Question 7: Sliding Window Technique and Its Applications in String Problems

Concept

The sliding window technique is a method that uses two pointers to create a window that slides over data to process it in chunks. It's particularly efficient for solving problems involving arrays or strings where we need to find a subarray or substring that satisfies certain conditions.

Types of Sliding Windows

1. Fixed-Size Window

- Window size remains constant throughout
- Used for problems like "find maximum sum subarray of size k"

2. Variable-Size Window

- Window size can change during processing
- Used for problems like "find smallest subarray with sum greater than x"

Applications in String Problems

1. **Finding Longest Substring with K Distinct Characters**
   - Using a variable-size window to track frequency of characters
   - Expanding and contracting window based on the distinct character count
2. **Minimum Window Substring**
   - Finding smallest substring containing all characters of another string
   - Using a frequency map and two-pointer approach
3. **Longest Substring Without Repeating Characters**
   - Using a set or map to track characters in the current window
   - Expanding the window until a duplicate is found, then contracting
4. **Substring with Concatenation of All Words**
   - Using fixed-length windows and frequency maps
   - Sliding the window to find valid concatenations
5. **Permutation in String**
   - Using fixed-size window equal to the length of the pattern string
   - Checking if any window matches the pattern's character frequency

Implementation Example: Find All Anagrams in a String

cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>
using namespace std;

vector<int> findAnagrams(string s, string p) {
    vector<int> result;
    int n = s.length();
    int m = p.length();

    if (n < m) return result;

    unordered_map<char, int> pCount, sCount;

    // Count characters in pattern
    for (char c : p) {
        pCount[c]++;
    }

    // Initial window
    for (int i = 0; i < m; i++) {
        sCount[s[i]]++;
    }

    // Check if first window is anagram
    if (sCount == pCount) {
        result.push_back(0);
    }

    // Slide window
    for (int i = m; i < n; i++) {
        // Add one character from right
        sCount[s[i]]++;

        // Remove one character from left
        sCount[s[i-m]]--;
        if (sCount[s[i-m]] == 0) {
            sCount.erase(s[i-m]);
        }

        // Check if current window is anagram
        if (sCount == pCount) {
            result.push_back(i-m+1);
        }
    }
```

```
    return result;
}

int main() {
    string s = "cbaebabacd";
    string p = "abc";

    vector<int> indices = findAnagrams(s, p);

    cout << "Anagrams of \"" << p << "\" found at indices: ";
    for (int idx : indices) {
        cout << idx << " ";
    }
    cout << endl;

    return 0;
}
```

Benefits of Sliding Window Technique

1. **Efficiency**: Reduces time complexity from $O(n^2)$ to $O(n)$ in many cases
2. **Memory Optimization**: Typically requires $O(1)$ or $O(k)$ extra space
3. **Avoiding Redundant Work**: Reuses computation from previous steps
4. **Simplicity**: Conceptually straightforward once the pattern is understood

Common Variations

1. **Two-pointer Technique**: Both pointers move in the same direction
2. **Fast and Slow Pointers**: Pointers move at different speeds
3. **Opposite Direction Pointers**: Pointers move toward each other