DS 210 Discussion 9 - Leetcode Again!

A. 2671. Frequency Tracker

https://leetcode.com/problems/frequency-tracker/description/

After passing the question, submit your screenshot as instructions on the gradescope. (Deadline: Friday, Apr. 19, 11:59 PM)

A.1. Introduction

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

- `FrequencyTracker::new()`: Initializes the FrequencyTracker object with an empty array initially.
  - $D = \emptyset$, i.e, $D[\text{val}] = 0, \forall \text{val}$
- `fn add(val: i32)`: Adds one occurrence of val to the data structure.
  - $D[\text{val}] = D[\text{val}] + 1$
- `fn deleteOne(val: i32)`: Deletes one occurrence of val from the data structure. **The data structure may not contain val, and in this case nothing is deleted.**
  - $D[\text{val}] = D[\text{val}] - 1$, if $D[\text{val}] > 0$
- `fn hasFrequency(frequency: i32) -> bool`: Returns true if there is a number in the data structure that occurs frequency number of times, otherwise, it returns false.
  - Is there any val in D that $D[\text{val}] = \text{frequency}$?

A.2. Naive Approach

Hint: Use a `HashMap` from value to occurrences.

1. What will happen if the value is not in hashmap? In add and delete?
2. How will we do `hasFrequency` in the hashmap? How many elements we need to check?

A.3. The Solution

Hint: How can we handle `hasFrequency` in $O(1)$ time with an additional `HashMap`? How should it be updated in sync with the original hashmap?

```rust
use std::collections::HashMap;

struct FrequencyTracker {
    pub val_to_freq: HashMap<i32, i32>,
}

impl FrequencyTracker {
    fn new() -> Self {
        FrequencyTracker {
            val_to_freq: HashMap::new(),
        }
    }

    fn add(&mut self, val: i32) {
        self.val_to_freq
            .entry(val)
            .and_modify(|f| *f += 1)
            .or_insert(1);
    }

    fn delete_one(&mut self, val: i32) {
        self.val_to_freq.entry(val).and_modify(|f| {
            if *f > 0 {
                *f -= 1;
            }
        });
    }

    fn has_frequency(&self, frequency: i32) -> bool {
        for (_, freq) in self.val_to_freq.iter() {
            if *freq == frequency {
                return true;
            }
        }
        return false;
    }
}
```