Sub-Arrays With Equal Number Of Occurences

solved by	Senan
Platform	GeeksForGeeks
difficulty	Hard
tags	Hash Map Prefix Sum
language	C++
solved on	@22/10/2024
link	https://www.geeksforgeeks.org/problems/sub-arrays-with-equal-number-of-occurences3901/1
Completion	✓

Intuition

The problem aims to find how many subarrays exist in which the frequency of `x` equals the frequency of `y`. We can solve this by using a difference counter between the occurrences of `x` and `y`. If at any point, the difference between the counts of `x` and `y` becomes zero (i.e., `cnt(x) - cnt(y) = 0`), it implies that `x` and `y` have occurred the same number of times up to that point.

Approach

1. We use a counter (`cnt`) to track the difference between occurrences of `x` and `y` as we iterate through the array.
2. Whenever we encounter `x`, we increment the counter, and whenever we encounter `y`, we decrement the counter.
3. To efficiently count how many times a particular difference has been seen, we use a hash map (`unordered_map<int, int>`) where the key is the value of `cnt` and the value is how many times this difference has occurred so far.
4. Every time we encounter a difference we've seen before, it indicates that there exists a subarray between the current position and a previous position where the frequency of `x` and `y` are the same.
5. The total number of such subarrays is accumulated into `maxCount`.

Complexity

Time Complexity:

- The solution involves a single traversal of the array, which takes `O(n)`, where `n` is the size of the array.
- Each operation on the unordered map (insertion and lookup) takes average `O(1)` time. Thus, the overall time complexity is **O(n)**.

Space Complexity:

- We use an unordered map to store the frequency of differences, which in the worst case could store up to `n + 1` entries. Hence, the space complexity is **O(n)**.

Code

```
class Solution {
  public:
    int sameOccurrence(vector<int>& arr, int x, int y) {
        int n = arr.size();
        int maxCount = 0;
        int cnt = 0;

        unordered_map<int,int> mpp;
        mpp[0] = 1;

        for(int i=0; i<n; i++){
            if(arr[i] == x) cnt++;
            if(arr[i] == y) cnt--;
            if(mpp.count(cnt)){
                maxCount += mpp[cnt];
            }
            mpp[cnt]++;
        }

        return maxCount;
    }
};
```