Degree of an Array

Given a non-empty array of non-negative integers `nums`, the **degree** of this array is defined as the maximum frequency of any one of its elements.

Your task is to find the smallest possible length of a (contiguous) subarray of `nums`, that has the same degree as `nums`.

Example 1:

```
Input: [1, 2, 2, 3, 1]
Output: 2
Explanation:
The input array has a degree of 2 because both elements 1 and 2 appear twice.
Of the subarrays that have the same degree:
[1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2]
The shortest length is 2. So return 2.
```

Example 2:

```
Input: [1,2,2,3,1,4,2]
Output: 6
```

Note:

- `nums.length` will be between 1 and 50,000.
- `nums[i]` will be an integer between 0 and 49,999.

Solution 1

1. Get *degree* of array, frequency of all integers in array, and the indices of the first and last occurrence of each integer in the array
2. Return the minimum occurrence range of each integer which appears *degree* number of times in the array,

```
public int findShortestSubArray(int[] nums) {
    int degree = 0, n = nums.length, minSize = n;
    Map<Integer, Integer> map = new HashMap<>();
    Map<Integer, Integer[]> map2 = new HashMap<>();

    for (int i=0;i<n;i++) {
        map.put(nums[i], map.getOrDefault(nums[i], 0) + 1);
        degree = Math.max(degree, map.get(nums[i]));

        if (map2.get(nums[i]) == null) map2.put(nums[i], new Integer[2]);
        Integer[] numRange = map2.get(nums[i]);
        if (numRange[0] == null) numRange[0] = i;
        numRange[1] = i;
    }

    for (Map.Entry<Integer, Integer> entry : map.entrySet()) {
        if (entry.getValue() != degree) continue;
        Integer[] range = map2.get(entry.getKey());
        minSize = Math.min(minSize, range[1] - range[0] + 1);
    }
    return minSize;
}
```

written by compton_scatter original link here

Solution 2

```python
def findShortestSubArray(self, nums):
        c = collections.Counter(nums)
        first, last = {}, {}
        for i, v in enumerate(nums):
            first.setdefault(v, i)
            last[v] = i
        degree = max(c.values())
        return min(last[v] - first[v] + 1 for v in c if c[v] == degree)
```

written by lee215 original link here

Solution 3

Using two hash map.
One records the starting index for the character.
The other records the frequency of the character.

Once a certain character's frequency is bigger than others. we update the variable len. When more than two character have the same frequency, just compare their length, chose the shorter one.

```
class Solution {
public:
    int findShortestSubArray(vector<int>& nums) {
        if (nums.size() < 2) return nums.size();
        int res = nums.size();
        unordered_map<int, int> startIndex, count;
        int len = nums.size(), fre = 0;
        for (int i = 0; i < nums.size() ;i++) {
            if (startIndex.count(nums[i]) == 0) startIndex[nums[i]] = i;
            count[nums[i]]++;
            if (count[nums[i]] == fre){
                len = min(i - startIndex[nums[i]] + 1, len);
            }
            if (count[nums[i]] > fre){
                len = i - startIndex[nums[i]] + 1;
                fre = count[nums[i]];
            }
        }
        return len;
    }
};
```

written by angiehoo original link here