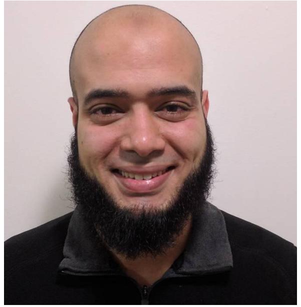
Cracking Coding Interviews

Subarray Sum Equals K

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



Leetcode 560 - Subarray Sum Equals K

- Given an array of integers and an integer k, return the total number of continuous subarrays where sum equals to k.
- Input ⇒ Output
    - [1, 1, 1], k = 2 ⇒ 2 (1+1), (1+1)
    - [1, 2, 3], k = 3 ⇒ 2 (1+2), (3)
- Signature
    - C++: int subarraySum(vector<int>& nums, int k)
    - Python: def subarraySum(nums: List[int], k: int) -> int:
    - Java: public int subarraySum(int[] nums, int k)

Your turn

- Ask the right questions, if any, or state your assumptions
- Develop some test cases

Questions & test cases

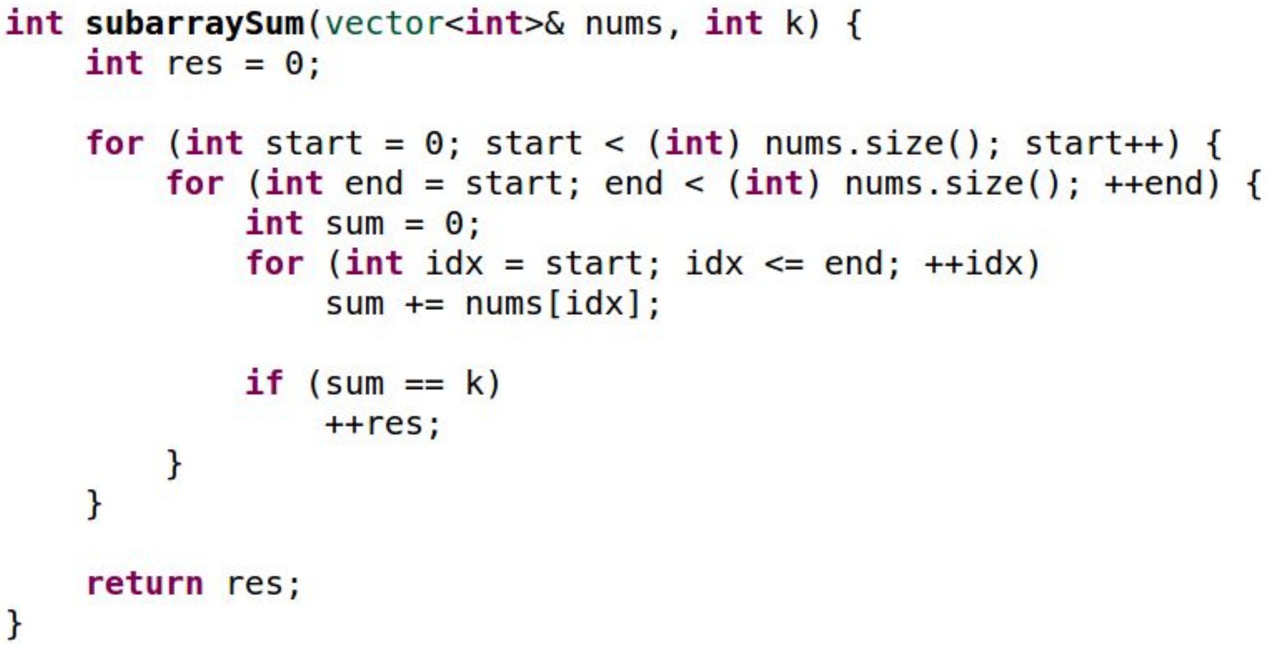
- The problem seems clear
- The test cases provided are a little too simplified.
- Let's develop a more substantial test case.
  - {9, -9, 2, 3, 4}, k = 9 ⇒ 3
    - (9), (2, 3, 4), (9, -9, 2, 3, 4)
    - One solution is simply the value of an element in the array; in this case, the first element (9).
    - Another solution equates to the sum of the entire array.

Your turn

- Can we approach it with brute-force? If so, how?

Brute-force it!

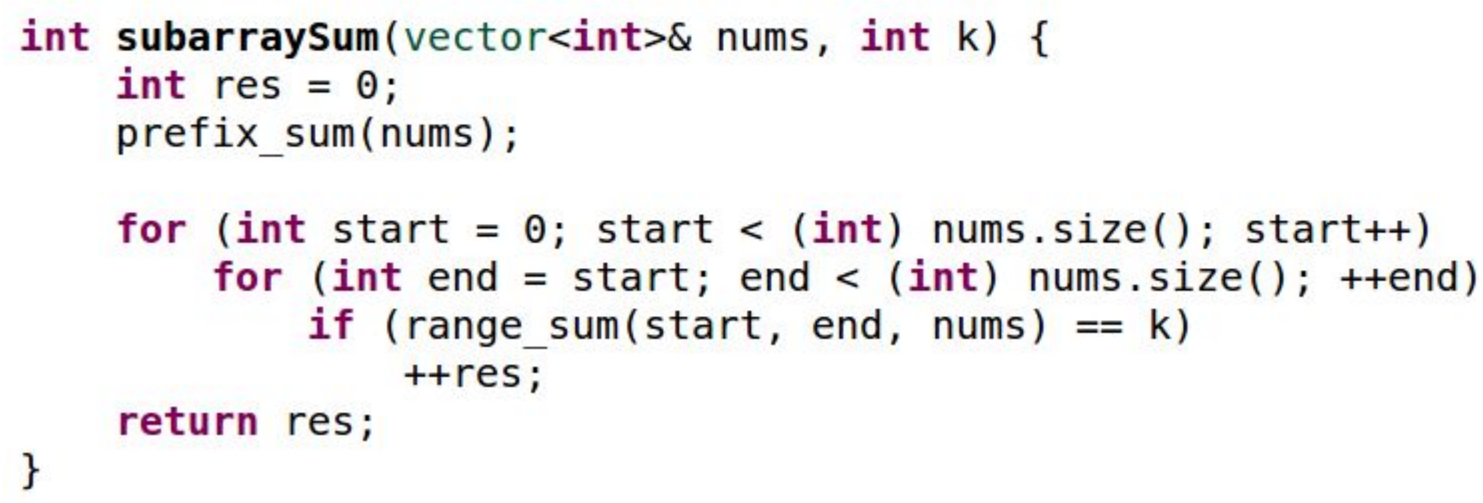
- Since this is a range-based problem, we can try all ranges, sum the range values and compare to K!
- O(n^3)
- Optimize?



```
int subarraySum(vector<int>& nums, int k) {
    int res = 0;

    for (int start = 0; start < (int) nums.size(); start++) {
        for (int end = start; end < (int) nums.size(); ++end) {
            int sum = 0;
            for (int idx = start; idx <= end; ++idx)
                sum += nums[idx];

            if (sum == k)
                ++res;
        }
    }

    return res;
}
```

Brute-force it: Optimizations

- The 3rd loop is just summing a range. Prefix-sum is a direct application
    - We can even do the accumulation inside the 2nd loop in an easy way
- Now O(n^2). Still slow. Can you do it in O(n)?



```
int subarraySum(vector<int>& nums, int k) {
    int res = 0;
    prefix_sum(nums);

    for (int start = 0; start < (int) nums.size(); start++)
        for (int end = start; end < (int) nums.size(); ++end)
            if (range_sum(start, end, nums) == k)
                ++res;
    return res;
}
```

Your turn

- Try to discover some observations / properties

Your turn

- Try to discover some observations / properties
- Let's say nothing obvious comes to mind
- Really stuck? Let’s ask the interviewer for a hint!
- Interviewer:
  - Assume at the ith index, we have the current prefix sum S
  - How can we evaluate the **number of valid ranges** that end at position i with the target sum T?
- Your turn
  - Always make use of the hints
  - Try to pick an example and trace considering the hint
  - If the hint is not clear, feel free to ask for more clarification

Let’s take an example

- We want to use the hint
- Let’s develop a case where we have a prefix S
    - {**0, 1, 2, 3, 4, 5, 6, 7**, 8}, k = 18
    - At position 7, our current prefix sum is: 28
    - But we target only 18
    - How is that useful?

Let’s take an example

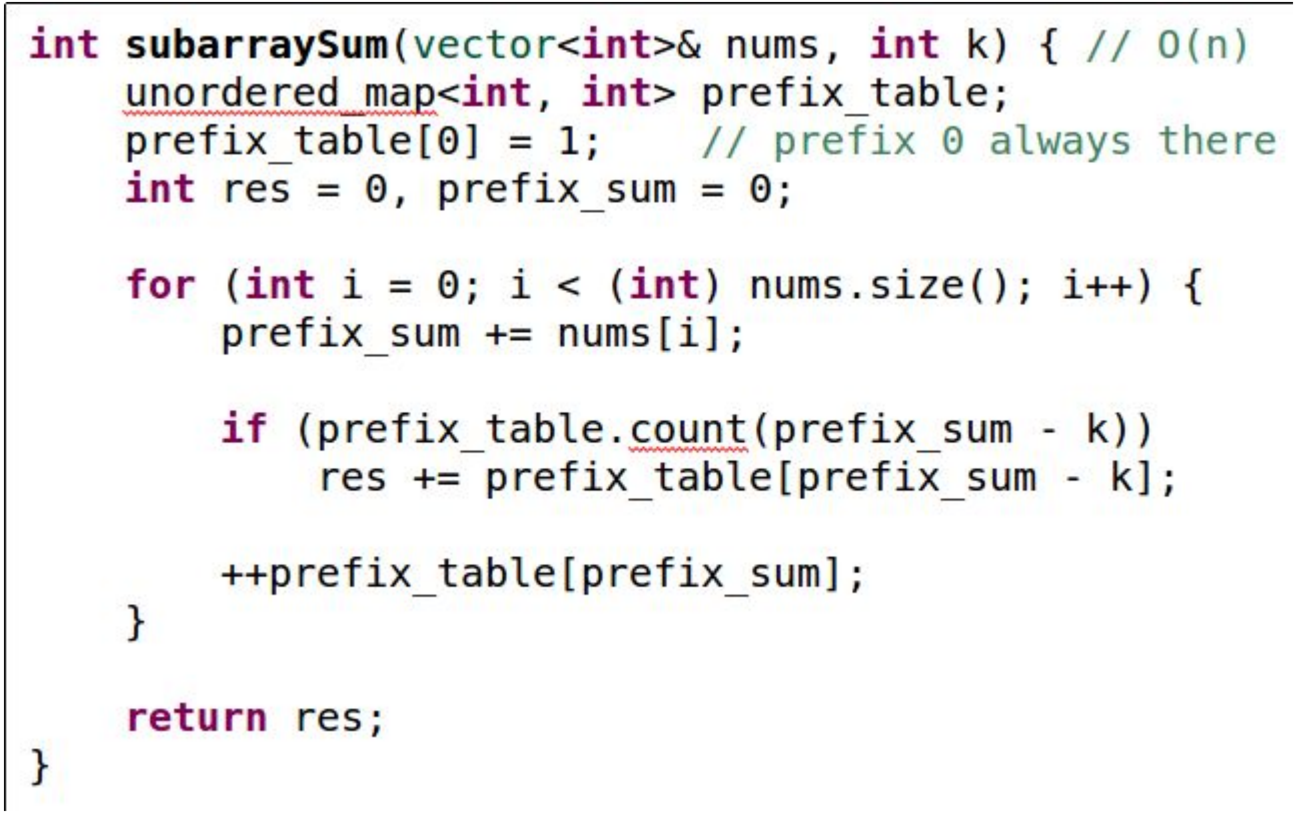
- We want to use the hint
- Let’s develop a case where we have a prefix S
  - {**0, 1, 2, 3, 4, 5, 6, 7**, 8}, k = 18
  - At position 7, our current prefix sum is: 28
  - But we target only 18
  - How is that useful?
  - We need to remove a prefix of 10
  - Do we have an earlier prefix of 10? If so, we have a window
  - Yes: {0, 1, 2, 3, 4}
  - Then the remaining: {5, 6, 7} is a correct window
- So, if there is an earlier prefix of S-T, we have a valid window
  - We can have many. So we need to know how many previous prefixes have value V

Approach

- We iterate on the array, accumulate the prefix
- With each prefix, we need to mark that we found this prefix
- Actually, we need to know how many times this value appeared
- The value can be large or -ve
- The best handling makes use of a hash table!

Approach

- In each step, use your table carefully.
- We're looking for all instances where we have recorded a prefix sum **equal to the target value K** in our hash table.



```
int subarraySum(vector<int>& nums, int k) { // O(n)
    unordered_map<int, int> prefix_table;
    prefix_table[0] = 1;    // prefix 0 always there
    int res = 0, prefix_sum = 0;

    for (int i = 0; i < (int) nums.size(); i++) {
        prefix_sum += nums[i];

        if (prefix_table.count(prefix_sum - k))
            res += prefix_table[prefix_sum - k];

        ++prefix_table[prefix_sum];
    }

    return res;
}
```

Tip

- Sometimes, we need to store some information about the current prefix while processing
  - For example, the prefix sum itself, number of even numbers so far, etc...
  - A hash table is the most viable form of storage
- Then we try to find all ranges **ending** at the current position
  - By searching the hash table for a specific value
- The nature of the problem should allow **range cancellation** to use this style
  - E.g. something that is countable (summation), product, etc

Leetcode 525 - Contiguous Array

- Given a binary array, return the maximum length of a contiguous subarray with an **equal number** of 0 and 1.
  - [0,1,0] ⇒ 2
  - [0,1,0,1] ⇒ 4
- This problem asks about the maximum NOT number of subarrays
- Regardless of that, how can you **convert** it to the previous problem?

Leetcode 525 - Contiguous Array

- Given a binary array, return the maximum length of a contiguous subarray with an **equal number** of 0 and 1.
  - [0,1,0] ⇒ 2
  - [0,1,0,1] ⇒ 4
- This problem asks about the maximum NOT number of subarrays
- Regardless of that, how can you convert it to the previous problem?
- If you replace each zero with -1, now the problem is the maximum length of a contiguous subarray with zero-sum!
  - Almost the same code now!
  - For the maximum (not count): keep the **earliest** index for a specific prefix sum

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."