subarray sum hackerrank solution

Subarray sum hackerrank solution is a common problem that challenges developers and coding enthusiasts on platforms like HackerRank. It tests your understanding of arrays, subarrays, and the cumulative sum techniques. In this article, we will explore the problem in detail, analyze various approaches to solve it, and provide a comprehensive solution. By the end, you will gain a solid understanding of how to tackle similar problems in your coding journey.

Understanding the Problem

The subarray sum problem typically requires you to find the number of contiguous subarrays that sum to a given target value. A subarray is defined as a contiguous segment of an array. For instance, in an array of integers, you may need to determine how many subarrays have a sum equal to a specified number, such as zero or any positive integer.

Problem Definition

Given an array of integers and a target sum, you need to count how many contiguous subarrays have a sum equal to that target. For example, consider the following:

- Input:
- Array: [1, 2, 3, -3, 2]
- Target Sum: 3
- Output: 2
- Explanation: The subarrays that sum to 3 are [1, 2] and [3].

Approaches to Solve the Problem

There are several approaches to solve the subarray sum problem. Here, we will discuss the most effective methods:

Brute Force Approach

The simplest way to solve this problem is using a brute force approach. This involves checking all possible subarrays and calculating their sums. Although it is straightforward, it is also inefficient, especially for large arrays.

- Steps:
1. Loop through each element in the array as the starting point.
2. For each starting point, loop through the subsequent elements to create subarrays.
3. Calculate the sum of each subarray and compare it to the target sum.

- Complexity:
- Time Complexity: $O(n^2)$, where n is the number of elements in the array.

- Space Complexity: O(1), as it uses a constant amount of space.

Using Prefix Sums

A more efficient approach involves using prefix sums. The idea is to keep track of the cumulative sum of elements up to each index in the array.

- Steps:
1. Create an empty hash map to store the count of prefix sums.
2. Initialize a variable to keep track of the current sum.
3. Iterate through the array, updating the current sum and checking if the difference between the current sum and the target sum exists in the hash map.
4. If it does, add the count from the hash map to the result.
5. Update the hash map with the current sum.

- Complexity:
- Time Complexity: O(n).
- Space Complexity: O(n), due to the hash map used to store prefix sums.

Implementing the Solution

Here is a Python implementation of the prefix sums approach to solve the subarray sum problem:

```python
def subarray_sum(arr, target):
count = 0
current_sum = 0
prefix_sum_map = {0: 1} Initialize with sum of 0

for num in arr:
current_sum += num

Check if (current_sum - target) is in the prefix sum map
if (current_sum - target) in prefix_sum_map:
count += prefix_sum_map[current_sum - target]

Update the prefix sum map
if current_sum in prefix_sum_map:
prefix_sum_map[current_sum] += 1
else:
prefix_sum_map[current_sum] = 1

return count

Example usage
arr = [1, 2, 3, -3, 2]
target = 3
print(subarray_sum(arr, target)) Output: 2
```

Explanation of the Code

1. Initialization: The function starts by initializing the count of subarrays, the current cumulative sum, and a hash map that keeps track of the counts of different prefix sums.

2. Iterating through the array: For each element in the array, the current sum is updated. The code then checks if the difference between the current sum and the target value exists in the hash map. If it does, it means there are subarrays that sum to the target value, and the count is incremented accordingly.

3. Updating the hash map: Finally, the current sum is added to the hash map, either by incrementing its count if it already exists or by initializing it to 1.

Testing the Solution

Once you implement the solution, it is important to test it with various input cases to ensure its correctness and efficiency. Here are some test cases to consider:

- Test Case 1:
  - Input: [1, 2, 1, 2, 1], Target: 3
  - Expected Output: 4
- Test Case 2:
  - Input: [0, 0, 0, 0], Target: 0
  - Expected Output: 10
- Test Case 3:
  - Input: [-1, -1, 1], Target: 0
  - Expected Output: 1

Conclusion

The **subarray sum hackerrank solution** is a practical problem that helps you understand the concepts of arrays and cumulative sums. By using the prefix sum technique, you can efficiently count the number of contiguous subarrays that sum to a target value. This approach significantly reduces the time complexity compared to the brute force method. Mastering this problem will not only enhance your problem-solving skills but also prepare you for more advanced challenges in competitive programming and technical interviews.

Frequently Asked Questions

What is the Subarray Sum problem on HackerRank?

The Subarray Sum problem on HackerRank requires you to find the count of contiguous subarrays within an array that sum to a given value.

What is the time complexity of an optimal solution for the Subarray Sum problem?

An optimal solution for the Subarray Sum problem can be achieved with a time complexity of O(n) using a hash map to store the cumulative sums.

How do you handle negative numbers in the Subarray Sum problem?

To handle negative numbers, you maintain a cumulative sum and use a hash map to track how many times each sum has occurred, allowing you to account for negative contributions to the sum.

What data structure is typically used to solve the Subarray Sum problem efficiently?

A hash map (or dictionary) is typically used to store the cumulative sums and their frequencies, allowing for efficient lookup and updates.

Can the Subarray Sum problem be solved using a brute-force approach?

Yes, you can solve the Subarray Sum problem using a brute-force approach by checking all possible subarrays, but this has a time complexity of O(n^2), which is inefficient for larger arrays.

What is a common mistake when implementing the Subarray Sum solution?

A common mistake is not properly handling the case where the sum of subarrays equals the target at the beginning of the array, which can lead to incorrect counts.

How do you test your solution for the Subarray Sum problem?

You can test your solution by using various test cases, including edge cases such as an empty array, all negative numbers, and cases where no subarray sums to the target.

What programming languages are commonly used to solve the Subarray Sum problem on HackerRank?

Common programming languages used for solving the Subarray Sum problem on HackerRank include Python, Java, C++, and JavaScript.

Subarray Sum Hackerrank Solution

Subarray Sum Hackerrank Solution

Related Articles

- story of a secret state
- strange days my life with and without jim morrison
- success through a positive mental attitude

Back to Home