,7,7,7,7,7,7,3,3] firstUnique.add(7);   // the queue is now [7,7,7,7,7,7,7,3,3,7] firstUnique.add(17);   // the queue is now [7,7,7,7,7,7,7,3,3,7,17] firstUnique.showFirstUnique(); // return 17

Example 3:

Input:

["FirstUnique","showFirstUnique","add","showFirstUnique"] [[[809]],[],[809],[]]

Output:

[null,809,null,-1]

Explanation:

FirstUnique firstUnique = new FirstUnique([809]); firstUnique.showFirstUnique(); // return 809 firstUnique.add(809); // the queue is now [809,809] firstUnique.showFirstUnique(); // return -1

Constraints:

1 <= nums.length <= 10^5

1 <= nums[i] <= 10^8

1 <= value <= 10^8

At most

50000

calls will be made to

showFirstUnique

and

add

.

Code Snippets

C++:

```
class FirstUnique {
public:
FirstUnique(vector<int>& nums) {

}

int showFirstUnique() {

}

void add(int value) {

}
};

/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique* obj = new FirstUnique(nums);
* int param_1 = obj->showFirstUnique();
* obj->add(value);
*/
```

Java:

```
class FirstUnique {

public FirstUnique(int[] nums) {

}

public int showFirstUnique() {

}

public void add(int value) {

}
}
```

```
/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique obj = new FirstUnique(nums);
* int param_1 = obj.showFirstUnique();
* obj.add(value);
*/
```

Python3:

```
class FirstUnique:

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



def showFirstUnique(self) -> int:



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




# Your FirstUnique object will be instantiated and called as such:
# obj = FirstUnique(nums)
# param_1 = obj.showFirstUnique()
# obj.add(value)
```

Python:

```
class FirstUnique(object):

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



def showFirstUnique(self):
"""
:rtype: int
"""
```

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



# Your FirstUnique object will be instantiated and called as such:
# obj = FirstUnique(nums)
# param_1 = obj.showFirstUnique()
# obj.add(value)
```

JavaScript:

```
/**
* @param {number[]} nums
*/
var FirstUnique = function(nums) {

};

/**
* @return {number}
*/
FirstUnique.prototype.showFirstUnique = function() {

};

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

};

/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
```

```
*/
```

TypeScript:

```
class FirstUnique {
constructor(nums: number[]) {

}

showFirstUnique(): number {

}

add(value: number): void {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

C#:

```
public class FirstUnique {

public FirstUnique(int[] nums) {

}

public int ShowFirstUnique() {

}

public void Add(int value) {

}
}
```

```
/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique obj = new FirstUnique(nums);
* int param_1 = obj.ShowFirstUnique();
* obj.Add(value);
*/
```

C:

```
typedef struct {

} FirstUnique;


FirstUnique* firstUniqueCreate(int* nums, int numsSize) {

}

int firstUniqueShowFirstUnique(FirstUnique* obj) {

}

void firstUniqueAdd(FirstUnique* obj, int value) {

}

void firstUniqueFree(FirstUnique* obj) {

}

/**
* Your FirstUnique struct will be instantiated and called as such:
* FirstUnique* obj = firstUniqueCreate(nums, numsSize);
* int param_1 = firstUniqueShowFirstUnique(obj);

* firstUniqueAdd(obj, value);

* firstUniqueFree(obj);
*/
```

Go:

```
type FirstUnique struct {

}


func Constructor(nums []int) FirstUnique {

}


func (this *FirstUnique) ShowFirstUnique() int {

}


func (this *FirstUnique) Add(value int) {

}


/**
* Your FirstUnique object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.ShowFirstUnique();
* obj.Add(value);
*/
```

Kotlin:

```
class FirstUnique(nums: IntArray) {

fun showFirstUnique(): Int {

}

fun add(value: Int) {

}

}
```

```
/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

Swift:

```
class FirstUnique {

init(_ nums: [Int]) {

}

func showFirstUnique() -> Int {

}

func add(_ value: Int) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* let obj = FirstUnique(nums)
* let ret_1: Int = obj.showFirstUnique()
* obj.add(value)
*/
```

Rust:

```
struct FirstUnique {

}



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

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

}

fn show_first_unique(&self) -> i32 {

}

fn add(&self, value: i32) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* let obj = FirstUnique::new(nums);
* let ret_1: i32 = obj.show_first_unique();
* obj.add(value);
*/
```

Ruby:

```
class FirstUnique

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

end



=begin
:rtype: Integer
=end
def show_first_unique()

end
```

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

end



end

# Your FirstUnique object will be instantiated and called as such:
# obj = FirstUnique.new(nums)
# param_1 = obj.show_first_unique()
# obj.add(value)
```

PHP:

```
class FirstUnique {
/**
* @param Integer[] $nums
*/
function __construct($nums) {

}

/**
* @return Integer
*/
function showFirstUnique() {

}

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

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* $obj = FirstUnique($nums);
* $ret_1 = $obj->showFirstUnique();
* $obj->add($value);
*/
```

Scala:

```
class FirstUnique(_nums: Array[Int]) {

def showFirstUnique(): Int = {

}

def add(value: Int) {

}

}

/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

Racket:

```
(define first-unique%
(class object%
(super-new)

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

; show-first-unique : -> exact-integer?
(define/public (show-first-unique)
)
; add : exact-integer? -> void?
(define/public (add value)

)))

;; Your first-unique% object will be instantiated and called as such:
;; (define obj (new first-unique% [nums nums]))
;; (define param_1 (send obj show-first-unique))
;; (send obj add value)
```

Solutions

C++ Solution:

```
/*
* Problem: First Unique Number
* Difficulty: Medium
* Tags: array, hash, queue
*
* 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 FirstUnique {
public:
FirstUnique(vector<int>& nums) {

}

int showFirstUnique() {

}

void add(int value) {

}
};
```

```
/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique* obj = new FirstUnique(nums);
* int param_1 = obj->showFirstUnique();
* obj->add(value);
*/
```

Java Solution:

```
/**
* Problem: First Unique Number
* Difficulty: Medium
* Tags: array, hash, queue
*
* 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 FirstUnique {

public FirstUnique(int[] nums) {

}

public int showFirstUnique() {

}

public void add(int value) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique obj = new FirstUnique(nums);
* int param_1 = obj.showFirstUnique();
* obj.add(value);
*/
```

Python3 Solution:

```
"""
Problem: First Unique Number
Difficulty: Medium
Tags: array, hash, queue

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 FirstUnique:

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



def showFirstUnique(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class FirstUnique(object):

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



def showFirstUnique(self):
"""
:rtype: int
"""



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

```
# Your FirstUnique object will be instantiated and called as such:
# obj = FirstUnique(nums)
# param_1 = obj.showFirstUnique()
# obj.add(value)
```

JavaScript Solution:

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

/**
* @param {number[]} nums
*/
var FirstUnique = function(nums) {

};

/**
* @return {number}
*/
FirstUnique.prototype.showFirstUnique = function() {

};

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

};
```

```
/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

TypeScript Solution:

```
/**
* Problem: First Unique Number
* Difficulty: Medium
* Tags: array, hash, queue
*
* 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 FirstUnique {
constructor(nums: number[]) {

}

showFirstUnique(): number {

}

add(value: number): void {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

C# Solution:

```
/*
* Problem: First Unique Number
* Difficulty: Medium
* Tags: array, hash, queue
*
* 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 FirstUnique {

public FirstUnique(int[] nums) {

}

public int ShowFirstUnique() {

}

public void Add(int value) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* FirstUnique obj = new FirstUnique(nums);
* int param_1 = obj.ShowFirstUnique();
* obj.Add(value);
*/
```

C Solution:

```
/*
* Problem: First Unique Number
* Difficulty: Medium
* Tags: array, hash, queue
*
* 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 {

} FirstUnique;


FirstUnique* firstUniqueCreate(int* nums, int numsSize) {

}

int firstUniqueShowFirstUnique(FirstUnique* obj) {

}

void firstUniqueAdd(FirstUnique* obj, int value) {

}

void firstUniqueFree(FirstUnique* obj) {

}

/**
* Your FirstUnique struct will be instantiated and called as such:
* FirstUnique* obj = firstUniqueCreate(nums, numsSize);
* int param_1 = firstUniqueShowFirstUnique(obj);

* firstUniqueAdd(obj, value);

* firstUniqueFree(obj);
*/
```

Go Solution:

```
// Problem: First Unique Number
// Difficulty: Medium
// Tags: array, hash, queue
//
// 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 FirstUnique struct {

}


func Constructor(nums []int) FirstUnique {

}


func (this *FirstUnique) ShowFirstUnique() int {

}


func (this *FirstUnique) Add(value int) {

}


/**
* Your FirstUnique object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.ShowFirstUnique();
* obj.Add(value);
*/
```

Kotlin Solution:

```
class FirstUnique(nums: IntArray) {

fun showFirstUnique(): Int {

}
fun add(value: Int) {

}

}

/**
* Your FirstUnique object will be instantiated and called as such:
* var obj = FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

Swift Solution:

```
class FirstUnique {

init(_ nums: [Int]) {

}

func showFirstUnique() -> Int {

}

func add(_ value: Int) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* let obj = FirstUnique(nums)
* let ret_1: Int = obj.showFirstUnique()
* obj.add(value)
*/
```

Rust Solution:

```
// Problem: First Unique Number
// Difficulty: Medium
// Tags: array, hash, queue
//
// 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 FirstUnique {

}



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

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

}

fn show_first_unique(&self) -> i32 {

}

fn add(&self, value: i32) {

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* let obj = FirstUnique::new(nums);
* let ret_1: i32 = obj.show_first_unique();
* obj.add(value);
*/
```

Ruby Solution:

```
class FirstUnique

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

end



=begin
:rtype: Integer
=end
def show_first_unique()

end



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

end



end

# Your FirstUnique object will be instantiated and called as such:
# obj = FirstUnique.new(nums)
# param_1 = obj.show_first_unique()
# obj.add(value)
```

PHP Solution:

```
class FirstUnique {
/**
* @param Integer[] $nums
*/
function __construct($nums) {
}

/**
* @return Integer
*/
function showFirstUnique() {

}

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

}
}

/**
* Your FirstUnique object will be instantiated and called as such:
* $obj = FirstUnique($nums);
* $ret_1 = $obj->showFirstUnique();
* $obj->add($value);
*/
```

Scala Solution:

```
class FirstUnique(_nums: Array[Int]) {

def showFirstUnique(): Int = {

}

def add(value: Int) {

}

}

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

```
* var obj = new FirstUnique(nums)
* var param_1 = obj.showFirstUnique()
* obj.add(value)
*/
```

Racket Solution:

```
(define first-unique%
(class object%
(super-new)

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

; show-first-unique : -> exact-integer?
(define/public (show-first-unique)

)
; add : exact-integer? -> void?
(define/public (add value)

)))

;; Your first-unique% object will be instantiated and called as such:
;; (define obj (new first-unique% [nums nums]))
;; (define param_1 (send obj show-first-unique))
;; (send obj add value)
```