Problem 706: Design HashMap

Problem Information

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

Problem Description

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

Implement the

MyHashMap

class:

MyHashMap()

initializes the object with an empty map.

void put(int key, int value)

inserts a

(key, value)

pair into the HashMap. If the

key

already exists in the map, update the corresponding

value

.

int get(int key)

returns the

value

to which the specified

key

is mapped, or

-1

if this map contains no mapping for the

key

.

void remove(key)

removes the

key

and its corresponding

value

if the map contains the mapping for the

key

.

Example 1:

Input

["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"] [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]

Output

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

Explanation

MyHashMap myHashMap = new MyHashMap(); myHashMap.put(1, 1); // The map is now [[1,1]] myHashMap.put(2, 2); // The map is now [[1,1], [2,2]] myHashMap.get(1); // return 1, The map is now [[1,1], [2,2]] myHashMap.get(3); // return -1 (i.e., not found), The map is now [[1,1], [2,2]] myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing value) myHashMap.get(2); // return 1, The map is now [[1,1], [2,1]] myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]] myHashMap.get(2); // return -1 (i.e., not found), The map is now [[1,1]]

Constraints:

0 <= key, value <= 10

6

At most

10

4

calls will be made to

put

,

get

, and

remove

.

Code Snippets

C++:

```
class MyHashMap {
public:
MyHashMap() {

}

void put(int key, int value) {

}

int get(int key) {

}

void remove(int key) {

}
};

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap* obj = new MyHashMap();
* obj->put(key,value);
* int param_2 = obj->get(key);
* obj->remove(key);
*/
```

Java:

```
class MyHashMap {
```

```
public MyHashMap() {

}

public void put(int key, int value) {

}

public int get(int key) {

}

public void remove(int key) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = new MyHashMap();
* obj.put(key,value);
* int param_2 = obj.get(key);
* obj.remove(key);
*/
```

Python3:

```
class MyHashMap:

def __init__(self):



def put(self, key: int, value: int) -> None:



def get(self, key: int) -> int:



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

```
# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap()
# obj.put(key,value)
# param_2 = obj.get(key)
# obj.remove(key)
```

Python:

```
class MyHashMap(object):

def __init__(self):



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



def get(self, key):
"""
:type key: int
:rtype: int
"""



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




# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap()
# obj.put(key,value)
# param_2 = obj.get(key)
# obj.remove(key)
```

JavaScript:

```
var MyHashMap = function() {

};

/**
* @param {number} key
* @param {number} value
* @return {void}
*/
MyHashMap.prototype.put = function(key, value) {

};

/**
* @param {number} key
* @return {number}
*/
MyHashMap.prototype.get = function(key) {

};

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

};

/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = new MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

TypeScript:

```
class MyHashMap {
constructor() {

}

put(key: number, value: number): void {

}

get(key: number): number {

}

remove(key: number): void {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = new MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

C#:

```
public class MyHashMap {

public MyHashMap() {

}

public void Put(int key, int value) {

}

public int Get(int key) {

}

public void Remove(int key) {
}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = new MyHashMap();
* obj.Put(key,value);
* int param_2 = obj.Get(key);
* obj.Remove(key);
*/
```

C:

```
typedef struct {

} MyHashMap;


MyHashMap* myHashMapCreate() {

}

void myHashMapPut(MyHashMap* obj, int key, int value) {

}

int myHashMapGet(MyHashMap* obj, int key) {

}

void myHashMapRemove(MyHashMap* obj, int key) {

}

void myHashMapFree(MyHashMap* obj) {

}
```

```
/**
* Your MyHashMap struct will be instantiated and called as such:
* MyHashMap* obj = myHashMapCreate();
* myHashMapPut(obj, key, value);

* int param_2 = myHashMapGet(obj, key);

* myHashMapRemove(obj, key);

* myHashMapFree(obj);
*/
```

Go:

```
type MyHashMap struct {

}


func Constructor() MyHashMap {

}


func (this *MyHashMap) Put(key int, value int) {

}


func (this *MyHashMap) Get(key int) int {

}


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

}


/**
* Your MyHashMap object will be instantiated and called as such:
* obj := Constructor();
* obj.Put(key,value);
* param_2 := obj.Get(key);
* obj.Remove(key);
*/
```

Kotlin:

```
class MyHashMap() {

fun put(key: Int, value: Int) {

}

fun get(key: Int): Int {

}

fun remove(key: Int) {

}

}

/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

Swift:

```
class MyHashMap {

init() {

}

func put(_ key: Int, _ value: Int) {
}

func get(_ key: Int) -> Int {

}

func remove(_ key: Int) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* let obj = MyHashMap()
* obj.put(key, value)
* let ret_2: Int = obj.get(key)
* obj.remove(key)
*/
```

Rust:

```
struct MyHashMap {

}



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

fn new() -> Self {

}

fn put(&self, key: i32, value: i32) {

}

fn get(&self, key: i32) -> i32 {
}

fn remove(&self, key: i32) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* let obj = MyHashMap::new();
* obj.put(key, value);
* let ret_2: i32 = obj.get(key);
* obj.remove(key);
*/
```

Ruby:

```
class MyHashMap
def initialize()

end



=begin
:type key: Integer
:type value: Integer
:rtype: Void
=end
def put(key, value)

end



=begin
:type key: Integer
:rtype: Integer
=end
def get(key)

end
```

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

end



end

# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap.new()
# obj.put(key, value)
# param_2 = obj.get(key)
# obj.remove(key)
```

PHP:

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

}

/**
* @param Integer $key
* @param Integer $value
* @return NULL
*/
function put($key, $value) {

}

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

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

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* $obj = MyHashMap();
* $obj->put($key, $value);
* $ret_2 = $obj->get($key);
* $obj->remove($key);
*/
```

Dart:

```
class MyHashMap {

MyHashMap() {

}

void put(int key, int value) {

}

int get(int key) {

}

void remove(int key) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = MyHashMap();
* obj.put(key,value);
* int param2 = obj.get(key);
* obj.remove(key);
*/
```

Scala:

```
class MyHashMap() {

def put(key: Int, value: Int): Unit = {

}

def get(key: Int): Int = {

}

def remove(key: Int): Unit = {

}

}

/**
* Your MyHashMap object will be instantiated and called as such:
* val obj = new MyHashMap()
* obj.put(key,value)
* val param_2 = obj.get(key)
* obj.remove(key)
*/
```

Elixir:

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

end

@spec put(key :: integer, value :: integer) :: any
def put(key, value) do
```

```
end

@spec get(key :: integer) :: integer
def get(key) do

end

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

end
end

# Your functions will be called as such:
# MyHashMap.init_()
# MyHashMap.put(key, value)
# param_2 = MyHashMap.get(key)
# MyHashMap.remove(key)

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

Erlang:

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

-spec my_hash_map_put(Key :: integer(), Value :: integer()) -> any().
my_hash_map_put(Key, Value) ->
.

-spec my_hash_map_get(Key :: integer()) -> integer().
my_hash_map_get(Key) ->
.

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



%% Your functions will be called as such:
```

```
%% my_hash_map_init_(),
%% my_hash_map_put(Key, Value),
%% Param_2 = my_hash_map_get(Key),
%% my_hash_map_remove(Key),

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

Racket:

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

(init-field)

; put : exact-integer? exact-integer? -> void?
(define/public (put key value)
)
; get : exact-integer? -> exact-integer?
(define/public (get key)
)
; remove : exact-integer? -> void?
(define/public (remove key)
)))

;; Your my-hash-map% object will be instantiated and called as such:
;; (define obj (new my-hash-map%))
;; (send obj put key value)
;; (define param_2 (send obj get key))
;; (send obj remove key)
```

Solutions

C++ Solution:

```
/*
* Problem: Design HashMap
* 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 MyHashMap {
public:
MyHashMap() {

}

void put(int key, int value) {

}

int get(int key) {

}

void remove(int key) {

}
};

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap* obj = new MyHashMap();
* obj->put(key,value);
* int param_2 = obj->get(key);
* obj->remove(key);
*/
```

Java Solution:

```
/**
* Problem: Design HashMap
* 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 MyHashMap {

public MyHashMap() {

}

public void put(int key, int value) {

}

public int get(int key) {

}

public void remove(int key) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = new MyHashMap();
* obj.put(key,value);
* int param_2 = obj.get(key);
* obj.remove(key);
*/
```

Python3 Solution:

```
"""
Problem: Design HashMap
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 MyHashMap:

def __init__(self):



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

Python Solution:

```
class MyHashMap(object):

def __init__(self):



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



def get(self, key):
"""
:type key: int
:rtype: int
"""



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

```
# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap()
# obj.put(key,value)
# param_2 = obj.get(key)
# obj.remove(key)
```

JavaScript Solution:

```
/**
* Problem: Design HashMap
* 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 MyHashMap = function() {

};

/**
* @param {number} key
* @param {number} value
* @return {void}
*/
MyHashMap.prototype.put = function(key, value) {

};

/**
* @param {number} key
* @return {number}
*/
MyHashMap.prototype.get = function(key) {

};

/**
```

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

};

/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = new MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

TypeScript Solution:

```
/**
* Problem: Design HashMap
* 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 MyHashMap {
constructor() {

}

put(key: number, value: number): void {

}

get(key: number): number {

}

remove(key: number): void {
}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = new MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

C# Solution:

```
/*
* Problem: Design HashMap
* 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 MyHashMap {

public MyHashMap() {

}

public void Put(int key, int value) {

}

public int Get(int key) {

}

public void Remove(int key) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = new MyHashMap();
* obj.Put(key,value);
* int param_2 = obj.Get(key);
* obj.Remove(key);
*/
```

C Solution:

```
/*
* Problem: Design HashMap
* 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 {

} MyHashMap;



MyHashMap* myHashMapCreate() {

}

void myHashMapPut(MyHashMap* obj, int key, int value) {

}

int myHashMapGet(MyHashMap* obj, int key) {

}
```

```
void myHashMapRemove(MyHashMap* obj, int key) {

}

void myHashMapFree(MyHashMap* obj) {

}

/**
* Your MyHashMap struct will be instantiated and called as such:
* MyHashMap* obj = myHashMapCreate();
* myHashMapPut(obj, key, value);

* int param_2 = myHashMapGet(obj, key);

* myHashMapRemove(obj, key);

* myHashMapFree(obj);
*/
```

Go Solution:

```
// Problem: Design HashMap
// 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 MyHashMap struct {

}


func Constructor() MyHashMap {

}
```

```
func (this *MyHashMap) Put(key int, value int) {

}


func (this *MyHashMap) Get(key int) int {

}


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

}


/**
* Your MyHashMap object will be instantiated and called as such:
* obj := Constructor();
* obj.Put(key,value);
* param_2 := obj.Get(key);
* obj.Remove(key);
*/
```

Kotlin Solution:

```
class MyHashMap() {

fun put(key: Int, value: Int) {

}

fun get(key: Int): Int {

}

fun remove(key: Int) {

}

}
```

```
/**
* Your MyHashMap object will be instantiated and called as such:
* var obj = MyHashMap()
* obj.put(key,value)
* var param_2 = obj.get(key)
* obj.remove(key)
*/
```

Swift Solution:

```
class MyHashMap {

init() {

}

func put(_ key: Int, _ value: Int) {

}

func get(_ key: Int) -> Int {

}

func remove(_ key: Int) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* let obj = MyHashMap()
* obj.put(key, value)
* let ret_2: Int = obj.get(key)
* obj.remove(key)
*/
```

Rust Solution:

```
// Problem: Design HashMap
// 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 MyHashMap {

}



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

fn new() -> Self {

}

fn put(&self, key: i32, value: i32) {

}

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

}

fn remove(&self, key: i32) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* let obj = MyHashMap::new();
* obj.put(key, value);
* let ret_2: i32 = obj.get(key);
* obj.remove(key);
*/
```

Ruby Solution:

```
class MyHashMap
def initialize()

end


=begin
:type key: Integer
:type value: Integer
:rtype: Void
=end
def put(key, value)

end


=begin
:type key: Integer
:rtype: Integer
=end
def get(key)

end


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

end


end

# Your MyHashMap object will be instantiated and called as such:
# obj = MyHashMap.new()
# obj.put(key, value)
# param_2 = obj.get(key)
# obj.remove(key)
```

PHP Solution:

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

}

/**
* @param Integer $key
* @param Integer $value
* @return NULL
*/
function put($key, $value) {

}

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

}

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

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* $obj = MyHashMap();
* $obj->put($key, $value);
* $ret_2 = $obj->get($key);
* $obj->remove($key);
*/
```

Dart Solution:

```
class MyHashMap {

MyHashMap() {

}

void put(int key, int value) {

}

int get(int key) {

}

void remove(int key) {

}
}

/**
* Your MyHashMap object will be instantiated and called as such:
* MyHashMap obj = MyHashMap();
* obj.put(key,value);
* int param2 = obj.get(key);
* obj.remove(key);
*/
```

Scala Solution:

```
class MyHashMap() {

def put(key: Int, value: Int): Unit = {

}
def get(key: Int): Int = {

}

def remove(key: Int): Unit = {

}

}

/**
* Your MyHashMap object will be instantiated and called as such:
* val obj = new MyHashMap()
* obj.put(key,value)
* val param_2 = obj.get(key)
* obj.remove(key)
*/
```

Elixir Solution:

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

end

@spec put(key :: integer, value :: integer) :: any
def put(key, value) do

end

@spec get(key :: integer) :: integer
def get(key) do

end

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

end
end
```

```
# Your functions will be called as such:
# MyHashMap.init_()
# MyHashMap.put(key, value)
# param_2 = MyHashMap.get(key)
# MyHashMap.remove(key)

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

Erlang Solution:

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

-spec my_hash_map_put(Key :: integer(), Value :: integer()) -> any().
my_hash_map_put(Key, Value) ->
.

-spec my_hash_map_get(Key :: integer()) -> integer().
my_hash_map_get(Key) ->
.

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


%% Your functions will be called as such:
%% my_hash_map_init_(),
%% my_hash_map_put(Key, Value),
%% Param_2 = my_hash_map_get(Key),
%% my_hash_map_remove(Key),

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

Racket Solution:

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

(init-field)

; put : exact-integer? exact-integer? -> void?
(define/public (put key value)
)
; get : exact-integer? -> exact-integer?
(define/public (get key)
)
; remove : exact-integer? -> void?
(define/public (remove key)
)))

;; Your my-hash-map% object will be instantiated and called as such:
;; (define obj (new my-hash-map%))
;; (send obj put key value)
;; (define param_2 (send obj get key))
;; (send obj remove key)
```