Saturday, December 25, 2021 12:21 PM

Zero Sum Subarrays

Medium Accuracy: 50.41% Submissions: 28275 Points: 4

You are given an array arr[] of size n. Find the total count of sub-arrays having their sum equal to 0.

Example 1:

Input:
n = 6
arr[] = {0,0,5,5,0,0}
Output: 6
Explanation: The 6 subarrays are
[0], [0], [0], [0], [0,0], and [0,0].

Example 2:

Input:
n = 10
arr[] = {6,-1,-3,4,-2,2,4,6,-12,-7}
Output: 4
Explanation: The 4 subarrays are [-1 -3 4]
[-2 2], [2 4 6 -12], and [-1 -3 4 -2 2]

Your Task:

You don't need to read input or print anything. Complete the function **findSubarray()** that takes the array arr and its size n as input parameters and returns the total number of sub-arrays with 0 sum.

Expected Time Complexity : O(n)

Expected Auxilliary Space : O(n)

Constraints:
$1 <= n <= 10^7$
$-10^{10} <= arr_i <= 10^{10}$

View Bookmarked Problems

Company Tags

Amazon Microsoft OYO Rooms

Topic Tags

Arrays Hash

From <https://practice.geeksforgeeks.org/problems/zero-sum-subarrays1825/1>

Given an array, print all subarrays in the array which has sum 0.

Examples:

```
Input:  arr = [6, 3, -1, -3, 4, -2, 2,
4, 6, -12, -7]
Output:
Subarray found from Index 2 to 4
Subarray found from Index 2 to 6
Subarray found from Index 5 to 6
Subarray found from Index 6 to 9
Subarray found from Index 0 to 10
```

Related posts: Find if there is a subarray with 0 sum

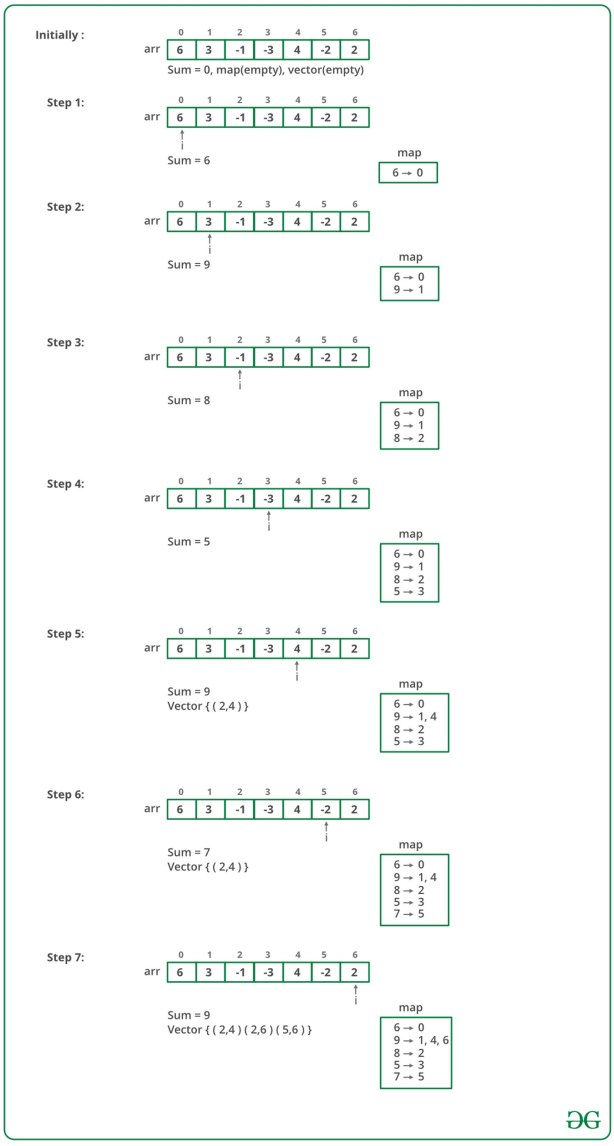
A simple solution is to consider all subarrays one by one and check if sum of every subarray is equal to 0 or not. The complexity of this solution would be O(n^2).

A better approach is to use Hashing.

Do following for each element in the array

1. Maintain sum of elements encountered so far in a variable (say sum).
2. If current sum is 0, we found a subarray starting from index 0 and ending at index current index
3. Check if current sum exists in the hash table or not.
4. If current sum already exists in the hash table then it indicates that this sum was the sum of some sub-array elements arr[0]...arr[i] and now the same sum is obtained for the current sub-array arr[0]...arr[j] which means that the sum of the sub-array arr[i+1]...arr[j] must be 0.
5. Insert current sum into the hash table

Below is a dry run of the above approach:



Below is the implementation of the above approach:

From <https://practice.geeksforgeeks.org/problems/zero-sum-subarrays1825/1#>

```
// Java program to print all subarrays
// in the array which has sum 0
import java.io.*;
import java.util.*;
```

```
// User defined pair class
class Pair
{
   int first, second;
   Pair(int a, int b)
   {
      first = a;
      second = b;
   }
}

public class GFG
{
   // Function to print all subarrays in the array which
   // has sum 0
   static ArrayList<Pair> findSubArrays(int[] arr, int n)
   {
         // create an empty map
         HashMap<Integer,ArrayList<Integer>> map = new HashMap<>();

         // create an empty vector of pairs to store
         // subarray starting and ending index
         ArrayList<Pair> out = new ArrayList<>();

         // Maintains sum of elements so far
         int sum = 0;

         for (int i = 0; i < n; i++)
         {
            // add current element to sum
            sum += arr[i];

            // if sum is 0, we found a subarray starting
            // from index 0 and ending at index i
            if (sum == 0)
               out.add(new Pair(0, i));
            ArrayList<Integer> al = new ArrayList<>();

            // If sum already exists in the map there exists
            // at-least one subarray ending at index i with
            // 0 sum
            if (map.containsKey(sum))
            {
               // map[sum] stores starting index of all subarrays
               al = map.get(sum);
               for (int it = 0; it < al.size(); it++)
               {
                     out.add(new Pair(al.get(it) + 1, i));
               }
            }
            al.add(i);
            map.put(sum, al);
         }
         return out;
   }

   // Utility function to print all subarrays with sum 0
   static void print(ArrayList<Pair> out)
   {
         for (int i = 0; i < out.size(); i++)
         {
            Pair p = out.get(i);
            System.out.println("Subarray found from Index "
                  + p.first + " to " + p.second);
         }
   }

   // Driver code
   public static void main(String args[])
   {
         int[] arr = {6, 3, -1, -3, 4, -2, 2, 4, 6, -12, -7};
         int n = arr.length;

         ArrayList<Pair> out = findSubArrays(arr, n);

         // if we did not find any subarray with 0 sum,
         // then subarray does not exists
         if (out.size() == 0)
            System.out.println("No subarray exists");
         else
            print(out);
   }
}

// This code is contributed by rachana soma
```

Find subarray with given sum | Set 1 (Nonnegative Numbers)

- Difficulty Level : Medium

- Last Updated : 07 Sep, 2021

Given an unsorted array of nonnegative integers, find a continuous subarray which adds to a given number.

Examples :

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Output: Sum found between indexes 2 and 4
Sum of elements between indices
2 and 4 is 20 + 3 + 10 = 33
Input: arr[] = {1, 4, 0, 0, 3, 10, 5}, sum = 7
Output: Sum found between indexes 1 and 4
Sum of elements between indices
1 and 4 is 4 + 0 + 0 + 3 = 7
```

```
Input: arr[] = {1, 4}, sum = 0
Output: No subarray found
There is no subarray with 0 sum
```

There may be more than one subarrays with sum as the given sum. The following solutions print first such subarray.

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Simple Approach: A simple solution is to consider all subarrays one by one and check the sum of every subarray. Following program implements the simple solution. Run two loops: the outer loop picks a starting point I and the inner loop tries all subarrays starting from i.

Algorithm:

1. Traverse the array from start to end.
2. From every index start another loop from *i* to the end of array to get all subarray starting from i, keep a variable sum to calculate the sum.
3. For every index in inner loop update *sum = sum + array[j]*
4. If the sum is equal to the given sum then print the subarray.

- C++
- C
- Java
- Python3
- C#
- PHP
- Javascript

```
class SubarraySum {
    /* Returns true if the there is a
subarray of arr[] with a sum equal to
       'sum' otherwise returns false.
Also, prints the result */
    int subArraySum(int arr[], int n, int sum)
    {
        int curr_sum, i, j;

        // Pick a starting point
        for (i = 0; i < n; i++) {
            curr_sum = arr[i];

            // try all subarrays starting with 'i'
            for (j = i + 1; j <= n; j++) {
                if (curr_sum == sum) {
                    int p = j - 1;
                    System.out.println(
                        "Sum found between indexes " + i
                        + " and " + p);
                    return 1;
                }
                if (curr_sum > sum || j == n)
                    break;
```

```
                curr_sum = curr_sum + arr[j];
            }
        }

        System.out.println("No subarray found");
        return 0;
    }

    public static void main(String[] args)
    {
        SubarraySum arraysum = new SubarraySum();
        int arr[] = { 15, 2, 4, 8, 9, 5, 10, 23 };
        int n = arr.length;
        int sum = 23;
        arraysum.subArraySum(arr, n, sum);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Output :

```
Sum found between indexes 1 and 4
```

Complexity Analysis:

- **Time Complexity:** O(n^2) in worst case.
  Nested loop is used to traverse the array so the time complexity is O(n^2)
- **Space Complexity:** O(1).
  As constant extra space is required.
  Efficient Approach: There is an idea if all the elements of the array are positive. If a subarray has sum greater than the given sum then there is no possibility that adding elements to the current subarray the sum will be *x* (given sum). Idea is to use a similar approach to a sliding window. Start with an empty subarray, add elements to the subarray until the sum is less than *x*. If the sum is greater than *x*, remove elements from the start of the current subarray.

Algorithm:

1. Create three variables, *l=0, sum = 0*
2. Traverse the array from start to end.
3. Update the variable sum by adding current element, *sum = sum + array[i]*
4. If the sum is greater than the given sum, update the variable sum as *sum = sum – array[l]*, and update l as, l++.
5. If the sum is equal to given sum, print the subarray and break the loop.

- C++
- C
- Java
- Python3
- C#
- PHP
- Javascript

```
class SubarraySum {
    /* Returns true if the there is
a subarray of arr[] with sum equal to
       'sum' otherwise returns false.
Also, prints the result */
    int subArraySum(int arr[], int n, int sum)
    {
        int curr_sum = arr[0], start = 0, i;

        // Pick a starting point
        for (i = 1; i <= n; i++) {
            // If curr_sum exceeds the sum,
            // then remove the starting elements
            while (curr_sum > sum && start < i - 1) {
                curr_sum = curr_sum - arr[start];
                start++;
            }

            // If curr_sum becomes equal to sum,
            // then return true
            if (curr_sum == sum) {
                int p = i - 1;
                System.out.println(
                    "Sum found between indexes " + start
                    + " and " + p);
                return 1;
            }

            // Add this element to curr_sum
            if (i < n)
                curr_sum = curr_sum + arr[i];
        }

        System.out.println("No subarray found");
        return 0;
    }

    public static void main(String[] args)
    {
```

```
        SubarraySum arraysum = new SubarraySum();
        int arr[] = { 15, 2, 4, 8, 9, 5, 10, 23 };
        int n = arr.length;
        int sum = 23;
        arraysum.subArraySum(arr, n, sum);
    }
}

// This code has been contributed by Mayank Jaiswal(mayank_24)
```

Output :

```
Sum found between indexes 1 and 4
```

Complexity Analysis:

- **Time Complexity :** O(n).
  Only one traversal of the array is required. So the time complexity is O(n).
- **Space Complexity:** O(1).
  As constant extra space is required.
  https://youtu.be/GY-KULykGaw?list=PLqM7alHXFySEQDk2MDfbwEdjd2svVJH9p

  The above solution doesn't handle negative numbers. We can use hashing to handle negative numbers. See below set 2.

- Find subarray with given sum | Set 2 (Handles Negative Numbers)
- Find subarray with given sum with negatives allowed in constant space

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

From <https://www.cdn.geeksforgeeks.org/find-subarray-with-given-sum/>

Find subarray with given sum | Set 2 (Handles Negative Numbers)

- Difficulty Level : Medium
- Last Updated : 15 Dec, 2021

Given an unsorted array of integers, find a subarray that adds to a given number. If there is more than one subarray with the sum of the given number, print any of them.

Examples:

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Output: Sum found between indexes 2 and 4
Explanation: Sum of elements between indices
2 and 4 is 20 + 3 + 10 = 33
Input: arr[] = {10, 2, -2, -20, 10}, sum = -10
Output: Sum found between indexes 0 to 3
Explanation: Sum of elements between indices
0 and 3 is 10 + 2 - 2 - 20 = -10
```

```
Input: arr[] = {-10, 0, 2, -2, -20, 10}, sum = 20
Output: No subarray with given sum exists
Explanation: There is no subarray with the given sum
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Note: We have discussed a solution that does not handle negative integers here. In this post, negative integers are also handled.

Simple Approach: A simple solution is to consider all subarrays one by one and check the sum of every subarray. Following program implements the simple solution. Run two loops: the outer loop picks a starting point I and the inner loop tries all subarrays starting from i.

Algorithm:

1. Traverse the array from start to end.
2. From every index start another loop from i to the end of the array to get all subarray starting from i, keep a variable sum to calculate the sum. For every index in the inner loop

update sum = sum + array[j]If the sum is equal to the given sum then print the subarray.
3. For every index in the inner loop update sum = sum + array[j]
4. If the sum is equal to the given sum then print the subarray.

From <https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/>

Java

```
// Java program for the above approach
import java.util.*;

class GFG
{

/* Returns true if the there is a subarray
of arr[] with sum equal to 'sum' otherwise
returns false. Also, prints the result */
static int subArraySum(int arr[], int n, int sum)
{
    int curr_sum, i, j;

    // Pick a starting point
    for (i = 0; i < n; i++) {
        curr_sum = 0;

        // try all subarrays starting with 'i'
        for (j = i ; j < n; j++) {
           curr_sum = curr_sum + arr[j];

            if (curr_sum == sum) {
                System.out.print( "Sum found between indexes "
                     + i + " and " + j);
                return 1;
            }


        }
    }

    System.out.print("No subarray found");
    return 0;
}

// Driver Code
public static void main (String[] args)
{
    int arr[] = { 15, 2, 4, 8, 9, 5, 10, 23 };
    int n = arr.length;
    int sum = 23;
    subArraySum(arr, n, sum);
}
}

// This code is contributed by code_hunt.
```

Output

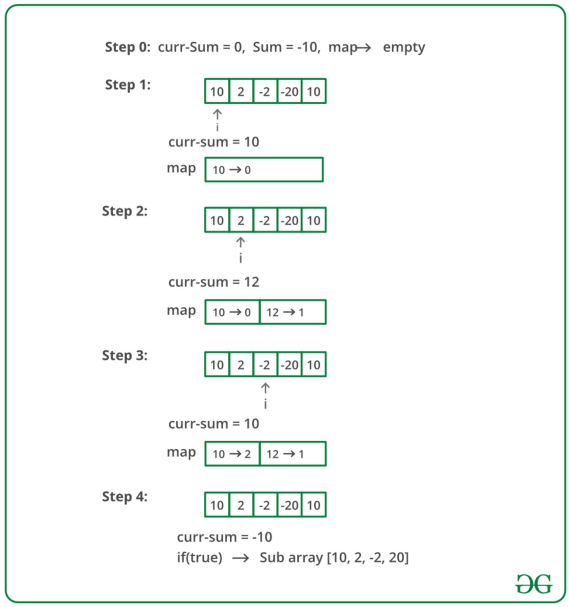
```
Sum found between indexes 1 and 4
```

Approach: The idea is to store the sum of elements of every prefix of the array in a hashmap, i.e, every index stores the sum of elements up to that index hashmap. So to check if there is a subarray with a sum equal to *s*, check for every index i, and sum up to that index as *x*. If there is a prefix with a sum equal to *x – s*, then the subarray with the given sum is found.

Algorithm:

5. Create a Hashmap (*hm*) to store a key-value pair, i.e, key = prefix sum and value = its index, and a variable to store the current sum (*sum = 0*) and the sum of the subarray as *s*
6. Traverse through the array from start to end.
7. For every element update the sum, i.e *sum = sum + array[i]*
8. If the sum is equal to s then print that the subarray with the given sum is from 0 to i
9. If there is any key in the HashMap which is equal to *sum – s* then print that the subarray with the given sum is from hm[sum – s] to i
10. Put the sum and index in the hashmap as a key-value pair.

Dry-run of the above approach:



Implementation:

From <https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/>

Java

```
// Java program to print subarray with sum as given sum
import java.util.*;

class GFG {

    public static void subArraySum(int[] arr, int n, int sum) {
        //cur_sum to keep track of cumulative sum till that point
        int cur_sum = 0;
        int start = 0;
        int end = -1;
        HashMap<Integer, Integer> hashMap = new HashMap<>();

        for (int i = 0; i < n; i++) {
            cur_sum = cur_sum + arr[i];
            //check whether cur_sum - sum = 0, if 0 it means
            //the sub array is starting from index 0- so stop
            if (cur_sum - sum == 0) {
                start = 0;
                end = i;
                break;
            }
            //if hashMap already has the value, means we already
            // have subarray with the sum - so stop
            if (hashMap.containsKey(cur_sum - sum)) {
                start = hashMap.get(cur_sum - sum) + 1;
                end = i;
                break;
            }
            //if value is not present then add to hashmap
            hashMap.put(cur_sum, i);

        }
        // if end is -1 : means we have reached end without the sum
        if (end == -1) {
            System.out.println("No subarray with given sum exists");
        } else {
            System.out.println("Sum found between indexes "
                               + start + " to " + end);
        }

    }

    // Driver code
    public static void main(String[] args) {
        int[] arr = {10, 2, -2, -20, 10};
        int n = arr.length;
        int sum = -10;
        subArraySum(arr, n, sum);

    }
}
```

From <https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/>

Complexity Analysis:

- **Time complexity:** O(N).

If hashing is performed with the help of an array, then this is the time complexity. In case the elements cannot be hashed in an array a hash map can also be used as shown in the above code.

- **Auxiliary space:** O(n).
  As a HashMap is needed, this takes linear space.

From <https://www.geeksforgeeks.org/find-subarray-with-given-sum-in-array-of-integers/>