Problem 2502: Design Memory Allocator

Problem Information

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

Problem Description

You are given an integer

n

representing the size of a

0-indexed

memory array. All memory units are initially free.

You have a memory allocator with the following functionalities:

Allocate

a block of

size

consecutive free memory units and assign it the id

mID

.

Free

all memory units with the given id

mID

.

Note

that:

Multiple blocks can be allocated to the same

mID

.

You should free all the memory units with

mID

, even if they were allocated in different blocks.

Implement the

Allocator

class:

Allocator(int n)

Initializes an

Allocator

object with a memory array of size

n

.

int allocate(int size, int mID)

Find the

leftmost

block of

size

consecutive

free memory units and allocate it with the id

mID

. Return the block's first index. If such a block does not exist, return

-1

.

int freeMemory(int mID)

Free all memory units with the id

mID

. Return the number of memory units you have freed.

Example 1:

Input

["Allocator", "allocate", "allocate", "allocate", "freeMemory", "allocate", "allocate", "allocate", "freeMemory", "allocate", "freeMemory"] [[10], [1, 1], [1, 2], [1, 3], [2], [3, 4], [1, 1], [1, 1], [1], [10, 2], [7]]

Output

[null, 0, 1, 2, 1, 3, 1, 6, 3, -1, 0]

Explanation

Allocator loc = new Allocator(10); // Initialize a memory array of size 10. All memory units are initially free. loc.allocate(1, 1); // The leftmost block's first index is 0. The memory array becomes [

1

,_,_,_,_,_,_,_,_,_]. We return 0. loc.allocate(1, 2); // The leftmost block's first index is 1. The memory array becomes [1,

2

,_,_,_,_,_,_,_,_]. We return 1. loc.allocate(1, 3); // The leftmost block's first index is 2. The memory array becomes [1,2,

3

,_,_,_,_,_,_,_]. We return 2. loc.freeMemory(2); // Free all memory units with mID 2. The memory array becomes [1,_, 3,_,_,_,_,_,_,_]. We return 1 since there is only 1 unit with mID 2. loc.allocate(3, 4); // The leftmost block's first index is 3. The memory array becomes [1,_,3,

4

,

4

,

4

,_,_,_,_]. We return 3. loc.allocate(1, 1); // The leftmost block's first index is 1. The memory array becomes [1,

1

,3,4,4,4,_,_,_,_]. We return 1. loc.allocate(1, 1); // The leftmost block's first index is 6. The memory array becomes [1,1,3,4,4,4,

1

,_,_,_]. We return 6. loc.freeMemory(1); // Free all memory units with mID 1. The memory array becomes [_,_,3,4,4,4,_,_,_,_]. We return 3 since there are 3 units with mID 1. loc.allocate(10, 2); // We can not find any free block with 10 consecutive free memory units, so we return -1. loc.freeMemory(7); // Free all memory units with mID 7. The memory array remains the same since there is no memory unit with mID 7. We return 0.

Constraints:

1 <= n, size, mID <= 1000

At most

1000

calls will be made to

allocate

and

freeMemory

.

Code Snippets

C++:

```
class Allocator {
public:
Allocator(int n) {

}
```

```
int allocate(int size, int mID) {

}

int freeMemory(int mID) {

}
};

/**
* Your Allocator object will be instantiated and called as such:
* Allocator* obj = new Allocator(n);
* int param_1 = obj->allocate(size,mID);
* int param_2 = obj->freeMemory(mID);
*/
```

Java:

```
class Allocator {

public Allocator(int n) {

}

public int allocate(int size, int mID) {

}

public int freeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = new Allocator(n);
* int param_1 = obj.allocate(size,mID);
* int param_2 = obj.freeMemory(mID);
*/
```

Python3:

```
class Allocator:

def __init__(self, n: int):


def allocate(self, size: int, mID: int) -> int:


def freeMemory(self, mID: int) -> int:



# Your Allocator object will be instantiated and called as such:
# obj = Allocator(n)
# param_1 = obj.allocate(size,mID)
# param_2 = obj.freeMemory(mID)
```

Python:

```
class Allocator(object):

def __init__(self, n):
"""
:type n: int
"""


def allocate(self, size, mID):
"""
:type size: int
:type mID: int
:rtype: int
"""


def freeMemory(self, mID):
"""
:type mID: int
:rtype: int
"""
```

```
# Your Allocator object will be instantiated and called as such:
# obj = Allocator(n)
# param_1 = obj.allocate(size,mID)
# param_2 = obj.freeMemory(mID)
```

JavaScript:

```
/**
* @param {number} n
*/
var Allocator = function(n) {

};

/**
* @param {number} size
* @param {number} mID
* @return {number}
*/
Allocator.prototype.allocate = function(size, mID) {

};

/**
* @param {number} mID
* @return {number}
*/
Allocator.prototype.freeMemory = function(mID) {

};

/**
* Your Allocator object will be instantiated and called as such:
* var obj = new Allocator(n)
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
*/
```

TypeScript:

```
class Allocator {
constructor(n: number) {
```

```
}

allocate(size: number, mID: number): number {

}

freeMemory(mID: number): number {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* var obj = new Allocator(n)
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
*/
```

C#:

```
public class Allocator {

public Allocator(int n) {

}

public int Allocate(int size, int mID) {

}

public int FreeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = new Allocator(n);
* int param_1 = obj.Allocate(size,mID);
* int param_2 = obj.FreeMemory(mID);
*/
```

C:

```
typedef struct {

} Allocator;


Allocator* allocatorCreate(int n) {

}

int allocatorAllocate(Allocator* obj, int size, int mID) {

}

int allocatorFreeMemory(Allocator* obj, int mID) {

}

void allocatorFree(Allocator* obj) {

}

/**
* Your Allocator struct will be instantiated and called as such:
* Allocator* obj = allocatorCreate(n);
* int param_1 = allocatorAllocate(obj, size, mID);

* int param_2 = allocatorFreeMemory(obj, mID);

* allocatorFree(obj);
*/
```

Go:

```
type Allocator struct {

}
```

```
func Constructor(n int) Allocator {

}


func (this *Allocator) Allocate(size int, mID int) int {

}


func (this *Allocator) FreeMemory(mID int) int {

}


/**
* Your Allocator object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Allocate(size,mID);
* param_2 := obj.FreeMemory(mID);
*/
```

Kotlin:

```
class Allocator(n: Int) {

fun allocate(size: Int, mID: Int): Int {

}

fun freeMemory(mID: Int): Int {

}

}

/**
* Your Allocator object will be instantiated and called as such:
* var obj = Allocator(n)
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
```

```
*/
```

Swift:

```
class Allocator {

init(_ n: Int) {

}

func allocate(_ size: Int, _ mID: Int) -> Int {

}

func freeMemory(_ mID: Int) -> Int {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* let obj = Allocator(n)
* let ret_1: Int = obj.allocate(size, mID)
* let ret_2: Int = obj.freeMemory(mID)
*/
```

Rust:

```
struct Allocator {

}


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

fn new(n: i32) -> Self {
```

```
}

fn allocate(&self, size: i32, m_id: i32) -> i32 {

}

fn free_memory(&self, m_id: i32) -> i32 {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* let obj = Allocator::new(n);
* let ret_1: i32 = obj.allocate(size, mID);
* let ret_2: i32 = obj.free_memory(mID);
*/
```

Ruby:

```
class Allocator

=begin
:type n: Integer
=end
def initialize(n)

end


=begin
:type size: Integer
:type m_id: Integer
:rtype: Integer
=end
def allocate(size, m_id)

end


=begin
:type m_id: Integer
```

```
:rtype: Integer
=end
def free_memory(m_id)

end



end

# Your Allocator object will be instantiated and called as such:
# obj = Allocator.new(n)
# param_1 = obj.allocate(size, m_id)
# param_2 = obj.free_memory(m_id)
```

PHP:

```
class Allocator {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @param Integer $size
* @param Integer $mID
* @return Integer
*/
function allocate($size, $mID) {

}

/**
* @param Integer $mID
* @return Integer
*/
function freeMemory($mID) {

}
}
```

```
/**
* Your Allocator object will be instantiated and called as such:
* $obj = Allocator($n);
* $ret_1 = $obj->allocate($size, $mID);
* $ret_2 = $obj->freeMemory($mID);
*/
```

Dart:

```
class Allocator {

Allocator(int n) {

}

int allocate(int size, int mID) {

}

int freeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = Allocator(n);
* int param1 = obj.allocate(size,mID);
* int param2 = obj.freeMemory(mID);
*/
```

Scala:

```
class Allocator(_n: Int) {

def allocate(size: Int, mID: Int): Int = {

}

def freeMemory(mID: Int): Int = {

}
```

```
}

/**
* Your Allocator object will be instantiated and called as such:
* val obj = new Allocator(n)
* val param_1 = obj.allocate(size,mID)
* val param_2 = obj.freeMemory(mID)
*/
```

Elixir:

```
defmodule Allocator do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec allocate(size :: integer, m_id :: integer) :: integer
def allocate(size, m_id) do

end

@spec free_memory(m_id :: integer) :: integer
def free_memory(m_id) do

end
end

# Your functions will be called as such:
# Allocator.init_(n)
# param_1 = Allocator.allocate(size, m_id)
# param_2 = Allocator.free_memory(m_id)

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

Erlang:

```
-spec allocator_init_(N :: integer()) -> any().
allocator_init_(N) ->
.
```

```
-spec allocator_allocate(Size :: integer(), MID :: integer()) -> integer().
allocator_allocate(Size, MID) ->
.

-spec allocator_free_memory(MID :: integer()) -> integer().
allocator_free_memory(MID) ->
.


%% Your functions will be called as such:
%% allocator_init_(N),
%% Param_1 = allocator_allocate(Size, MID),
%% Param_2 = allocator_free_memory(MID),

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

Racket:

```
(define allocator%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; allocate : exact-integer? exact-integer? -> exact-integer?
(define/public (allocate size m-id)
)
; free-memory : exact-integer? -> exact-integer?
(define/public (free-memory m-id)
)))

;; Your allocator% object will be instantiated and called as such:
;; (define obj (new allocator% [n n]))
;; (define param_1 (send obj allocate size m-id))
;; (define param_2 (send obj free-memory m-id))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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 Allocator {
public:
Allocator(int n) {

}

int allocate(int size, int mID) {

}

int freeMemory(int mID) {

}
};

/**
* Your Allocator object will be instantiated and called as such:
* Allocator* obj = new Allocator(n);
* int param_1 = obj->allocate(size,mID);
* int param_2 = obj->freeMemory(mID);
*/
```

Java Solution:

```
/**
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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 Allocator {

public Allocator(int n) {

}

public int allocate(int size, int mID) {

}

public int freeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = new Allocator(n);
* int param_1 = obj.allocate(size,mID);
* int param_2 = obj.freeMemory(mID);
*/
```

Python3 Solution:

```
"""
Problem: Design Memory Allocator
Difficulty: Medium
Tags: array, 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 Allocator:
```

```
def __init__(self, n: int):


def allocate(self, size: int, mID: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Allocator(object):

def __init__(self, n):
"""
:type n: int
"""



def allocate(self, size, mID):
"""
:type size: int
:type mID: int
:rtype: int
"""



def freeMemory(self, mID):
"""
:type mID: int
:rtype: int
"""




# Your Allocator object will be instantiated and called as such:
# obj = Allocator(n)
# param_1 = obj.allocate(size,mID)
# param_2 = obj.freeMemory(mID)
```

JavaScript Solution:

```
/**
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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
*/

/**
* @param {number} n
*/
var Allocator = function(n) {

};

/**
* @param {number} size
* @param {number} mID
* @return {number}
*/
Allocator.prototype.allocate = function(size, mID) {

};

/**
* @param {number} mID
* @return {number}
*/
Allocator.prototype.freeMemory = function(mID) {

};

/**
* Your Allocator object will be instantiated and called as such:
* var obj = new Allocator(n)
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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 Allocator {
constructor(n: number) {

}

allocate(size: number, mID: number): number {

}

freeMemory(mID: number): number {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* var obj = new Allocator(n)
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
*/
```

C# Solution:

```
/*
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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 Allocator {

public Allocator(int n) {

}

public int Allocate(int size, int mID) {

}

public int FreeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = new Allocator(n);
* int param_1 = obj.Allocate(size,mID);
* int param_2 = obj.FreeMemory(mID);
*/
```

C Solution:

```
/*
* Problem: Design Memory Allocator
* Difficulty: Medium
* Tags: array, 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 {

} Allocator;
```

```
Allocator* allocatorCreate(int n) {

}

int allocatorAllocate(Allocator* obj, int size, int mID) {

}

int allocatorFreeMemory(Allocator* obj, int mID) {

}

void allocatorFree(Allocator* obj) {

}

/**
* Your Allocator struct will be instantiated and called as such:
* Allocator* obj = allocatorCreate(n);
* int param_1 = allocatorAllocate(obj, size, mID);

* int param_2 = allocatorFreeMemory(obj, mID);

* allocatorFree(obj);
*/
```

Go Solution:

```
// Problem: Design Memory Allocator
// Difficulty: Medium
// Tags: array, 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 Allocator struct {

}
```

```
func Constructor(n int) Allocator {

}


func (this *Allocator) Allocate(size int, mID int) int {

}


func (this *Allocator) FreeMemory(mID int) int {

}


/**
* Your Allocator object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Allocate(size,mID);
* param_2 := obj.FreeMemory(mID);
*/
```

Kotlin Solution:

```
class Allocator(n: Int) {

fun allocate(size: Int, mID: Int): Int {

}

fun freeMemory(mID: Int): Int {

}

}

/**
* Your Allocator object will be instantiated and called as such:
* var obj = Allocator(n)
```

```
* var param_1 = obj.allocate(size,mID)
* var param_2 = obj.freeMemory(mID)
*/
```

Swift Solution:

```
class Allocator {

init(_ n: Int) {

}

func allocate(_ size: Int, _ mID: Int) -> Int {

}

func freeMemory(_ mID: Int) -> Int {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* let obj = Allocator(n)
* let ret_1: Int = obj.allocate(size, mID)
* let ret_2: Int = obj.freeMemory(mID)
*/
```

Rust Solution:

```
// Problem: Design Memory Allocator
// Difficulty: Medium
// Tags: array, 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 Allocator {
```

```
}


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

fn new(n: i32) -> Self {

}

fn allocate(&self, size: i32, m_id: i32) -> i32 {

}

fn free_memory(&self, m_id: i32) -> i32 {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* let obj = Allocator::new(n);
* let ret_1: i32 = obj.allocate(size, mID);
* let ret_2: i32 = obj.free_memory(mID);
*/
```

Ruby Solution:

```
class Allocator

=begin
:type n: Integer
=end
def initialize(n)

end
```

```
=begin
:type size: Integer
:type m_id: Integer
:rtype: Integer
=end
def allocate(size, m_id)

end


=begin
:type m_id: Integer
:rtype: Integer
=end
def free_memory(m_id)

end



end

# Your Allocator object will be instantiated and called as such:
# obj = Allocator.new(n)
# param_1 = obj.allocate(size, m_id)
# param_2 = obj.free_memory(m_id)
```

PHP Solution:

```
class Allocator {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @param Integer $size
* @param Integer $mID
* @return Integer
*/
```

```
function allocate($size, $mID) {

}

/**
* @param Integer $mID
* @return Integer
*/
function freeMemory($mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* $obj = Allocator($n);
* $ret_1 = $obj->allocate($size, $mID);
* $ret_2 = $obj->freeMemory($mID);
*/
```

Dart Solution:

```
class Allocator {

Allocator(int n) {

}

int allocate(int size, int mID) {

}

int freeMemory(int mID) {

}
}

/**
* Your Allocator object will be instantiated and called as such:
* Allocator obj = Allocator(n);
* int param1 = obj.allocate(size,mID);
```

```
* int param2 = obj.freeMemory(mID);
*/
```

Scala Solution:

```
class Allocator(_n: Int) {

def allocate(size: Int, mID: Int): Int = {

}

def freeMemory(mID: Int): Int = {

}

}

/**
* Your Allocator object will be instantiated and called as such:
* val obj = new Allocator(n)
* val param_1 = obj.allocate(size,mID)
* val param_2 = obj.freeMemory(mID)
*/
```

Elixir Solution:

```
defmodule Allocator do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec allocate(size :: integer, m_id :: integer) :: integer
def allocate(size, m_id) do

end

@spec free_memory(m_id :: integer) :: integer
def free_memory(m_id) do

end
```

```
end

# Your functions will be called as such:
# Allocator.init_(n)
# param_1 = Allocator.allocate(size, m_id)
# param_2 = Allocator.free_memory(m_id)

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

Erlang Solution:

```
-spec allocator_init_(N :: integer()) -> any().
allocator_init_(N) ->
.

-spec allocator_allocate(Size :: integer(), MID :: integer()) -> integer().
allocator_allocate(Size, MID) ->
.

-spec allocator_free_memory(MID :: integer()) -> integer().
allocator_free_memory(MID) ->
.


%% Your functions will be called as such:
%% allocator_init_(N),
%% Param_1 = allocator_allocate(Size, MID),
%% Param_2 = allocator_free_memory(MID),

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

Racket Solution:

```
(define allocator%
(class object%
(super-new)

; n : exact-integer?
(init-field
```

```
n)

; allocate : exact-integer? exact-integer? -> exact-integer?
(define/public (allocate size m-id)
)
; free-memory : exact-integer? -> exact-integer?
(define/public (free-memory m-id)
)))

;; Your allocator% object will be instantiated and called as such:
;; (define obj (new allocator% [n n]))
;; (define param_1 (send obj allocate size m-id))
;; (define param_2 (send obj free-memory m-id))
```