Subarray Sum Equals K

Given an array of integers and an integer **k**, you need to find the total number of continuous subarrays whose sum equals to **k**.

Example 1:

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

Note:

1. The length of the array is in range [1, 20,000].
2. The range of numbers in the array is [-1000, 1000] and the range of the integer **k** is [-1e7, 1e7].

Solution 1

Solution 1. Brute force. We just need two loops (i, j) and test if `SUM[i, j]` = k. Time complexity O(n^2), Space complexity O(1). I bet this solution will TLE.

Solution 2. From solution 1, we know the key to solve this problem is `SUM[i, j]`. So if we know `SUM[0, i - 1]` and `SUM[0, j]`, then we can easily get `SUM[i, j]`. To achieve this, we just need to go through the array, calculate the current sum and save number of all seen `PreSum` to a HashMap. Time complexity O(n), Space complexity O(n).

```
public class Solution {
    public int subarraySum(int[] nums, int k) {
        int sum = 0, result = 0;
        Map<Integer, Integer> preSum = new HashMap<>();
        preSum.put(0, 1);

        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            if (preSum.containsKey(sum - k)) {
                result += preSum.get(sum - k);
            }
            preSum.put(sum, preSum.getOrDefault(sum, 0) + 1);
        }

        return result;
    }
}
```

written by shawngao original link here

Solution 2

Let's remember count[V], the number of previous prefix sums with value V. If our newest prefix sum has value W, and W-V == K, then we add count[V] to our answer.

This is because at time t, `A[0] + A[1] + ... + A[t-1] = W`, and there are `count[V]` indices `j` with `j < t-1` and `A[0] + A[1] + ... + A[j] = V`. Thus, there are `count[V]` subarrays `A[j+1] + A[j+2] + ... + A[t-1] = K`.

```
def subarraySum(self, A, K):
    count = collections.Counter()
    count[0] = 1
    ans = su = 0
    for x in A:
        su += x
        ans += count[su-K]
        count[su] += 1
    return ans
```

written by awice original link here

Solution 3

basically use a hashmap to store how many subarrays that can sum up to a number.

```
public int subarraySum(int[] a, int k) {
        int sum = 0;
  HashMap<Integer, Integer> map = new HashMap<>();
  map.put(0, 1);
  int count = 0;
  for (int i = 0; i < a.length; i++) {
   sum += a[i];
   if (map.containsKey(sum - k)) {
    count += map.get(sum-k);
   }
   if (!map.containsKey(sum)) {
    map.put(sum, 1);
   } else {
       map.put(sum, map.get(sum) + 1);
   }
  }
  return count;
    }
```

written by xmztzt original link here