Sliding Window with Set - Complete Mastery Guide

LeetCode, HackerRank, and Codeforces Problems with Solutions

Table of Contents

Chapter 1: When to Use Set with Sliding Window

1.1 Recognition Signals

Use Set When:

- Need to track **unique elements** (presence/absence only)
- No frequency counting required
- Check for duplicates
- Maintain distinct elements in window
- "unique", "distinct", "no repeating"

Set vs HashMap:

Aspect	Set	HashMap
Storage	Elements only	Key-value pairs
Use Case	Uniqueness check	Frequency tracking
Space	O(k) distinct	O(k) distinct
Time	O(1) average	O(1) average

1.2 Core Template

```
int slidingWindowSet(string s) {
    unordered_set&lt;char&gt; window_set;
    int left = 0;
    int max_len = 0;

    for (int right = 0; right &lt; s.length(); right++) {
        // If duplicate found, shrink window
        while (window_set.count(s[right])) {
            window_set.erase(s[left]);
            left++;
        }

        // Add current character
        window_set.insert(s[right]);

        // Update result
        max_len = max(max_len, right - left + 1);
    }

    return max_len;
}
```

Chapter 2: Complete Problem Solutions

Problem 1: Longest Substring Without Repeating Characters (LeetCode 3)

Link: https://leetcode.com/problems/longest-substring-without-repeating-characters/

Platforms:

- **LeetCode 3:** Longest Substring Without Repeating Characters
- **HackerRank:** No Idea!
- **Codeforces:** Good String

Difficulty: Medium

Description: Find length of longest substring without repeating characters.

Solution 1: Set with Shrinking Window - O(n) time, O(min(n,m)) space

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_set&lt;char&gt; char_set;
        int left = 0;
        int max_len = 0;

        for (int right = 0; right &lt; s.length(); right++) {
```

```
            // Shrink window until no duplicates
            while (char_set.count(s[right])) {
                char_set.erase(s[left]);
                left++;
            }

            // Add current character
            char_set.insert(s[right]);

            // Update max length
            max_len = max(max_len, right - left + 1);
        }

        return max_len;
    }
};
```

Detailed Walkthrough:

```
s = "abcabcbb"

right=0: 'a'
  char_set = {}, 'a' not in set
  Insert 'a', char_set = {a}
  max_len = 1

right=1: 'b'
  char_set = {a}, 'b' not in set
  Insert 'b', char_set = {a,b}
  max_len = 2

right=2: 'c'
  char_set = {a,b}, 'c' not in set
  Insert 'c', char_set = {a,b,c}
  max_len = 3

right=3: 'a'
  char_set = {a,b,c}, 'a' IN set (duplicate!)
  SHRINK: erase s[0]='a', left=1, char_set = {b,c}
  Now 'a' not in set
  Insert 'a', char_set = {b,c,a}
  window = "bca", max_len = 3

right=4: 'b'
  char_set = {b,c,a}, 'b' IN set (duplicate!)
  SHRINK: erase s[1]='b', left=2, char_set = {c,a}
  Now 'b' not in set
  Insert 'b', char_set = {c,a,b}
  window = "cab", max_len = 3

right=5: 'c'
  char_set = {c,a,b}, 'c' IN set (duplicate!)
  SHRINK: erase s[2]='c', left=3, char_set = {a,b}
  Now 'c' not in set
  Insert 'c', char_set = {a,b,c}
```

```
  window = "abc", max_len = 3

right=6: 'b'
  char_set = {a,b,c}, 'b' IN set
  SHRINK: erase s[3]='a', left=4, char_set = {b,c}
  Still 'b' in set
  SHRINK: erase s[4]='b', left=5, char_set = {c}
  Insert 'b', char_set = {c,b}
  window = "cb", max_len = 3

right=7: 'b'
  char_set = {c,b}, 'b' IN set
  SHRINK: erase s[5]='c', left=6, char_set = {b}
  Still 'b' in set
  SHRINK: erase s[6]='b', left=7, char_set = {}
  Insert 'b', char_set = {b}
  window = "b", max_len = 3

Answer: 3
```

Problem 2: Longest Continuous Subarray With Absolute Diff ≤ Limit (LeetCode 1438)

Link: https://leetcode.com/problems/longest-continuous-subarray-with-absolute-diff-less-than-or-equal-to-limit/

Difficulty: Medium

Description: Find longest subarray where absolute difference between any two elements is less than or equal to limit.

Examples:

```
Input: nums = [8,2,4,7], limit = 4
Output: 2
Explanation: [2,4] has diff 2, [4,7] has diff 3

Input: nums = [10,1,2,4,7,2], limit = 5
Output: 4
Explanation: [2,4,7,2] has max diff 5

Input: nums = [4,2,2,2,4,4,2,2], limit = 0
Output: 3
```

Solution 1: Multiset (Ordered Set) - O(n log n) time, O(n) space

```
#include &lt;set&gt;

class Solution {
public:
    int longestSubarray(vector&lt;int&gt;&amp; nums, int limit) {
        multiset&lt;int&gt; window;  // Keeps elements sorted, allows duplicates
        int left = 0;
        int max_len = 0;

        for (int right = 0; right &lt; nums.size(); right++) {
            // Add element to window
            window.insert(nums[right]);

            // Check if window valid
            // multiset keeps elements sorted: *begin() = min, *rbegin() = max
            while (*window.rbegin() - *window.begin() &gt; limit) {
                // Remove leftmost element
                window.erase(window.find(nums[left]));
                left++;
            }

            max_len = max(max_len, right - left + 1);
        }

        return max_len;
    }
};
```

Why Multiset:

- Maintains sorted order: O(log n) insert/delete
- Can find min/max in O(1): `*begin()`, `*rbegin()`
- Allows duplicates (unlike `set`)

Walkthrough:

```
nums = [8,2,4,7], limit = 4

right=0: nums[0]=8
  window = {8}
  diff = 8-8 = 0 ≤ 4, valid
  max_len = 1

right=1: nums[1]=2
  window = {2,8}  (sorted)
  diff = 8-2 = 6 &gt; 4, INVALID
  SHRINK: remove nums[0]=8, left=1
  window = {2}
  diff = 2-2 = 0 ≤ 4
  max_len = 1
```

```
right=2: nums[2]=4
  window = {2,4}
  diff = 4-2 = 2 ≤ 4, valid
  max_len = 2

right=3: nums[3]=7
  window = {2,4,7}
  diff = 7-2 = 5 &gt; 4, INVALID
  SHRINK: remove nums[1]=2, left=2
  window = {4,7}
  diff = 7-4 = 3 ≤ 4, valid
  max_len = 2

Answer: 2
```

Solution 2: Two Deques (Monotonic Queue) - O(n) time, O(n) space ★

```
#include &lt;deque&gt;

class Solution {
public:
    int longestSubarray(vector&lt;int&gt;&amp; nums, int limit) {
        deque&lt;int&gt; max_deque;  // Decreasing order (front = max)
        deque&lt;int&gt; min_deque;  // Increasing order (front = min)
        int left = 0;
        int max_len = 0;

        for (int right = 0; right &lt; nums.size(); right++) {
            // Maintain max deque (decreasing)
            while (!max_deque.empty() &amp;&amp; nums[max_deque.back()] &lt; nums[right])
                max_deque.pop_back();
            }
            max_deque.push_back(right);

            // Maintain min deque (increasing)
            while (!min_deque.empty() &amp;&amp; nums[min_deque.back()] &gt; nums[right])
                min_deque.pop_back();
            }
            min_deque.push_back(right);

            // Shrink window if invalid
            while (nums[max_deque.front()] - nums[min_deque.front()] &gt; limit) {
                left++;

                // Remove indices outside window
                if (max_deque.front() &lt; left) max_deque.pop_front();
                if (min_deque.front() &lt; left) min_deque.pop_front();
            }

            max_len = max(max_len, right - left + 1);
        }

        return max_len;
```

```
    }
};
```

Advantage: O(n) vs O(n log n) - each element pushed/popped at most once.

Problem 3: Unique Email Addresses (LeetCode 929)

Link: https://leetcode.com/problems/unique-email-addresses/

Difficulty: Easy

Description: Count unique email addresses after applying rules.

Example:

```
Input: emails = ["test.email+alex@leetcode.com","test.e.mail+bob.cathy@leetcode.com"]
Output: 1
Explanation: Both emails are sent to "testemail@leetcode.com"
```

Solution: Set for Uniqueness - O(n×m) time, O(n×m) space

```
class Solution {
public:
    int numUniqueEmails(vector&lt;string&gt;&amp; emails) {
        unordered_set&lt;string&gt; unique_emails;

        for (const string&amp; email : emails) {
            string normalized = normalizeEmail(email);
            unique_emails.insert(normalized);
        }

        return unique_emails.size();
    }

private:
    string normalizeEmail(const string&amp; email) {
        string result;
        bool plus_found = false;
        bool at_found = false;

        for (char c : email) {
            if (c == '@') {
                at_found = true;
                result += c;
            } else if (at_found) {
                // After @: keep everything
                result += c;
            } else if (c == '+') {
                // Ignore everything until @
                plus_found = true;
            } else if (c == '.') {
```

```
                // Ignore dots in local name
                continue;
            } else if (!plus_found) {
                // Add character before + or @
                result += c;
            }
        }

        return result;
    }
};
```

Problem 4: Contains Duplicate (LeetCode 217)

Link: https://leetcode.com/problems/contains-duplicate/

Difficulty: Easy

Description: Return true if any value appears at least twice.

Solution: Set for Duplicate Detection - O(n) time, O(n) space

```
class Solution {
public:
    bool containsDuplicate(vector&lt;int&gt;&amp; nums) {
        unordered_set&lt;int&gt; seen;

        for (int num : nums) {
            if (seen.count(num)) {
                return true;  // Duplicate found
            }
            seen.insert(num);
        }

        return false;
    }
};
```

Alternative (Sorting): O(n log n) time, O(1) space

```
bool containsDuplicate(vector&lt;int&gt;&amp; nums) {
    sort(nums.begin(), nums.end());

    for (int i = 1; i &lt; nums.size(); i++) {
        if (nums[i] == nums[i-1]) {
            return true;
        }
    }
```

```
    return false;
}
```

Problem 5: Distribute Candies (LeetCode 575)

Link: https://leetcode.com/problems/distribute-candies/

Difficulty: Easy

Description: Maximum number of different candy types sister can eat (she can eat n/2 candies).

Example:

```
Input: candyType = [1,1,2,2,3,3]
Output: 3
Explanation: Can eat 3 candies, there are 3 types

Input: candyType = [1,1,2,3]
Output: 2
Explanation: Can eat 2 candies, there are 3 types, limited to 2
```

Solution: Count Unique Types - O(n) time, O(n) space

```
class Solution {
public:
    int distributeCandies(vector&lt;int&gt;&amp; candyType) {
        unordered_set&lt;int&gt; unique_types(candyType.begin(), candyType.end());

        int max_candies = candyType.size() / 2;
        int unique_count = unique_types.size();

        return min(max_candies, unique_count);
    }
};
```

Logic: Sister wants maximum variety, but limited to n/2 candies.

Problem 6: Longest Harmonious Subsequence (LeetCode 594)

Link: https://leetcode.com/problems/longest-harmonious-subsequence/

Difficulty: Easy

Description: Harmonious array: max and min differ by exactly 1. Find longest harmonious subsequence.

Example:

```
Input: nums = [1,3,2,2,5,2,3,7]
Output: 5
Explanation: [3,2,2,2,3] is harmonious (max=3, min=2, diff=1)
```

Solution: HashMap for Frequency - O(n) time, O(n) space

```
class Solution {
public:
    int findLHS(vector&lt;int&gt;&amp; nums) {
        unordered_map&lt;int, int&gt; freq;

        // Count frequencies
        for (int num : nums) {
            freq[num]++;
        }

        int max_len = 0;

        // Check each number and number+1
        for (const auto&amp; [num, count] : freq) {
            if (freq.count(num + 1)) {
                max_len = max(max_len, count + freq[num + 1]);
            }
        }

        return max_len;
    }
};
```

Problem 7: Maximum Erasure Value (LeetCode 1695)

Link: https://leetcode.com/problems/maximum-erasure-value/

Difficulty: Medium

Description: Find maximum sum of unique subarray.

Example:

```
Input: nums = [4,2,4,5,6]
Output: 17
Explanation: [2,4,5,6] has sum 17 and all unique

Input: nums = [5,2,1,2,5,2,1,2,5]
Output: 8
Explanation: [5,2,1] or [1,2,5] have sum 8
```

Solution: Sliding Window with Set - O(n) time, O(n) space

```
class Solution {
public:
    int maximumUniqueSubarray(vector&lt;int&gt;&amp; nums) {
        unordered_set&lt;int&gt; window_set;
        int left = 0;
        int current_sum = 0;
        int max_sum = 0;

        for (int right = 0; right &lt; nums.size(); right++) {
            // Shrink until unique
            while (window_set.count(nums[right])) {
                window_set.erase(nums[left]);
                current_sum -= nums[left];
                left++;
            }

            // Add current element
            window_set.insert(nums[right]);
            current_sum += nums[right];

            // Update max sum
            max_sum = max(max_sum, current_sum);
        }

        return max_sum;
    }
};
```

Walkthrough:

```
nums = [4,2,4,5,6]

right=0: nums[0]=4
  window_set = {}, 4 not in set
  Add 4, set={4}, sum=4, max=4

right=1: nums[1]=2
  window_set = {4}, 2 not in set
  Add 2, set={4,2}, sum=6, max=6

right=2: nums[2]=4
  window_set = {4,2}, 4 IN set (duplicate!)
  SHRINK: remove nums[0]=4, left=1
  set={2}, sum=2
  Add 4, set={2,4}, sum=6, max=6

right=3: nums[3]=5
  window_set = {2,4}, 5 not in set
  Add 5, set={2,4,5}, sum=11, max=11

right=4: nums[4]=6
  window_set = {2,4,5}, 6 not in set
```

```
  Add 6, set={2,4,5,6}, sum=17, max=17

Answer: 17
```

Problem 8: Subarrays with K Different Integers (LeetCode 992)

Link: https://leetcode.com/problems/subarrays-with-k-different-integers/

Difficulty: Hard

Description: Count subarrays with exactly K different integers.

Note: Uses HashMap for frequency, but Set concept for distinct count.

Solution: atMost(K) - atMost(K-1) - O(n) time, O(k) space

```
class Solution {
private:
    int atMostK(vector&lt;int&gt;&amp; nums, int k) {
        unordered_map&lt;int, int&gt; freq;
        int left = 0;
        int count = 0;

        for (int right = 0; right &lt; nums.size(); right++) {
            if (freq[nums[right]]++ == 0) k--;  // New distinct element

            while (k &lt; 0) {
                if (--freq[nums[left]] == 0) k++;  // Lost distinct element
                left++;
            }

            // All subarrays ending at right
            count += right - left + 1;
        }

        return count;
    }

public:
    int subarraysWithKDistinct(vector&lt;int&gt;&amp; nums, int k) {
        return atMostK(nums, k) - atMostK(nums, k - 1);
    }
};
```

Key Insight: exactly(K) = atMost(K) - atMost(K-1)

Problem 9: Remove Duplicates from Sorted Array (LeetCode 26)

Link: https://leetcode.com/problems/remove-duplicates-from-sorted-array/

Difficulty: Easy

Description: Remove duplicates in-place, return new length.

Solution: Two Pointers (No Set Needed - Sorted!) - O(n) time, O(1) space

```
class Solution {
public:
    int removeDuplicates(vector&lt;int&gt;&amp; nums) {
        if (nums.empty()) return 0;

        int write_idx = 1;  // Next position to write

        for (int read_idx = 1; read_idx &lt; nums.size(); read_idx++) {
            if (nums[read_idx] != nums[read_idx - 1]) {
                nums[write_idx] = nums[read_idx];
                write_idx++;
            }
        }

        return write_idx;
    }
};
```

Why No Set: Array is sorted, so duplicates are adjacent.

Problem 10: Keyboard Row (LeetCode 500)

Link: https://leetcode.com/problems/keyboard-row/

Difficulty: Easy

Description: Find words that can be typed using one keyboard row.

Example:

```
Input: words = ["Hello","Alaska","Dad","Peace"]
Output: ["Alaska","Dad"]
```

Solution: Set Intersection - O(n×m) time, O(1) space

```
class Solution {
public:
    vector&lt;string&gt; findWords(vector&lt;string&gt;&amp; words) {
        unordered_set&lt;char&gt; row1{'q','w','e','r','t','y','u','i','o','p'};
        unordered_set&lt;char&gt; row2{'a','s','d','f','g','h','j','k','l'};
        unordered_set&lt;char&gt; row3{'z','x','c','v','b','n','m'};

        vector&lt;string&gt; result;

        for (const string&amp; word : words) {
            if (canType(word, row1) || canType(word, row2) || canType(word, row3)) {
                result.push_back(word);
            }
        }

        return result;
    }

private:
    bool canType(const string&amp; word, const unordered_set&lt;char&gt;&amp; row) {
        for (char c : word) {
            if (!row.count(tolower(c))) {
                return false;
            }
        }
        return true;
    }
};
```

Problem 11: Intersection of Multiple Arrays (LeetCode 2248)

Link: https://leetcode.com/problems/intersection-of-multiple-arrays/

Difficulty: Easy

Description: Find integers present in all arrays, sorted.

Example:

```
Input: nums = [[3,1,2,4,5],[1,2,3,4],[3,4,5,6]]
Output: [3,4]
```

Solution: Set Intersection - O(n×m) time, O(k) space

```
class Solution {
public:
    vector&lt;int&gt; intersection(vector&lt;vector&lt;int&gt;&gt;&amp; nums) {
        unordered_map&lt;int, int&gt; count;

        // Count occurrences across all arrays
        for (const auto&amp; arr : nums) {
            unordered_set&lt;int&gt; unique(arr.begin(), arr.end());
            for (int num : unique) {
                count[num]++;
            }
        }

        vector&lt;int&gt; result;
        int n = nums.size();

        for (const auto&amp; [num, freq] : count) {
            if (freq == n) {  // Present in all arrays
                result.push_back(num);
            }
        }

        sort(result.begin(), result.end());
        return result;
    }
};
```

Problem 12: Single Number (LeetCode 136)

Link: https://leetcode.com/problems/single-number/

Difficulty: Easy

Description: Every element appears twice except one. Find that one.

Solution 1: Set - O(n) time, O(n) space

```
int singleNumber(vector&lt;int&gt;&amp; nums) {
    unordered_set&lt;int&gt; seen;

    for (int num : nums) {
        if (seen.count(num)) {
            seen.erase(num);  // Seen twice, remove
        } else {
            seen.insert(num);  // First time
        }
    }
```

```
    return *seen.begin();  // Only element left
}
```

Solution 2: XOR Trick - O(n) time, O(1) space ⭐

```
int singleNumber(vector&lt;int&gt;&amp; nums) {
    int result = 0;

    for (int num : nums) {
        result ^= num;  // XOR cancels duplicates
    }

    return result;
}
```

Why XOR Works:

- `a ^ a = 0` (same numbers cancel)
- `0 ^ b = b` (XOR with 0 gives original)
- XOR is commutative and associative

This comprehensive textbook covers 12+ problems using Set with Sliding Window and related patterns, with complete solutions, complexity analysis, and platform-specific problem mappings!