1726. Tuple with Same Product

Solved

Medium · Topics · Companies · Hint

Given an array `nums` of **distinct** positive integers, return *the number of tuples* `(a, b, c, d)` *such that* `a * b = c * d` *where* `a`, `b`, `c`, *and* `d` *are elements of* `nums`, *and* `a != b != c != d`.

Example 1:

```
Input: nums = [2,3,4,6]
Output: 8
Explanation: There are 8 valid tuples:
(2,6,3,4) , (2,6,4,3) , (6,2,3,4) , (6,2,4,3)
(3,4,2,6) , (4,3,2,6) , (3,4,6,2) , (4,3,6,2)
```

Example 2:

```
Input: nums = [1,2,4,5,10]
Output: 16
Explanation: There are 16 valid tuples:
(1,10,2,5) , (1,10,5,2) , (10,1,2,5) , (10,1,5,2)
(2,5,1,10) , (2,5,10,1) , (5,2,1,10) , (5,2,10,1)
(2,10,4,5) , (2,10,5,4) , (10,2,4,5) , (10,2,5,4)
(4,5,2,10) , (4,5,10,2) , (5,4,2,10) , (5,4,10,2)
```

1. Problem Explanation

We are given an array `nums` of distinct positive integers. We need to find the number of tuples `(a, b, c, d)` such that:

- `a * b = c * d`
- `a`, `b`, `c`, and `d` are distinct elements from `nums`.

Example 1:

- Input: `nums = [2, 3, 4, 6]`
- Output: `8`
- Explanation: The valid tuples are:
  - `(2,6,3,4)`, `(2,6,4,3)`, `(6,2,3,4)`, `(6,2,4,3)`
  - `(3,4,2,6)`, `(4,3,2,6)`, `(3,4,6,2)`, `(4,3,6,2)`

Example 2:

- Input: `nums = [1, 2, 4, 5, 10]`
- Output: `16`
- Explanation: The valid tuples are:
  - `(1,10,2,5)`, `(1,10,5,2)`, `(10,1,2,5)`, `(10,1,5,2)`
  - `(2,5,1,10)`, `(2,5,10,1)`, `(5,2,1,10)`, `(5,2,10,1)`
  - `(2,10,4,5)`, `(2,10,5,4)`, `(10,2,4,5)`, `(10,2,5,4)`
  - `(4,5,2,10)`, `(4,5,10,2)`, `(5,4,2,10)`, `(5,4,10,2)`

2. Approach

Non-Optimal Approach (Brute Force):

- **Algorithm:**
  - Generate all possible quadruples `(a, b, c, d)` where `a`, `b`, `c`, and `d` are distinct elements from `nums`.
  - Check if `a * b = c * d`.
  - Count the number of valid tuples.
- **Data Structures:**
  - Nested loops to generate all possible combinations.
- **Time Complexity:** O(n^4), where `n` is the length of `nums`.
- **Space Complexity:** O(1), as we are not using any additional space.

Optimal Approach:

- **Algorithm:**
  - Use a hash map (dictionary) to store the frequency of each product `a * b` where `a` and `b` are distinct elements.
  - For each product, calculate the number of ways to form pairs `(a, b)` and `(c, d)` such that `a * b = c * d`.
  - Multiply the counts of pairs to get the total number of valid tuples.
- **Data Structures:**
  - Hash map (dictionary) to store product frequencies.
- **Time Complexity:** O(n^2), where `n` is the length of `nums`.
- **Space Complexity:** O(n^2), due to the hash map storing products.

3. Optimal Code with Comments

```python
from collections import defaultdict

def countTuples(nums):
    product_count = defaultdict(int)
    n = len(nums)

    # Step 1: Count the frequency of each product a * b
    for i in range(n):
        for j in range(i + 1, n):
            product = nums[i] * nums[j]
            product_count[product] += 1

    # Step 2: Calculate the total number of valid tuples
    total = 0
    for count in product_count.values():
        if count >= 2:
            # For each product, the number of ways to choose 2 pairs is count *
(count - 1)
            # Each pair can be arranged in 2 ways (a,b) and (b,a), so multiply
by 4
            total += count * (count - 1) * 4

    return total

# Example usage:
nums = [2, 3, 4, 6]
print(countTuples(nums))  # Output: 8
```

4. Explanation of the Algorithm

1. **Step 1: Counting Products**
   - We iterate through all pairs `(a, b)` in `nums` and calculate their product `a * b`.
   - We store the frequency of each product in a hash map (`product_count`).
2. **Step 2: Calculating Valid Tuples**
   - For each product that appears at least twice, we calculate the number of ways to form two distinct pairs `(a, b)` and `(c, d)`.
   - Each pair can be arranged in 2 ways (e.g., `(a, b)` and `(b, a)`), so we multiply by 4 to account for all permutations.
3. **Final Result**
   - The total number of valid tuples is the sum of all valid combinations for each product.

5. Dry Run for Optimal Code

Input: `nums = [2, 3, 4, 6]`

Step 1: Counting Products

- Pairs and their products:
  - (2, 3) → 6
  - (2, 4) → 8
  - (2, 6) → 12
  - (3, 4) → 12
  - (3, 6) → 18
  - (4, 6) → 24
- `product_count` becomes: `{6: 1, 8: 1, 12: 2, 18: 1, 24: 1}`

Step 2: Calculating Valid Tuples

- Only the product `12` appears twice.
- Number of ways to choose 2 pairs from 2 pairs: `2 * (2 - 1) = 2`
- Each pair can be arranged in 2 ways, so total tuples: `2 * 4 = 8`

Output: `8`

6. Time and Space Complexity

Non-Optimal Approach:

- **Time Complexity:** O(n^4)
  - We are generating all possible quadruples, which is computationally expensive.
- **Space Complexity:** O(1)
  - No additional space is used.

Optimal Approach:

- **Time Complexity:** O(n^2)
  - We are iterating through all pairs of elements, which is much more efficient.
- **Space Complexity:** O(n^2)
  - We are storing the frequency of each product, which can be up to `n^2` in the worst case.

Why Optimal is Better:

- The optimal approach reduces the time complexity from O(n^4) to O(n^2), making it feasible for larger inputs.
- The space complexity is higher, but it is a trade-off for the significant reduction in time complexity.

Data Structures and Algorithms Used:

- **Non-Optimal:** Nested loops (brute force).
- **Optimal:** Hash map (dictionary) for counting frequencies, nested loops for generating pairs.

This approach ensures that we efficiently count the number of valid tuples without generating all possible combinations explicitly.