Subarray with Given Sum

25 September 2025 07:20 PM

Given a 1-based indexing array **arr[]** of **non-negative** integers and an integer **sum**. You mainly need to return the left and right indexes(**1-based indexing**) of that subarray. In case of multiple subarrays, return the subarray indexes which come first on moving from left to right. If no such subarray exists return an array consisting of element **-1**.

Examples:

***Input**: arr[] = [15, 2, 4, 8, 9, 5, 10, 23], target = 23*

***Output**: [2, 5]*

***Explanation:** Sum of subarray arr[2...5] is 2 + 4 + 8 + 9 = 23.*

***Input**: arr[] = [1, 10, 4, 0, 3, 5], target = 7*

***Output**: [3, 5]*

***Explanation:** Sum of subarray arr[3...5] is 4 + 0 + 3 = 7.*

***Input**: arr[] = [1, 4], target = 0*

***Output**: [-1]*

***Explanation:** There is no subarray with 0 sum.*

Try it on GfG Practice

Table of Content

- [Naive Approach] Using Nested loop - O(n2) Time and O(1) Space
- [Expected Approach] Sliding Window - O(n) Time and O(1) Space
- [Alternate Approach] Hashing + Prefix Sum - O(n) Time and O(n) Space

[Naive Approach] Using Nested loop - O(n^2) Time and O(1) Space

The very basic idea is to use a nested loop where the outer loop picks a starting element, and the inner loop calculates the cumulative sum of elements starting from this element.

*For each starting element, the inner loop iterates through subsequent elements and adding each element to the cumulative sum until the given sum is found or the end of the array is reached. If at any point the cumulative sum equals the given **sum**, then return starting and ending indices (1-based). If no such sub-array is found after all iterations, then return -1.*

```python
# Function to find a continuous sub-array which adds up to
# a given number.
def subarraySum(arr, target):
    res = []
    n = len(arr)

    # Pick a starting point for a subarray
    for s in range(n):
        curr = 0

        # Consider all ending points
        # for the picked starting point
        for e in range(s, n):
            curr += arr[e]
            if curr == target:
                res.append(s + 1)
                res.append(e + 1)
                return res

    # If no subarray is found
    return [-1]

if __name__ == "__main__":
    arr = [15, 2, 4, 8, 9, 5, 10, 23]
    target = 23
    res = subarraySum(arr, target)

    for ele in res:
        print(ele, end=" ")
```

✓ 0.0s

```
2 5
```

```
# Function to find a continuous sub-array which adds up to
# a given number.
def subarraySum(arr, target):
    res = []
    n = len(arr)

    # Pick a starting point for a subarray
    for s in range(n):
        curr = 0

        # Consider all ending points
        # for the picked starting point
        for e in range(s, n):
            curr += arr[e]
            if curr == target:
                res.append(s + 1)
                res.append(e + 1)
                return res

    # If no subarray is found
    return [-1]

if __name__ == "__main__":
    arr = [15, 2, 4, 8, 9, 5, 10, 23]
    target = 23
    res = subarraySum(arr, target)

    for ele in res:
        print(ele, end=" ")
```

Output

```
2 5
```

[Expected Approach] Sliding Window - O(n) Time and O(1) Space

*The idea is simple, as we know that all the elements in subarray are positive so, If a subarray has sum greater than the **given sum** then there is no possibility that adding elements to the current subarray will be equal to the given sum. So the Idea is to use a similar approach to a **sliding window**.*

- *Start with an empty window*
- *add elements to the window while the current sum is less than **sum***
- *If the sum is greater than **sum**, remove elements from the **start** of the current window.*
- *If current sum becomes same as **sum,** return the result*

```
# Function to find a continuous sub-array which adds up to
# a given number.
def subarraySum(arr, target):
    # Initialize window
    s, e = 0, 0
    res = []

    curr = 0
    for i in range(len(arr)):
        curr += arr[i]

        # If current sum becomes more or equal,
        # set end and try adjusting start
        if curr >= target:
            e = i

            # While current sum is greater,
            # remove starting elements of current window
            while curr > target and s < e:
                curr -= arr[s]
                s += 1

            # If we found a subarray
            if curr == target:
                res.append(s + 1)
                res.append(e + 1)
                return res

    # If no subarray is found
    return [-1]

if __name__ == "__main__":
    arr = [15, 2, 4, 8, 9, 5, 10, 23]
    target = 23
    res = subarraySum(arr, target)

    print(" ".join(map(str, res)))
```

Output

```
2 5
```

[Alternate Approach] Hashing + Prefix Sum - O(n) Time and O(n) Space

The above solution does not work for arrays with negative numbers. To handle all cases, we use hashing and prefix sum. 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 ***target***, check for every index **i**, and sum up to that index as ***currSum***. If there is a prefix with a sum equal to ***(currSum – target)***, then the subarray with the given sum is found.

To know more about the implementation, please refer Subarray with Given Sum – Handles Negative Numbers.

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