Shortest Subarray With Sum At Least K

solved by	Senan
Platform	LeetCode
difficulty	Hard
Serial	862
tags	Deque Sliding Window
language	C++
solved on	@17/11/2024
link	https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/description/
Completion	✓

Intuition

The problem involves finding the shortest subarray with a sum greater than or equal to `k`. Using a sliding window or a monotonic deque is efficient for this, as it allows us to dynamically maintain potential subarrays and their sums. By leveraging prefix sums, we can quickly compute the sum of any subarray and optimize our solution.

Approach

1. **Prefix Sum**:
    - Compute the prefix sum array where `prefix[i]` represents the sum of the first `i` elements of `nums`.
    - This helps in computing subarray sums efficiently as `sum(l, r) = prefix[r+1] - prefix[l]`.
2. **Deque for Monotonicity**:
    - Maintain a deque to store indices of the `prefix` array in increasing order of values.
    - This ensures that we can efficiently find a subarray with the smallest length that satisfies the sum condition.
3. **Processing with the Deque**:
    - For each index `i` in the `prefix` array:
        - Remove indices from the front of the deque if the sum of the subarray they represent is greater than or equal to `k`. Update the minimum length for such subarrays.
        - Remove indices from the back of the deque if their `prefix` values are greater than or equal to the current `prefix[i]`. This keeps the deque in a monotonic increasing order.
        - Push the current index `i` to the deque.
4. **Final Check**:
    - If no subarray satisfies the condition, return `1`. Otherwise, return the smallest length found.

Complexity

Time Complexity:

- Computing the prefix sum takes **O(n)**.
- Each index is added and removed from the deque at most once, which results in **O(n)** operations for the deque.
- Overall: **O(n)**.

Space Complexity:

- Prefix sum array takes **O(n)**.
- Deque can store at most **O(n)** indices.
- Overall: **O(n)**.

Code

```
class Solution {
public:
    int shortestSubarray(vector<int>& nums, int k) {
        int n = nums.size();

        vector<long long> prefix(n + 1, 0);
        for (int i = 0; i < n; i++) {
            prefix[i + 1] = prefix[i] + nums[i];
        }

        deque<int> q;
        int mini = INT_MAX;

        for (int i = 0; i <= n; i++) {
            while (!q.empty() && prefix[i] - prefix[q.front()] >= k) {
                mini = min(mini, i - q.front());
                q.pop_front();
            }

            while (!q.empty() && prefix[i] <= prefix[q.back()]) {
                q.pop_back();
            }

            q.push_back(i);
        }

        return (mini == INT_MAX) ? -1 : mini;
    }
};
```