Documentation for "Arithmetic Slices II - Subsequence"

Table of Contents

1. Problem Statement

Given an integer array of nums, return the number of all arithmetic subsequences of nums. A sequence of numbers is called arithmetic if it consists of at least three elements and the difference between any two consecutive elements is the same.

Example 1:

Input: nums = [2,4,6,8,10]
Output: 7
Explanation: All arithmetic subsequence slices are:

- [2,4,6]
- [4,6,8]
- [6,8,10]
- [2,4,6,8]
- [4,6,8,10]
- [2,4,6,8,10]
- [2,6,10]

Example 2:

Input: nums = [7,7,7,7,7]
Output: 16
Explanation: Any subsequence of this array is arithmetic.

Constraints:

- 1 <= nums.length <= 1000
- -2^{31} <= nums[i] <= $2^{31} - 1$

2. Intuition

An arithmetic subsequence is a sequence of numbers where the difference between consecutive elements is constant. To count all possible arithmetic subsequences, we need to use dynamic programming (DP) to track subsequences of different differences.

Each subsequence in the array is defined by its last element and the common difference between consecutive elements. Maintaining a DP array can efficiently calculate the number of valid arithmetic subsequences.

3. Key Observations

i. Arithmetic Sequence Definition: A sequence is arithmetic if the difference between consecutive numbers is constant. For example, [2, 4, 6] has a common difference of 2.
ii. Subsequences: A subsequence can be formed by removing some elements from the array. However, the length of the subsequence must be at least 3 to count as a valid arithmetic subsequence.
iii. Dynamic Programming:
    a. Use a dynamic programming approach where each entry dp[i][diff] represents the number of subsequences ending at index i with a difference of diff.
    b. For each pair of indices i and j (where j < i), calculate the difference diff = nums[i] - nums[j]. The number of subsequences that can end at i with a difference diff can be derived from the subsequences that end at j with the same difference.

4. Approach

Steps:

i. Dynamic Programming Setup:
   a. Initialize dp as a list of defaultdicts to store counts of subsequences for each difference.
ii. Traverse Array:
   a. For each index i, iterate over all previous indices j (j < i).
   b. Calculate the difference diff = nums[i] - nums[j].
   c. The number of subsequences ending at i with the same difference can be obtained from the subsequences ending at j with that difference. Add these subsequences and update the count.
iii. Count Subsequences:
   a. We count the number of valid subsequences by considering all subsequences of length ≥ 3.

Algorithm:

i. Initialize an empty list dp where each element is a defaultdict to store subsequences ending at that index for each difference.
ii. For each pair of indices i and j, compute the difference diff and update the DP array accordingly.
iii. Add the valid subsequences (those of length ≥ 3) to the total count.

5. Edge Cases

- Single Element Array: If the array has fewer than 3 elements, there are no valid arithmetic subsequences.
- Identical Elements: If all elements in the array are the same, every subsequence of length ≥ 3 is a valid arithmetic subsequence.
- Array with No Arithmetic Sequences: If no valid subsequences exist, the answer will be 0.
- Large Arrays: The algorithm must efficiently handle arrays with a length of up to 1000.

6. Complexity Analysis

Time Complexity:

- The solution involves iterating over each pair of indices i and j (where i ranges from 0 to n-1, and j ranges from 0 to i-1), leading to a time complexity of $O(n^2)$, where n is the length of the input array nums.

Space Complexity:

- The space complexity is $O(n^2)$, as we maintain a DP array of size n where each element is a defaultdict that can store multiple differences.

7. Alternative Approaches

i. Brute Force Approach:
   a. A brute force solution would involve generating all possible subsequences and checking if they are arithmetic. This approach is inefficient, especially for large arrays, with a time complexity of $O(2^n)$, where n is the length of the array.

ii. Optimized Approach Using Hash Maps:
   a. Instead of using an array of defaultdicts, we could use a hash map for each element, reducing the space overhead. However, the time complexity would remain $O(n^2)$.

8. Test Cases

Test Case 1:

Input: nums = [2, 4, 6, 8, 10]
Expected Output: 7
Explanation: As described earlier, the arithmetic subsequences are:

- [2, 4, 6]
- [4, 6, 8]
- [6, 8, 10]
- [2, 4, 6, 8]

- [4, 6, 8, 10]
- [2, 4, 6, 8, 10]
- [2, 6, 10]

Test Case 2:

Input: nums = [7, 7, 7, 7, 7]
Expected Output: 16
Explanation: Every subsequence of length ≥ 3 is an arithmetic sequence.

9. Final Thoughts

This problem is a good exercise in dynamic programming and helps understand how to count subsequences based on specific conditions efficiently. The solution leverages hash maps to track subsequences with common differences, making it scalable for large inputs. Although the time complexity is quadratic, it is efficient enough for the input constraints.