Find Number of Subarrays that Fit an Exact Criteria — Coding Interview Notes (Light Theme)

General Pattern Template

```
from collections import defaultdict

def fn(arr, k):
    counts = defaultdict(int)
    counts[0] = 1
    ans = curr = 0

    for num in arr:
        # do logic to change curr
        ans += counts[curr - k]
        counts[curr] += 1

    return ans
```

Concept:
This pattern counts the number of subarrays that meet an **exact criterion** (e.g., sum equals *k* or XOR equals *k*) by maintaining a running cumulative state and a hash map of prefix counts.

At each step, we check how many previous prefixes lead to a valid subarray ending at the current index.
Time Complexity: O(n) **Space Complexity:** O(n)

Key Ideas

1. Use a cumulative value (sum, xor, etc.) to represent the state up to each index.
2. Maintain a dictionary mapping from cumulative value → frequency.
3. To find subarrays meeting a criterion (e.g., curr - k), look up previous cumulative values.
4. Add the current cumulative value to the dictionary after counting.
5. This pattern generalizes range-sum and prefix-based logic.

Example 1: Count Subarrays with Sum = k

Goal: Count contiguous subarrays where the sum equals *k*.
Approach: Maintain a running prefix sum and use hashmap to count how many previous prefixes satisfy sum[j] - sum[i] = k.

```
from collections import defaultdict

def subarray_sum(nums, k):
```

```
    counts = defaultdict(int)
    counts[0] = 1
    ans = curr = 0

    for num in nums:
        curr += num
        ans += counts[curr - k]
        counts[curr] += 1

    return ans

# Example
print(subarray_sum([1, 1, 1], 2))  # Output: 2
```

Example 2: Count Subarrays with XOR = k

Goal: Count subarrays whose bitwise XOR equals a given value *k*.
Approach: Replace sum logic with XOR accumulation and use the same counting principle.

```
from collections import defaultdict

def subarray_xor(nums, k):
    counts = defaultdict(int)
    counts[0] = 1
    ans = curr = 0

    for num in nums:
        curr ^= num
        ans += counts[curr ^ k]
        counts[curr] += 1

    return ans

# Example
print(subarray_xor([4, 2, 2, 6, 4], 6))  # Output: 4
```

Example 3: Count Subarrays with Sum = k in Binary Array

Goal: Given a binary array, count the number of subarrays that have exactly *k* ones.
Approach: Use prefix sum for running count of ones; check prefix_sum - k in dictionary.

```
from collections import defaultdict

def count_subarrays_with_k_ones(nums, k):
    counts = defaultdict(int)
    counts[0] = 1
    ans = curr = 0

    for num in nums:
        curr += num  # count ones
        ans += counts[curr - k]
```

```
            counts[curr] += 1

        return ans

# Example
print(count_subarrays_with_k_ones([1, 0, 1, 0, 1], 2))  # Output: 4
```

Summary Table

ProblemConditionStateComplexity Subarray sum = ksum[j] - sum[i] = kPrefix SumO(n) Subarray XOR = kxor[j] ^ xor[i] = kPrefix XORO(n) Binary array k onesprefix_count - kPrefix SumO(n)