Maximum Sum Of Distinct Subarrays With Length K

solved by	Senan
Platform	LeetCode
difficulty	Medium
Serial	2461
tags	Hash Map
language	C++
solved on	@19/11/2024
link	https://leetcode.com/problems/maximum-sum-of-distinct-subarrays-with-length-k/description/
Completion	✓

Intuition

The problem asks us to find the maximum sum of any subarray of size `k` such that all elements in the subarray are distinct. To achieve this, we need to keep track of the sum of the current subarray and ensure that the subarray contains distinct elements.

A sliding window approach works efficiently here, as it allows us to dynamically maintain a window of size `k` while checking for the distinctness of elements.

Approach

1. Use a sliding window of size `k` to iterate over the array.
2. Maintain a hash map (`mp`) to store the frequency of elements in the current window. This helps us track the distinctness of elements.
3. Compute the initial sum and populate the hash map for the first `k` elements.
4. Check if the window contains exactly `k` distinct elements by comparing the size of the hash map to `k`. If so, update the maximum sum.
5. Slide the window by:
    - Adding the next element to the current sum.
    - Removing the leftmost element of the window from the sum and updating the hash map.
    - If the frequency of the removed element becomes zero, delete it from the hash map.
6. After sliding the window, repeat the check for distinctness and update the maximum sum if the condition is met.
7. Return the maximum sum.

Complexity

Time Complexity:

- **Initial window setup: O(k)** — Summing the first **k** elements and populating the hash map.
- **Sliding the window: O(n - k)** — For each of the remaining elements, the hash map operations (insert, delete, update) are **O(1)** on average.

- **Overall: O(n)**, where **n** is the size of the array.

Space Complexity:

- The hash map stores up to **k** elements at any time.
- **Space complexity: O(k)**.

Code

```
class Solution {
public:
    long long maximumSubarraySum(vector<int>& nums, int k) {
        unordered_map<int, int> mp;

        long long maxSum = 0;
        long long sum = 0;

        for (int i = 0; i < k; i++) {
            sum += nums[i];
            mp[nums[i]]++;
        }

        if (mp.size() == k) maxSum = max(maxSum, sum);

        for (int i = k; i < nums.size(); i++) {
            sum += nums[i];
            mp[nums[i]]++;

            sum -= nums[i - k];
            mp[nums[i - k]]--;
            if (mp[nums[i - k]] == 0) {
                mp.erase(nums[i - k]);
            }

            if (mp.size() == k) maxSum = max(maxSum, sum);
        }

        return maxSum;
    }
};
```