560. Subarray Sum Equals K

Medium 13312 430 Add to List Share

Given an array of integers `nums` and an integer `k`, return *the total number of subarrays whose sum equals to* `k`.

A subarray is a contiguous **non-empty** sequence of elements within an array.

Example 1:

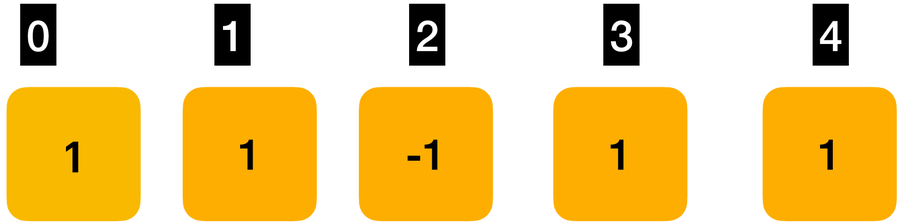
```
Input: nums = [1,1,1], k = 2
Output: 2
```

Example 2:

```
Input: nums = [1,2,3], k = 3
Output: 2
```

Constraints:

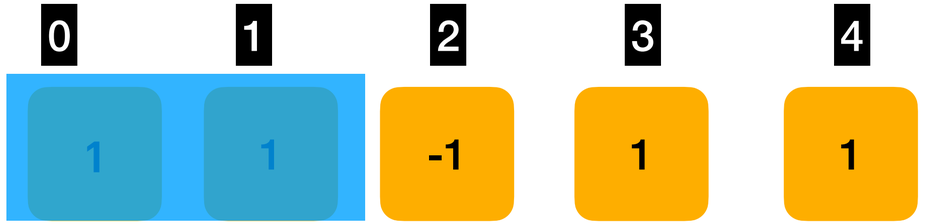
- $1 <= nums.length <= 2 * 10^4$
- $-1000 <= nums[i] <= 1000$
- $-10^7 <= k <= 10^7$



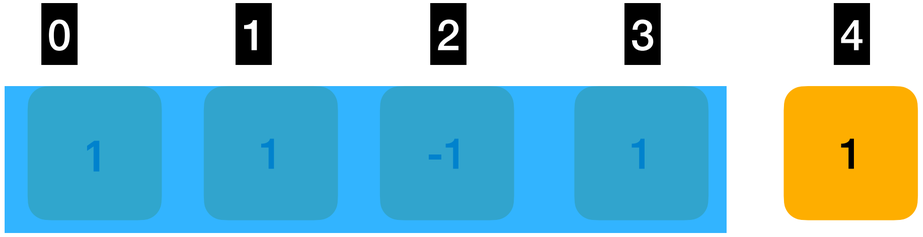
Return the count of
How many continuous subarrays can be made.

K=2

Return the count of
How many continuous subarrays can be made.

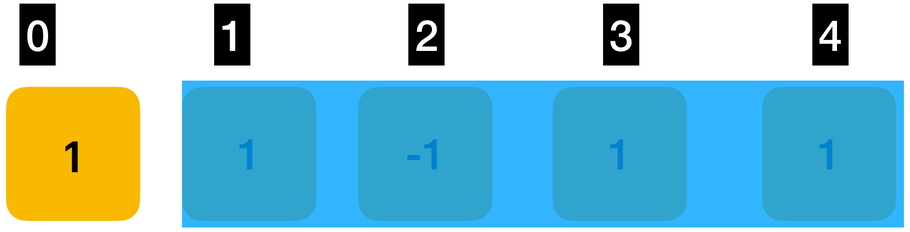


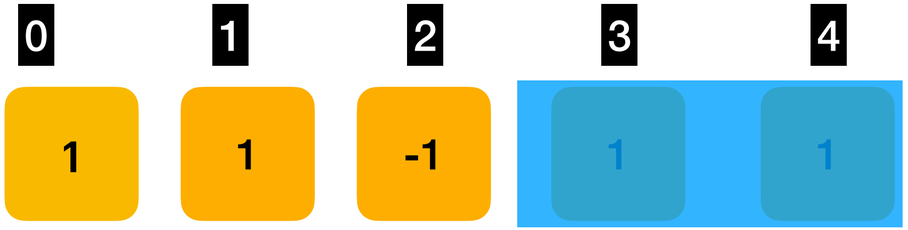
Output : 4



Algorithm :
For each move maintain the prefixSum in Map:
When sum-k presents in Map, you reach the target.
Update the Counter accordingly

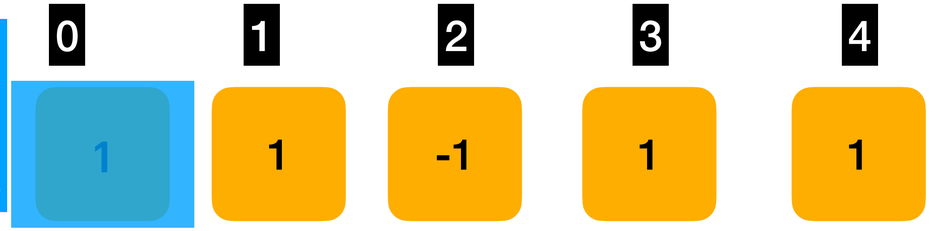
Time Complexity : O(n)
Space Complexity : O(n)





Counter = 0

Sum: 1
sum-k: 1-2 = -1
-1 does not exist in map
Upsert sum counter in map



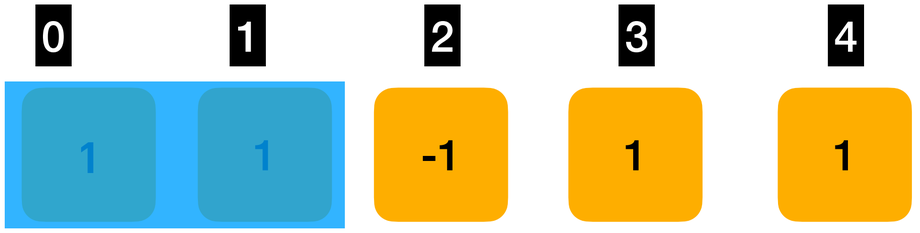
K=2

Return the count of
How many continuous subarrays can be made.

Map :

prefixSum. ——> Counter
0 —-> 1
1 —-> 1

Sum: 2
sum-k: 2-2 = 0
0 exist in map.
Update the counter
Counter = counter+map.get(0)
= 0 + 1 = 1
Counter = 1
Upsert sum counter in map

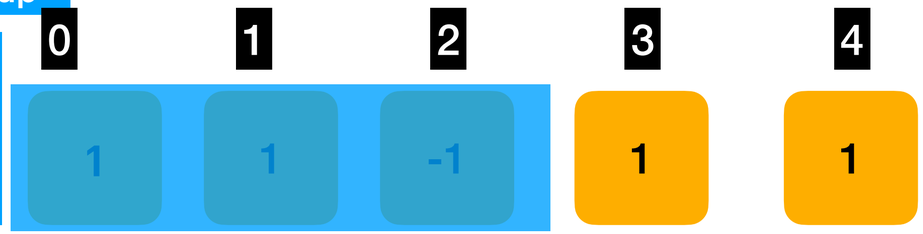


K=2

Map :

prefixSum. ——> Counter
0 —-> 1
1 —-> 1
2—> 1

Sum: 1
sum-k: 1-2 = -1
-1 does not exist in map
Upsert sum counter in map

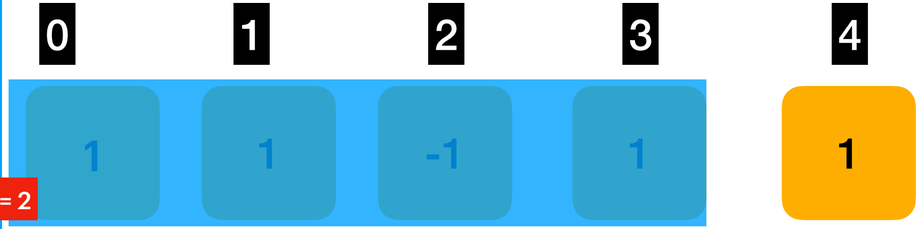


K=2

Map :

prefixSum. ——> Counter
0 —-> 1
1 —-> 2
2 —> 1

Sum: 2
sum-k: 2-2 = 0
0 existi in map
Counter = 1+map.get(0)
= 1 + 1 = 2
Counter = 2
Upsert sum counter in map

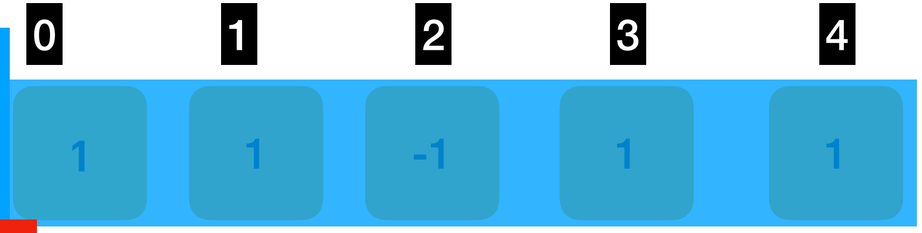


K=2

Map :

prefixSum. ——> Counter
0 —-> 1
1 —-> 2
2 —> 2

Sum: 3
sum-k: 3-2 = 1
1 existi in map
Counter = 2+map.get(1)
= 2 + 2= 4
Counter = 4
Upsert sum counter in map



Map :

prefixSum. ——> Counter
0 —-> 1
1 —-> 2
2 —> 2
3 —> 1

49. Group Anagrams

Medium 9522 318 Add to List Share

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

Constraints:

- `1 <= strs.length <= 10^4`
- `0 <= strs[i].length <= 100`
- `strs[i]` consists of lowercase English letters.

Input: strs = ["eat","tea","tan","ate","nat","bat"]

After sort each anagram returns the same key.

Anagrams —>
[
[“eat”, “tea”, “ate”],
[“nat”, “tan”],
[“bat”]
]

key: aet —-> value:[“eat”, “tea”, “ate”],

Key: ant -> value: [“nat”, “tan”],

Key: abt -> value: [“bat”],

Map<String, List<String> >

Algorithm : O(n*klogk)

1. Iterate string arr.
2. Sort each String : O(klogk)
3. Upserts key:[sortedString] value:[currentIteratedString] in the map

Time Complexity: O(n*klogk)
Space Complexity : O(n*k) ~ O(n)