0

Subarray Sums Divisible by K

JAVA | 2 approaches | Fully explained ✅

sourin_bruh 2068 Sep 21, 2022

Please Upvote :D

1. Brute force approach (TLE):

Java

```java
class Solution {
    public int subarraysDivByK(int[] nums, int k) {
        // check all possible subarrays,
        // take their sum, increment the count if divisible by k
        int total = 0;
        for (int i = 0; i < nums.length; i++) {
            int sum = 0;
            for (int j = i; j < nums.length; j++) {
                sum += nums[j];
                if (sum % k == 0) {
                    total++;
                }
            }
        }

        return total;
    }
}

// O(n ^ 2), SC: O(1)
```

2. Optimal approach (using hashmap):

We will be using a hashmap to store **remainders** of the running sum of our array at each index.

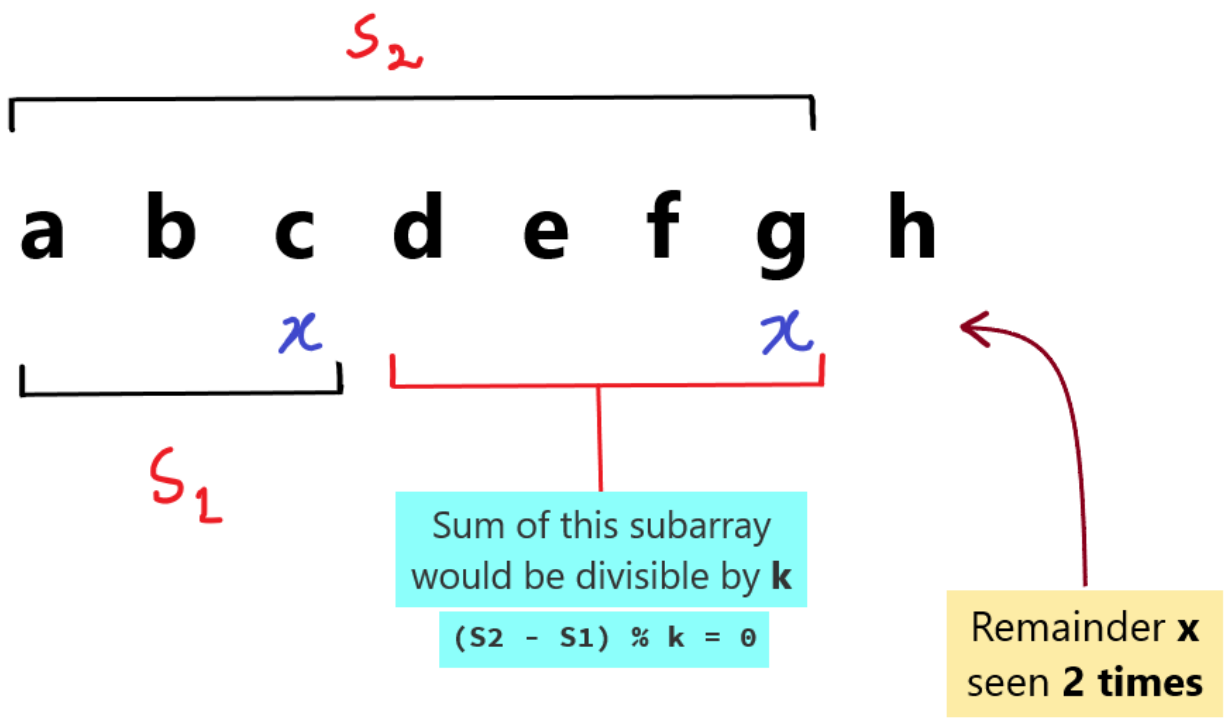
The logic behind this is that if we encounter a remainder which was already encountered before, it means that the *sum of the subarray from from the index right next to that point to our current point is divisible by*

25

Say at index `i` , we got a remainder `x` , after that we got the same remainder `x` at index `j` .
It means that the sum of the subarray from index `i + 1` to `j` is divisble by `k` (or we can say that the sum of the subarray from index `i + 1` to `j` yields a remainder `0` when divided by `k`).

Say sum till index `i` is `S1` and sum till index `j` is `S2` .
We observed that:

```
 Remainder at index i = Remainder at index j
     S2 % k = S1 % k
  => S2 % k - S1 % k = 0
  => (S2 - S1) % k = 0
  => (Sum of subarray from i + 1 to j) % k = 0
```


S_2
a b c d e f g h
x
x
S_1
Sum of this subarray would be divisible by **k**
(S2 - S1) % k = 0
Remainder **x** seen **2 times**

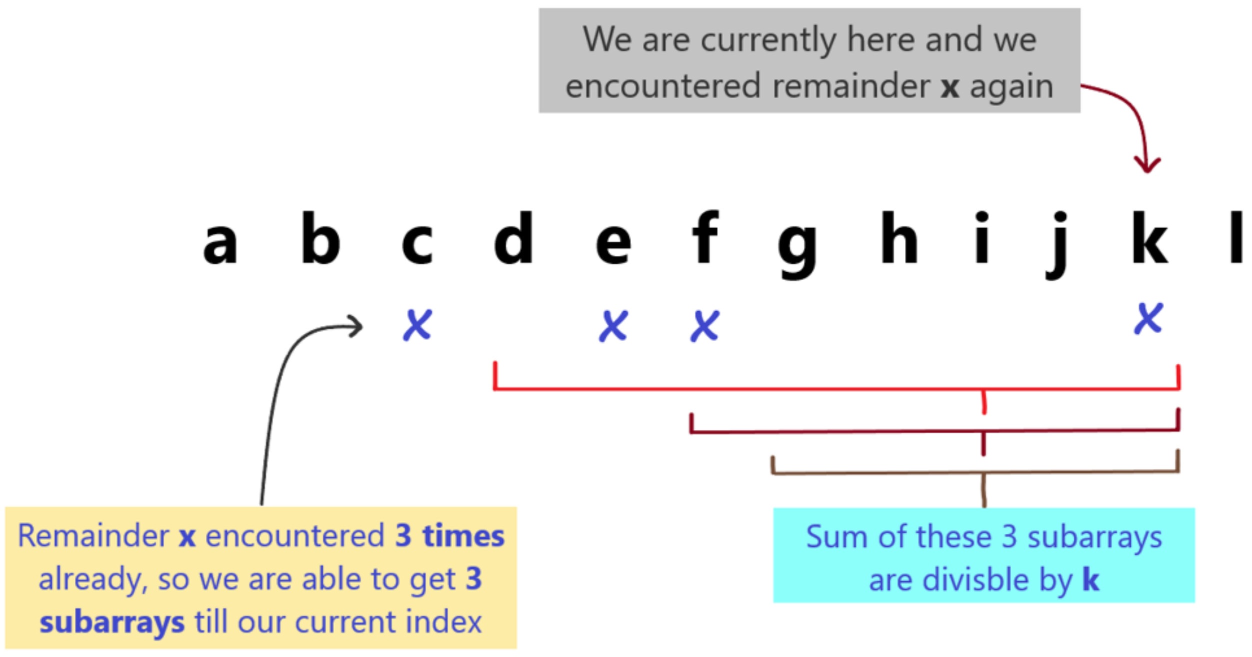
Say the frequency of a remainder `x` is `3` .
It means that **at `3` points, remainder `3` was encountered**.

So if we draw a subarray from the next index of all those `3` points to our current index, we will get a sum divisible by `k` , so we will add that `3` to our answer.

At the end we will increment `3` to `4` (the count of `x`) because our current index is one more point where we have encountered `x` .

So next time we enounter `x` again, we will be able to draw `4` subarrays from `4` points and add `4` to our answer.

25


We are currently here and we encountered remainder **x** again
a b c d e f g h i j k l
x x x x
Remainder **x** encountered **3 times** already, so we are able to get **3 subarrays** till our current index
Sum of these 3 subarrays are divisble by **k**

Code:

Java

```java
class Solution {
    public int subarraysDivByK(int[] nums, int k) {
        // map to store the remainders and number of times they've been encounte
        Map<Integer, Integer> map = new HashMap<>();
        // our sum is initially 0, and 0 is also divisible by k
        // there would be a case when remainder would actually be zero
        // so the array from the beginning to that index is our candidate subarı
        // so to address that case, so we put <0, 1> initially
        map.put(0, 1);
        int runningSum = 0, ans = 0;
        for (int n : nums) {
            runningSum += n;            // add the element to the running sum
            int rem = runningSum % k;   // get the remainder by k of our running
            if (rem < 0) {              // in case remainder < 0,
                rem += k;               // add divisor (k) to make it +ve
            }
            // if we already encountered the remainder, we add the frequency map
            // that frequency is nothing but the number of subarrays whose sum h
            ans += map.getOrDefault(rem, 0);
            // after that, we increment the frequency of that remainder,
            // because we have encountered it again so the number of subarrays i
            map.put(rem , 1 + map.getOrDefault(rem, 0));
        }

        return ans;     // return the answer
    }
}
```

25

```
// TC: O(n), SC: O(n)
```

Clean solution:

```
Java

class Solution {
    public int subarraysDivByK(int[] nums, int k) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, 1);
        int runningSum = 0, ans = 0;
        for (int n : nums) {
            runningSum += n;
            int rem = runningSum % k;
            if (rem < 0) {
                rem += k;
            }
            ans += map.getOrDefault(rem, 0);
            map.put(rem , 1 + map.getOrDefault(rem, 0));
        }

        return ans;
    }
}
```

25 Share Favorite

Previous ✅✅ C++ || Easy Solution || 💯 💯 Pref...

Next Day 19 || Explanation with Diagram - ...

Comments (1) Sort by: Best

Type comment here... (Markdown supported)

Preview Comment

racemus Jan 20, 2023

Thanks. It makes me understand why it works.

0 Show 1 Replies Reply

25

1

25