2470. Number of Subarrays With LCM Equal to K

Medium Array Math Number Theory

Leetcode Link

Problem Description

The problem provides us with two inputs: an array `nums` of integers and an integer `k`. We are required to find the number of contiguous non-empty sequences within the array (subarrays), such that the Least Common Multiple (LCM) of the numbers in each of these subarrays is exactly `k`.

A subarray is a sequence of consecutive elements from the array and has at least one element. It's not necessary for a subarray to include every element from the original array; it could just contain a part of it.

The Least Common Multiple (LCM) of an array is defined as the smallest positive integer divisible by each of the array elements. In other words, each element in the array is a factor of the LCM.

For example, if given `nums = [2,3,4]` and `k = 12`, we'd be looking for subarrays like `[2,3,4]` and `[3,4]` since the LCMs of these subarrays are 12.

Intuition

The intuition behind the solution is to iteratively explore all possible subarrays and calculate their LCM until we find the subarrays where the LCM is equal to `k`. This can be done by checking each possible pair of starting and ending positions within the `nums` array.

We initialize a variable to count the number of desirable subarrays. Starting with the first element of the array, we calculate the LCM of this single element and then extend the subarray one element at a time by including the next element in sequence and updating the LCM of the broader subarray to include this new element. We compare the updated LCM to `k`. If it's equal, we increment our count.

This process is repeated, expanding the subarray one element at a time, until we've considered all elements from the current starting position. We then shift our starting position by one and repeat the entire process until we've explored all possible starting positions in the array.

The provided solution does not directly call a built-in 'lcm' function since it was pseudocode, where 'lcm' would represent a standard function to calculate the Least Common Multiple. In an actual Python implementation, you would have to define such a function or use math algorithms to calculate the LCM for each step.

Solution Approach

The implementation begins by setting a counter `ans` to 0, which will keep a tally of the number of subarrays that have `k` as their LCM. The outer loop iterates over each `start` position of a potential subarray within `nums`. The inner loop then iterates through each `end` position, starting from the current `start` position and extending to the end of `nums`.

No complex data structures are necessary for this approach; we only require variables to store the ongoing LCM (`a`) and the answer counter (`ans`).

The `lcm(a, b)` function is called at each step, which calculates the LCM of the current accumulated LCM value `a` and the new element `b` from the array. If `lcm(a, b)` returns a value that equals `k`, we know we have found a valid subarray, and we increment `ans` by 1. The accumulator `a` is then updated to the new LCM to reflect the LCM of the subarray from the `start` position to the current position `i` in `nums`. This ensures we are always calculating the LCM of the current subarray under consideration.

The algorithm's efficiency could vary widely based on the implementation of the LCM calculation and the range of numbers in `nums`. It's a brute force approach and may not be efficient for larger arrays or larger values of `k` due to its time complexity, which is O(n^2) in the number of elements `n` in `nums` because of the nested loops. Each extension of the subarray involves computing an LCM, which itself can be costly depending on the size of the numbers involved.

Note: In an actual Python implementation, since the provided pseudocode assumes the existence of an `lcm` function, one would either define a function to calculate the LCM or utilize a library function such as `math.gcd()` to calculate the Greatest Common Divisor (GCD) and then use the relation `lcm(a, b) = abs(a*b) // gcd(a, b)` to find the LCM of two numbers.

Example Walkthrough

Let's illustrate the solution approach with a small example:

Suppose we are given the following inputs: `nums = [2, 6, 8]` and `k = 24`. We want to find all contiguous subarrays where the LCM of the numbers is exactly 24.

1. We start with the first element 2. The LCM of just [2] is 2, which is not equal to 24. So, we move on.
2. Next, we consider the subarray [2, 6]. The LCM of this set is 6 (since 6 is a multiple of 2 and it is the smallest one that is), which is still not equal to 24. We iterate further.
3. Now, we extend our subarray to [2, 6, 8]. The LCM of [2, 6, 8] is 24, which matches k. We increment our counter `ans` to 1.
4. Since we have reached the end of the array for this starting position, we reset to the next starting position in the array, which is element 6.
5. Starting at the element 6, the LCM of [6] is 6. Since this does not match 24, we continue.
6. We then look at [6, 8]. The LCM of this subarray is 24, which does match k. We increment `ans` to 2.
7. We've gone through all elements, so we move on to the starting element 8. However, the LCM of [8] is 8, which does not match 24.

After completing the iterations, we find that there are 2 subarrays whose LCM is 24: `[2, 6, 8]` and `[6, 8]`. Therefore, the final answer (`ans`) is 2.

Python Solution

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

class Solution:
    def subarrayLCM(self, nums: List[int], k: int) -> int:
        n = len(nums)  # Length of the nums list
        count = 0  # Initialize count of subarrays with LCM equals to k

        # Helper function to calculate LCM of two numbers
        def lcm(x, y):
            return (x * y) // gcd(x, y)

        # Iterate over each starting point of subarrays
        for i in range(n):
            # Initialize the LCM of the current subarray
            subarray_lcm = nums[i]
            # Extend the subarray to include subsequent elements
            for j in range(i, n):
                # Calculate the LCM of the current subarray
                subarray_lcm = lcm(subarray_lcm, nums[j])
                # Check if the current subarray LCM is equal to k
                if subarray_lcm == k:
                    count += 1
                # If LCM exceeds k, no need to continue extending the subarray
                if subarray_lcm > k:
                    break

        return count  # Return the count of valid subarrays

```

Java Solution

```java
class Solution {
    // This method finds the number of contiguous subarrays where the Least Common Multiple (LCM)
    // of their elements is equal to a given integer k
    public int subarrayLCM(int[] nums, int k) {
        int length = nums.length; // Length of the input array
        int count = 0; // To keep track of the number of subarrays with LCM equal to k

        // Iterate over the array to consider every possible subarray
        for (int i = 0; i < length; ++i) {
            int currentLCM = nums[i]; // Initialize the LCM with the first element of subarray
            // Consider all subarrays starting at index i
            for (int j = i; j < length; ++j) {
                int nextElement = nums[j]; // Next element in the subarray to consider
                currentLCM = lcm(currentLCM, nextElement); // Update the current LCM

                // If the current LCM is equal to k, increment the count
                if (currentLCM == k) {
                    ++count;
                }
            }
        }
        return count; // Return the total count of such subarrays
    }

    // This helper method calculates the Least Common Multiple of two integers a and b
    private int lcm(int a, int b) {
        return a * (b / gcd(a, b)); // LCM formula: a * b / GCD(a, b)
    }

    // This recursive helper method calculates the Greatest Common Divisor of two integers a and b
    private int gcd(int a, int b) {
        return b == 0 ? a : gcd(b, a % b); // GCD calculation using Euclidean algorithm
    }
}
```

C++ Solution

```cpp
#include <numeric> // Required for std::lcm function

class Solution {
public:
    // Function to count subarrays whose Least Common Multiple (LCM) is equal to `k`.
    int subarrayLCM(vector<int>& nums, int k) {
        int numElements = nums.size(); // Size of the input array
        int count = 0; // Variable to store the count of valid subarrays

        // Iterate over the array using two pointers to check every subarray's LCM
        for (int start = 0; start < numElements; ++start) {
            // Start with the first element of the subarray
            int currentLCM = nums[start];

            // Expand the subarray towards the right
            for (int end = start; end < numElements; ++end) {
                int nextElement = nums[end];

                // Calculate the LCM of the current subarray
                currentLCM = std::lcm(currentLCM, nextElement);

                // If the LCM is equal to `k`, increment the count
                if (currentLCM == k) {
                    ++count;
                }
            }
        }

        // Return the total count of subarrays whose LCM is `k`
        return count;
    }
};
```

Typescript Solution

```typescript
// Import the Math library for Least Common Multiple computation
import * as Math from "mathjs"

// Function to count subarrays whose Least Common Multiple (LCM) is equal to `k`
function subarrayLCM(nums: number[], k: number): number {
    let numElements = nums.length; // Size of the input array
    let count = 0; // Variable to store the count of valid subarrays

    // Iterate over the array using two pointers to check every subarray's LCM
    for (let start = 0; start < numElements; ++start) {
        // Start with the first element of the subarray
        let currentLCM = nums[start];

        // Expand the subarray towards the right
        for (let end = start; end < numElements; ++end) {
            let nextElement = nums[end];

            // Calculate the LCM of the current subarray
            // Note: mathjs library function lcm is used for LCM calculation
            currentLCM = Math.lcm(currentLCM, nextElement) as number;

            // If the LCM is equal to `k`, increment the count
            if (currentLCM === k) {
                count++;
            }
        }
    }

    // Return the total count of subarrays whose LCM is `k`
    return count;
}
```

Time and Space Complexity

The provided Python code calculates the number of contiguous subarrays in an array `nums` where the Least Common Multiple (LCM) of the subarray is equal to a given integer `k`. The code is a brute-force approach where it iterates through every possible subarray using nested loops.

Time Complexity

The time complexity of the code is determined by the nested loops and the computation of the LCM within the inner loop. There are `O(n)` possible starting points for a subarray (where `n` is the length of the array) and, for each start point, up to `O(n)` ending points. Therefore, we are considering `O(n^2)` contiguous subarrays.

Computing the LCM within the inner loop has a varying complexity depending on the implementation of the `lcm` function, which is not shown in the code snippet. Assuming the `lcm` function uses the Greatest Common Divisor (GCD) algorithm which, on average, has a time complexity of `O(log(min(a, b)))` where `a` and `b` are the input numbers, the overall time complexity of the inner loop is `O(log(min(a, b)))`.

However, since the computation of the LCM is done in a successive manner, where each computed LCM is then used with the next array element, in the worst case, the numbers can grow exponentially with respect to their indices. But this behavior is still bounded by the small input bounds due to integer limits in a practical scenario, and the average case would often be much better.

Given these nested loops and assuming the average complexity for LCM computations, the overall time complexity can be considered `O(n^2 * log(min(a, b)))`.

Space Complexity

The space complexity of the code is `O(1)` as there are only a few integer variables being used, and there is no additional space being allocated that grows with the input size. The variables `n`, `ans`, `a`, and `x` are used to track the length of the array, the result count, the running LCM of the current subarray, and the LCM of the current elements respectively.