Problem 380: Insert Delete GetRandom O(1)

Problem Information

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

Problem Description

Implement the

RandomizedSet

class:

RandomizedSet()

Initializes the

RandomizedSet

object.

bool insert(int val)

Inserts an item

val

into the set if not present. Returns

true

if the item was not present,

false

otherwise.

bool remove(int val)

Removes an item

val

from the set if present. Returns

true

if the item was present,

false

otherwise.

int getRandom()

Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the

same probability

of being returned.

You must implement the functions of the class such that each function works in

average

O(1)

time complexity.

Example 1:

Input

["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"] [[], [1], [2], [2], [], [1], [2], []]

Output

[null, true, false, true, 2, true, false, 2]

Explanation

RandomizedSet randomizedSet = new RandomizedSet(); randomizedSet.insert(1); // Inserts 1 to the set. Returns true as 1 was inserted successfully. randomizedSet.remove(2); // Returns false as 2 does not exist in the set. randomizedSet.insert(2); // Inserts 2 to the set, returns true. Set now contains [1,2]. randomizedSet.getRandom(); // getRandom() should return either 1 or 2 randomly. randomizedSet.remove(1); // Removes 1 from the set, returns true. Set now contains [2]. randomizedSet.insert(2); // 2 was already in the set, so return false. randomizedSet.getRandom(); // Since 2 is the only number in the set, getRandom() will always return 2.

Constraints:

-2

31

<= val <= 2

31

- 1

At most

2 *

10

5

calls will be made to

insert

,

remove

, and

getRandom

.

There will be

at least one

element in the data structure when

getRandom

is called.

Code Snippets

C++:

```
class RandomizedSet {
public:
RandomizedSet() {

}

bool insert(int val) {

}
```

```
bool remove(int val) {

}

int getRandom() {

}
};

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet* obj = new RandomizedSet();
* bool param_1 = obj->insert(val);
* bool param_2 = obj->remove(val);
* int param_3 = obj->getRandom();
*/
```

Java:

```
class RandomizedSet {

public RandomizedSet() {

}

public boolean insert(int val) {

}

public boolean remove(int val) {

}

public int getRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = new RandomizedSet();
* boolean param_1 = obj.insert(val);
```

```
* boolean param_2 = obj.remove(val);
* int param_3 = obj.getRandom();
*/
```

Python3:

```
class RandomizedSet:

def __init__(self):


def insert(self, val: int) -> bool:


def remove(self, val: int) -> bool:


def getRandom(self) -> int:



# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()
```

Python:

```
class RandomizedSet(object):

def __init__(self):


def insert(self, val):
"""
:type val: int
:rtype: bool
"""


def remove(self, val):
```

```
        """
        :type val: int
        :rtype: bool
        """


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



# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()
```

JavaScript:

```
var RandomizedSet = function() {

};

/**
* @param {number} val
* @return {boolean}
*/
RandomizedSet.prototype.insert = function(val) {

};

/**
* @param {number} val
* @return {boolean}
*/
RandomizedSet.prototype.remove = function(val) {

};
```

```
/**
* @return {number}
*/
RandomizedSet.prototype.getRandom = function() {

};

/**
* Your RandomizedSet object will be instantiated and called as such:
* var obj = new RandomizedSet()
* var param_1 = obj.insert(val)
* var param_2 = obj.remove(val)
* var param_3 = obj.getRandom()
*/
```

TypeScript:

```
class RandomizedSet {
constructor() {

}

insert(val: number): boolean {

}

remove(val: number): boolean {

}

getRandom(): number {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* var obj = new RandomizedSet()
* var param_1 = obj.insert(val)
* var param_2 = obj.remove(val)
* var param_3 = obj.getRandom()
*/
```

C#:

```
public class RandomizedSet {

public RandomizedSet() {

}

public bool Insert(int val) {

}

public bool Remove(int val) {

}

public int GetRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = new RandomizedSet();
* bool param_1 = obj.Insert(val);
* bool param_2 = obj.Remove(val);
* int param_3 = obj.GetRandom();
*/
```

C:

```
typedef struct {

} RandomizedSet;


RandomizedSet* randomizedSetCreate() {

}
```

```
bool randomizedSetInsert(RandomizedSet* obj, int val) {

}

bool randomizedSetRemove(RandomizedSet* obj, int val) {

}

int randomizedSetGetRandom(RandomizedSet* obj) {

}

void randomizedSetFree(RandomizedSet* obj) {

}

/**
* Your RandomizedSet struct will be instantiated and called as such:
* RandomizedSet* obj = randomizedSetCreate();
* bool param_1 = randomizedSetInsert(obj, val);

* bool param_2 = randomizedSetRemove(obj, val);

* int param_3 = randomizedSetGetRandom(obj);

* randomizedSetFree(obj);
*/
```

Go:

```
type RandomizedSet struct {

}


func Constructor() RandomizedSet {

}


func (this *RandomizedSet) Insert(val int) bool {
```

```
}


func (this *RandomizedSet) Remove(val int) bool {

}


func (this *RandomizedSet) GetRandom() int {

}


/**
* Your RandomizedSet object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Insert(val);
* param_2 := obj.Remove(val);
* param_3 := obj.GetRandom();
*/
```

Kotlin:

```
class RandomizedSet() {

fun insert(`val`: Int): Boolean {

}

fun remove(`val`: Int): Boolean {

}

fun getRandom(): Int {

}

}

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

```
* var obj = RandomizedSet()
* var param_1 = obj.insert(`val`)
* var param_2 = obj.remove(`val`)
* var param_3 = obj.getRandom()
*/
```

Swift:

```
class RandomizedSet {

init() {

}

func insert(_ val: Int) -> Bool {

}

func remove(_ val: Int) -> Bool {

}

func getRandom() -> Int {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* let obj = RandomizedSet()
* let ret_1: Bool = obj.insert(val)
* let ret_2: Bool = obj.remove(val)
* let ret_3: Int = obj.getRandom()
*/
```

Rust:

```
struct RandomizedSet {

}
```

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

fn new() -> Self {

}

fn insert(&self, val: i32) -> bool {

}

fn remove(&self, val: i32) -> bool {

}

fn get_random(&self) -> i32 {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* let obj = RandomizedSet::new();
* let ret_1: bool = obj.insert(val);
* let ret_2: bool = obj.remove(val);
* let ret_3: i32 = obj.get_random();
*/
```

Ruby:

```
class RandomizedSet
def initialize()

end


=begin
:type val: Integer
```

```
:rtype: Boolean
=end
def insert(val)

end


=begin
:type val: Integer
:rtype: Boolean
=end
def remove(val)

end


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

end


end

# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet.new()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.get_random()
```

PHP:

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

}

/**
```

```
* @param Integer $val
* @return Boolean
*/
function insert($val) {

}

/**
* @param Integer $val
* @return Boolean
*/
function remove($val) {

}

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

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* $obj = RandomizedSet();
* $ret_1 = $obj->insert($val);
* $ret_2 = $obj->remove($val);
* $ret_3 = $obj->getRandom();
*/
```

Dart:

```
class RandomizedSet {

RandomizedSet() {

}

bool insert(int val) {

}
```

```
bool remove(int val) {

}

int getRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = RandomizedSet();
* bool param1 = obj.insert(val);
* bool param2 = obj.remove(val);
* int param3 = obj.getRandom();
*/
```

Scala:

```
class RandomizedSet() {

def insert(`val`: Int): Boolean = {

}

def remove(`val`: Int): Boolean = {

}

def getRandom(): Int = {

}

}

/**
* Your RandomizedSet object will be instantiated and called as such:
* val obj = new RandomizedSet()
* val param_1 = obj.insert(`val`)
* val param_2 = obj.remove(`val`)
* val param_3 = obj.getRandom()
```

```
*/
```

Elixir:

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

end

@spec insert(val :: integer) :: boolean
def insert(val) do

end

@spec remove(val :: integer) :: boolean
def remove(val) do

end

@spec get_random() :: integer
def get_random() do

end
end

# Your functions will be called as such:
# RandomizedSet.init_()
# param_1 = RandomizedSet.insert(val)
# param_2 = RandomizedSet.remove(val)
# param_3 = RandomizedSet.get_random()

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

Erlang:

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

-spec randomized_set_insert(Val :: integer()) -> boolean().
```

```
randomized_set_insert(Val) ->
.

-spec randomized_set_remove(Val :: integer()) -> boolean().
randomized_set_remove(Val) ->
.

-spec randomized_set_get_random() -> integer().
randomized_set_get_random() ->
.


%% Your functions will be called as such:
%% randomized_set_init_(),
%% Param_1 = randomized_set_insert(Val),
%% Param_2 = randomized_set_remove(Val),
%% Param_3 = randomized_set_get_random(),

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

Racket:

```
(define randomized-set%
(class object%
(super-new)

(init-field)

; insert : exact-integer? -> boolean?
(define/public (insert val)
)
; remove : exact-integer? -> boolean?
(define/public (remove val)
)
; get-random : -> exact-integer?
(define/public (get-random)
)))

;; Your randomized-set% object will be instantiated and called as such:
;; (define obj (new randomized-set%))
;; (define param_1 (send obj insert val))
```

```
;; (define param_2 (send obj remove val))
;; (define param_3 (send obj get-random))
```

Solutions

C++ Solution:

```
/*
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 RandomizedSet {
public:
RandomizedSet() {

}

bool insert(int val) {

}

bool remove(int val) {

}

int getRandom() {

}
};

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet* obj = new RandomizedSet();
* bool param_1 = obj->insert(val);
```

```
* bool param_2 = obj->remove(val);
* int param_3 = obj->getRandom();
*/
```

Java Solution:

```
/**
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 RandomizedSet {

public RandomizedSet() {

}

public boolean insert(int val) {

}

public boolean remove(int val) {

}

public int getRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = new RandomizedSet();
* boolean param_1 = obj.insert(val);
* boolean param_2 = obj.remove(val);
* int param_3 = obj.getRandom();
```

```
*/
```

Python3 Solution:

```
"""
Problem: Insert Delete GetRandom O(1)
Difficulty: Medium
Tags: array, math, 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 RandomizedSet:

def __init__(self):


def insert(self, val: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class RandomizedSet(object):

def __init__(self):


def insert(self, val):
"""
:type val: int
:rtype: bool
"""


def remove(self, val):
"""
:type val: int
:rtype: bool
```

```
"""


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



# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.getRandom()
```

JavaScript Solution:

```
/**
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 RandomizedSet = function() {

};

/**
* @param {number} val
* @return {boolean}
*/
RandomizedSet.prototype.insert = function(val) {

};
```

```
/**
* @param {number} val
* @return {boolean}
*/
RandomizedSet.prototype.remove = function(val) {

};

/**
* @return {number}
*/
RandomizedSet.prototype.getRandom = function() {

};

/**
* Your RandomizedSet object will be instantiated and called as such:
* var obj = new RandomizedSet()
* var param_1 = obj.insert(val)
* var param_2 = obj.remove(val)
* var param_3 = obj.getRandom()
*/
```

TypeScript Solution:

```
/**
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 RandomizedSet {
constructor() {

}

insert(val: number): boolean {
```

```
}

remove(val: number): boolean {

}

getRandom(): number {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* var obj = new RandomizedSet()
* var param_1 = obj.insert(val)
* var param_2 = obj.remove(val)
* var param_3 = obj.getRandom()
*/
```

C# Solution:

```
/*
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 RandomizedSet {

public RandomizedSet() {

}

public bool Insert(int val) {

}
```

```
public bool Remove(int val) {

}

public int GetRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = new RandomizedSet();
* bool param_1 = obj.Insert(val);
* bool param_2 = obj.Remove(val);
* int param_3 = obj.GetRandom();
*/
```

C Solution:

```
/*
* Problem: Insert Delete GetRandom O(1)
* Difficulty: Medium
* Tags: array, math, 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 {

} RandomizedSet;


RandomizedSet* randomizedSetCreate() {

}
```

```
bool randomizedSetInsert(RandomizedSet* obj, int val) {

}

bool randomizedSetRemove(RandomizedSet* obj, int val) {

}

int randomizedSetGetRandom(RandomizedSet* obj) {

}

void randomizedSetFree(RandomizedSet* obj) {

}

/**
* Your RandomizedSet struct will be instantiated and called as such:
* RandomizedSet* obj = randomizedSetCreate();
* bool param_1 = randomizedSetInsert(obj, val);

* bool param_2 = randomizedSetRemove(obj, val);

* int param_3 = randomizedSetGetRandom(obj);

* randomizedSetFree(obj);
*/
```

Go Solution:

```
// Problem: Insert Delete GetRandom O(1)
// Difficulty: Medium
// Tags: array, math, 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 RandomizedSet struct {
```

```
}


func Constructor() RandomizedSet {

}


func (this *RandomizedSet) Insert(val int) bool {

}


func (this *RandomizedSet) Remove(val int) bool {

}


func (this *RandomizedSet) GetRandom() int {

}


/**
* Your RandomizedSet object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Insert(val);
* param_2 := obj.Remove(val);
* param_3 := obj.GetRandom();
*/
```

Kotlin Solution:

```
class RandomizedSet() {

fun insert(`val`: Int): Boolean {

}

fun remove(`val`: Int): Boolean {
```

```
}

fun getRandom(): Int {

}

}

/**
* Your RandomizedSet object will be instantiated and called as such:
* var obj = RandomizedSet()
* var param_1 = obj.insert(`val`)
* var param_2 = obj.remove(`val`)
* var param_3 = obj.getRandom()
*/
```

Swift Solution:

```
class RandomizedSet {

init() {

}

func insert(_ val: Int) -> Bool {

}

func remove(_ val: Int) -> Bool {

}

func getRandom() -> Int {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* let obj = RandomizedSet()
```

```
* let ret_1: Bool = obj.insert(val)
* let ret_2: Bool = obj.remove(val)
* let ret_3: Int = obj.getRandom()
*/
```

Rust Solution:

```
// Problem: Insert Delete GetRandom O(1)
// Difficulty: Medium
// Tags: array, math, 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 RandomizedSet {

}


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

fn new() -> Self {

}

fn insert(&self, val: i32) -> bool {

}

fn remove(&self, val: i32) -> bool {

}

fn get_random(&self) -> i32 {

}
```

```
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* let obj = RandomizedSet::new();
* let ret_1: bool = obj.insert(val);
* let ret_2: bool = obj.remove(val);
* let ret_3: i32 = obj.get_random();
*/
```

Ruby Solution:

```
class RandomizedSet
def initialize()

end


=begin
:type val: Integer
:rtype: Boolean
=end
def insert(val)

end


=begin
:type val: Integer
:rtype: Boolean
=end
def remove(val)

end


=begin
:rtype: Integer
=end
def get_random()
```

```
end


end

# Your RandomizedSet object will be instantiated and called as such:
# obj = RandomizedSet.new()
# param_1 = obj.insert(val)
# param_2 = obj.remove(val)
# param_3 = obj.get_random()
```

PHP Solution:

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

}

/**
* @param Integer $val
* @return Boolean
*/
function insert($val) {

}

/**
* @param Integer $val
* @return Boolean
*/
function remove($val) {

}

/**
* @return Integer
*/
function getRandom() {
```

```
}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* $obj = RandomizedSet();
* $ret_1 = $obj->insert($val);
* $ret_2 = $obj->remove($val);
* $ret_3 = $obj->getRandom();
*/
```

Dart Solution:

```
class RandomizedSet {

RandomizedSet() {

}

bool insert(int val) {

}

bool remove(int val) {

}

int getRandom() {

}
}

/**
* Your RandomizedSet object will be instantiated and called as such:
* RandomizedSet obj = RandomizedSet();
* bool param1 = obj.insert(val);
* bool param2 = obj.remove(val);
* int param3 = obj.getRandom();
*/
```

Scala Solution:

```
class RandomizedSet() {

def insert(`val`: Int): Boolean = {

}

def remove(`val`: Int): Boolean = {

}

def getRandom(): Int = {

}

}

/**
* Your RandomizedSet object will be instantiated and called as such:
* val obj = new RandomizedSet()
* val param_1 = obj.insert(`val`)
* val param_2 = obj.remove(`val`)
* val param_3 = obj.getRandom()
*/
```

Elixir Solution:

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

end

@spec insert(val :: integer) :: boolean
def insert(val) do

end

@spec remove(val :: integer) :: boolean
def remove(val) do

end
```

```
@spec get_random() :: integer
def get_random() do

end
end

# Your functions will be called as such:
# RandomizedSet.init_()
# param_1 = RandomizedSet.insert(val)
# param_2 = RandomizedSet.remove(val)
# param_3 = RandomizedSet.get_random()

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

Erlang Solution:

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

-spec randomized_set_insert(Val :: integer()) -> boolean().
randomized_set_insert(Val) ->
.

-spec randomized_set_remove(Val :: integer()) -> boolean().
randomized_set_remove(Val) ->
.

-spec randomized_set_get_random() -> integer().
randomized_set_get_random() ->
.


%% Your functions will be called as such:
%% randomized_set_init_(),
%% Param_1 = randomized_set_insert(Val),
%% Param_2 = randomized_set_remove(Val),
%% Param_3 = randomized_set_get_random(),

%% randomized_set_init_ will be called before every test case, in which you
```

```
can do some necessary initializations.
```

Racket Solution:

```
(define randomized-set%
(class object%
(super-new)

(init-field)

; insert : exact-integer? -> boolean?
(define/public (insert val)
)
; remove : exact-integer? -> boolean?
(define/public (remove val)
)
; get-random : -> exact-integer?
(define/public (get-random)
)))

;; Your randomized-set% object will be instantiated and called as such:
;; (define obj (new randomized-set%))
;; (define param_1 (send obj insert val))
;; (define param_2 (send obj remove val))
;; (define param_3 (send obj get-random))
```