Experiment – 3.1

Student Name: Avinash Jena **UID: 20BCS2690**

Branch: CSE **Section/Group: 707/B**

Subject Name: Competitive Coding II **Subject Code: 20CSP-351**

Aim: Remove Duplicate Letters

Objective:

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.

Example 1:

Input: s = "bcabc"

Output: "abc"

Example 2:

Input: s = "cbacdcbc"

Output: "acdb"

Constraints:

1 <= s.length <= 104

s consists of lowercase English letters.

Code:

```
class Solution {
    public String removeDuplicateLetters(String s) {
        int[] lastIndex = new int[26];
        for (int i = 0; i < s.length(); i++){
            lastIndex[s.charAt(i) - 'a'] = i;
        }

        boolean[] seen = new boolean[26];
        Stack<Integer> st = new Stack();

        for (int i = 0; i < s.length(); i++) {
            int curr = s.charAt(i) - 'a';
            if (seen[curr]) continue;
            while (!st.isEmpty() && st.peek() > curr && i < lastIndex[st.peek()]){
                seen[st.pop()] = false;
            }
            st.push(curr);
            seen[curr] = true;
        }

        StringBuilder sb = new StringBuilder();
        while (!st.isEmpty())
            sb.append((char) (st.pop() + 'a'));
        return sb.reverse().toString();
    }
}
```

Output:

Java • Auto

```java
class Solution {
    public String removeDuplicateLetters(String s) {
        int[] lastIndex = new int[26];
        for (int i = 0; i < s.length(); i++){
            lastIndex[s.charAt(i) - 'a'] = i; // track the lastIndex of character presence
        }

        boolean[] seen = new boolean[26]; // keep track seen
        Stack<Integer> st = new Stack();

        for (int i = 0; i < s.length(); i++) {
            int curr = s.charAt(i) - 'a';
            if (seen[curr]) continue; // if seen continue as we need to pick one char only
            while (!st.isEmpty() && st.peek() > curr && i < lastIndex[st.peek()]){
                seen[st.pop()] = false; // pop out and mark unseen
            }
            st.push(curr); // add into stack
            seen[curr] = true; // mark seen
        }
```

Testcase | Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

s =
"cbacdcbc"

Output

"acdb"

Expected

"acdb"

Console

Submitted By – Avinash Jena

Aim: Assign Cookies

Objective:

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.

Example 1:

Input: g = [1,2,3], s = [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Example 2:

Input: g = [1,2], s = [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.

You have 3 cookies and their sizes are big enough to gratify all of the children,

You need to output 2.

Code:

```
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        Arrays.sort(g);
        Arrays.sort(s);
        int contentChildren = 0;
        int i = 0;
        int j = 0;
        while (i < g.length && j < s.length) {
            if (s[j] >= g[i]) {
                contentChildren++;
                i++;
            }
            j++;
        }
    return contentChildren;
    }
}
```

Output:

Java • Auto

```java
class Solution {
    public int findContentChildren(int[] g, int[] s) {
        Arrays.sort(g);
        Arrays.sort(s);
        int contentChildren = 0;
        int i = 0;
        int j = 0;
        while (i < g.length && j < s.length) {
            if (s[j] >= g[i]) {
                contentChildren++;
                i++;
            }
            j++;
        }
    return contentChildren;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

g =
[1,2]

s =
[1,2,3]

Output

2

Expected

2

Submitted By – Avinash Jena