Make Sum Divisible By P

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	1590
tags	Hash Map Prefix Sum Vector
language	C++
solved on	@03/10/2024
link	https://leetcode.com/problems/make-sum-divisible-by-p/
Completion	☑

Intuition

The goal is to find the shortest subarray whose sum, when removed, makes the sum of the remaining elements divisible by `p`. This can be achieved using the **prefix sum** and **modulo arithmetic** concepts. By keeping track of the cumulative sums modulo `p`, we can find subarrays that satisfy the condition efficiently.

Approach

1. **Compute total sum modulo `p`:** First, calculate the total sum of the array modulo `p`. If the sum is already divisible by `p`, return `0`.
2. **Prefix sum and map:** Use a prefix sum approach with a hash map (`unordered_map`) to track the indices where each prefix modulo `p` appears.
3. **Target subarray sum:** For each element in the array, compute the necessary remainder to adjust the current prefix sum so that removing this subarray will make the rest of the array sum divisible by `p`. Check if this target exists in the map, and if so, update the minimum length.
4. **Final check:** If a valid subarray is found, return its length. Otherwise, return `1`.

Complexity

Time Complexity:

- **First loop (total sum calculation):** This runs in O(n) since it iterates through the entire array.
- **Second loop (prefix sum processing):** This also runs in O(n), iterating through the array once while performing constant-time operations like modulo calculation and hash map lookups.

Hence, the overall time complexity is: $O(n)$

Space Complexity:

The space complexity is dominated by the hash map (`unordered_map`), which at most stores `n` entries (one for each prefix sum). Therefore, the space complexity is: $O(n)$

Code

```
class Solution {
public:
    int minSubarray(vector<int>& nums, int p) {
        int n = nums.size();
        int total = 0;
        for(auto num: nums) total = (total + num) % p;
        if(total == 0) return 0;

        unordered_map<int, int> modulo;
        modulo[0] = -1;

        int minLength = n;
        int prefix = 0;
        int search;

        // Iterate through the array to compute prefix sums and search for target suba
rrays
        for(int i = 0; i < n; i++){
            prefix = (prefix + nums[i]) % p;
            search = (prefix - total + p) % p;

            if(modulo.count(search)) {
                minLength = min(minLength, i - modulo[search]);

            }
            modulo[prefix] = i;
        }

        return (minLength == n) ? -1 : minLength;
    }
};
```