Equivalent Subarrays in C++

```
#include <iostream>
#include <unordered_map>
#include <unordered_set>
#include <vector>

using namespace std;

int main() {
    int ans = 0;
    vector<int> arr = {2, 1, 3, 2, 3};
    unordered_set<int> set;

    // Insert unique elements into the set
    for (int i = 0; i < arr.size(); i++) {
        set.insert(arr[i]);
    }

    int k = set.size();
    int i = -1;
    int j = -1;
    unordered_map<int, int> map;

    while (true) {
        bool f1 = false;
        bool f2 = false;

        // Expand the window until all unique elements
are covered
        while (i < arr.size() - 1) {
            f1 = true;
            i++;
            map[arr[i]] = map[arr[i]] + 1; // Add current
element to the map
            if (map.size() == k) { // If all unique elements
are covered
                ans += arr.size() - i; // Add the number of
valid subarrays ending at index i
                break;
            }
        }

        // Slide the window to the right until the
uniqueness condition is violated
        while (j < i) {
            f2 = true;
            j++;
            if (map[arr[j]] == 1) {
                map.erase(arr[j]); // Remove element from
map if its count is reduced to 0
            } else {
                map[arr[j]] = map[arr[j]] - 1; // Decrease the
count of the element
            }

            // If the map size matches k, add the number of
valid subarrays again
            if (map.size() == k) {
                ans += arr.size() - i;
            } else {
                break;
            }
```

Input:

arr = {2, 1, 3, 2, 3}

We are looking for subarrays with **all unique elements** in the array.

Initial Setup:

- arr = {2, 1, 3, 2, 3}
- We initialize an unordered_set called set to store the unique elements of the array.
- We calculate k = set.size(), which is the number of unique elements in the array.
    - set = {2, 1, 3} -> k = 3 (3 unique elements).

Algorithm Steps:

1. **Initialization:**
    - i = -1, j = -1 (start indices for the sliding window).
    - map = {} (tracks the frequency of elements in the current window).
    - ans = 0 (tracks the number of valid subarrays).
2. **Start the outer while (true) loop:**
    - The outer loop keeps expanding and shrinking the window until we can no longer process subarrays.

First pass through the outer loop:

- **Expand the window (moving i):**
    - Initially i = -1, we move i to 0 (i.e., arr[i] = 2).
    - Add arr[i] = 2 to the map:
        - map = {2: 1}
    - Now, i = 0, and map.size() = 1 (we have only one unique element, so we move on).
    - Move i to 1 (i.e., arr[i] = 1).
    - Add arr[i] = 1 to the map:
        - map = {2: 1, 1: 1}
    - Now, i = 1, and map.size() = 2 (we still have not covered all unique elements, so continue expanding).
    - Move i to 2 (i.e., arr[i] = 3).
    - Add arr[i] = 3 to the map:
        - map = {2: 1, 1: 1, 3: 1}
    - Now, i = 2, and map.size() = 3 (we have all 3 unique elements).
    - At this point, we have a valid subarray from arr[0] to arr[2]. All unique elements are covered.
    - **Count valid subarrays:**
        - Subarrays ending at i = 2:
            - The number of subarrays is

```
        }

        // If both windows cannot be expanded or
contracted further, break the loop
        if (!f1 && !f2) {
            break;
        }
    }

    // Print the total number of equivalent subarrays
    cout << ans << endl;

    return 0;
}
```

calculated as arr.size() - i = 5 - 2 = 3.
- ans += 3 → ans = 3.

- **Shrink the window (moving j):**
  - Now we start shrinking the window by moving the left pointer (j).
  - Move j to 0 (i.e., arr[j] = 2):
    - Since map[arr[j]] = 1, we remove arr[j] from the map:
      - map = {1: 1, 3: 1}
  - Move j to 1 (i.e., arr[j] = 1):
    - Since map[arr[j]] = 1, we remove arr[j] from the map:
      - map = {3: 1}
  - Now, map.size() = 1, which is less than k. We stop shrinking.

Second pass through the outer loop:

- Now, we repeat the steps again by expanding and shrinking the window, but the map.size() will no longer match k (the number of unique elements).

Output:-
0