The Problem:

"Find the length of the longest substring in a given string `s` that contains no repeating characters."

Q1: Why is this a classic interview problem?

1. Tests your understanding of **strings** and **hashmaps.**
2. Introduces the powerful **Sliding Window** technique.
3. Makes you think about **edge cases**.

Q2: Brute force first: How does it work?

Check **every substring** in the string, then figure out which is the longest without duplicates.
💀 Time complexity? **O(n³)** (checking all substrings is slow!).

Q3: What's the clever approach?

Enter **Sliding Window**, the magic wand for substring problems:

- Use a **moving window** to expand and contract based on rules.
- Keep track of characters already in the window using a **hashmap** or **set**.

Q4: Sliding Window in Plain English, please?

1. Start with an empty window and two pointers, `start` and `end`.
2. Move the `end` pointer to expand the window.
3. If a duplicate is found, shrink the window by moving `start`.
4. Track the max length of a substring without duplicates.

Optimal Solution with Sliding Window

python
Copier le code

```
def length_of_longest_substring(s):
    char_map = {}  # To store the last seen index of each character
    start = 0
```

```
    max_length = 0

    for end, char in enumerate(s):
        if char in char_map and char_map[char] >= start:
            # Move start pointer to avoid duplicate
            start = char_map[char] + 1
        char_map[char] = end
        max_length = max(max_length, end - start + 1)

    return max_length

# Example
print(length_of_longest_substring("abcabcbb"))  # Output: 3 ("abc")
```

Q5: Does it scale well?

- **Time Complexity**: **O(n)** → Each character is processed at most twice.
- **Space Complexity**: **O(k)**, where k is the number of unique characters in the string (ASCII = 128).

Q6: Sliding Window + Variations to Know

🟢 Longest Substring Without Repeating Characters

Classic problem. HashMap tracks seen characters. Return the maximum window size.

🟡 Longest Substring With At Most K Distinct Characters

- **Twist**: Keep track of only k distinct characters.
- Use a **HashMap** to store character counts, shrink the window when distinct > k.

python
Copier le code

```
def length_of_longest_k_distinct(s, k):
    char_count = {}
    start = 0
    max_length = 0

    for end, char in enumerate(s):
        char_count[char] = char_count.get(char, 0) + 1
```

```
        while len(char_count) > k:
            char_count[s[start]] -= 1
            if char_count[s[start]] == 0:
                del char_count[s[start]]
            start += 1
        max_length = max(max_length, end - start + 1)

    return max_length

# Example
print(length_of_longest_k_distinct("eceba", 2))  # Output: 3 ("ece")
```

🔴 Longest Substring With At Least K Repeating Characters

- Split string into valid substrings where every character appears **at least k times**.
- Use **divide and conquer** + frequency analysis.

Q7: Edge Cases You Should Know

1. Empty string: `s = ""`. Answer? `0`.
2. Single character string: `s = "a"`. Answer? `1`.
3. All repeating characters: `s = "bbbbbb"`. Answer? `1`.
4. Unique characters: `s = "abcdef"`. Answer? Length of the string!

Quick Facts About Strings + Sliding Window

Strings 📜

1. Strings in Python are **immutable.**
2. **Common functions**:
    - Membership check: `"a" in "abc"` → O(n).
    - `s.find("a")`, `s.split()`, `s.replace()`.
3. **Pattern Matching:** Use slices + hashmaps for problems like substring palindromes or rotations.

Sliding Window 🔎

1. Applies to **arrays and strings**.

2. Solve problems involving **contiguous segments**:
    - Longest substring (no repeats, at most k distinct).
    - Smallest window containing all characters.
    - Maximum sum subarray.

Practice Problems Related to Sliding Window

1. **Minimum Window Substring** - Find the smallest window that contains all characters of another string.
2. **Substring with Concatenation of All Words** - Check if concatenation of words exists in a string.
3. **Max Consecutive Ones II** (array problem).