Sheryians Coding School

Live Cohort

Day 91

Topic: Contiguous Array (Prefix Sum + Hash Map)

Definition:

A contiguous array problem involves finding the maximum length of a subarray where the number of 0s and 1s is equal. This is solved using the running sum technique, where we convert 0 to -1, and look for subarrays whose total sum is 0.

Code(Examples):

```
var findMaxLength = function(nums) {
    let map = new Map();
    map.set(0, -1); // sum 0 at index -1
    let maxLen = 0;
    let sum = 0;

    for (let i = 0; i < nums.length; i++) {
        sum += nums[i] === 0 ? -1 : 1;

        if (map.has(sum)) {
            maxLen = Math.max(maxLen, i - map.get(sum));
        } else {
            map.set(sum, i);
        }
    }

    return maxLen;
};
```

Use Case:

- Balancing binary data (0s and 1s) in problems related to parity, signal processing, or pattern detection.

- Can be applied in questions where equal distribution or canceling values are involved (like +1 and -1 patterns).

Interview Q&A:

Q: What is the core idea behind solving Contiguous Array with 0s and 1s?

A: Replace 0 with -1 and track the running sum. When the same sum repeats, the subarray between indices has equal 0s and 1s.

Q: Why do we initialize the map with map.set(0, -1)?

A: It accounts for the full subarray starting from index 0. Without it, we'd miss valid cases at the beginning.

Topic: Longest Consecutive Sequence (HashSet)

Definition:

Given an unsorted array of integers, find the length of the longest consecutive sequence (like [100, 101, 102, 103]). The key is using a Set for constant-time lookups and detecting sequence starts.

Code(Examples):

```
var longestConsecutive = function(nums) {
    let numSet = new Set(nums);
    let longest = 0;

    for (let num of numSet) {
        if (!numSet.has(num - 1)) { // start of sequence
            let currentNum = num;
            let streak = 1;

            while (numSet.has(currentNum + 1)) {
                currentNum++;
                streak++;
            }

            longest = Math.max(longest, streak);
        }
    }

    return longest;
};
```

Use Case:

- Useful when determining range-like properties in an unsorted dataset.
- Applied in leaderboards, calendar dates, sensor data, or streak counters.

Interview Q&A:

Q: Why do we check !numSet.has(num - 1)?

A: It ensures we only start counting once per sequence, avoiding duplicate checks and reducing time complexity.

Q: What is the time complexity of this algorithm?

A: O(n) — each number is visited only once due to the Set.

Q: Can this be done with sorting instead of a Set?

A: Yes, but sorting would make the time complexity O(n log n), which is less optimal than the O(n) Set approach.

Count Distinct Elements in Every Window (Sliding Window + Hash Map)

Definition:

Given an array and a window size k, determine the number of distinct elements in every sliding window of size k. This is a classic application of a sliding window with frequency map.

Code(Examples):

```
function countDistinctInWindow(arr, k) {
    const result = [];
    const freqMap = new Map();

    // Initial window
    for (let i = 0; i < k; i++) {
        freqMap.set(arr[i], (freqMap.get(arr[i]) || 0) + 1);
    }
    result.push(freqMap.size);

    // Slide the window
    for (let i = k; i < arr.length; i++) {
        let outElem = arr[i - k];
        if (freqMap.get(outElem) === 1) {
            freqMap.delete(outElem);
        } else {
            freqMap.set(outElem, freqMap.get(outElem) - 1);
        }
        freqMap.set(arr[i], (freqMap.get(arr[i]) || 0) + 1);
        result.push(freqMap.size);
    }

    return result;
}
```

Use Case:

- Analytics: track how many unique visitors/ products appear in a time-based window.
- Streaming platforms: measure diversity of data in limited buffer size.
- Efficient in real-time monitoring with fixed memory.

Interview Q&A:

Q: What's the purpose of the frequency map?

A: To keep count of each element inside the current window for constant-time update when sliding.

Q: How do we update the map when the window moves?

A: We decrement frequency of the element that exits and increment the one that enters the window.

Q: What is the time complexity?

A: O(n), because each element is added and removed at most once from the map.