Equilibrium Index

Author:	Prakash JC

Problem Statement

You are given an array of integers `arr[]` of size `n`. Your task is to **find the first equilibrium index** in the array.

An **equilibrium index** is an index such that the **sum of all elements before it is equal to the sum of all elements after it**. The element at the index itself is **not** included in either sum.

Return the **first such index (0-based)** if it exists. If no such index exists, return `-1`.

Examples:

Input: arr[] = [1, 2, 0, 3]
Output: 2
Explanation: The sum of left of index 2 is 1 + 2 = 3 and sum on right of index 2 is 3.

Input: arr[] = [1, 1, 1, 1]
Output: -1
Explanation: There is no equilibrium index in the array.

Input: arr[] = [-7, 1, 5, 2, -4, 3, 0]
Output: 3
Explanation: The sum of left of index 3 is -7 + 1 + 5 = -1 and sum on right of index 3 is -4 + 3 + 0 = -1.

Constraints:

3 <= arr.size() <= 10^5

-10^4 <= arr[i] <= 10^4

Editorial

Brute force Approach - Nested Loop

Approach

- For every index in the array, calculate left sum and right sum.
- If left sum = right sum, return that index.

Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int findEquilibrium(vector<int>& arr) {
```

```
        for (int i = 0; i < arr.size(); ++i) {
            int leftSum = 0;
            for (int j = 0; j < i; j++)
                leftSum += arr[j];

            int rightSum = 0;
            for (int j = i + 1; j < arr.size(); j++)
                rightSum += arr[j];

            if (leftSum == rightSum)
                return i;
        }
        return -1;
    }
};

int main() {
    vector<int> arr = {-7, 1, 5, 2, -4, 3, 0};
    Solution s1;
    cout << s1.findEquilibrium(arr);
    return 0;
}
```

Complexity Analysis

Time Complexity: `O(n^2)` as we are using two nested loops.

Space Complexity: `O(1)` as we are using only a few integer variables.

Better Approach - Prefix Sum and Suffix Sum Array

Approach

For every index,

- Calculate Prefix sum(sum of all the elements till that index) and Suffix sum.
- If `prefixSum[i - 1] = suffixSum[i + 1]` return `i`.
- Note: `prefixSum[i - 1] = suffixSum[i + 1]` can also be written as `prefixSum[i] = suffixSum[i]` . This avoids unnecessary boundary checks.

Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int findEquilibrium(vector<int>& arr) {
        int n = arr.size();
        vector<int> pref(n, 0);
        vector<int> suff(n, 0);
```

```
        pref[0] = arr[0];
        suff[n - 1] = arr[n - 1];

        for (int i = 1; i < n; i++)
            pref[i] = pref[i - 1] + arr[i];

        for (int i = n - 2; i >= 0; i--)
            suff[i] = suff[i + 1] + arr[i];

        for (int i = 0; i < n; i++) {
            if (pref[i] == suff[i])
                return i;
        }

        return -1;
    }
};

int main() {
    vector<int> arr = {-7, 1, 5, 2, -4, 3, 0};
    Solution s1;
    cout << s1.findEquilibrium(arr) << "\n";
    return 0;
}
```

Complexity Analysis

Time Complexity: `O(n)` as we are using linear loops.

Space Complexity: `O(n)` as we are using two additional arrays.

Expected Approach - Running Prefix Sum and Suffix Sum

Approach

Remove extra arrays for prefix sum and suffix sum. Instead, use two integer variables `prefixSum` and `suffixSum` that stores the current prefix and suffix sums.

And, Total Sum = Prefix Sum + Present index (pivot) + Suffix Sum.

For every index:

- Calculate Prefix Sum and Suffix Sum (Total - Prefix sum - Present index).
- If Prefix Sum = Suffix Sum, return present index.

Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
public:
    int equilibriumPoint(vector<int>& arr) {
        int prefSum = 0, total = 0;
```

```
        for (int ele : arr) {
            total += ele;
        }
        for (int pivot = 0; pivot < arr.size(); pivot++) {
            int suffSum = total - prefSum - arr[pivot];
            if (prefSum == suffSum)
                return pivot;
            prefSum += arr[pivot];
        }
        return -1;
    }
};

int main() {
    vector<int> arr = {1, 7, 3, 6, 5, 6};
    Solution s1;
    cout << s1.equilibriumPoint(arr) << endl;
    return 0;
}
```

Complexity Analysis

Time Complexity: `O(n)` as we are using linear loops.

Space Complexity: `O(1)` as we are using only a few integer variables.

THE END