Problem 2349: Design a Number Container System

Problem Information

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

Problem Description

Design a number container system that can do the following:

Insert

or

Replace

a number at the given index in the system.

Return

the smallest index for the given number in the system.

Implement the

NumberContainers

class:

NumberContainers()

Initializes the number container system.

void change(int index, int number)

Fills the container at

index

with the

number

. If there is already a number at that

index

, replace it.

int find(int number)

Returns the smallest index for the given

number

, or

-1

if there is no index that is filled by

number

in the system.

Example 1:

Input

["NumberContainers", "find", "change", "change", "change", "change", "find", "change", "find"]
[[], [10], [2, 10], [1, 10], [3, 10], [5, 10], [10], [1, 20], [10]]

Output

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

Explanation

NumberContainers nc = new NumberContainers(); nc.find(10); // There is no index that is filled with number 10. Therefore, we return -1. nc.change(2, 10); // Your container at index 2 will be filled with number 10. nc.change(1, 10); // Your container at index 1 will be filled with number 10. nc.change(3, 10); // Your container at index 3 will be filled with number 10. nc.change(5, 10); // Your container at index 5 will be filled with number 10. nc.find(10); // Number 10 is at the indices 1, 2, 3, and 5. Since the smallest index that is filled with 10 is 1, we return 1. nc.change(1, 20); // Your container at index 1 will be filled with number 20. Note that index 1 was filled with 10 and then replaced with 20. nc.find(10); // Number 10 is at the indices 2, 3, and 5. The smallest index that is filled with 10 is 2. Therefore, we return 2.

Constraints:

1 <= index, number <= 10

9

At most

10

5

calls will be made

in total

to

change

and

find

.

Code Snippets

C++:

```
class NumberContainers {
public:
NumberContainers() {

}

void change(int index, int number) {

}

int find(int number) {

}
};

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers* obj = new NumberContainers();
* obj->change(index,number);
* int param_2 = obj->find(number);
*/
```

Java:

```
class NumberContainers {

public NumberContainers() {

}

public void change(int index, int number) {

}

public int find(int number) {

}

}

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = new NumberContainers();
* obj.change(index,number);
* int param_2 = obj.find(number);
*/
```

Python3:

```
class NumberContainers:

def __init__(self):


def change(self, index: int, number: int) -> None:


def find(self, number: int) -> int:



# Your NumberContainers object will be instantiated and called as such:
# obj = NumberContainers()
# obj.change(index,number)
# param_2 = obj.find(number)
```

Python:

```
class NumberContainers(object):

def __init__(self):


def change(self, index, number):
"""
:type index: int
:type number: int
:rtype: None
"""
```

```
def find(self, number):
"""
:type number: int
:rtype: int
"""



# Your NumberContainers object will be instantiated and called as such:
# obj = NumberContainers()
# obj.change(index,number)
# param_2 = obj.find(number)
```

JavaScript:

```
var NumberContainers = function() {

};

/**
* @param {number} index
* @param {number} number
* @return {void}
*/
NumberContainers.prototype.change = function(index, number) {

};

/**
* @param {number} number
* @return {number}
*/
NumberContainers.prototype.find = function(number) {

};

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = new NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

TypeScript:

```
class NumberContainers {
constructor() {

}

change(index: number, number: number): void {

}

find(number: number): number {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = new NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

C#:

```
public class NumberContainers {

public NumberContainers() {

}

public void Change(int index, int number) {

}

public int Find(int number) {

}
}
```

```
/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = new NumberContainers();
* obj.Change(index,number);
* int param_2 = obj.Find(number);
*/
```

C:

```
typedef struct {

} NumberContainers;


NumberContainers* numberContainersCreate() {

}

void numberContainersChange(NumberContainers* obj, int index, int number) {

}

int numberContainersFind(NumberContainers* obj, int number) {

}

void numberContainersFree(NumberContainers* obj) {

}

/**
* Your NumberContainers struct will be instantiated and called as such:
* NumberContainers* obj = numberContainersCreate();
* numberContainersChange(obj, index, number);

* int param_2 = numberContainersFind(obj, number);

* numberContainersFree(obj);
```

```
}
```

```

*/
```

Go:

```
type NumberContainers struct {

}


func Constructor() NumberContainers {

}


func (this *NumberContainers) Change(index int, number int) {

}


func (this *NumberContainers) Find(number int) int {

}


/**
* Your NumberContainers object will be instantiated and called as such:
* obj := Constructor();
* obj.Change(index,number);
* param_2 := obj.Find(number);
*/
```

Kotlin:

```
class NumberContainers() {

fun change(index: Int, number: Int) {

}

fun find(number: Int): Int {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

Swift:

```
class NumberContainers {

init() {

}

func change(_ index: Int, _ number: Int) {

}

func find(_ number: Int) -> Int {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* let obj = NumberContainers()
* obj.change(index, number)
* let ret_2: Int = obj.find(number)
*/
```

Rust:

```
struct NumberContainers {

}
```

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

fn new() -> Self {

}

fn change(&self, index: i32, number: i32) {

}

fn find(&self, number: i32) -> i32 {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* let obj = NumberContainers::new();
* obj.change(index, number);
* let ret_2: i32 = obj.find(number);
*/
```

Ruby:

```
class NumberContainers
def initialize()

end



=begin
:type index: Integer
:type number: Integer
:rtype: Void
=end
def change(index, number)

end
```

```
=begin
:type number: Integer
:rtype: Integer
=end
def find(number)

end



end

# Your NumberContainers object will be instantiated and called as such:
# obj = NumberContainers.new()
# obj.change(index, number)
# param_2 = obj.find(number)
```

PHP:

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

}

/**
* @param Integer $index
* @param Integer $number
* @return NULL
*/
function change($index, $number) {

}

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

/**
* Your NumberContainers object will be instantiated and called as such:
* $obj = NumberContainers();
* $obj->change($index, $number);
* $ret_2 = $obj->find($number);
*/
```

Dart:

```
class NumberContainers {

NumberContainers() {

}

void change(int index, int number) {

}

int find(int number) {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = NumberContainers();
* obj.change(index,number);
* int param2 = obj.find(number);
*/
```

Scala:

```
class NumberContainers() {

def change(index: Int, number: Int): Unit = {

}
def find(number: Int): Int = {

}

}

/**
* Your NumberContainers object will be instantiated and called as such:
* val obj = new NumberContainers()
* obj.change(index,number)
* val param_2 = obj.find(number)
*/
```

Elixir:

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

end

@spec change(index :: integer, number :: integer) :: any
def change(index, number) do

end

@spec find(number :: integer) :: integer
def find(number) do

end
end

# Your functions will be called as such:
# NumberContainers.init_()
# NumberContainers.change(index, number)
# param_2 = NumberContainers.find(number)

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

Erlang:

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

-spec number_containers_change(Index :: integer(), Number :: integer()) ->
any().
number_containers_change(Index, Number) ->
.

-spec number_containers_find(Number :: integer()) -> integer().
number_containers_find(Number) ->
.



%% Your functions will be called as such:
%% number_containers_init_(),
%% number_containers_change(Index, Number),
%% Param_2 = number_containers_find(Number),

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

Racket:

```
(define number-containers%
(class object%
(super-new)

(init-field)

; change : exact-integer? exact-integer? -> void?
(define/public (change index number)
)
; find : exact-integer? -> exact-integer?
(define/public (find number)
)))

;; Your number-containers% object will be instantiated and called as such:
;; (define obj (new number-containers%))
;; (send obj change index number)
;; (define param_2 (send obj find number))
```

Solutions

C++ Solution:

```
/*
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
*
* 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 NumberContainers {
public:
NumberContainers() {

}

void change(int index, int number) {

}

int find(int number) {

}
};

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers* obj = new NumberContainers();
* obj->change(index,number);
* int param_2 = obj->find(number);
*/
```

Java Solution:

```
/**
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
*
```

```
* 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 NumberContainers {

public NumberContainers() {

}

public void change(int index, int number) {

}

public int find(int number) {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = new NumberContainers();
* obj.change(index,number);
* int param_2 = obj.find(number);
*/
```

Python3 Solution:

```
"""
Problem: Design a Number Container System
Difficulty: Medium
Tags: hash, queue, heap

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 NumberContainers:
```

```

def __init__(self):


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

Python Solution:

```
class NumberContainers(object):

def __init__(self):


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


def find(self, number):
"""
:type number: int
:rtype: int
"""



# Your NumberContainers object will be instantiated and called as such:
# obj = NumberContainers()
# obj.change(index,number)
# param_2 = obj.find(number)
```

JavaScript Solution:

```
/**
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
```

```
*
* 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 NumberContainers = function() {

};

/**
* @param {number} index
* @param {number} number
* @return {void}
*/
NumberContainers.prototype.change = function(index, number) {

};

/**
* @param {number} number
* @return {number}
*/
NumberContainers.prototype.find = function(number) {

};

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = new NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

TypeScript Solution:

```
/**
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
```

```
*
* 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 NumberContainers {
constructor() {

}

change(index: number, number: number): void {

}

find(number: number): number {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = new NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

C# Solution:

```
/*
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
*
* 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 NumberContainers {
```

```
public NumberContainers() {

}

public void Change(int index, int number) {

}

public int Find(int number) {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = new NumberContainers();
* obj.Change(index,number);
* int param_2 = obj.Find(number);
*/
```

C Solution:

```
/*
* Problem: Design a Number Container System
* Difficulty: Medium
* Tags: hash, queue, heap
*
* 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 {

} NumberContainers;


NumberContainers* numberContainersCreate() {


}

void numberContainersChange(NumberContainers* obj, int index, int number) {

}

int numberContainersFind(NumberContainers* obj, int number) {

}

void numberContainersFree(NumberContainers* obj) {

}

/**
* Your NumberContainers struct will be instantiated and called as such:
* NumberContainers* obj = numberContainersCreate();
* numberContainersChange(obj, index, number);

* int param_2 = numberContainersFind(obj, number);

* numberContainersFree(obj);
*/
```

Go Solution:

```
// Problem: Design a Number Container System
// Difficulty: Medium
// Tags: hash, queue, heap
//
// 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 NumberContainers struct {

}



func Constructor() NumberContainers {
}


func (this *NumberContainers) Change(index int, number int) {

}


func (this *NumberContainers) Find(number int) int {

}


/**
* Your NumberContainers object will be instantiated and called as such:
* obj := Constructor();
* obj.Change(index,number);
* param_2 := obj.Find(number);
*/
```

Kotlin Solution:

```
class NumberContainers() {

fun change(index: Int, number: Int) {

}

fun find(number: Int): Int {

}

}

/**
* Your NumberContainers object will be instantiated and called as such:
* var obj = NumberContainers()
* obj.change(index,number)
* var param_2 = obj.find(number)
*/
```

Swift Solution:

```
class NumberContainers {

init() {

}

func change(_ index: Int, _ number: Int) {

}

func find(_ number: Int) -> Int {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* let obj = NumberContainers()
* obj.change(index, number)
* let ret_2: Int = obj.find(number)
*/
```

Rust Solution:

```
// Problem: Design a Number Container System
// Difficulty: Medium
// Tags: hash, queue, heap
//
// 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 NumberContainers {

}



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

fn new() -> Self {

}

fn change(&self, index: i32, number: i32) {

}

fn find(&self, number: i32) -> i32 {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* let obj = NumberContainers::new();
* obj.change(index, number);
* let ret_2: i32 = obj.find(number);
*/
```

Ruby Solution:

```
class NumberContainers
def initialize()

end



=begin
:type index: Integer
:type number: Integer
:rtype: Void
=end
def change(index, number)

end
```

```
=begin
:type number: Integer
:rtype: Integer
=end
def find(number)

end


end

# Your NumberContainers object will be instantiated and called as such:
# obj = NumberContainers.new()
# obj.change(index, number)
# param_2 = obj.find(number)
```

PHP Solution:

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

}

/**
* @param Integer $index
* @param Integer $number
* @return NULL
*/
function change($index, $number) {

}

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

/**
* Your NumberContainers object will be instantiated and called as such:
* $obj = NumberContainers();
* $obj->change($index, $number);
* $ret_2 = $obj->find($number);
*/
```

Dart Solution:

```
class NumberContainers {

NumberContainers() {

}

void change(int index, int number) {

}

int find(int number) {

}
}

/**
* Your NumberContainers object will be instantiated and called as such:
* NumberContainers obj = NumberContainers();
* obj.change(index,number);
* int param2 = obj.find(number);
*/
```

Scala Solution:

```
class NumberContainers() {

def change(index: Int, number: Int): Unit = {

}
def find(number: Int): Int = {

}

}

/**
* Your NumberContainers object will be instantiated and called as such:
* val obj = new NumberContainers()
* obj.change(index,number)
* val param_2 = obj.find(number)
*/
```

Elixir Solution:

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

end

@spec change(index :: integer, number :: integer) :: any
def change(index, number) do

end

@spec find(number :: integer) :: integer
def find(number) do

end
end

# Your functions will be called as such:
# NumberContainers.init_()
# NumberContainers.change(index, number)
# param_2 = NumberContainers.find(number)

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

Erlang Solution:

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

-spec number_containers_change(Index :: integer(), Number :: integer()) ->
any().
number_containers_change(Index, Number) ->
.

-spec number_containers_find(Number :: integer()) -> integer().
number_containers_find(Number) ->
.



%% Your functions will be called as such:
%% number_containers_init_(),
%% number_containers_change(Index, Number),
%% Param_2 = number_containers_find(Number),

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

Racket Solution:

```
(define number-containers%
(class object%
(super-new)

(init-field)

; change : exact-integer? exact-integer? -> void?
(define/public (change index number)
)
; find : exact-integer? -> exact-integer?
(define/public (find number)
)))

;; Your number-containers% object will be instantiated and called as such:
;; (define obj (new number-containers%))
;; (send obj change index number)
;; (define param_2 (send obj find number))
```