Sliding Window Problem Templates and LeetCode Questions with Code

1. Sliding Window for Substring Problems

1. Minimum Window Substring
2. Longest Substring Without Repeating Characters
3. Find All Anagrams in a String
4. Substring with Concatenation of All Words
5. Longest Substring with At Most K Distinct Characters
6. Longest Substring with At Most Two Distinct Characters
7. Permutation in String
8. Longest Repeating Character Replacement
9. Count Number of Nice Subarrays
10. Smallest Substring Containing
11. Longest Substring with At Least K Repeating Characters
12. Sliding Window Maximum
13. Maximum Number of Vowels in a Substring of Given Length
14. Binary Subarrays with Sum
15. Repeated DNA Sequences
16. Smallest Range Covering Elements from K Lists
17. Replace the Substring for Balanced String
18. Check Inclusions

Template Code

```
// Given a string s, find the length of the longest substring without repeating characters

public int lengthOfLongestSubstring(String s) {
```

```
    Set<Character> window = new HashSet<>(); or Using Map DataStructure
    int left = 0, right = 0, maxLength = 0;

    while (right < s.length()) {
        // Expand the window by moving right
        if (!window.contains(s.charAt(right))) {
            window.add(s.charAt(right));
            right++;
            maxLength = Math.max(maxLength, right - left);
        } else {
            // Shrink the window by moving left
            window.remove(s.charAt(left));
            left++;
        }
    }

    return maxLength;
}
```

2.Fixed-Size Sliding Window Problems

1. Sliding Window Maximum
2. Maximum Average Subarray I
3. Maximum Sum of Two Non-Overlapping Subarrays
4. Maximum Average Subarray II
5. Subarrays of Size K with Distinct Elements
6. Longest Continuous Subarray with Absolute Diff Less Than or Equal to Limit

Template Code

```
// Example: Maximum Average Subarray I

public double findMaxAverage(int[] nums, int k) {

    double maxSum = 0, currentSum = 0;

    for (int i = 0; i < k; i++) {

        currentSum += nums[i];

}

    maxSum = currentSum;

    for (int i = k; i < nums.length; i++) {

        currentSum += nums[i] - nums[i - k];

        maxSum = Math.max(maxSum, currentSum);

}

    return maxSum / k;

}
```

3.Variable-Size Sliding Window Problems

1. Longest Substring with At Most K Distinct Characters

2. Longest Subarray of 1's After Deleting One Element

3. Fruit Into Baskets

4. Maximum Length of Repeated Subarray

5. Longest Subarray with Sum at Most K

6. Maximum Consecutive Ones III

Template Code

```
// Example: Longest Substring with At Most K Distinct Characters

public int lengthOfLongestSubstringKDistinct(String s, int k) {

Map<Character, Integer> charCount = new HashMap<>();

    int left = 0, right = 0, maxLen = 0;

    while (right < s.length()) {

        char r = s.charAt(right);

        charCount.put(r, charCount.getOrDefault(r, 0) + 1);

        while (charCount.size() > k) {

            char l = s.charAt(left);

            charCount.put(l, charCount.get(l) - 1);

            if (charCount.get(l) == 0) charCount.remove(l);

            left++;

}

        maxLen = Math.max(maxLen, right - left + 1);

        right++;

}

    return maxLen;

}
```

4.Sliding Window with Frequency Counting

1. Find All Anagrams in a String

2. Permutation in String

3. Longest Substring with At Most Two Distinct Characters

4. Count Number of Substrings Containing K Distinct Characters

5. Subarrays with K Different Integers

6. Longest Substring Without Repeating Characters (using frequency counting)

Template Code

```
// Example: Find All Anagrams in a String

public List<Integer> findAnagrams(String s, String p) {

List<Integer> result = new ArrayList<>();

    if (s.length() < p.length()) return result;

    int[] pCount = new int[26];

    int[] sCount = new int[26];

    for (char c : p.toCharArray()) pCount[c - 'a']++;

    for (int i = 0; i < s.length(); i++) {

        sCount[s.charAt(i) - 'a']++;

        if (i >= p.length()) sCount[s.charAt(i - p.length()) - 'a']--;

        if (Arrays.equals(pCount, sCount)) result.add(i - p.length() + 1);

}

    return result;

}
```