

LeetCode #560

Subarray Sum Equals K

Prefix sum stored in a hash table

The first idea to solve this problem is a brute force approach. To achieve this we need to check every subarray in the input array using a loop and an inner loop that starts from $i + 1$ index. However, considering that the input array might have a size of up to $2 * 10^4$. This gives us Time Limit Exceeded (TLE) as the result since the total number of operations exceeds 10^8 . The next idea is to use a sliding window approach. But since the values in the array might be negative, it could lead to incorrect results. This is because sliding windows don't guarantee that the sum will always increase or decrease consistently, which makes them unsuitable for this problem. However, we can optimize the brute force solution and reduce the time complexity from $O(n^2)$ to $O(n)$ by using a hash map. The hash map will store the prefix sums as keys and the count of occurrences of each prefix sum as values. We start by adding an empty prefix to the hash map. This is important because the empty prefix has a sum of zero and if the sum of some of the first elements equals k , we would miss this subarray if we don't handle this base case. Then we iterate through the input array and accumulate the sum of each element to a local variable. On each iteration we check whether $\text{sum} - k$ exists in the hash map. If it does, we add the value associated with $\text{sum} - k$ to the result, since it indicates how many times a subarray sum has occurred up to that point. Also, on each iteration we increment the count of the current sum in the hash map. After the iteration, we return the result.

nums

| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|

map_sum

| key | value |
|-----|-------|
| 0 | 1 |
| | |
| | |
| | |
| | |

k = 3

prefix = 0

res = 0

nums

| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|

map_sum

k = 3

| key | value |
|-----|-------|
| 0 | 1 |
| 1 | 1 |
| | |
| | |
| | |


prefix += nums[i] = 1

res += map_sum[prefix - k] if prefix - k in map_sum else 0

res += map_sum[-2] if -2 in map_sum else 0 = 0

map_sum[prefix] += 1 = map_sum[1] = 1

nums



| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|

map_sum

k = 3

| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 1 |
| | |
| | |
| | |

prefix += nums[i] = 0

res += map_sum[prefix - k] if prefix - k in map_sum else 0

res += map_sum[-3] if -3 in map_sum else 0 = 0

map_sum[prefix] += 1 = map_sum[0] = 2

nums

| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|

map_sum

k = 3

| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 2 |
| | |
| | |
| | |

prefix += nums[i] = 1

res += map_sum[prefix - k] if prefix - k in map_sum else 0

res += map_sum[-2] if -2 in map_sum else 0 = 0

map_sum[prefix] += 1 = map_sum[1] = 2

nums

| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|

map_sum

k = 3

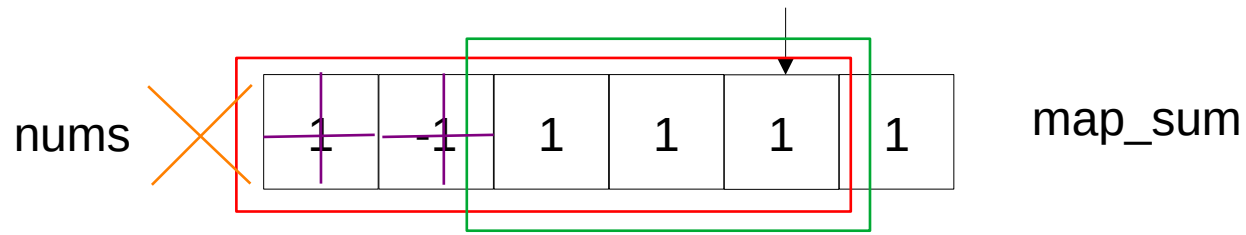
| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 2 |
| 2 | 1 |
| | |
| | |

prefix += nums[i] = 2

res += map_sum[prefix - k] if prefix - k in map_sum else 0

res += map_sum[-1] if -1 in map_sum else 0 = 0

map_sum[prefix] += 1 = map_sum[2] = 1



$k = 3$

Since we already have two prefixes with a sum 0 – the empty prefix (orange cross) and the prefix from 0 to 1 (purple cross), we add 2 to the result. This works because there are 2 subarrays (red and green frames) with a sum equal to k .

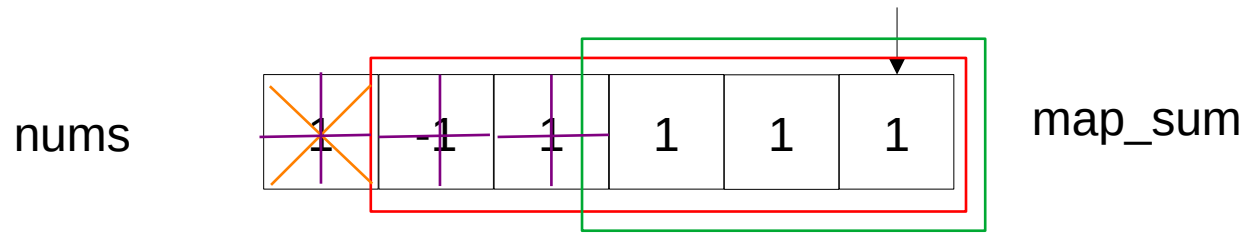
| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| | |

$\text{prefix} += \text{nums}[i] = 3$

$\text{res} += \text{map_sum}[\text{prefix} - k]$ if $\text{prefix} - k$ in map_sum else 0

$\text{res} += \text{map_sum}[0]$ if 0 in map_sum else 0 = 2

$\text{map_sum}[\text{prefix}] += 1 = \text{map_sum}[3] = 1$



k = 3

Since we already have two prefixes with a sum 1 – the prefix at index 0 (orange cross) and the prefix from 0 to 2 (purple cross), we add 2 to the result. This works because there are 2 subarrays (red and green frames) with a sum equal to k.

| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |

prefix += nums[i] = 4

res += map_sum[prefix - k] if prefix - k in map_sum else 0

res += map_sum[1] if 0 in map_sum else 0 = 4

map_sum[prefix] += 1 = map_sum[4] = 1

nums

| | | | | | |
|---|----|---|---|---|---|
| 1 | -1 | 1 | 1 | 1 | 1 |
|---|----|---|---|---|---|



map_sum

k = 3

| key | value |
|-----|-------|
| 0 | 2 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |

since the result is 4, we return 4