1819. Number of Different Subsequences GCDs

Hard · Array · Math · Counting · Number Theory — Leetcode Link

Problem Description

In this problem, you are given an array of positive integers called `nums`. You are asked to calculate the number of different Greatest Common Divisors (GCDs) that can be obtained from all possible non-empty subsequences of the array.

The term GCD refers to the largest integer that can evenly divide all the numbers in a sequence. For instance, the GCD of the sequence `[4,6,16]` is `2`, since 2 is the greatest integer that can divide 4, 6, and 16 without leaving a remainder.

A subsequence is defined as a sequence that can be derived from the array by removing some elements (or potentially no elements at all). For example, `[2,5,10]` is a subsequence of `[1,2,1,2,4,1,5,10]`.

The goal is to determine how many distinct GCDs you can find among all non-empty subsequences of the `nums` array.

Intuition

The solution is based on the insight that to find the GCD of a sequence, one does not necessarily need to consider all elements of the sequence; it's enough to consider only those elements that are multiples of a potential GCD candidate.

To employ this insight, we iterate over all integers `x` from 1 up to the maximum number in `nums`, considering each `x` as a potential GCD. For each potential GCD `x`, we want to find if there's a subsequence of nums whose GCD is exactly `x`. We do this by iterating through all multiples of `x` up to the maximum number in `nums` and calculating their GCD. If at any point our calculated GCD equals `x`, we know that we've found a subsequence whose GCD is `x`, and we count that.

In more detail, we follow these steps:

1. Find the maximum number in `nums`. This bounds the largest possible GCD.
2. Create a visibility set `vis` that contains all numbers present in `nums` to allow for O(1) lookups.
3. Initialize a counter `ans` to keep track of how many different GCDs we have found.
4. Iterate over each integer `x` starting from 1 up to the maximum number (inclusive). For each `x`, iterate through its multiples.
5. If a multiple `y` of `x` is in `vis`, calculate the GCD of `y` and the current GCD value `g`.
6. As soon as the GCD equals `x`, increment the counter `ans` and stop considering further multiples of `x` since we have found a valid subsequence for this GCD.
7. Return the counter `ans`, which now contains the number of distinct GCDs.

By doing this, we systematically check each number as a potential GCD and use the presence of its multiples in `nums` to see if that number is the GCD of some subsequence. With this approach, we are able to arrive at the solution efficiently.

Solution Approach

The implementation of the solution uses the mathematical concept of the Greatest Common Divisor (GCD) and some basic set operations for efficient lookup. The general approach involves iterating over all possible GCD values, checking for their existence by examining the multiples within the given `nums` array, and calculating the actual GCD. Here is a deeper dive into the approach:

1. **Calculate the Maximum Value (`mx`):** Determine the maximum value in the `nums` array, which sets a bound for the largest possible GCD.

2. **Visibility Set (`vis`):** Convert the `nums` array into a set. This `vis` set allows for constant-time complexity (`O(1)`) checks to determine if a multiple of the current GCD candidate `x` is in the `nums` array.

3. **GCD Function:** The built-in `gcd` function from Python's `math` module is used to compute the Greatest Common Divisor of two numbers.

4. **Iterate Over Potential GCDs (`x`):** For every integer `x` from 1 to `mx`, perform the following:

   - Initialize a variable `g` to 0, which will store the current GCD as we iterate through the multiples of `x`.
   - Loop through each multiple of `x`—`y`—starting from `x` to `mx` (inclusive) and incremented by `x`. We only need to consider multiples of `x` because a GCD will always have to be a factor of the numbers in the subsequence.

5. **Check Multiples and Calculate GCD:**

   - If the multiple `y` is present in the `vis` set, compute the GCD of the current GCD value `g` and `y`. This GCD represents the GCD of the subsequence consisting of the multiples of `x` encountered so far. As such, `g` is incrementally updated to always reflect the GCD of this growing subsequence.
   - If at any point `g` becomes equal to `x`, it means we've found a subsequence whose GCD is the current candidate `x`. When this happens, increment the counter `ans` by 1 to signify that another unique GCD has been found.
   - Break out of the loop for multiples of `x` because we've accomplished our goal for the current GCD candidate `x`. Continuing the loop would be redundant since we only count distinct GCDs.

6. **Return Result:** After going through all integers from 1 to `mx`, the counter `ans` holds the number of different GCDs. Returning `ans` completes the implementation and gives us the desired result.

The time complexity of the solution comes down to the number of iterations we perform, which depends on the range of numbers (`mx`) and the number of divisors they have. By using a set for lookups and the efficient `gcd` function, the implementation maintains acceptable performance for the given constraints.

Example Walkthrough

Let's consider a simple example to illustrate the solution approach. We will use the array `nums = [3, 6, 12]`.

Here's a step-by-step walkthrough using the provided solution approach:

1. **Calculate the Maximum Value (`mx`):** The maximum value in the `nums` array is `12`.

2. **Visibility Set (`vis`):** Convert the array to a set: `vis = {3, 6, 12}`.

3. **GCD Function:** We will use Python's built-in `math.gcd` function to compute the Greatest Common Divisor.

4. **Iterate Over Potential GCDs (`x`):** We iterate over each integer from `1` to `12` (inclusive). Let's take a few iterations as an example:

   - For `x = 1`:
     - Initialize `g = 0`.
     - Loop through multiples of `1`, i.e., all numbers from `1` to `12`. For each multiple, update `g` with the GCD value of `g` and the multiple if it exists in `vis`.
     - Since everything divides by `1`, the GCD will stay `1`.
     - We find a subsequence: `[3]`, `[6]`, `[12]`, where the GCD is `1`.
   - For `x = 2`:
     - Initialize `g = 0`.
     - Loop through multiples of `2`: `2`, `4`, `6`, `8`, `10`, `12`.
     - When we reach `6` (which exists in `vis`), we calculate the GCD of `g` (which is `0` initially) and `6`, resulting in a GCD of `6`.
     - Since `6` is not `2`, we go on until we reach `12`. The GCD of `6` and `12` is still `6`. Since we never reached a GCD of `2`, `2` is not counted.
   - For `x = 3`:
     - Initialize `g = 0`.
     - Loop through multiples of `3`: `3`, `6`, `9`, `12`. The first multiple we encounter is `3`, which exists in `vis`. The GCD of `0` and `3` is `3`.
     - As `g = x`, we've found a subsequence with this GCD and increment `ans` by `1`.
     - The same will hold true at multiple `6` and `12`, we already have `g = 3`, so we stop checking further multiples.
   - Continue this process for each integer `x` up to `12`.

5. **Check Multiples and Calculate GCD:** As shown in steps for `x = 1`, `2`, and `3`, we calculate the GCD for each multiple within `vis`, and if at any point `g` equals `x`, we count it in our answer and stop checking further multiples of `x`.

6. **Return Result:** After iterating over all numbers from `1` to `12`, we count all the unique GCDs that we found. In this example, the final count would be the GCDs obtained from all the unique subsequences, which would include `1` (from every possible subsequence), `3` (from `[3]`, `[3,6]`, `[3,12]`, `[3,6,12]`), and `6` (from `[6,12]`), giving us a total of 3 distinct GCDs.

In conclusion, the returned value for this `nums` array would be `3`, as there are three different possible GCDs for the non-empty subsequences in the array.

Python Solution

```python
from math import gcd
from typing import List

class Solution:
    def countDifferentSubsequenceGCDs(self, nums: List[int]) -> int:
        # Find the maximum value in the list to set the range for checking
        max_num = max(nums)

        # Create a set from the list for faster membership testing
        unique_nums = set(nums)

        # Initialize the answer count
        count = 0

        # Iterate over all possible gcd values
        for x in range(1, max_num + 1):
            # Initialize greatest common divisor for the current number x
            current_gcd = 0

            # Iterate through multiples of x
            for y in range(x, max_num + 1, x):
                # If multiple y exists in the original list, calculate its gcd with the current gcd
                if y in unique_nums:
                    current_gcd = gcd(current_gcd, y)
                    # If the gcd equals the current number x, then increment the count
                    if current_gcd == x:
                        count += 1
                        break

        return count
```

Java Solution

```java
import java.util.Arrays;

class Solution {
    // Method to count the number of different subsequence GCDs in the given array.
    public int countDifferentSubsequenceGCDs(int[] nums) {
        // Find the maximum value in the array to define the range of possible GCDs
        int maxVal = Arrays.stream(nums).max().getAsInt();

        // Initialize an array to keep track of visited numbers within the range
        boolean[] visited = new boolean[maxVal + 1];

        // Mark numbers that are present in the input array
        for (int num : nums) {
            visited[num] = true;
        }

        // Counter for the number of distinct subsequence GCDs
        int count = 0;

        // Iterate through all possible values to check if they can be a GCD of a subsequence
        for (int candidate = 1; candidate <= maxVal; ++candidate) {
            int gcdValue = 0;
            // Check multiples of the candidate if they are visited and calculate the GCD
            for (int multiple = candidate; multiple <= maxVal; multiple += candidate) {
                if (visited[multiple]) {
                    gcdValue = gcd(gcdValue, multiple);
                    // If the GCD equals the candidate, increment count and exit loop
                    if (candidate == gcdValue) {
                        ++count;
                        break;
                    }
                }
            }
        }

        // Return the total count of different subsequence GCDs
        return count;
    }

    // Helper method to calculate the GCD of two numbers using Euclidean algorithm
    private int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b);
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    // Function to count the number of different subsequences with distinct GCDs.
    int countDifferentSubsequenceGCDs(vector<int>& nums) {
        // Find the maximum element in the nums array.
        int maxElement = *max_element(nums.begin(), nums.end());
        // Initialize a boolean vector to mark visited elements in the range up to maxElement.
        vector<bool> visited(maxElement + 1, false);
        // Mark the existing elements in nums as visited.
        for (int num : nums) {
            visited[num] = true;
        }
        // Initialize a counter for the number of different GCDs.
        int totalCount = 0;
        // Iterate over each possible number x from 1 to maxElement.
        for (int x = 1; x <= maxElement; ++x) {
            // Initialize gcd to 0 for this iteration.
            int gcd = 0;
            // Check multiples of x within the range up to maxElement for GCD calculations.
            for (int multiple = x; multiple <= maxElement; multiple += x) {
                // If the current multiple has been visited, it is in the original nums array.
                if (visited[multiple]) {
                    gcd = std::gcd(gcd, multiple); // Compute GCD of current gcd and the multiple.
                    // If GCD equals x at any point, increment totalCount and break out of the loop.
                    if (gcd == x) {
                        totalCount++;
                        break;
                    }
                }
            }
        }
        // Return the total count of distinct GCDs.
        return totalCount;
    }
};

// Note: The method 'std::gcd' is part of the numeric header and C++17 standard library.
// Therefore, it is assumed that the appropriate header file is included:
// #include <numeric>
```

Typescript Solution

```typescript
// Import necessary functions from math-related libraries.
import { max, gcd } from 'mathjs';

// Function to count the number of different subsequences with distinct GCDs.
function countDifferentSubsequenceGCDs(nums: number[]): number {
    // Find the maximum element in the nums array.
    const maxElement = max(nums) as number;
    // Initialize an array to mark whether an element in the range up to maxElement was visited.
    const visited: boolean[] = new Array(maxElement + 1).fill(false);
    // Mark the existing elements in nums as visited.
    nums.forEach(num => {
        visited[num] = true;
    });
    // Initialize a counter for the number of different GCDs.
    let totalCount = 0;
    // Iterate over each possible number x from 1 to maxElement.
    for (let x = 1; x <= maxElement; ++x) {
        // Initialize gcd to 0 for this iteration.
        let currentGCD = 0;
        // Check multiples of x within the range up to maxElement for GCD calculations.
        for (let multiple = x; multiple <= maxElement; multiple += x) {
            // If the current multiple has been visited, it is in the original nums array.
            if (visited[multiple]) {
                // Compute GCD of current gcd and the multiple.
                currentGCD = gcd(currentGCD, multiple) as number;
                // If GCD equals x at any point, increment totalCount and break out of the loop.
                if (currentGCD === x) {
                    totalCount++;
                    break;
                }
            }
        }
    }
    // Return the total count of distinct GCDs.
    return totalCount;
}

// For TypeScript, we assume that an appropriate math library (like 'mathjs') is being used,
// and it should be imported at the beginning of the file.
// Note: The function 'gcd' used here refers to the greatest common divisor function
// which may be a part of a third-party mathematics library in TypeScript.
```

Time and Space Complexity

The code defines a function that counts the number of different greatest common divisors (GCDs) of all subsequences of the input array.

Time Complexity:

Let's denote `n` as the length of the array `nums` and `mx` as the maximum value in `nums`.

First, the function computes the maximum value from `nums` using `max(nums)`, which takes `O(n)` time.

Next, the function iterates from `1` to `mx` with nested loops:

- The outer loop runs `mx` times.
- The inner loop runs at most `mx / x` times for each value of `x`, because it increments by `x` on each iteration.

Within the inner loop, operations include checking membership in a set `vis` and computing the GCD, both of which are `O(1)` operations on average due to hashing for set membership and efficient algorithms for computing GCD (like Euclid's algorithm).

The costly part comes from the nested loops, so the total time complexity is: `O(n) + O(mx * ∑(1 .. mx) (1/x))` which simplifies to `O(n) + O(mx * ln(mx))` because the harmonic series `∑(1 .. mx) (1/x)` roughly approximates the natural logarithm of `mx`.

Thus, the time complexity of the function is `O(n + mx * ln(mx))`.

Space Complexity:

The space is spent on:

- Storing `vis`, which is a set of elements in `nums`. It therefore requires `O(n)` space.
- Storing the `mx`, `g`, and `ans` variables, which require `O(1)` space.

Consequently, the space complexity of the function is `O(n)`.