Prime Subtraction Operation

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	2601
tags	Primes
language	C++
solved on	@11/11/2024
link	https://leetcode.com/problems/prime-subtraction-operation/description/
Completion	☑

Intuition

The problem involves modifying elements in the `nums` array such that each element is smaller than the next one by subtracting a prime number from it, if needed. By ensuring each element satisfies this condition, we can verify if the sequence can become strictly increasing using the smallest necessary adjustments.

Approach

1. **Generate Prime Numbers**: We first generate all prime numbers up to a reasonable upper limit (1000 in this case) using the Sieve of Eratosthenes. This list will help us perform efficient checks and subtractions for each element in `nums`.
2. **Iterate Backwards and Adjust**: Starting from the second-to-last element, we check if each element is less than the next one. If it isn't, we try to subtract the smallest prime such that the current element becomes strictly less than the next element. If no prime can satisfy this condition, we return `false`.
3. **Return Result**: If we successfully adjust all elements to satisfy the strictly increasing property, we return `true`.

Complexity

Time Complexity:

- **Prime Generation**: **O(**$\text{upperLimit} \cdot \log \log \text{upperLimit}$**)** for the Sieve of Eratosthenes.
- **Main Loop**: **O(nm)**, where **n** is the size of `nums` and **m** is the number of primes. In the worst case, we might need to try multiple primes for each element in `nums`.

The overall time complexity is roughly **O(nm)**, where **n** is the size of `nums` and **m** is small due to the upper limit on primes.

Space Complexity:

- **Auxiliary Space**: **O(**upperLimit**)** for the `primes` vector used to store all prime numbers up to `upperLimit`.

So, the space complexity is **O(**upperLimit**)**.

Code

```
class Solution {
    // Function to generate all primes up to upperLimit using Sieve of Eratosthenes
    vector<int> getPrimes(int upperLimit) {
        vector<bool> primes(upperLimit + 1, true);
        vector<int> prime;

        for (int i = 2; i <= upperLimit; i++) {
            if (primes[i]) {
                prime.push_back(i);
                for (int j = 2 * i; j <= upperLimit; j += i)
                    primes[j] = false;
            }
        }
        return prime;
    }

public:
    bool primeSubOperation(vector<int>& nums) {
        vector<int> primes = getPrimes(1000);

        for (int i = nums.size() - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) continue;

            bool updated = false;
            for (int j = 0; j < primes.size() && primes[j] < nums[i]; j++) {
                if (nums[i] - primes[j] < nums[i + 1]) {
                    nums[i] -= primes[j];
                    updated = true;
                    break;
                }
            }
            if (!updated) return false;
        }
        return true;
    }
};
```