“**Experiment 3.1”**

**Aim:**

To demonstrate the concept of Greedy approach.

**Objective:**

• The objective is to build problem solving capability and to learn the basic concepts of data structures.

• The implementation of ‘Remove Duplicate Letters’ problem brushes up the concept of greedy approach.

• The implementation of ‘Assign Cookies’ problem brushes up the concept of greedy approach.

**Problem 1: “Remove Duplicate Letters”**

<https://leetcode.com/problems/remove-duplicate-letters/description/>

Given a string s, remove duplicate letters so that every letter appears once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

**Code:**

class Solution {

public:

    string removeDuplicateLetters(string s) {

        int len = s.size();

        string res = "";

        unordered\_map<char, int> M;

        unordered\_map<char, bool> V;

        stack<int> S;

        for (auto c : s) {

            if (M.find(c) == M.end()) M[c] = 1;

            else M[c]++;

        }

        for (unordered\_map<char, int>::iterator iter=M.begin(); iter!=M.end();iter++) V[iter->first] = false;

        cout<<M.size()<<V.size()<<endl;

        for (int i=0; i<len; i++) {

            M[s[i]]--;

            if (V[s[i]] == true) continue;

            while (!S.empty() and s[i] < s[S.top()] and M[s[S.top()]] > 0) {

                V[s[S.top()]] = false;

                S.pop();

            }

            S.push(i);

            V[s[i]] = true;

        }

        while (!S.empty()) {

            res = s[S.top()] + res;

            S.pop();

        }

        return res;

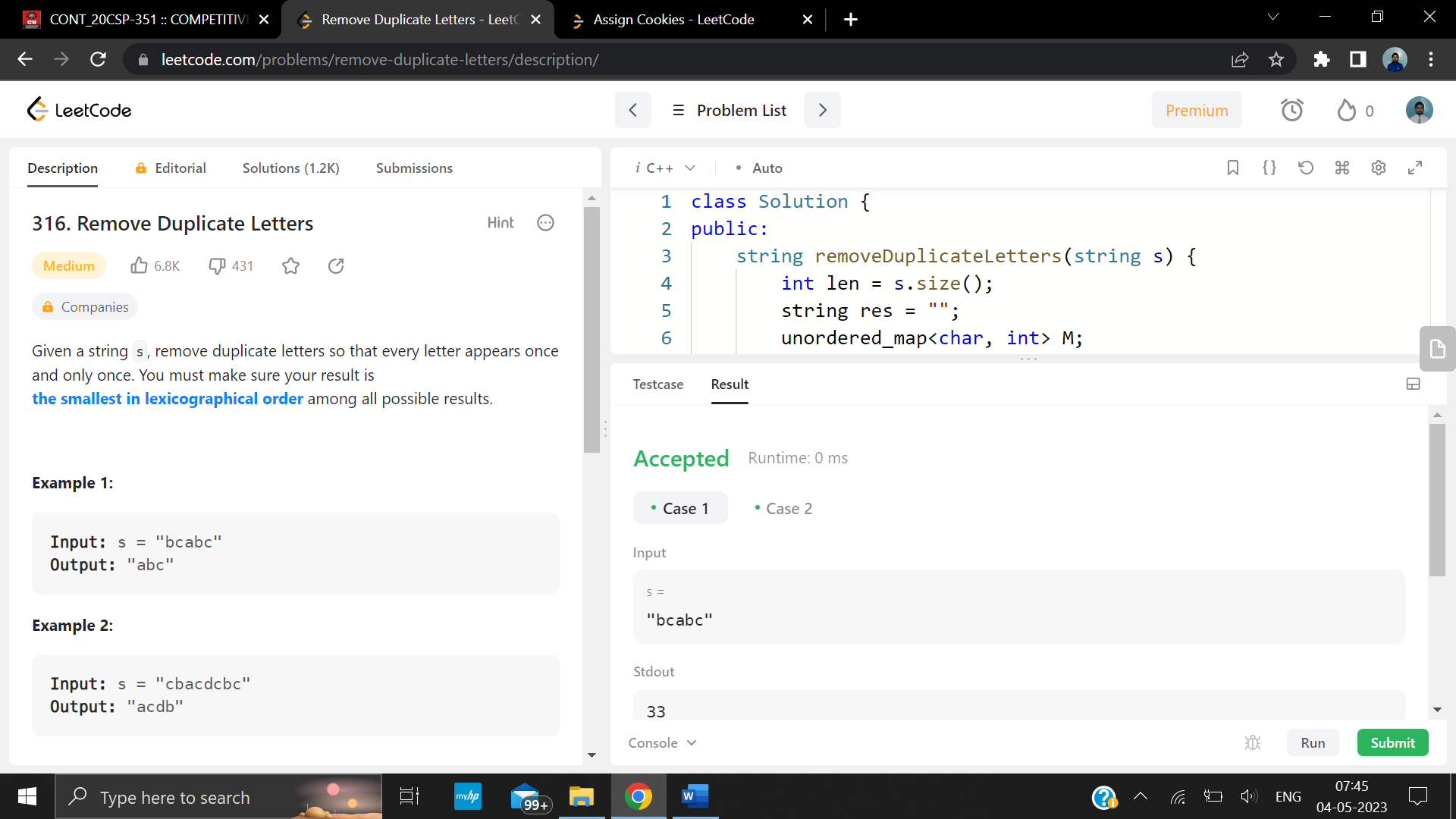
    }

};

// Time Complexity O(N)

// Space Complexity O(N)

**Output:**



**Problem 2: “Assign Cookies”**

<https://leetcode.com/problems/assign-cookies/>

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

If targetCapacity liters of water are measurable, you must have targetCapacity liters of water contained within one or both buckets by the end.

Operations allowed:

* Fill any of the jugs with water.
* Empty any of the jugs.
* Pour water from one jug into another till the other jug is completely full, or the first jug itself is empty

**Code:**

class Solution {

public:

    int findContentChildren(vector<int>& g, vector<int>& s) {

        sort(g.begin(), g.end());

        sort(s.begin(), s.end());

        int sizeG = g.size(), sizeS = s.size(), i, j, lastJ=0, count=0,found;

        for(i = 0; i < sizeG; ++i) {

            for(j = lastJ, found=0; j < sizeS; ++j) {

                if (s[j] >= g[i]) {

                    ++count;

                    lastJ=j+1;

                    found = 1;

                    break;

                }

            }

            if(found == 0) {

                break;

            }

        }

        return count;

    }

};

// Time Complexity: O(nlogn)

// Space Complexity: O(1)

**Output:**

