76. Minimum Window Substring

Hard 5823 399 Add to List Share

Given two strings `s` and `t`, return *the minimum window in* `s` *which will contain all the characters in* `t`. If there is no such window in `s` that covers all characters in `t`, return *the empty string* `""`.

Note that If there is such a window, it is guaranteed that there will always be only one unique minimum window in `s`.

Example 1:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
```

Example 2:

```
Input: s = "a", t = "a"
Output: "a"
```

Constraints:

- `1 <= s.length, t.length <= 10^5`
- `s` and `t` consist of English letters.

Follow up: Could you find an algorithm that runs in `O(n)` time?

```
void slidingWindow(string s, string t) {
    unordered_map<char, int> need, window;
    for (char c : t) need[c]++;

    int left = 0, right = 0;
    int valid = 0;
    while (right < s.size()) {
        // c is the char adding to window
        char c = s[right];
        // move right side of window
        right++;
        // update data in window
        ...
        // output debug information
        printf("window: [%d, %d)\n", left, right);

        // check whether shrink the left side of window
        while (window needs shrink) {
            // d is the char removing from window
            char d = s[left];
            // move left side of window
            left++;
            // update data in window
            ...
        }
    }
}
```

① unordered_map is hash table
map.contain(key)
map[key]

② Key idea:
a. left = right = 0, window is [left, right)
b. add right to augment window until satisfied
c. add left to shrink window until unsatisfied
d. repeat b. and c. until right takes the end

③ While add right, update window counter.
When valid satisfies need, shrink window.
While add left, reduce window counter.
When shrink window, update the final result.

```cpp
class Solution {
public:
    string minWindow(string s, string t) {
        unordered_map<char, int> need, window;
        for (char c : t) {
            need[c]++;
        }

        int left = 0, right = 0;
        int valid = 0;

        int min_window_size = INT_MAX;
        string min_window;

        while (right < s.size()) {
            // c is the char adding to window
            char c = s[right];
            // move the right side of window
            right++;
            // update the window counter and valid
            if (need.count(c)) {
                window[c]++;
                if (window[c] == need[c]) {
                    valid++;
                }
            }

    //        cout << "window: [" << left << "," << right << ")" << endl;

            while(valid >= need.size()) {
                // d is the char removing from the window
                char d = s[left];
                // move the left side of window
                left++;
                // update the window counter and valid
                if (need.count(d)) {
                    if (window[d] == need[d]) {
                        valid--;
                    }
                    window[d]--;
                }
                if (right - left + 1 < min_window_size) {
                    min_window_size = right - left + 1;
                    min_window = s.substr(left - 1, min_window_size);
                }
            }
        }

        if (min_window_size == INT_MAX) {
            min_window = "";
        }

        return min_window;
    }
};
```

Most tricky part.
Avoid to traverse unordered_map.
(It's time consuming.)