Split Array With Same Average

This problem involves dividing an array into two subarrays with equal averages. The goal is to determine if such a split is possible.

by MS Gaming

Understanding the Problem Statement

We are given an array of integers, 'nums'. Our task is to determine if we can divide this array into two non-empty subarrays, 'A' and 'B', such that the average of 'A' is equal to the average of 'B'.

Input

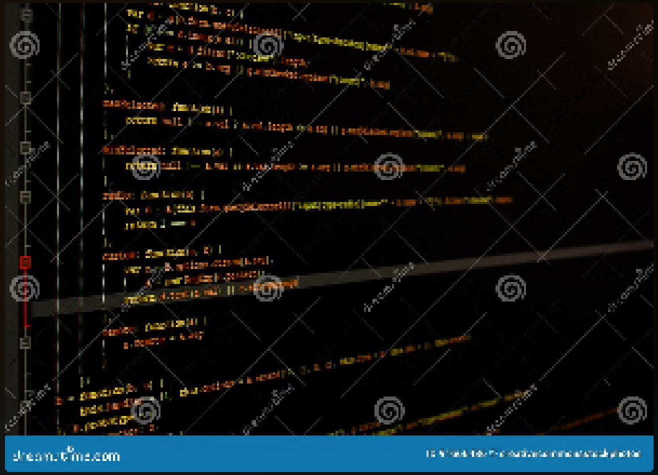
An integer array 'nums'.

Output

True if a split is possible, false otherwise.

Example

For 'nums' = [1,2,3,4,5,6,7,8], we can split it into [1,4,5,8] and [2,3,6,7], both with an average of 4.5.



Approach 1: Brute Force

We can try all possible combinations of dividing the array into two subarrays. For each combination, we can calculate the average of both subarrays and check if they are equal. This approach is computationally expensive, especially for larger arrays.

1. **Generate Subarrays**

   Generate all possible combinations of dividing the array into two subarrays.

2. **Calculate Averages**

   For each combination, calculate the average of the two subarrays.

3. **Compare Averages**

   Check if the averages of the two subarrays are equal.

Approach 2: Dynamic Programming

Dynamic programming can be used to solve this problem more efficiently. We can create a 2D table, 'dp', where 'dp[i][j]' represents whether a subarray ending at index 'i' with a sum of 'j' can be created. This table can be used to determine if a split with equal averages is possible.

i	0	1	2	3	4	5	6	7	8
0	True	False	False	False	False	False	False	False	False
1	True	True	False	False	False	False	False	False	False
2	True	True	True	False	False	False	False	False	False
3	True	True	True	True	False	False	False	False	False
4	True	True	True	True	True	False	False	False	False
5	True	True	True	True	True	True	False	False	False
6	True	True	True	True	True	True	True	False	False
7	True	True	True	True	True	True	True	True	False

Optimizing the Dynamic Programming Solution

The dynamic programming solution can be optimized by calculating the total sum of the array and the desired sum for each subarray. This allows us to efficiently check for the existence of two subarrays with equal averages.

1

Calculate Total Sum

Calculate the total sum of the elements in the array 'nums'.

2

Calculate Target Sum

Calculate the target sum for each subarray, which is half the total sum.

3

Populate DP Table

Populate the dynamic programming table to check for the existence of subarrays with the target sum.

Handling Edge Cases

It is essential to consider edge cases. If the array is empty or has only one element, it cannot be split into two non-empty subarrays. Similarly, if the total sum of the array is odd, it is impossible to achieve equal averages for the subarrays.

1. **Empty Array**

   If the array is empty, return false.

2. **Single Element Array**

   If the array has only one element, return false.

3. **Odd Total Sum**

   If the sum of the array is odd, return false.

Time and Space Complexity Analysis

The time complexity of the optimized dynamic programming solution is O(n * sum), where 'n' is the length of the array and 'sum' is the total sum of the elements. The space complexity is also O(n * sum) due to the dynamic programming table.

Time Complexity

O(n * sum)

Space Complexity

O(n * sum)

Conclusion and Key Takeaways

Dynamic programming provides an efficient solution to determine if an array can be split into two subarrays with equal averages. Understanding edge cases is crucial for ensuring correct handling of all possible inputs. This problem highlights the power of dynamic programming in solving complex problems.

Key Points

- Brute force solution is computationally expensive.
- Dynamic programming offers a more efficient solution.
- Edge cases must be handled carefully.

Applications

This problem has applications in areas such as resource allocation, data partitioning, and load balancing.