3.Hard\05.Largest_subarray_with_0sum.cpp

```cpp
/*
QUESTION:
Given an array with both positive and negative integers, we need to compute the length of the largest subarray with a sum of 0.

Example:
Input:
N = 8
A[] = {15, -2, 2, -8, 1, 7, 10, 23}
Output: 5
Explanation: The largest subarray with a sum of 0 will be -2, 2, -8, 1, 7.

APPROACH:
To find the length of the largest subarray with a sum of 0, we can use a technique called prefix sum.
1. Create a prefix sum array of the same size as the input array.
2. Initialize a map to store the prefix sum and its corresponding index. Initialize it with an entry for prefix sum 0 and index -1.
3. Iterate through the input array and calculate the prefix sum by adding each element.
4. For each prefix sum encountered, check if it exists in the map. If it does, update the answer by taking the maximum of the current answer and the difference between the current index and the index stored in the map for that prefix sum.
5. If the prefix sum is not found in the map, add it to the map with its corresponding index.
6. Finally, return the answer as the length of the largest subarray with a sum of 0.

CODE:
*/

int maxLen(vector<int> &A, int n)
{
    unordered_map<int, int> mp;
    mp[0] = -1;
    int pref_sum = 0;
    int ans = 0;

    for (int i = 0; i < n; i++)
    {
        pref_sum += A[i];
        if (mp.find(pref_sum) != mp.end())
        {
            ans = max(ans, i - mp[pref_sum]);
        }
        else
        {
            mp[pref_sum] = i;
        }
    }

    return ans;
}

/*
```

```
TIME COMPLEXITY: O(n), where n is the size of the input array A.
SPACE COMPLEXITY: O(n), as we are using a map to store the prefix sums and their
corresponding indices.
*/

```