Problem 2502: Design Memory Allocator

Problem Information

Difficulty: Medium
Acceptance Rate: 49.26%
Paid Only: No
Tags: Array, Hash Table, Design, Simulation

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:

1. **Allocate** a block of `size` consecutive free memory units and assign it the id `mID`. 2. **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)
```