★★★★★ Average Rating: 4.90 (272 votes) Premium

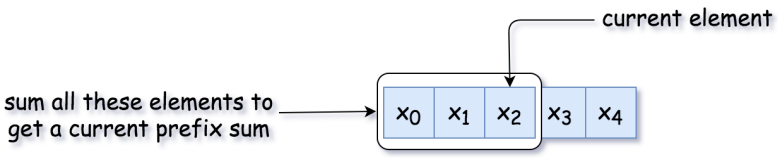
Solution

Prefix Sum: What is it

In this article, we're going to discuss simple but powerful prefix sum technique: one pass + linear time complexity.

> *Prefix sum* is a sum of the current value with all previous elements starting from the beginning of the structure.

It could be defined for 1D arrays (sum the current value with all the previous integers),



input array	x_0	x_1	x_2	x_3	x_4
prefix sum	x_0	$x_0 + x_1$	$x_0 + x_1 + x_2$	$x_0 + x_1 + x_2 + x_3$	$x_0 + x_1 + x_2 + x_3 + x_4$

Figure 1. Prefix sum for 1D array.

for 2D arrays (sum of the current value with the integers above or on the left)

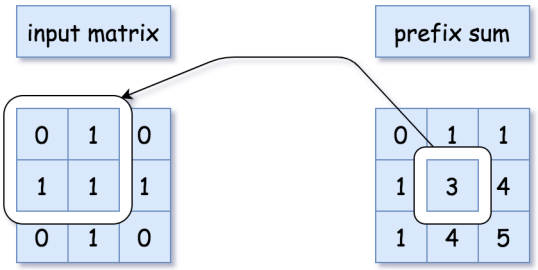


Figure 2. Prefix sum for 2D array.

or for the binary trees (sum the values of the current node and all parent nodes),

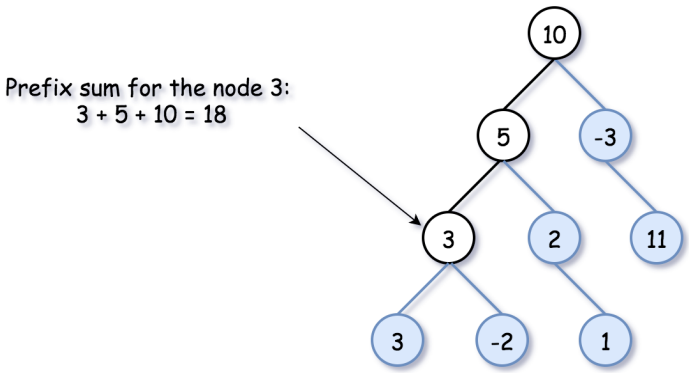


Figure 3. Prefix sum for the binary tree.

Prefix Sum: How to Use: Number of Continuous Subarrays that Sum to Target

You might want to use the prefix sum technique for the problems like "Find a number of *continuous* subarrays/submatrices/tree paths that sum to target".

Before going to the current problem with the tree, let's check the idea on a simpler example Find a number of continuous subarrays

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    // use prefix sum with hashmap
    HashMap<Long, Integer> map;
    int count;
    long prefixSum;

    public int pathSum(TreeNode root, int sum) {
        if (root == null) return 0;
        this.map = new HashMap<Long, Integer>();
        this.count = 0;
        this.prefixSum = 0;
        // init for subtree starting from root
        map.put((long) 0, 1);
        calPrefixSum(root, sum);
        return count;
    }

    private void calPrefixSum(TreeNode root, int sum) {
        if (root == null) return;
        prefixSum += root.val;
        count += map.get(prefixSum - sum) == null? 0 : map.get(prefixSum-sum);
        map.put(prefixSum, map.getOrDefault(prefixSum, 0)+1);


        calPrefixSum(root.left, sum);

        calPrefixSum(root.right, sum);

        map.put(prefixSum, map.getOrDefault(prefixSum, 0)-1);
        prefixSum -= root.val;
    }
}
```

Your previous code was restored from your local stor

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [1000000000,1000000000,nu 0

Output 0 Diff

Expected 0

Quick Navigation

Prefix Sum: How to Use: Number of Continuous Subarrays that Sum to Target

You might want to use the prefix sum technique for the problems like "Find a number of *continuous* subarrays/submatrices/tree paths that sum to target".

Before going to the current problem with the tree, let's check the idea on a simpler example Find a number of continuous subarrays that sum to target.

- Use a variable to track the current prefix sum and a hashmap "prefix sum -> how many times was it seen so far".

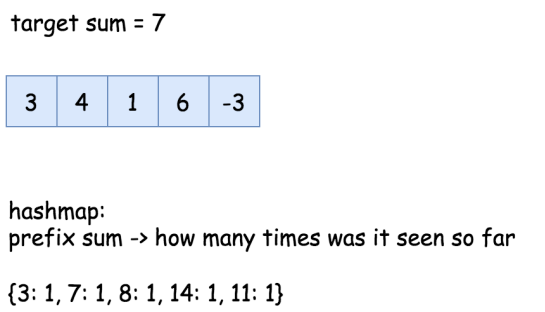


Figure 4. Find a number of continuous subarrays that sum to target.

- Parse the input structure and count the requested subarrays/submatrices/tree paths along the way with the help of that hashmap. How to count?

There could be two situations. In situation 1, the subarray with the target sum starts from the beginning of the array. That means that the current prefix sum is equal to the target sum, and we increase the counter by 1.

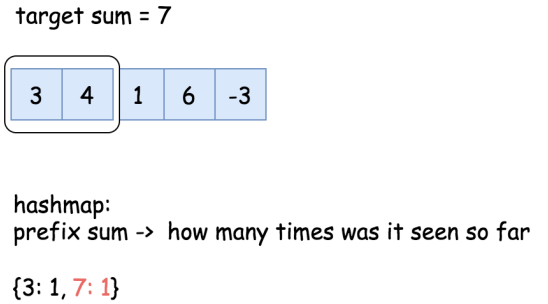


Figure 5. Situation 1: The subarray starts from the beginning of the array.

In situation 2, the subarray with the target sum starts somewhere in the middle. That means we should add to the counter the number of times we have seen the prefix sum `curr_sum - target` so far: `count += h[curr_sum - target]`.

The logic is simple: the current prefix sum is `curr_sum`, and some elements before the prefix sum was `curr_sum - target`. All the elements in between sum up to `curr_sum - (curr_sum - target) = target`.

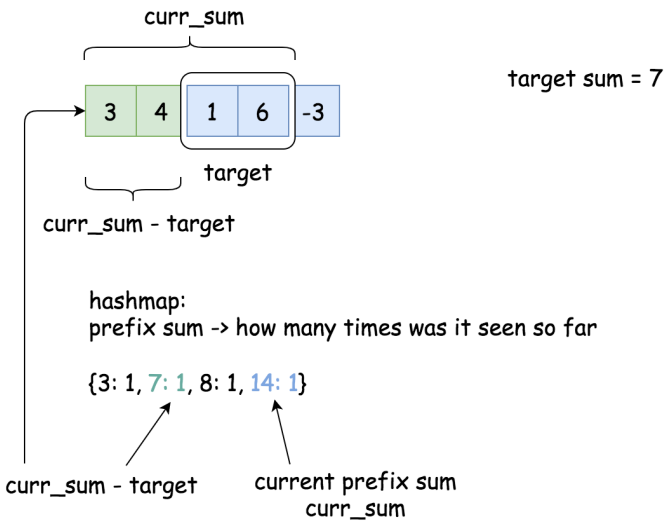


Figure 6. Situation 2: The subarray starts somewhere in the middle.

Solution for Number of Continuous Subarrays that Sum to Target

Here is an implementation of the solution for Find a number of continuous subarrays that sum to target.

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
public class Solution {
    // use prefix sum with hashmap
    HashMap<Long, Integer> map;
    int count;
    long prefixSum;

    public int pathSum(TreeNode root, int sum) {
        if (root == null) return 0;
        this.map = new HashMap<Long, Integer>();
        this.count = 0;
        this.prefixSum = 0;
        // init for subtree starting from root
        map.put((long) 0, 1);
        calPrefixSum(root, sum);
        return count;
    }

    private void calPrefixSum(TreeNode root, int sum) {
        if (root == null) return;
        prefixSum += root.val;
        count += map.get(prefixSum - sum) == null? 0 : map.get(prefixSum-sum);
        map.put(prefixSum, map.getOrDefault(prefixSum, 0)+1);


        calPrefixSum(root.left, sum);

        calPrefixSum(root.right, sum);

        map.put(prefixSum, map.getOrDefault(prefixSum, 0)-1);
        prefixSum -= root.val;
    }
}
```


Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [1000000000,1000000000,nu 0

Output 0 Diff

Expected 0