Hashtable

1. **Fast Data Access**:
    - You need to store and retrieve data in **constant average time complexity**, i.e., **O(1)** for operations like insert, delete, and search.
2. **Mapping Keys to Values**:
    - When there is a need to store **key-value pairs** and perform lookups based on keys, such as in dictionaries, caches, or configurations.
3. **Handling Large Data Efficiently**:
    - Hash tables are effective for managing **large datasets** where quick lookups are essential.
4. **Counting Frequencies**:
    - To count occurrences of elements (e.g., in a string or array) or to track element uniqueness.

Example: Counting character frequencies in a string.

5. **Avoiding Duplicates**:
    - To check for **duplicate elements** in a collection efficiently.
6. **Set-Like Behavior**:
    - When you need the behavior of a **set** to check if an element is present, without maintaining a specific order.
7. **Applications in Algorithms**:
    - Hash tables are widely used in algorithmic problems like:
        - Detecting **anagrams**.
        - Finding **two numbers that add up to a target sum**.
        - Solving problems related to **prefix sums** or **subarrays**.
        - Implementing **LRU Cache**.
8. **Custom Key Lookups**:
    - When you need to index elements by a property rather than their position, e.g., searching for a student by ID rather than array index.

Hashset in c++

```
#include<iostream>
#include <unordered_set>              // 0. include the library
using namespace std;
int main() {
   // 1. initialize a hash set
   unordered_set<int> hashset;
   // 2. insert a new key
   hashset.insert(3);
   hashset.insert(2);
   hashset.insert(1);
   // 3. delete a key
   hashset.erase(2);
   // 4. check if the key is in the hash set
   if (hashset.count(2) <= 0) {
      cout << "Key 2 is not in the hash set." << endl;
   }
   // 5. get the size of the hash set
   cout << "The size of hash set is: " << hashset.size() << endl;
   // 6. iterate the hash set
   for (auto it = hashset.begin(); it != hashset.end(); ++it) {
      cout << (*it) << " ";
   }
   cout << "are in the hash set." << endl;
   // 7. clear the hash set
   hashset.clear();
   // 8. check if the hash set is empty
   if (hashset.empty()) {
      cout << "hash set is empty now!" << endl;
   }
}
```

In java

```
// "static void main" must be defined in a public class.
import java.util.HashSet; // Import the HashSet class
import java.util.Set;     // Import the Set interface
class Main {
   public static void main(String[] args) {
      // 1. initialize the hash set
      Set<Integer> hashSet = new HashSet<>();
      // 2. add a new key
      hashSet.add(3);
      hashSet.add(2);
      hashSet.add(1);
      // 3. remove the key
      hashSet.remove(2);
      // 4. check if the key is in the hash set
```

```
        if (!hashSet.contains(2)) {
            System.out.println("Key 2 is not in the hash set.");
        }
        // 5. get the size of the hash set
        System.out.println("The size of has set is: " + hashSet.size());
        // 6. iterate the hash set
        for (Integer i : hashSet) {
            System.out.print(i + " ");
        }
        System.out.println("are in the hash set.");
        // 7. clear the hash set
        hashSet.clear();
        // 8. check if the hash set is empty
        if (hashSet.isEmpty()) {
            System.out.println("hash set is empty now!");
        }
    }
}
```

Find Duplicates By Hash Set

Given an array of integers, find if the array contains any duplicates.

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

bool findDuplicates(vector<int>& keys) {
    unordered_set<int> hashset;
    for (int key : keys) {
        if (hashset.count(key) > 0) {
            return true;
        }
        hashset.insert(key);
    }
    return false;
}

int main() {
    vector<int> keys = {1, 2, 3, 4, 5, 2}; // Example input with duplicate
    if (findDuplicates(keys)) {
        cout << "Duplicates found!" << endl;
    } else {
        cout << "No duplicates found!" << endl;
    }

    return 0;
}
```

Java

```
import java.util.*;

public class Main {
   public static <Type> boolean findDuplicates(List<Type> keys) {
      Set<Type> hashset = new HashSet<>();
      for (Type key : keys) {
         if (hashset.contains(key)) {
            return true;
         }
         hashset.add(key);
      }
      return false;
   }

   public static void main(String[] args) {
      List<Integer> keys = Arrays.asList(1, 2, 3, 4, 5, 3); // Example input
      if (findDuplicates(keys)) {
         System.out.println("Duplicates found.");
      } else {
         System.out.println("No duplicates found.");
      }
   }
}
```

Q **217. Contains Duplicate**

Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true
Explanation:
The element 1 occurs at the indices 0 and 3.
Example 2:
Input: nums = [1,2,3,4]
Output: false
Explanation:
All elements are distinct.
Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

```
bool containsDuplicate(vector<int>& nums) {
      unordered_set<int> s(nums.begin(), nums.end());
      return s.size() < nums.size();
}
```

Java

```
public boolean containsDuplicate(int[] nums) {
      Set<Integer> set = new HashSet<>();
      for (int num : nums) {
         if (set.contains(num)) {
            return true;
         }
         set.add(num);
      }
      return false;
   }
```

[349. Intersection of Two Arrays](#)

Given two integer arrays nums1 and nums2, return *an array of their intersection*
. Each element in the result must be **unique** and you may return the result in **any order**.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Explanation: [4,9] is also accepted.

```
 vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
    unordered_set<int> set1(nums1.begin(), nums1.end());
   unordered_set<int> resultSet;
   for (int num : nums2) {
      if (set1.count(num)) {
         resultSet.insert(num);
      }
   }
   return vector<int>(resultSet.begin(), resultSet.end());
   }
```

Java

```
public int[] intersection(int[] nums1, int[] nums2) {
    Set<Integer> set1 = new HashSet<>();
   for (int num : nums1) {
      set1.add(num);
   }
   Set<Integer> resultSet = new HashSet<>();
   for (int num : nums2) {
```

```
    if (set1.contains(num)) {
        resultSet.add(num);
    }
  }
  int[] result = new int[resultSet.size()];
  int index = 0;
  for (int num : resultSet) {
    result[index++] = num;
  }
  return result;
  }
```

202. Happy Number

```
bool isHappy(int n) {
    unordered_set<int> seen;

  while (n != 1 && seen.find(n) == seen.end()) {
    seen.insert(n);
    int sum = 0;
    while (n > 0) {
      int digit = n % 10;
      sum += digit * digit;
      n /= 10;
    }
    n = sum;
  }

  return n == 1;
  }
```

Java

```
 public boolean isHappy(int n) {
  Set<Integer> seen = new HashSet<>();

  while (n != 1 && !seen.contains(n)) {
    seen.add(n);
    int sum = 0;
    while (n > 0) {
      int digit = n % 10;
      sum += digit * digit;
      n /= 10;
    }
    n = sum;
  }

  return n == 1;
  }
```

1. Two Sum

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.
You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.
You can return the answer in any order.

Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]
Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]

```
 vector<int> twoSum(vector<int>& nums, int target) {
     unordered_map<int, int> map;

   for(int i = 0; i < nums.size(); i++) {
      int complement = target - nums[i];

      if(map.find(complement) != map.end()) {
         return {map[complement], i};
      }
      map[nums[i]] = i;
   }
   return {};
}
```

Java

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

   for(int i = 0; i < nums.length; i++) {
      int complement = target - nums[i];

      if(map.containsKey(complement)) {
         return new int[] {map.get(complement), i};
      }
      map.put(nums[i], i);
   }
   return new int[]{};
}
```

205. Isomorphic Strings

```
bool isIsomorphic(string s, string t) {
   vector<int> mapS(256, -1);
   vector<int> mapT(256, -1);

   for(int i = 0; i < s.length(); i++) {
      if(mapS[s[i]] != mapT[t[i]]) {
         return false;
      }
      mapS[s[i]] = i;
      mapT[t[i]] = i;
   }
   return true;
}
```

Java

```
 public boolean isIsomorphic(String s, String t) {
   int[] mapS = new int[256];
   int[] mapT = new int[256];

   for(int i = 0; i < s.length(); i++) {
      if(mapS[s.charAt(i)] != mapT[t.charAt(i)]) {
         return false;
      }
      mapS[s.charAt(i)] = i + 1;
      mapT[t.charAt(i)] = i + 1;
   }
   return true;
   }
```

599. Minimum Index Sum of Two Lists

```
 vector<string> findRestaurant(vector<string>& list1, vector<string>& list2) {
   unordered_map<string, int> map;
   vector<string> result;
   int minSum = INT_MAX;

   for(int i = 0; i < list1.size(); i++) {
      map[list1[i]] = i;
   }

   for(int i = 0; i < list2.size(); i++) {
```

```
        if(map.count(list2[i])) {
            int sum = i + map[list2[i]];
            if(sum < minSum) {
                result.clear();
                result.push_back(list2[i]);
                minSum = sum;
            }
            else if(sum == minSum) {
                result.push_back(list2[i]);
            }
        }
    }
    return result;
    }
```

Java

```
 public String[] findRestaurant(String[] list1, String[] list2) {
   Map<String, Integer> map = new HashMap<>();
    List<String> result = new ArrayList<>();
    int minSum = Integer.MAX_VALUE;

    for(int i = 0; i < list1.length; i++) {
        map.put(list1[i], i);
    }

    for(int i = 0; i < list2.length; i++) {
        if(map.containsKey(list2[i])) {
            int sum = i + map.get(list2[i]);
            if(sum < minSum) {
                result.clear();
                result.add(list2[i]);
                minSum = sum;
            }
            else if(sum == minSum) {
                result.add(list2[i]);
            }
        }
    }
    return result.toArray(new String[0]);
    }
```

387. First Unique Character in a String

```
 int firstUniqChar(string s) {
    unordered_map<char, int> charCount;
```

```
    for (char c : s) {
        charCount[c]++;
    }

    for (int i = 0; i < s.size(); ++i) {
        if (charCount[s[i]] == 1) {
            return i;
        }
    }

    return -1;
    }
```

Java

```
public int firstUniqChar(String s) {
     Map<Character, Integer> charCount = new HashMap<>();

    for (char c : s.toCharArray()) {
        charCount.put(c, charCount.getOrDefault(c, 0) + 1);
    }

    for (int i = 0; i < s.length(); i++) {
        if (charCount.get(s.charAt(i)) == 1) {
            return i;
        }
    }

    return -1;
}
```

350. Intersection of Two Arrays II

```
 vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
      unordered_map<int, int> map;
    vector<int> result;

    for(int num : nums1) {
        map[num]++;
    }

    for(int num : nums2) {
        if(map[num] > 0) {
            result.push_back(num);
            map[num]--;
        }
    }
```

```
    return result;
    }
```

Java

```
public int[] intersect(int[] nums1, int[] nums2) {
     Map<Integer, Integer> map = new HashMap<>();
    List<Integer> result = new ArrayList<>();

    for(int num : nums1) {
        map.put(num, map.getOrDefault(num, 0) + 1);
    }

    for(int num : nums2) {
        if(map.containsKey(num) && map.get(num) > 0) {
            result.add(num);
            map.put(num, map.get(num) - 1);
        }
    }

    return result.stream().mapToInt(i -> i).toArray();
    }
```

219. Contains Duplicate II

```
bool containsNearbyDuplicate(vector<int>& nums, int k) {
    unordered_map<int, int> map;

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

    return false;
    }
```

Java

```
public boolean containsNearbyDuplicate(int[] nums, int k) {
   Map<Integer, Integer> map = new HashMap<>();

    for(int i = 0; i < nums.length; i++) {
        if(map.containsKey(nums[i]) && i - map.get(nums[i]) <= k) {
            return true;
        }
```

```
    map.put(nums[i], i);
}

return false;
}
```