1207.Unique Number of Occurrences

Problem Understanding

Statement:

Hume ek integer array `arr` diya gaya hai. Har value kitni baar aayi hai uska count lena hai, aur check karna hai ki **kisi do value ka occurrence count same na ho**.

- Agar har element ka occurrence count unique hai → return `true`
- Agar kisi bhi do element ka occurrence count same hai → return `false`

Example 1

```
arr = [1,2,2,1,1,3]

Frequency:
1 → 3 times
2 → 2 times
3 → 1 time

Occurrences = {3, 2, 1} → all unique → Output = true
```

Your Approach

Tumne **2 data structures** use kiye:

1. `unordered_map<int,int> mp` → Store karta hai har element ka frequency count.
    - Key = element
    - Value = frequency
2. `unordered_set<int> s` → Store karta hai unique elements of array.

Algorithm Flow

1. **Frequency count & unique elements store**
   - Loop through `arr`
     - `mp[arr[i]]++` → frequency badhao
     - `s.insert(arr[i])` → unique element add
2. **Check frequency uniqueness**
   - Har element `ele` in `s` ke liye:
     - Uska frequency `freq = mp[ele]` nikalo
     - Fir poore `mp` ko loop karo:
       - Agar current `mp_ele.first == ele` → skip (apne aap se compare nahi karte)
       - Agar `freq == mp_ele.second` → matlab kisi aur element ka frequency same hai → return `false`
3. Agar pura loop complete hua without conflicts → return `true`.

Code (tumhara hi)

```
class Solution {
public:
    bool uniqueOccurrences(vector<int>& arr) {
        unordered_map<int,int>mp;
        unordered_set<int>s;
        for(int i=0;i<arr.size();i++){
            mp[arr[i]]++;
            s.insert(arr[i]);
        }
        for(int ele : s){
            int freq = mp[ele];
            for(auto mp_ele : mp){
                int freq2 = mp_ele.second;
```

```
            if(mp_ele.first == ele) continue;
            if(freq == freq2) return false;
          }
        }
      return true;
    }
};
```

Dry Run

Example:

```
arr = [1,2,2,1,1,3]
```

Step 1 – Build map & set

Loop i from 0 to 5:

i	arr[i]	mp after step	s after step
0	1	{1:1}	{1}
1	2	{1:1, 2:1}	{1, 2}
2	2	{1:1, 2:2}	{1, 2}
3	1	{1:2, 2:2}	{1, 2}
4	1	{1:3, 2:2}	{1, 2}
5	3	{1:3, 2:2, 3:1}	{1, 2, 3}

Step 2 – Check frequency uniqueness

Outer loop on `s`:

- **ele = 1**
  - freq = mp[1] = 3
  - Inner loop mp:

        - (1, 3) → skip
        - (2, 2) → not match
        - (3, 1) → not match

          → No duplicate found
- **ele = 2**
    - freq = 2
    - Inner loop mp:
        - (1, 3) → not match
        - (2, 2) → skip
        - (3, 1) → not match

          → No duplicate found
- **ele = 3**
    - freq = 1
    - Inner loop mp:
        - (1, 3) → not match
        - (2, 2) → not match
        - (3, 1) → skip

          → No duplicate found

Step 3 – Return result

- Koi bhi frequency duplicate nahi mili → return `true`.

Time Complexity

- Step 1: O(n) (map + set insertions)
- Step 2: $O(k^2)$ where k = number of unique elements (worst case sab alag ho to $k \approx n$)

- Overall worst case: $O(n^2)$ (agar saare elements unique ho)

Space Complexity

- Map stores at most n elements → O(n)
- Set stores at most n elements → O(n)

Possible Optimization

Tumne har element ka frequency compare har dusre ke saath kiya, isko optimize karke sirf **frequency ko ek set me store karke check kar sakte** ho, jo O(n) time me ho jayega.

(Lekin tumne bola tha code change nahi karna, isliye yeh sirf note hai.)
