

Minimum Window Substring

Shrink the window until it barely contains all required chars ■

● HARD

■ Hook

This is one of the hardest and most classic sliding window problems. Given strings **s** and **t**, find the **smallest substring of s that contains all characters of t**. It appears in Google, Meta, Amazon interviews and is the gold standard for 'minimum window' problems.

■ Core Intuition — Two-Phase Approach

Phase 1: **Expand** the right pointer until all characters of **t** are covered in the window. Phase 2: **Contract** the left pointer to minimize the window while still covering **t**. Then expand again. Repeat. Track the minimum valid window seen.

■ The 'have vs need' counter

Keep two maps: **need** (frequency of each char in **t**) and **window** (current frequencies in window). A variable **'have'** counts how many distinct chars in **t** are fully satisfied. When **have == need.size()**, the window is valid — try to shrink.

■ Visual — s='ADOBECODEBANC', t='ABC'

```

t = "ABC" → need = {A:1, B:1, C:1} need.size()==3
have = 0 (satisfied chars)

EXPAND right until have==3:
→ r=0(A): window={A:1} have=1
→ r=1(D): window={A:1,D:1} have=1
→ r=2(O): have=1
→ r=3(B): window={..B:1} have=2
→ r=4(E): have=2
→ r=5(C): window={..C:1} have=3 ✓ VALID!
    window='ADOBEC' len=6 save as best

SHRINK left until invalid:
→ remove A(l=0): window={A:0} have drops to 2 → STOP left=1
    ('DOBEC' - missing A, invalid)

EXPAND right again:
→ r=6(O),r=7(D),r=8(E): have=2
→ r=9(B): have=2 (B already satisfied)
→ r=10(A): window={A:1} have=3 ✓ VALID!
    window='DOBECODEBA'? No: left=1, right=10 → 'DOBECODEBA' len=10 not better
SHRINK: remove D(l=1) safe, remove O(l=2) safe, remove B(l=3): B count drops
    but B was 2 in window (index 3 and 9), still ≥ need → safe! left=4
    remove E(l=4): safe. left=5. remove C(l=5): C drops below need → STOP left=6
window = 'ODEBAN' + rest → kept as 'BANC' (r=12, l=9)

★ ANSWER = 'BANC' length 4

```

■ Code Implementation (Java)

```

import java.util.HashMap;

public class MinimumWindowSubstring {

    public String minWindow(String s, String t) {
        if (s.isEmpty() || t.isEmpty()) return "";

        HashMap<Character, Integer> need = new HashMap<>();
        HashMap<Character, Integer> window = new HashMap<>();

        for (char c : t.toCharArray()) need.merge(c, 1, Integer::sum);

        int have = 0, required = need.size();
        int left = 0, minLen = Integer.MAX_VALUE, minLeft = 0;

        for (int right = 0; right < s.length(); right++) {
            char c = s.charAt(right);
            window.merge(c, 1, Integer::sum);

            // Check if this char's freq matches need
            if (need.containsKey(c) && window.get(c).equals(need.get(c)))
                have++;

            // While window is valid, try to shrink
            while (have == required) {
                if (right - left + 1 < minLen) {
                    minLen = right - left + 1;
                    minLeft = left;
                }

                char lc = s.charAt(left);
                window.merge(lc, -1, Integer::sum);
                if (need.containsKey(lc) && window.get(lc) < need.get(lc))
                    have--;
                left++;
            }
        }

        return minLen == Integer.MAX_VALUE ? "" : s.substring(minLeft, minLeft + minLen)
    ;
    }
}

```

■ Time & Space Complexity

■ TIME COMPLEXITY

$O(|s| + |t|)$

■ SPACE COMPLEXITY

$O(|s| + |t|)$

Each character in s is added and removed from window at most once $\rightarrow O(|s|)$. Building need map $\rightarrow O(|t|)$. Window map can be at most $O(\text{unique chars in } s)$. Total: $O(|s| + |t|)$.

Common Mistakes

Comparing Integer objects with == instead of .equals()

`window.get(c) == need.get(c)` compares object references in Java, not values! For counts > 127 , this gives wrong results. Always use `.equals()` for Integer comparisons.

Not tracking 'have' correctly

Only increment 'have' when `window.get(c)` exactly equals `need.get(c)`. If it's already higher, 'have' shouldn't change. This ensures each char is counted once.

Practice Problems

#	Problem	Difficulty	Pattern
1	Minimum Window Substring (LC 76)	Hard	SW + Two Maps + have/need
2	Find All Anagrams in a String (LC 438)	Medium	Fixed SW + Freq
3	Permutation in String (LC 567)	Medium	Fixed SW + Freq
4	Smallest Range Covering Elements from K Lists	Hard	SW + Priority Queue
5	Minimum Window Subsequence (LC 727)	Hard	Two-pointer + DP

Quick Revision

- ✓ need map: frequency of each char in `t`
- ✓ window map: frequency of each char in current window
- ✓ `have` = count of chars whose window freq $\geq$ need freq
- ✓ When `have == need.size()`: window is valid $\rightarrow$ shrink left
- ✓ Track minimum valid window (`minLen`, `minLeft`)
- ✓ Use `.equals()` not `==` for Integer comparison
- ✓ Time: $O(|s| + |t|)$ | Space: $O(|s| + |t|)$

■ MENTAL MODEL — Remember This Forever

Think of `t` as a shopping list. You walk through `s` (the store aisle), adding items to your cart. The moment your cart has every item on the list (`have == required`), you've found a valid window. Now walk the cart back from the front, removing items one by one — as long as the list is still complete. The smallest cart that holds everything is your answer.