|  |  |
| --- | --- |
| **Largest Subarray with 0sum in C++** | |
| #include<bits/stdc++.h>  using namespace std;  int largest2(vector<int> arr, int n) {  int max\_len = 0;  for (int i = 0; i < n; i++) {  int sum = 0;  for (int j = i; j < n; j++) {  sum += arr[j];  if (sum == 0) {  max\_len = max(max\_len, j - i + 1);  }  }  }  return max\_len;  }  int largest3(vector<int> arr, int n) {  map<int, int> mapp;  mapp[0]=-1;  int sum=0;  int ans=0;  for (int i = 0; i < n; i++)  {  sum+=arr[i];  if(mapp.find(sum)!=mapp.end()){  auto it=mapp[sum];  ans=max(ans,i- it);  }  else{  mapp[sum]=i;  }  }  return ans;  }  int largestSubarrayWithZeroSum(vector<int>& arr) {  unordered\_map<int, int> hm; // Maps sum to index  int sum = 0;  int max\_len = 0;    hm[0] = -1; // Initialize to handle the case where sum becomes 0 at the start    for (int i = 0; i < arr.size(); i++) {  sum += arr[i];    if (hm.find(sum) != hm.end()) {  int len = i - hm[sum];  if (len > max\_len) {  max\_len = len;  }  } else {  hm[sum] = i;  }  }    return max\_len;  }  int main() {  vector<int> arr = {2, 8, -3, -5, 2, -4, 6, 1, 2, 1, -3, 4};  int max\_length = largestSubarrayWithZeroSum(arr);  cout << max\_length << endl; // Output: 5  int n=arr.size();  int res=largest2(arr,n);  cout<<res<<endl;  int res3=largest3(arr,n);  cout<<res3<<endl;    return 0;  } | **Step 1: Understanding the Problem**  * We need to find the **largest subarray with sum = 0**. * The input array is:   {2, 8, -3, -5, 2, -4, 6, 1, 2, 1, -3, 4}   * The program runs **three different implementations** for this:   1. **largestSubarrayWithZeroSum()** → Optimized using unordered\_map.   2. **largest2()** → Brute-force approach.   3. **largest3()** → Using map.  **Step 2: Dry Run for** largestSubarrayWithZeroSum() **(Optimized Hashing Approach)**  | **Index (i)** | **arr[i]** | **Sum** | **hm (Sum → Index)** | **Max Length (max\_len)** | | --- | --- | --- | --- | --- | | 0 | 2 | 2 | {0:-1, 2:0} | 0 | | 1 | 8 | 10 | {0:-1, 2:0, 10:1} | 0 | | 2 | -3 | 7 | {0:-1, 2:0, 10:1, 7:2} | 0 | | 3 | -5 | 2 | **Found 2 at index 0 → 3 - 0 = 3** | **3** | | 4 | 2 | 4 | {0:-1, 2:0, 10:1, 7:2, 4:4} | 3 | | 5 | -4 | 0 | **Found 0 at index -1 → 5 - (-1) = 6** | **6** | | 6 | 6 | 6 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6} | 6 | | 7 | 1 | 7 | **Found 7 at index 2 → 7 - 2 = 5** | 6 | | 8 | 2 | 9 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8} | 6 | | 9 | 1 | 10 | **Found 10 at index 1 → 9 - 1 = 8** | **8** | | 10 | -3 | 7 | **Found 7 at index 2 → 10 - 2 = 8** | 8 | | 11 | 4 | 11 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8, 11:11} | 8 |  ****Final Output of**** largestSubarrayWithZeroSum() ****→**** 8**Step 3: Dry Run for** largest2() **(Brute-force approach)**  * **Time Complexity:** O(N²) → Iterates over all possible subarrays. * Iterates over each possible subarray and calculates its sum.  | **i** | **j** | **Subarray** | **Sum** | **Max Length (max\_len)** | | --- | --- | --- | --- | --- | | 0 | 1 | {2, 8} | 10 | 0 | | 0 | 2 | {2, 8, -3} | 7 | 0 | | 0 | 3 | {2, 8, -3, -5} | 2 | 0 | | 0 | 5 | {2, 8, -3, -5, 2, -4} | **0** | **6** | | 1 | 5 | {8, -3, -5, 2, -4} | **0** | **6** | | 3 | 9 | { -5, 2, -4, 6, 1, 2, 1 } | **0** | **7** | | 1 | 9 | { 8, -3, -5, 2, -4, 6, 1, 2, 1 } | **0** | **8** |  ****Final Output of**** largest2() ****→**** 8**Step 4: Dry Run for** largest3() **(Map-based approach)**  * Similar to largestSubarrayWithZeroSum(), but uses map<int, int> instead of unordered\_map<int, int>.  | **Index (i)** | **arr[i]** | **Sum** | **mapp (Sum → Index)** | **Max Length (ans)** | | --- | --- | --- | --- | --- | | 0 | 2 | 2 | {0:-1, 2:0} | 0 | | 1 | 8 | 10 | {0:-1, 2:0, 10:1} | 0 | | 2 | -3 | 7 | {0:-1, 2:0, 10:1, 7:2} | 0 | | 3 | -5 | 2 | **Found 2 at index 0 → 3 - 0 = 3** | **3** | | 4 | 2 | 4 | {0:-1, 2:0, 10:1, 7:2, 4:4} | 3 | | 5 | -4 | 0 | **Found 0 at index -1 → 5 - (-1) = 6** | **6** | | 6 | 6 | 6 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6} | 6 | | 7 | 1 | 7 | **Found 7 at index 2 → 7 - 2 = 5** | 6 | | 8 | 2 | 9 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8} | 6 | | 9 | 1 | 10 | **Found 10 at index 1 → 9 - 1 = 8** | **8** | | 10 | -3 | 7 | **Found 7 at index 2 → 10 - 2 = 8** | 8 | | 11 | 4 | 11 | {0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8, 11:11} | 8 |  ****Final Output of**** largest3() ****→**** 8**Final Outputs**  | **Function** | **Approach** | **Output** | | --- | --- | --- | | largestSubarrayWithZeroSum() | Hashing (unordered\_map) | 8 | | largest2() | Brute-force (O(N²)) | 8 | | largest3() | Hashing (map) | 8 | |
| Output: 8  8  8 | |