GeeksforGeeks
A computer science portal for geeks
Practice IDE Q&A GeeksQuiz

Find subarray with given sum

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

Examples:

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Ouptut: Sum found between indexes 2 and 4

Input: arr[] = {1, 4, 0, 0, 3, 10, 5}, sum = 7
Ouptut: Sum found between indexes 1 and 4

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

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

Source: Google Interview Question

Method 1 (Simple)

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

```
/* A simple program to print subarray with sum as given sum */
#include<stdio.h>

/* 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, 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)
            {
                printf ("Sum found between indexes %d and %d", i, j-1);
                return 1;
            }
            if (curr_sum > sum || j == n)
                break;
           curr_sum = curr_sum + arr[j];
        }
    }

    printf("No subarray found");
    return 0;
}

// Driver program to test above function
int main()
{
    int arr[] = {15, 2, 4, 8, 9, 5, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = 23;
    subArraySum(arr, n, sum);
    return 0;
}
```

Run on IDE

Output:

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

Time Complexity: O(n^2) in worst case.

Method 2 (Efficient)

Initialize a variable curr_sum as first element. curr_sum indicates the sum of current subarray. Start from the second element and add all elements one by one to the curr_sum. If curr_sum becomes equal to sum, then print the solution. If curr_sum exceeds the sum, then remove trailing elemnents while curr_sum is greater than sum.

Following is C implementation of the above approach.

```
/* An efficient program to print subarray with sum as given sum */
#include<stdio.h>

/* 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)
{
    /* Initialize curr_sum as value of first element
       and starting point as 0 */
    int curr_sum = arr[0], start = 0, i;

    /* Add elements one by one to curr_sum and if the curr_sum exceeds the
       sum, then remove starting element */
    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)
        {
            printf ("Sum found between indexes %d and %d", start, i-1);
            return 1;
        }

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

    // If we reach here, then no subarray
    printf("No subarray found");
    return 0;
}

// Driver program to test above function
int main()
{
    int arr[] = {15, 2, 4, 8, 9, 5, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = 23;
    subArraySum(arr, n, sum);
    return 0;
}
```

Run on IDE

Output:

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

Time complexity of method 2 looks more than O(n), but if we take a closer look at the program, then we can figure out the time complexity is O(n). We can prove it by counting the number of operations performed on every element of arr[] in worst case. There are at most 2 operations performed on every element: (a) the element is added to the curr_sum (b) the element is subtracted from curr_sum. So the upper bound on number of operations is 2n which is O(n).

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

134 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3.1 Average Difficulty : **3.1/5.0**
Based on **7** vote(s)

Add to TODO List

Mark as DONE

Like Share 13 people like this. Be the first of your friends.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

@geeksforgeeks, Some rights reserved Contact Us! About Us! Advertise with us!