1590. Make Sum Divisible by P

Solved

Medium | Topics | Companies | Hint

Given an array of positive integers `nums`, remove the **smallest** subarray (possibly **empty**) such that the **sum** of the remaining elements is divisible by `p`. It is **not** allowed to remove the whole array.

Return *the length of the smallest subarray that you need to remove, or* `-1` *if it's impossible.*

A **subarray** is defined as a contiguous block of elements in the array.

Example 1:

```
Input: nums = [3,1,4,2], p = 6
Output: 1
Explanation: The sum of the elements in nums is 10, which is not
divisible by 6. We can remove the subarray [4], and the sum of the
remaining elements is 6, which is divisible by 6.
```

Example 2:

```
Input: nums = [6,3,5,2], p = 9
Output: 2
Explanation: We cannot remove a single element to get a sum divisible by
9. The best way is to remove the subarray [5,2], leaving us with [6,3]
with sum 9.
```

Example 3:

```
Input: nums = [1,2,3], p = 3
Output: 0
Explanation: Here the sum is 6. which is already divisible by 3. Thus we
do not need to remove anything.
```

Constraints:

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^9`
- `1 <= p <= 10^9`

Python:

```python
class Solution:
    def minSubarray(self, nums: List[int], p: int) -> int:
        totalSum = sum(nums)
        rem = totalSum % p

        if rem == 0:
            return 0

        prefixMod = {0: -1}
        prefixSum = 0
        minLength = len(nums)

        for i, num in enumerate(nums):
            prefixSum += num
            currentMod = prefixSum % p
            targetMod = (currentMod - rem + p) % p

            if targetMod in prefixMod:
                minLength = min(minLength, i - prefixMod[targetMod])

            prefixMod[currentMod] = i

        return minLength if minLength < len(nums) else -1
```

JavaScript:

```javascript
/**
 * @param {number[]} nums
 * @param {number} p
 * @return {number}
 */
var minSubarray = function(nums, p) {
    let totalSum = nums.reduce((a, b) => a + b, 0);

    // Find the remainder when total sum is divided by p
    let rem = totalSum % p;
    if (rem === 0) return 0;  // If the remainder is 0, no subarray needs to be removed

    let prefixMod = new Map();
    prefixMod.set(0, -1);  // Initialize to handle full prefix
    let prefixSum = 0;
    let minLength = nums.length;

    for (let i = 0; i < nums.length; i++) {
```

```
        prefixSum += nums[i];
        let currentMod = prefixSum % p;
        let targetMod = (currentMod - rem + p) % p;

        if (prefixMod.has(targetMod)) {
            minLength = Math.min(minLength, i - prefixMod.get(targetMod));
        }

        prefixMod.set(currentMod, i);
    }

    return minLength === nums.length ? -1 : minLength;
};
```

Java:

```
import java.util.HashMap;

class Solution {
    public int minSubarray(int[] nums, int p) {
        long totalSum = 0;
        for (int num : nums) {
            totalSum += num;
        }

        // Find remainder when total sum is divided by p
        int rem = (int)(totalSum % p);
        if (rem == 0) return 0; // If remainder is 0, no subarray needs to be removed

        HashMap<Integer, Integer> prefixMod = new HashMap<>();
        prefixMod.put(0, -1);  // Initialize to handle full prefix
        long prefixSum = 0;
        int minLength = nums.length;

        for (int i = 0; i < nums.length; ++i) {
            prefixSum += nums[i];
            int currentMod = (int)(prefixSum % p);
            int targetMod = (currentMod - rem + p) % p;

            if (prefixMod.containsKey(targetMod)) {
                minLength = Math.min(minLength, i - prefixMod.get(targetMod));
            }

            prefixMod.put(currentMod, i);
        }
```

```
        return minLength == nums.length ? -1 : minLength;
    }
}
```