🗺️ Hash Map Interview Quick-Check Pattern

Your go-to framework for spotting and solving hash-based problems fast and clean.

---

1. 🧠 When to Reach for a Hash Map?

Ask yourself:

- ❓ Do I need to **track frequency**?
- ❓ Am I looking for **duplicates**, **first/last seen**, or **index mapping**?
- ❓ Is **fast lookup (O(1))** needed?
- ❓ Am I comparing two collections?
- ❓ Is there a need for **grouping**?

> 💡 If you said yes to any — a Hash Map (or Set) is your weapon.

---

2. 🔧 Most Common Hash Map Use Cases

Problem Type	Technique
Frequency Count	Map or Object
First Non-Repeating	Map + Ordered Iteration
Group by Key	Map of arrays
Lookup in Constant Time	Map/Set
Remove Duplicates	Use a Set
Index Mapping	Value → Index
Sliding Window Combo	Hash + Two Pointers

🧠 Hash Maps are versatile — they store **relationships**, **frequency**, **existence**, or **grouping**.

---

3. 🔍 Understand the Shape of the Map

- ✅ `value → count` (e.g., frequency map)
- ✅ `char → index` (e.g., first seen location)
- ✅ `key → [grouped values]` (e.g., group anagrams)
- ✅ `number → boolean` (e.g., check if seen before)

---

4. 🧪 Must-Know Templates

✅ Frequency Count

```
function buildFreqMap(arr) {
  const map = {};
  for (let item of arr) {
    map[item] = (map[item] || 0) + 1;
  }
  return map;
}
```

✅ First Unique Character

```
function firstUniqChar(s) {
  const freq = {};
  for (let c of s) freq[c] = (freq[c] || 0) + 1;
  for (let i = 0; i < s.length; i++) {
    if (freq[s[i]] === 1) return i;
  }
```

```
  return -1;
}
```

✅ Group Anagrams

```
function groupAnagrams(strs) {
  const map = {};
  for (let word of strs) {
    const key = word.split('').sort().join('');
    if (!map[key]) map[key] = [];
    map[key].push(word);
  }
  return Object.values(map);
}
```

✅ Two Sum (Value → Index Map)

```
function twoSum(nums, target) {
  const map = {};
  for (let i = 0; i < nums.length; i++) {
    let diff = target - nums[i];
    if (map[diff] !== undefined) return [map[diff], i];
    map[nums[i]] = i;
  }
}
```

✅ Longest Substring Without Repeating Characters

```
function lengthOfLongestSubstring(s) {
  let map = {}, left = 0, maxLen = 0;
  for (let right = 0; right < s.length; right++) {
    if (map[s[right]] !== undefined && map[s[right]] >= left) {
      left = map[s[right]] + 1;
    }
```

```
    map[s[right]] = right;
    maxLen = Math.max(maxLen, right - left + 1);
  }
  return maxLen;
}
```

---

5. 🧱 Edge Cases to Watch For

- Empty input or missing keys
- Key collisions or overwriting
- Key not found → `undefined`
- Map vs Object pitfalls (use `Map` when keys aren't strings)
- Deleting keys while iterating
- Large datasets → memory usage
- Case sensitivity (e.g., `"A"` vs `"a"`)

---

6. 🧠 Mental Model for Hash Map Problems

🎯 Ask yourself:

Question	Pattern to Apply
Do I need to **count something**?	Frequency Map
Do I need **O(1) lookup**?	Hash Map or Set
Do I need to track **index of items**?	Value → Index Mapping
Do I need to check for **duplicates**?	Use Set
Do I need to **group items** by pattern?	Map of Arrays

🔁 Problem Solving Loop:

1. 📌 What's the **key** I care about?
2. 📎 What's the **value** I want to associate?
3. 🔄 Do I need to **update**, **delete**, or **count** that key?
4. 🚨 What happens if key doesn't exist?
5. 🧱 What are possible **collisions or edge cases**?

✅ Final Interview Checklist

- Am I using the correct key → value pairing?
- Is my key **unique** enough (e.g., sorted string for anagram)?
- Do I need to **return an array or group** of values?
- Am I using `hasOwnProperty` if needed with raw objects?
- Do I handle **missing keys** or **duplicates** safely?