Problem 2671: Frequency Tracker

Problem Information

Difficulty: Medium
Acceptance Rate: 30.45%
Paid Only: No
Tags: Hash Table, Design

Problem Description

Design a data structure that keeps track of the values in it and answers some queries regarding their frequencies.

Implement the `FrequencyTracker` class.

* `FrequencyTracker()`: Initializes the `FrequencyTracker` object with an empty array initially. * `void add(int number)`: Adds `number` to the data structure. * `void deleteOne(int number)`: Deletes **one** occurrence of `number` from the data structure. The data structure **may not contain** `number`, and in this case nothing is deleted. * `bool hasFrequency(int frequency)`: Returns `true` if there is a number in the data structure that occurs `frequency` number of times, otherwise, it returns `false`.

Example 1:

Input ["FrequencyTracker", "add", "add", "hasFrequency"] [[], [3], [3], [2]] **Output** [null, null, null, true] **Explanation** FrequencyTracker frequencyTracker = new FrequencyTracker(); frequencyTracker.add(3); // The data structure now contains [3] frequencyTracker.add(3); // The data structure now contains [3, 3] frequencyTracker.hasFrequency(2); // Returns true, because 3 occurs twice

Example 2:

Input ["FrequencyTracker", "add", "deleteOne", "hasFrequency"] [[], [1], [1], [1]] **Output** [null, null, null, false] **Explanation** FrequencyTracker frequencyTracker = new FrequencyTracker(); frequencyTracker.add(1); // The data structure now contains [1] frequencyTracker.deleteOne(1); // The data structure becomes empty []

frequencyTracker.hasFrequency(1); // Returns false, because the data structure is empty

Example 3:

Input ["FrequencyTracker", "hasFrequency", "add", "hasFrequency"] [[], [2], [3], [1]] **Output** [null, false, null, true] **Explanation** FrequencyTracker frequencyTracker = new FrequencyTracker(); frequencyTracker.hasFrequency(2); // Returns false, because the data structure is empty frequencyTracker.add(3); // The data structure now contains [3] frequencyTracker.hasFrequency(1); // Returns true, because 3 occurs once

Constraints:

* `1 <= number <= 105` * `1 <= frequency <= 105` * At most, `2 * 105` calls will be made to `add`, `deleteOne`, and `hasFrequency` in **total**.

Code Snippets

C++:

```
class FrequencyTracker {
public:
FrequencyTracker() {

}

void add(int number) {

}

void deleteOne(int number) {

}

bool hasFrequency(int frequency) {

}
};

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

```
* FrequencyTracker* obj = new FrequencyTracker();
* obj->add(number);
* obj->deleteOne(number);
* bool param_3 = obj->hasFrequency(frequency);
*/
```

Java:

```
class FrequencyTracker {

public FrequencyTracker() {

}

public void add(int number) {

}

public void deleteOne(int number) {

}

public boolean hasFrequency(int frequency) {

}
}

/**
* Your FrequencyTracker object will be instantiated and called as such:
* FrequencyTracker obj = new FrequencyTracker();
* obj.add(number);
* obj.deleteOne(number);
* boolean param_3 = obj.hasFrequency(frequency);
*/
```

Python3:

```
class FrequencyTracker:

def __init__(self):
```

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


def deleteOne(self, number: int) -> None:


def hasFrequency(self, frequency: int) -> bool:



# Your FrequencyTracker object will be instantiated and called as such:
# obj = FrequencyTracker()
# obj.add(number)
# obj.deleteOne(number)
# param_3 = obj.hasFrequency(frequency)
```