🔷 C++: unordered_map

Header:

#include <unordered_map>

✅ Declaration:

unordered_map<int, int> mp; // map from int to int

unordered_map<string, int> freq; // map from string to int

✅ Common Operations:

mp[key] = value; // insert or update

int val = mp[key]; // access value (default 0 if key not present)

mp[key]++; // increment frequency

mp.erase(key); // delete key

mp.count(key); // returns 1 if key exists, else 0

mp.find(key) != mp.end() // check existence

mp.clear(); // remove all

HashMap<Integer, Integer> hashMap = **new** HashMap<>();

In java

🔶 Java: HashMap

Import:

import java.util.HashMap;

✅ Declaration:

HashMap<Integer, Integer> map = new HashMap<>();

HashMap<String, Integer> freq = new HashMap<>();

✅ Common Operations:

map.put(key, value); // insert or update

int val = map.getOrDefault(key, 0); // access with default

map.put(key, map.getOrDefault(key, 0) + 1); // increment frequency

map.remove(key); // delete key

map.containsKey(key); // check if key exists

map.clear(); // remove all

✅ **Example IN JAVA**

```
import java.util.HashMap;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();

        int array[]  = new int[n];
        HashMap<Integer, Integer> hashMap = new HashMap<>();
        for (int i = 0; i < n; i++) {
            array[i] = scanner.nextInt();
            int g = hashMap.getOrDefault(array[i], 0);
            hashMap.put(array[i], g  + 1);
        }


        int q = scanner.nextInt();
        for (int i = 0; i < q; i++) {
            int query = scanner.nextInt();
            int count = hashMap.getOrDefault(query, 0);
            System.out.println(count);
        }
    }
}
// int max and int min in java
int maxiFreq = Integer.MIN_VALUE
int miniFreq = Integer.MAX_VALUE
```

Summary Table

Operation	C++ (`unordered_map`)	Java (`HashMap`)
Insert/Update	`mp[key] = val;`	`map.put(key, val);`
Access with default	`mp[key]` (default = 0)	`map.getOrDefault(key, 0)`
Check if key exists	`mp.count(key)` or `find()`	`map.containsKey(key)`
Delete key	`mp.erase(key);`	`map.remove(key);`
Iterate	`for (auto &p : mp)`	`for (String k : map.keySet())`

```
for (Map.Entry<Integer, Integer> num : mp.entrySet()){
if (num.getValue() >= maxiFreq){
maxiFreq = num.getValue();
maxiElement = num.getKey();
}
if (num.getValue() <= miniFreq){
miniFreq = num.getValue();
miniElement = num.getKey();
}
}
```

Q) Check if there are any two Equal numbers in an array at a distance less than or equal to k

IN C++:

```
bool containsNearbyDuplicateOptimized(const std::vector<int>& nums, int k) {

unordered_map<int, int> numIndices;

for (int i = 0; i < nums.size(); ++i) {
if (numIndices.find(nums[i]) != numIndices.end() && i - numIndices[nums[i]] <= k) {
return true;
```

```
    }

    numIndices[nums[i]] = i;

    }
    return false;
    }
```

IN JAVA:

```
public static boolean containsNearbyDuplicateOptimized(int[] nums, int k) {
      HashMap<Integer, Integer> numIndices = new HashMap<>();
      for (int i = 0; i < nums.length; ++i) {
         if (numIndices.containsKey(nums[i]) && i - numIndices.get(nums[i]) <= k) {
            return true;
         }
         numIndices.put(nums[i], i);
      }
      return false;
   }
```

Q) Count All ((i,j) pairs such that b[i] - b[j] == k (count of such pairs.) [i<j]

C++:

```
    int countPairsOptimized(const std::vector<int>& b, int k) {
    int count = 0;
    std::unordered_map<int, int> freqMap;
    for (int j = 0; j < b.size(); ++j) {
    int target = b[j] + k;
    if (freqMap.find(target) != freqMap.end()) {
    count += freqMap[target];
    }
    freqMap[b[j]]++;
    }
```

```
return count;
}
```

Q) Count pairs with absolute difference equal to k (LEETCODE)

***

[Expected Approach] Using Hash Map or Dictionary – O(n) Time and O(n) Space

*The idea is to count the frequency of each number in a hash map or dictionary as we go through the array Iterate over the array and for each element **arr[i]**, we need another element say **complement** such that **abs(arr[i] - complement) = k.** Now, we can have two cases:*

1. ***(arr[i] - complement) is positive:***
   - *arr[i] - complement = k*
   - *So, **complement = arr[i] - k***
2. ***(arr[i] - complement) is negative:***
   - *(arr[i] - complement) = -k*
   - *So, **complement = arr[i] + k***

C++:

```
int countKDifference(vector<int>& nums, int k) {
      int count=0;
      unordered_map<int,int> mp;
    for(int i=0;i<nums.size();i++){
        if(mp.find(nums[i]+k)!=mp.end()){
            count+=mp[nums[i]+k];
        }
        if(mp.find(nums[i]-k)!=mp.end()){
            count+=mp[nums[i]-k];
        }
        mp[nums[i]]++;
    }
    return count;
  }
```

JAVA:

```
public int countKDifference(int[] nums, int k) {
        int count=0;
        HashMap<Integer,Integer> mp=new HashMap<>();
      for(int i=0;i<nums.length;i++){
           if(mp.containsKey(nums[i]+k)){
               count+=mp.get(nums[i]+k);
           }
           if(mp.containsKey(nums[i]-k)){
               count+=mp.get(nums[i]-k);
           }
           mp.put(nums[i],mp.getOrDefault(nums[i],0)+1);
      }
      return count;
    }
```