EXPERIMENT - 8

Student Name: Sandeep Kumar **UID: 20BCS4885**

Branch: CSE **Section/Group: 603/A**

Semester: 6th semester **Subject: Competitive Coding**

Aim: Greedy Approach

Objective:

a) Remove Duplicates letters
b) Assign Cookies

Problem 1: Remove Duplicates letters

Solution code:

```
class Solution {
public:
    string removeDuplicateLetters(string s) {
        vector<int> cnt(26,0)  , vis(26,0);

        string res = "";
        int n = s.size();

        for(int i = 0; i<n; ++i)
            cnt[s[i] - 'a']++;

        for(int i = 0; i<n; ++i)
        {
            cnt[s[i] - 'a']--;
            if(!vis[s[i]- 'a'])
            {
                while(res.size() > 0 && res.back() > s[i] && cnt[res.back() - 'a'] > 0)
                {
                    vis[res.back() - 'a'] = 0;
                    res.pop_back();
                }

                res += s[i];
                vis[s[i] - 'a'] = 1;
            }
        }
        return res;
    }
};
```

Approach:

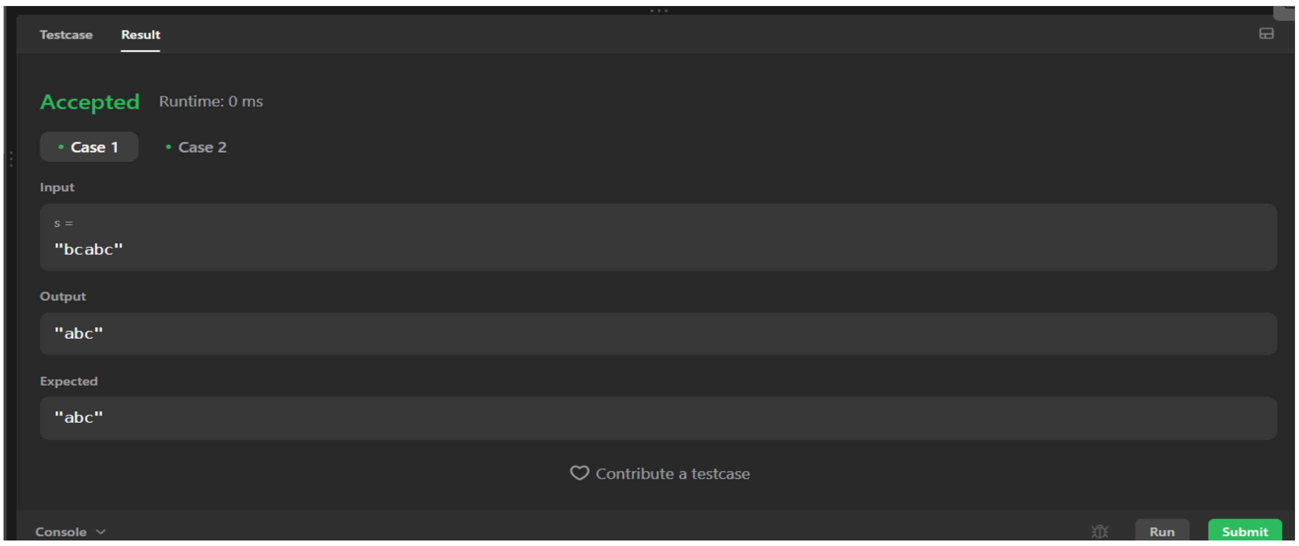
1. Initialize an string res for storing smallest lexicographical order.
2. Store the frequency of each character present in the given string in an vector cnt.
3. Maintain an vector vis for making the characters that are already present in resultant string res.
4. Traverse the given string s and for each character s[i] perform the following operation.
    - Decrease the frequency of the current character by 1.
    - If the current character is not mark as visited in the vector vis, then perform following,
    - If the last letter of res is less than s[i], add s[i] to res.
    - If the last letter of res is greater than s[i] and the cnt of the last letter of res exceeds 0, then remove that character & mark vis[s[i]] as 0 and continue this step till above condition satisfies.
    - After breaking out from above condition, add s[i] to res & mark vis[s[i]] as 1.
5. return res.

Complexity:

Time Complexity: O(n)

Space Complexity: O(n)

Output:



Accepted Runtime: 0 ms

Input

s =
"bcabc"

Output

"abc"

Expected

"abc"

Problem 2: Assign Cookies

Input Code:

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

Approach:

Using Greedy algorithm

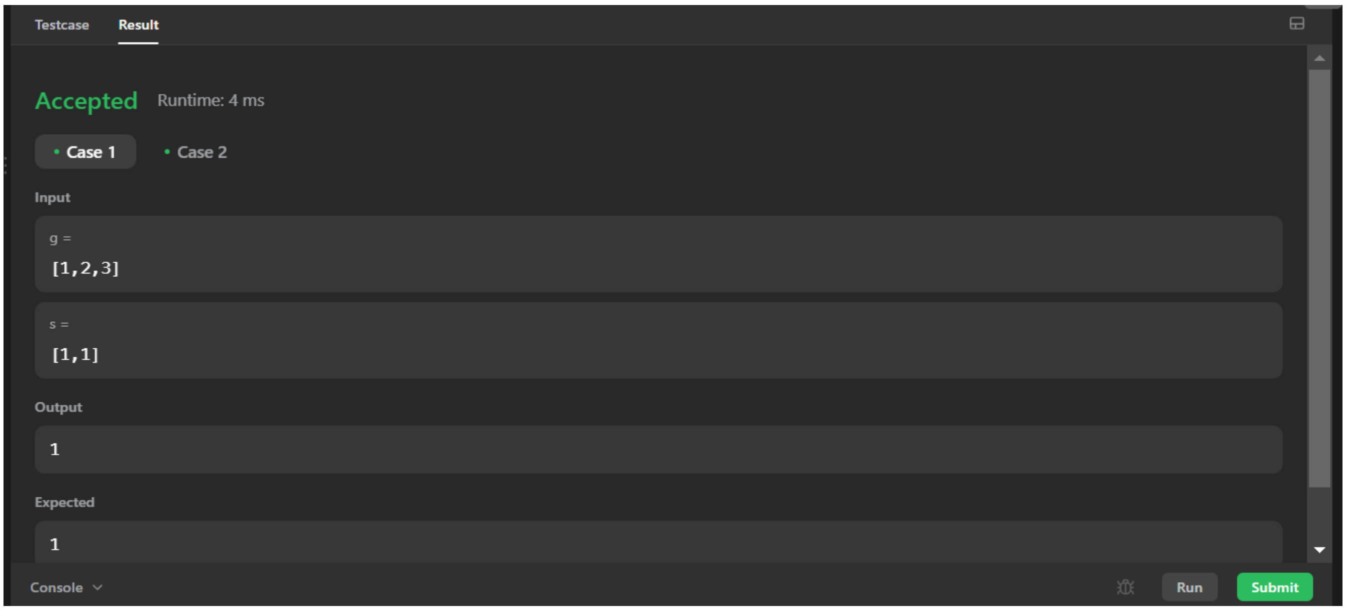
- Sort the greed factors of children and the sizes of cookies in non-decreasing order.
- Initialize a variable contentChildren to 0 to keep track of the number of children that can be content with a cookie.
- Use two pointers i and j to point to the current child's greed factor and the current cookie size, respectively.
- Iterate over the two sorted arrays, and for each cookie size, check if it can satisfy the current child's greed factor. If so, assign the cookie to the child, increment contentChildren and move on to the next child and cookie. If not, move on to the next cookie.
- Return contentChildren.

Complexity:

Time Complexity: O(n)

Space Complexity: O(1)

Output:



Accepted Runtime: 4 ms

Input

```
g =
[1,2,3]
```

```
s =
[1,1]
```

Output

```
1
```

Expected

```
1
```