Problem 705: Design HashSet

Problem Information

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

Problem Description

Design a HashSet without using any built-in hash table libraries.

Implement

MyHashSet

class:

void add(key)

Inserts the value

key

into the HashSet.

bool contains(key)

Returns whether the value

key

exists in the HashSet or not.

void remove(key)

Removes the value

key

in the HashSet. If

key

does not exist in the HashSet, do nothing.

Example 1:

Input

["MyHashSet", "add", "add", "contains", "contains", "add", "contains", "remove", "contains"] [[], [1], [2], [1], [3], [2], [2], [2], [2]]

Output

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

Explanation

MyHashSet myHashSet = new MyHashSet(); myHashSet.add(1); // set = [1] myHashSet.add(2); // set = [1, 2] myHashSet.contains(1); // return True myHashSet.contains(3); // return False, (not found) myHashSet.add(2); // set = [1, 2] myHashSet.contains(2); // return True myHashSet.remove(2); // set = [1] myHashSet.contains(2); // return False, (already removed)

Constraints:

0 <= key <= 10

6

At most

10

4

calls will be made to

add

,

remove

, and

contains

.

Code Snippets

C++:

```
class MyHashSet {
public:
MyHashSet() {

}

void add(int key) {

}

void remove(int key) {

}

bool contains(int key) {

}
};

/**
```

```
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet* obj = new MyHashSet();
* obj->add(key);
* obj->remove(key);
* bool param_3 = obj->contains(key);
*/
```

Java:

```
class MyHashSet {

public MyHashSet() {

}

public void add(int key) {

}

public void remove(int key) {

}

public boolean contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = new MyHashSet();
* obj.add(key);
* obj.remove(key);
* boolean param_3 = obj.contains(key);
*/
```

Python3:

```
class MyHashSet:

def __init__(self):
```

```
def add(self, key: int) -> None:


def remove(self, key: int) -> None:


def contains(self, key: int) -> bool:



# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

Python:

```
class MyHashSet(object):

def __init__(self):


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


def remove(self, key):
"""
:type key: int
:rtype: None
"""


def contains(self, key):
"""
:type key: int
:rtype: bool
```

```
"""



# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

JavaScript:

```
var MyHashSet = function() {

};

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

};

/**
* @param {number} key
* @return {void}
*/
MyHashSet.prototype.remove = function(key) {

};

/**
* @param {number} key
* @return {boolean}
*/
MyHashSet.prototype.contains = function(key) {

};

/**
```

```
* Your MyHashSet object will be instantiated and called as such:
* var obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

TypeScript:

```
class MyHashSet {
constructor() {

}

add(key: number): void {

}

remove(key: number): void {

}

contains(key: number): boolean {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* var obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

C#:

```
public class MyHashSet {

public MyHashSet() {

}
```

}

```
public void Add(int key) {

}

public void Remove(int key) {

}

public bool Contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = new MyHashSet();
* obj.Add(key);
* obj.Remove(key);
* bool param_3 = obj.Contains(key);
*/
```

C:

```
typedef struct {

} MyHashSet;


MyHashSet* myHashSetCreate() {

}

void myHashSetAdd(MyHashSet* obj, int key) {

}

void myHashSetRemove(MyHashSet* obj, int key) {
}

bool myHashSetContains(MyHashSet* obj, int key) {

}

void myHashSetFree(MyHashSet* obj) {

}

/**
* Your MyHashSet struct will be instantiated and called as such:
* MyHashSet* obj = myHashSetCreate();
* myHashSetAdd(obj, key);

* myHashSetRemove(obj, key);

* bool param_3 = myHashSetContains(obj, key);

* myHashSetFree(obj);
*/
```

Go:

```
type MyHashSet struct {

}


func Constructor() MyHashSet {

}


func (this *MyHashSet) Add(key int) {

}


func (this *MyHashSet) Remove(key int) {

}
```

```
func (this *MyHashSet) Contains(key int) bool {

}


/**
* Your MyHashSet object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(key);
* obj.Remove(key);
* param_3 := obj.Contains(key);
*/
```

Kotlin:

```
class MyHashSet() {

fun add(key: Int) {

}

fun remove(key: Int) {

}

fun contains(key: Int): Boolean {

}

}

/**
* Your MyHashSet object will be instantiated and called as such:
* var obj = MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

Swift:

```

class MyHashSet {

init() {

}

func add(_ key: Int) {

}

func remove(_ key: Int) {

}

func contains(_ key: Int) -> Bool {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* let obj = MyHashSet()
* obj.add(key)
* obj.remove(key)
* let ret_3: Bool = obj.contains(key)
*/
```

Rust:

```
struct MyHashSet {

}



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

```
fn new() -> Self {

}

fn add(&self, key: i32) {

}

fn remove(&self, key: i32) {

}

fn contains(&self, key: i32) -> bool {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* let obj = MyHashSet::new();
* obj.add(key);
* obj.remove(key);
* let ret_3: bool = obj.contains(key);
*/
```

Ruby:

```
class MyHashSet
def initialize()

end



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

end
```

```
=begin
:type key: Integer
:rtype: Void
=end
def remove(key)

end



=begin
:type key: Integer
:rtype: Boolean
=end
def contains(key)

end



end

# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet.new()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

PHP:

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

}

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

}
/**
* @param Integer $key
* @return NULL
*/
function remove($key) {

}

/**
* @param Integer $key
* @return Boolean
*/
function contains($key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* $obj = MyHashSet();
* $obj->add($key);
* $obj->remove($key);
* $ret_3 = $obj->contains($key);
*/
```

Dart:

```
class MyHashSet {

MyHashSet() {

}

void add(int key) {

}

void remove(int key) {

}
bool contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = MyHashSet();
* obj.add(key);
* obj.remove(key);
* bool param3 = obj.contains(key);
*/
```

Scala:

```
class MyHashSet() {

def add(key: Int): Unit = {

}

def remove(key: Int): Unit = {

}

def contains(key: Int): Boolean = {

}

}

/**
* Your MyHashSet object will be instantiated and called as such:
* val obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* val param_3 = obj.contains(key)
*/
```

Elixir:

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

end

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

end

@spec remove(key :: integer) :: any
def remove(key) do

end

@spec contains(key :: integer) :: boolean
def contains(key) do

end
end

# Your functions will be called as such:
# MyHashSet.init_()
# MyHashSet.add(key)
# MyHashSet.remove(key)
# param_3 = MyHashSet.contains(key)

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

Erlang:

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

-spec my_hash_set_add(Key :: integer()) -> any().
my_hash_set_add(Key) ->
.

-spec my_hash_set_remove(Key :: integer()) -> any().
my_hash_set_remove(Key) ->
```

```
.

-spec my_hash_set_contains(Key :: integer()) -> boolean().
my_hash_set_contains(Key) ->
.



%% Your functions will be called as such:
%% my_hash_set_init_(),
%% my_hash_set_add(Key),
%% my_hash_set_remove(Key),
%% Param_3 = my_hash_set_contains(Key),

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

Racket:

```
(define my-hash-set%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add key)
)
; remove : exact-integer? -> void?
(define/public (remove key)
)
; contains : exact-integer? -> boolean?
(define/public (contains key)
)))

;; Your my-hash-set% object will be instantiated and called as such:
;; (define obj (new my-hash-set%))
;; (send obj add key)
;; (send obj remove key)
;; (define param_3 (send obj contains key))
```

Solutions

C++ Solution:

```
/*
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 MyHashSet {
public:
MyHashSet() {

}

void add(int key) {

}

void remove(int key) {

}

bool contains(int key) {

}
};

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet* obj = new MyHashSet();
* obj->add(key);
* obj->remove(key);
* bool param_3 = obj->contains(key);
*/
```

Java Solution:

```
/**
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 MyHashSet {

public MyHashSet() {

}

public void add(int key) {

}

public void remove(int key) {

}

public boolean contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = new MyHashSet();
* obj.add(key);
* obj.remove(key);
* boolean param_3 = obj.contains(key);
*/
```

Python3 Solution:

```
"""
Problem: Design HashSet
Difficulty: Easy
```

```
Tags: array, hash, linked_list

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 MyHashSet:

def __init__(self):



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

Python Solution:

```
class MyHashSet(object):

def __init__(self):



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



def remove(self, key):
"""
:type key: int
:rtype: None
"""



def contains(self, key):
"""
:type key: int
:rtype: bool
```

```
"""



# Your MyHashSet object will be instantiated and called as such:
# obj = MyHashSet()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

JavaScript Solution:

```
/**
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 MyHashSet = function() {

};

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

};

/**
* @param {number} key
* @return {void}
*/
MyHashSet.prototype.remove = function(key) {
};

/**
* @param {number} key
* @return {boolean}
*/
MyHashSet.prototype.contains = function(key) {

};

/**
* Your MyHashSet object will be instantiated and called as such:
* var obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

TypeScript Solution:

```
/**
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 MyHashSet {
constructor() {

}

add(key: number): void {

}

remove(key: number): void {
}

contains(key: number): boolean {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* var obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

C# Solution:

```
/*
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 MyHashSet {

public MyHashSet() {

}

public void Add(int key) {

}

public void Remove(int key) {

}
public bool Contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = new MyHashSet();
* obj.Add(key);
* obj.Remove(key);
* bool param_3 = obj.Contains(key);
*/
```

C Solution:

```
/*
* Problem: Design HashSet
* Difficulty: Easy
* Tags: array, hash, linked_list
*
* 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 {

} MyHashSet;


MyHashSet* myHashSetCreate() {

}

void myHashSetAdd(MyHashSet* obj, int key) {

}
```

```
void myHashSetRemove(MyHashSet* obj, int key) {

}

bool myHashSetContains(MyHashSet* obj, int key) {

}

void myHashSetFree(MyHashSet* obj) {

}

/**
* Your MyHashSet struct will be instantiated and called as such:
* MyHashSet* obj = myHashSetCreate();
* myHashSetAdd(obj, key);

* myHashSetRemove(obj, key);

* bool param_3 = myHashSetContains(obj, key);

* myHashSetFree(obj);
*/
```

Go Solution:

```
// Problem: Design HashSet
// Difficulty: Easy
// Tags: array, hash, linked_list
//
// 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 MyHashSet struct {

}


func Constructor() MyHashSet {
}


func (this *MyHashSet) Add(key int) {

}


func (this *MyHashSet) Remove(key int) {

}


func (this *MyHashSet) Contains(key int) bool {

}


/**
* Your MyHashSet object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(key);
* obj.Remove(key);
* param_3 := obj.Contains(key);
*/
```

Kotlin Solution:

```
class MyHashSet() {

fun add(key: Int) {

}

fun remove(key: Int) {

}

fun contains(key: Int): Boolean {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* var obj = MyHashSet()
* obj.add(key)
* obj.remove(key)
* var param_3 = obj.contains(key)
*/
```

Swift Solution:

```
class MyHashSet {

init() {

}

func add(_ key: Int) {

}

func remove(_ key: Int) {

}

func contains(_ key: Int) -> Bool {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* let obj = MyHashSet()
* obj.add(key)
* obj.remove(key)
* let ret_3: Bool = obj.contains(key)
*/
```

Rust Solution:

```
// Problem: Design HashSet
// Difficulty: Easy
// Tags: array, hash, linked_list
//
// 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 MyHashSet {

}


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

fn new() -> Self {

}

fn add(&self, key: i32) {

}

fn remove(&self, key: i32) {

}

fn contains(&self, key: i32) -> bool {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* let obj = MyHashSet::new();
* obj.add(key);
* obj.remove(key);
* let ret_3: bool = obj.contains(key);
*/
```

Ruby Solution:

```
class MyHashSet
def initialize()

end



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

end



=begin
:type key: Integer
:rtype: Void
=end
def remove(key)

end



=begin
:type key: Integer
:rtype: Boolean
=end
def contains(key)

end



end

# Your MyHashSet object will be instantiated and called as such:
```

```
# obj = MyHashSet.new()
# obj.add(key)
# obj.remove(key)
# param_3 = obj.contains(key)
```

PHP Solution:

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

}

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

}

/**
* @param Integer $key
* @return NULL
*/
function remove($key) {

}

/**
* @param Integer $key
* @return Boolean
*/
function contains($key) {

}
}

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

```
* $obj = MyHashSet();
* $obj->add($key);
* $obj->remove($key);
* $ret_3 = $obj->contains($key);
*/
```

Dart Solution:

```
class MyHashSet {

MyHashSet() {

}

void add(int key) {

}

void remove(int key) {

}

bool contains(int key) {

}
}

/**
* Your MyHashSet object will be instantiated and called as such:
* MyHashSet obj = MyHashSet();
* obj.add(key);
* obj.remove(key);
* bool param3 = obj.contains(key);
*/
```

Scala Solution:

```
class MyHashSet() {

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

def remove(key: Int): Unit = {

}

def contains(key: Int): Boolean = {

}

}

/**
* Your MyHashSet object will be instantiated and called as such:
* val obj = new MyHashSet()
* obj.add(key)
* obj.remove(key)
* val param_3 = obj.contains(key)
*/
```

Elixir Solution:

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

end

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

end

@spec remove(key :: integer) :: any
def remove(key) do

end

@spec contains(key :: integer) :: boolean
def contains(key) do
```

```
end
end

# Your functions will be called as such:
# MyHashSet.init_()
# MyHashSet.add(key)
# MyHashSet.remove(key)
# param_3 = MyHashSet.contains(key)

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

Erlang Solution:

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

-spec my_hash_set_add(Key :: integer()) -> any().
my_hash_set_add(Key) ->
.

-spec my_hash_set_remove(Key :: integer()) -> any().
my_hash_set_remove(Key) ->
.

-spec my_hash_set_contains(Key :: integer()) -> boolean().
my_hash_set_contains(Key) ->
.



%% Your functions will be called as such:
%% my_hash_set_init_(),
%% my_hash_set_add(Key),
%% my_hash_set_remove(Key),
%% Param_3 = my_hash_set_contains(Key),

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

Racket Solution:

```
(define my-hash-set%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add key)
)
; remove : exact-integer? -> void?
(define/public (remove key)
)
; contains : exact-integer? -> boolean?
(define/public (contains key)
)))

;; Your my-hash-set% object will be instantiated and called as such:
;; (define obj (new my-hash-set%))
;; (send obj add key)
;; (send obj remove key)
;; (define param_3 (send obj contains key))
```