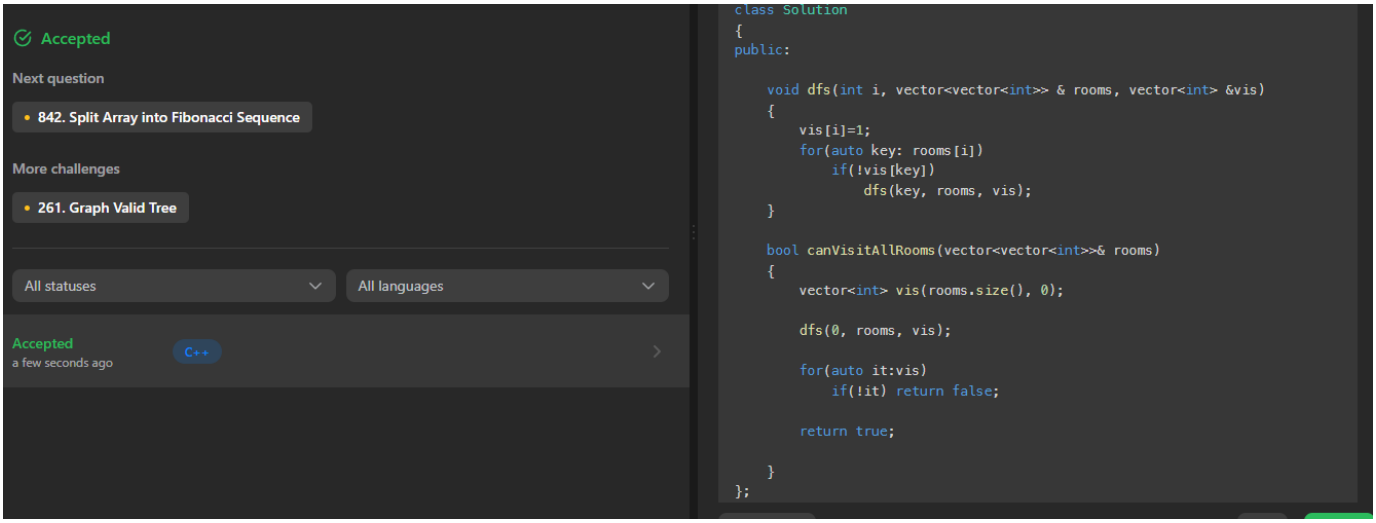
WORKSHEET 8

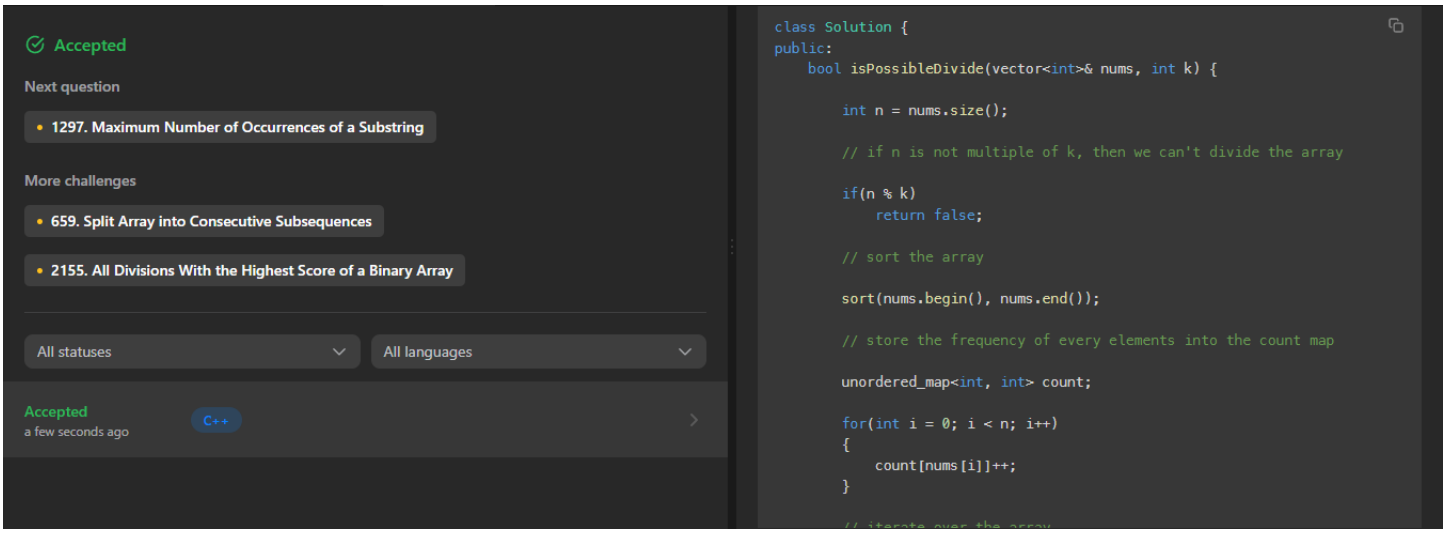
Student Name: Jannat Baweja **UID:** 20BCS2050
DOMAIN CAMP: 03-01-2023 to 14-01-2023 **Section/Group:** DWWC-43
Subject Name: IT Skills (DSA)

Question 1. KEYS AND ROOMS (DIVIDE AND CONQUER)




```
class Solution
{
public:

    void dfs(int i, vector<vector<int>> & rooms, vector<int> &vis)
    {
        vis[i]=1;
        for(auto key: rooms[i])
            if(!vis[key])
                dfs(key, rooms, vis);
    }

    bool canVisitAllRooms(vector<vector<int>>& rooms)
    {
        vector<int> vis(rooms.size(), 0);

        dfs(0, rooms, vis);

        for(auto it:vis)
            if(!it) return false;

        return true;

    }
};
```

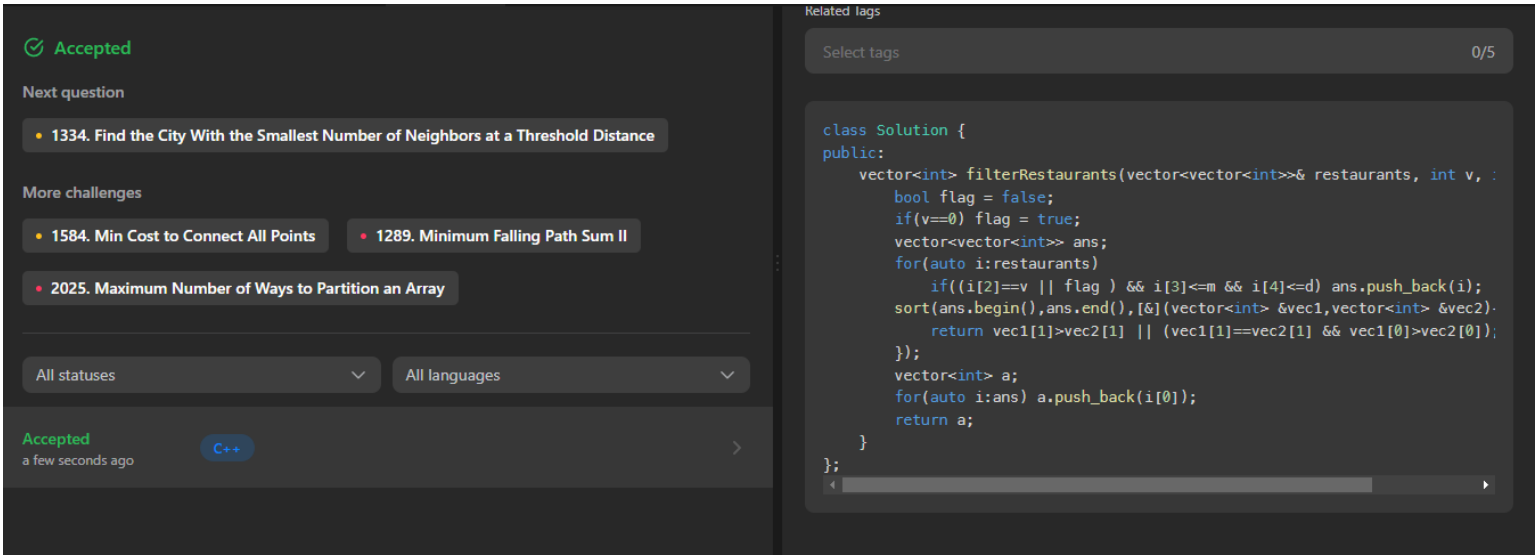
Question 2. DIVIDE ARRAY IN SETS OF K CONSECUTIVE NUMBERS (DIVIDE AND CONQUER)



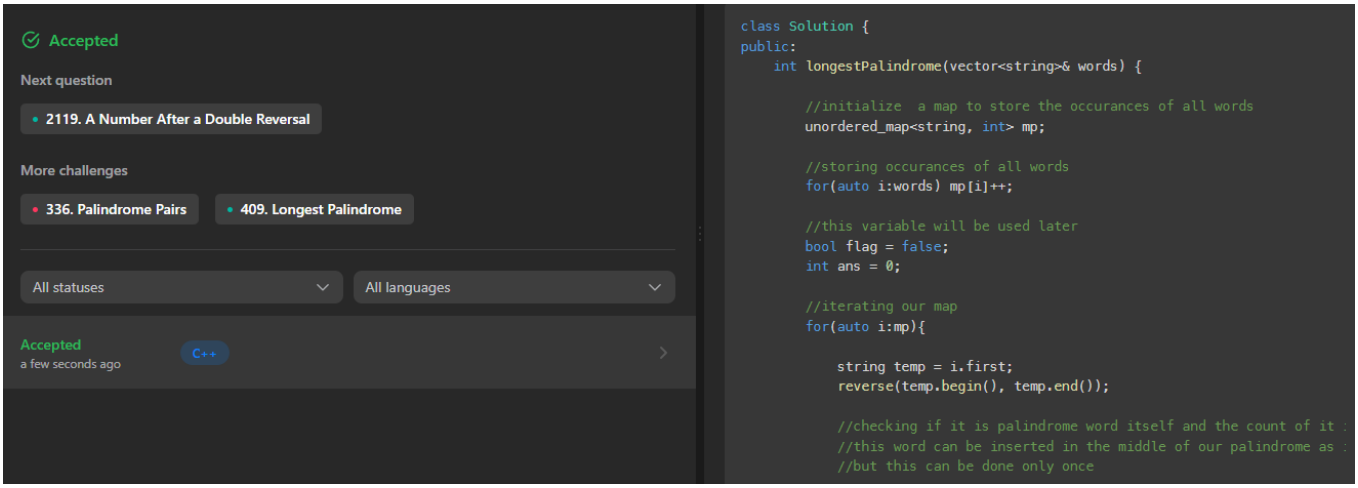

```
class Solution {
public:
    bool isPossibleDivide(vector<int>& nums, int k) {

        int n = nums.size();

        // if n is not multiple of k, then we can't divide the array

        if(n % k)
            return false;

        // sort the array

        sort(nums.begin(), nums.end());

        // store the frequency of every elements into the count map

        unordered_map<int, int> count;

        for(int i = 0; i < n; i++)
        {
            count[nums[i]]++;
        }

        // iterate over the array
```

```
        // iterate over the array

        for(int i = 0; i < n; i++)
        {
            // if all the occurance of curr element is include

            if(count[nums[i]] == 0)
                continue;

            // decrement the count of occurance of curr element

            count[nums[i]]--;

            // check can we make a set of k consecutive numbers

            for(int j = 1; j < k; j++)
            {
                // if nums[i] + j is not present in count map, then we can't make a set of k cons

                if(count[nums[i] + j] == 0)
                    return false;

                // decrement the count of occurance of nums[i] + j element

                count[nums[i] + j]--;
            }
        }

        return true;
    }
};
```

Question 3. FILTER RESTAURANTS BY VEGAN-FRIENDLY, PRICE AND DISTANCE (DIVIDE AND CONQUER)

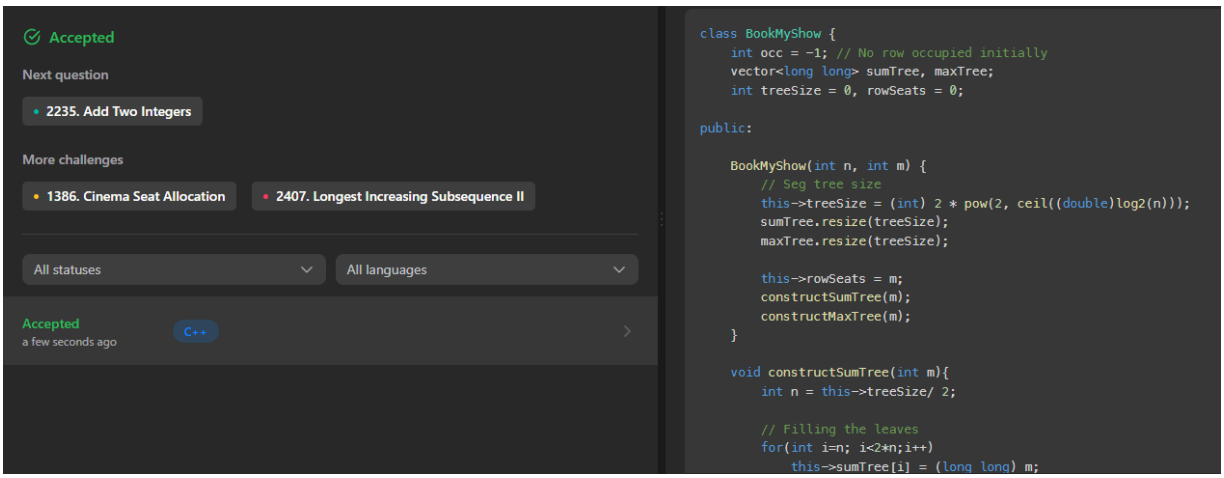


Question 4. LONGEST PALINDROME BY CONCATENATING TWO LETTER WORDS (DIVIDE AND CONQUER)




```cpp
class Solution {
public:
    int longestPalindrome(vector<string>& words) {

        //initialize  a map to store the occurances of all words
        unordered_map<string, int> mp;

        //storing occurances of all words
        for(auto i:words) mp[i]++;

        //this variable will be used later
        bool flag = false;
        int ans = 0;

        //iterating our map
        for(auto i:mp){

            string temp = i.first;
            reverse(temp.begin(), temp.end());

            //checking if it is palindrome word itself and the count of it
            //this word can be inserted in the middle of our palindrome as
            //but this can be done only once

            if(temp == i.first and i.second%2!=0 and flag == false){
                //we mark our flag to true because we want to insert this element onl
                flag = true;
                ans+=2;
                mp[temp]--;
            }

            //checking if the word as well its reverse exist in the map
            if(mp.find(i.first)!=mp.end() and mp.find(temp)!=mp.end()){

                //for the case when the word is not palindrome we will increase our a
                if(i.first!=temp){
                    ans += min(mp[i.first], mp[temp])*2*2;
                    mp[i.first]=mp[temp]=0;
                }
                //for the case when it is palindrome we just divide the count of the
                else{
                    ans+=mp[temp]/2 *2*2;
                    mp[i.first]=mp[temp]=0;

                }
            }
        }
        return ans;

    }
};
```

Question 5. BOOKING CONCERT TICKETS IN GROUPS (DIVIDE AND CONQUER)




```cpp
class BookMyShow {
    int occ = -1; // No row occupied initially
    vector<long long> sumTree, maxTree;
    int treeSize = 0, rowSeats = 0;

public:

    BookMyShow(int n, int m) {
        // Seg tree size
        this->treeSize = (int) 2 * pow(2, ceil((double)log2(n)));
        sumTree.resize(treeSize);
        maxTree.resize(treeSize);

        this->rowSeats = m;
        constructSumTree(m);
        constructMaxTree(m);
    }

    void constructSumTree(int m){
        int n = this->treeSize/ 2;

        // Filling the leaves
        for(int i=n; i<2*n;i++)
            this->sumTree[i] = (long long) m;
```

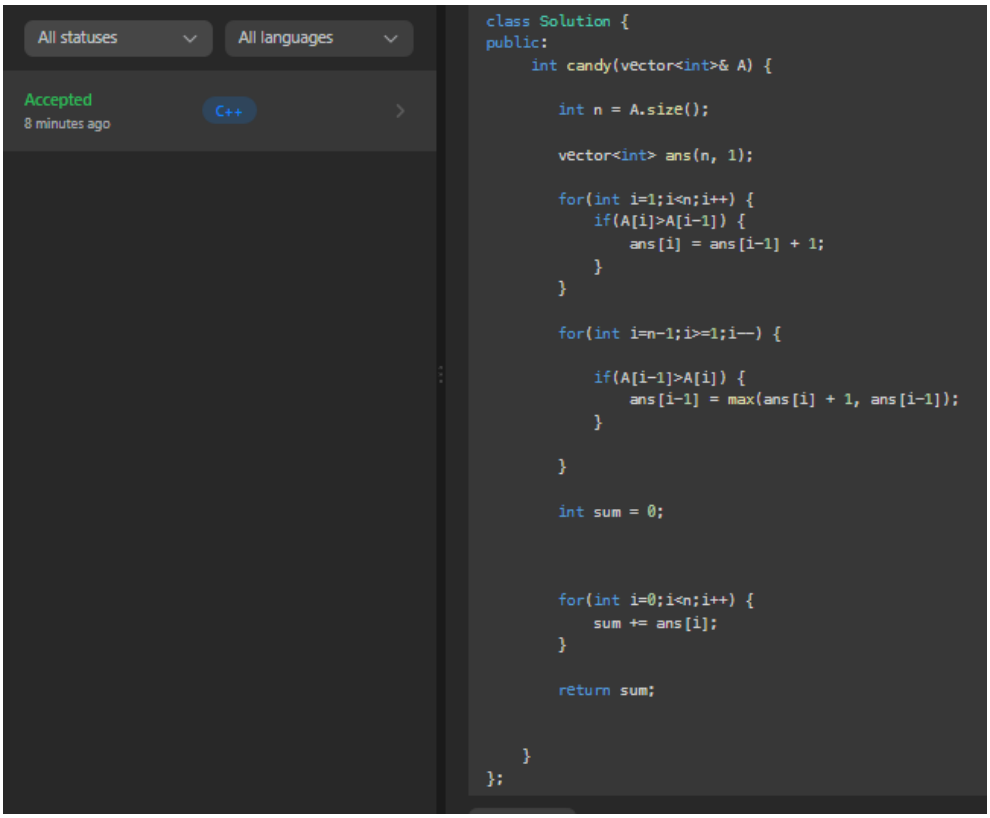
```
        // Forming the higher nodes
        for(int i=n-1;i>=1;i--)
            this->sumTree[i] = this->sumTree[2*i] + this->sumTree[2*i+1];
    }

    void constructMaxTree(int m){
        int n = this->treeSize/ 2;

        // Filling the leaves
        for(int i=n; i<2*n;i++)
            this->maxTree[i] = (long long) m;

        // Forming the higher nodes
        for(int i=n-1;i>=1;i--)
            this->maxTree[i] = max(this->maxTree[2*i], this->maxTree[2*i+1]);
    }

    long long rangeSum(int minRow, int maxRow){
        long long sum = 0;
        int n = treeSize / 2;
        minRow += n; maxRow += n;
        while(minRow <= maxRow){
            if(minRow % 2 == 1) sum += this->sumTree[minRow++]; // Right ch
            if(maxRow % 2 == 0) sum += this->sumTree[maxRow--]; // Left chi
            minRow /= 2; maxRow /= 2;
        }
```

```
        return sum;
    }

    void updateSumTree(int index, int newValue){
        int n = this->treeSize / 2;
        int temp = index;
        index += n;
        this->sumTree[index] = newValue; // Update leaf
        index /= 2;
        while(index > 0){
            this->sumTree[index] = this->sumTree[2*index] + this->sumTree[2
            index /= 2;
        }
    }

    long long rangeMax(int minRow, int maxRow){
        long long ans = 0;
        int n = this->treeSize / 2;
        minRow += n; maxRow += n;
        while(minRow <= maxRow){
            if(minRow % 2 == 1) ans = max(ans, this->maxTree[minRow++]); //
            if(maxRow % 2 == 0) ans = max(ans, this->maxTree[maxRow--]); //
            minRow /= 2; maxRow /= 2;
        }
```

```
        return ans;
    }

    void updateMaxTree(int index, int newValue){
        int n = this->treeSize / 2;
        int temp = index;
        index += n;
        this->maxTree[index] = newValue; // Update leaf
        index /= 2;
        while(index > 0){
            this->maxTree[index] = max(this->maxTree[2*index], this->maxTree[2*index + 1]);
            index /= 2;
        }
    }

    vector<int> gather(int k, int maxRow) {
        int minRow = occ + 1;
        if(maxRow < minRow) return {};
        if(rangeMax(minRow, maxRow) < k) return {};

        int minIndex = maxRow;
        int seats = 0;
        int low = minRow, high = maxRow;
        while(low <= high){
            int midRow = (low + high)/2;
            int maxSeats = rangeMax(minRow, midRow);
            if(maxSeats >= k){
                high = midRow - 1;
                seats = maxSeats;
                minIndex = midRow;
            }
            else low = midRow + 1;
        }
```

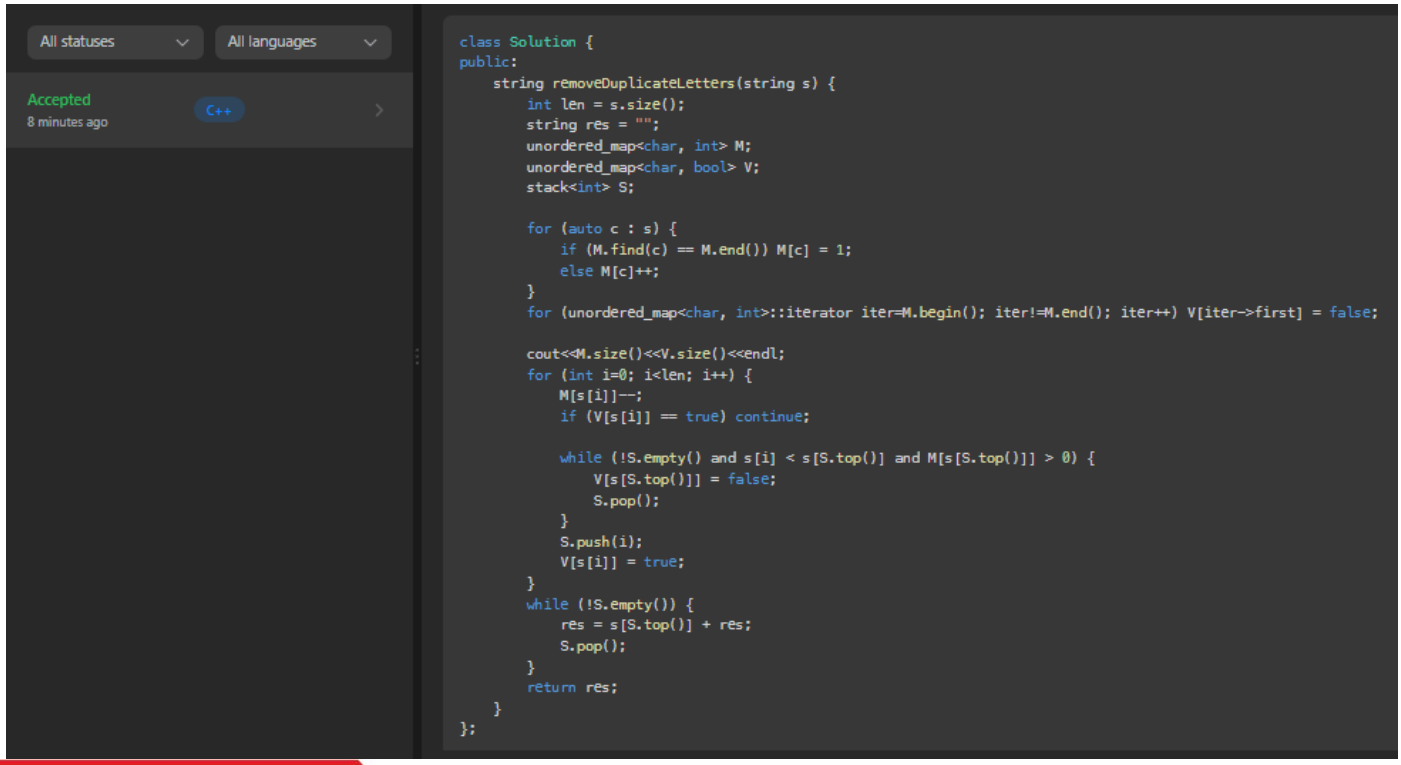
```
        int r = minIndex, c = this->rowSeats - seats;

        // Updatng the segment trees
        this->updateMaxTree(minIndex, seats - k);
        this->updateSumTree(minIndex, seats - k);
        return {r,c};
    }

    bool scatter(int k, int maxRow) {
        int minRow = occ + 1;
        if(maxRow < minRow) return false;
        if(rangeSum(minRow, maxRow) < k) return false;

        int minIndex = maxRow;
        long long seats = 0;
        int low = minRow, high = maxRow;
        while(low <= high){
            int midRow = (low + high)/2;
            long long rangeSeats = rangeSum(minRow, midRow);
            if(rangeSeats >= k){
                high = midRow - 1;
                seats = rangeSeats;
                minIndex = midRow;
            }
            else low = midRow + 1;
        }

        // Updating the occupied rows
        occ = minIndex - 1;

        // Updating the segment trees
        this->updateSumTree(minIndex, seats - k);
        this->updateMaxTree(minIndex, seats - k);
        return true;
    }
};
```

Question 6. CANDY (GREEDY APPROACH)



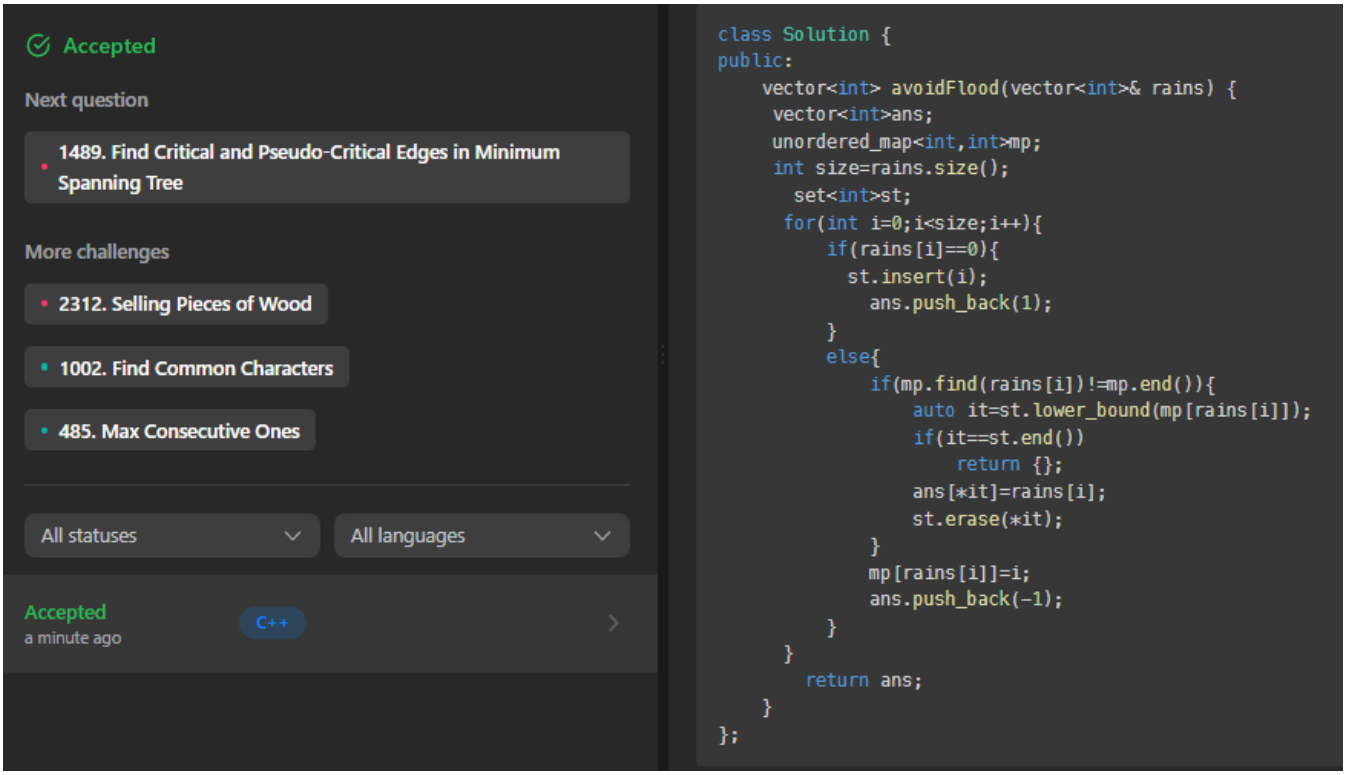

```cpp
class Solution {
public:
    int candy(vector<int>& A) {

        int n = A.size();

        vector<int> ans(n, 1);

        for(int i=1;i<n;i++) {
            if(A[i]>A[i-1]) {
                ans[i] = ans[i-1] + 1;
            }
        }

        for(int i=n-1;i>=1;i--) {

            if(A[i-1]>A[i]) {
                ans[i-1] = max(ans[i] + 1, ans[i-1]);
            }

        }

        int sum = 0;



        for(int i=0;i<n;i++) {
            sum += ans[i];
        }

        return sum;


    }
};
```

Question 7. REMOVE DUPLICATE LETTERS (GREEDY APPROACH)



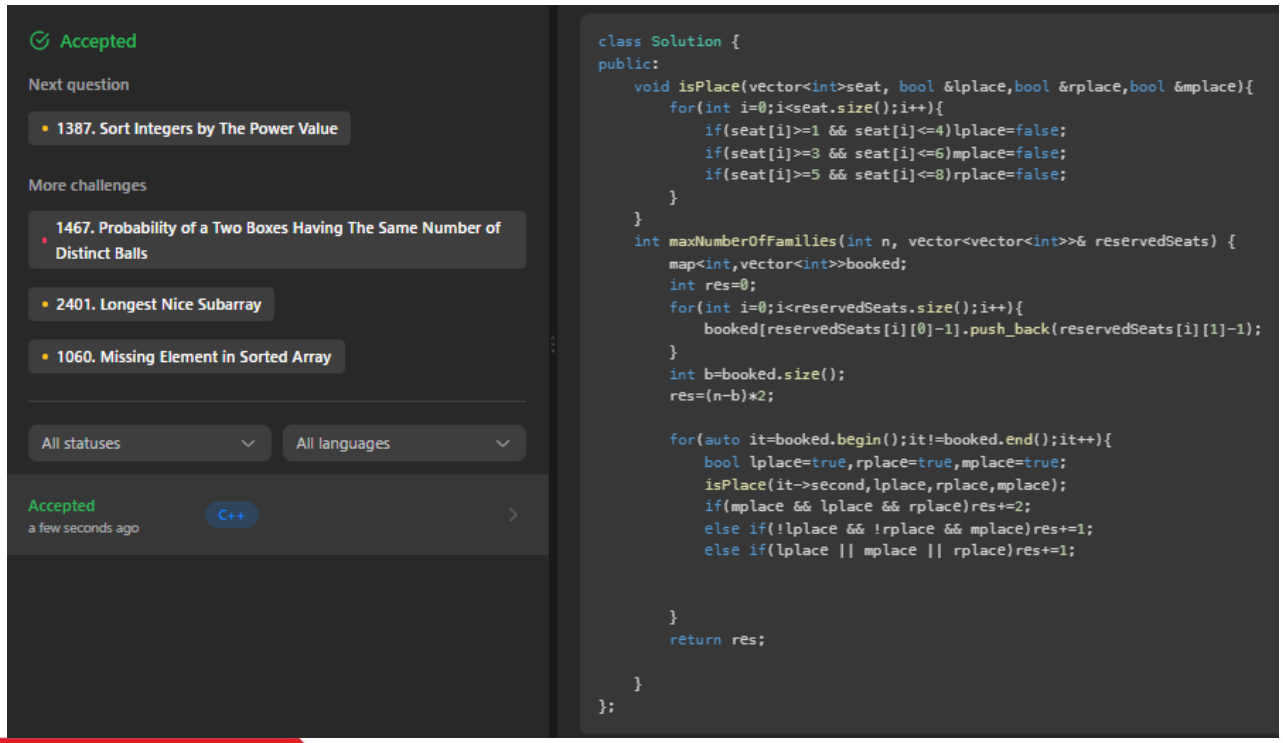

```cpp
class Solution {
public:
    string removeDuplicateLetters(string s) {
        int len = s.size();
        string res = "";
        unordered_map<char, int> M;
        unordered_map<char, bool> V;
        stack<int> S;

        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;
    }
};
```

Question 8. AVOID FLOOD IN THE CITY (GREEDY APPROACH)




```cpp
class Solution {
public:
    vector<int> avoidFlood(vector<int>& rains) {
     vector<int>ans;
     unordered_map<int,int>mp;
     int size=rains.size();
       set<int>st;
      for(int i=0;i<size;i++){
          if(rains[i]==0){
            st.insert(i);
              ans.push_back(1);
          }
          else{
              if(mp.find(rains[i])!=mp.end()){
                  auto it=st.lower_bound(mp[rains[i]]);
                  if(it==st.end())
                      return {};
                  ans[*it]=rains[i];
                  st.erase(*it);
              }
              mp[rains[i]]=i;
              ans.push_back(-1);
          }
      }
        return ans;
    }
};
```

Question 9. CINEMA SEAT ALLOCATION (GREEDY APPROACH)




```cpp
class Solution {
public:
    void isPlace(vector<int>seat, bool &lplace,bool &rplace,bool &mplace){
        for(int i=0;i<seat.size();i++){
            if(seat[i]>=1 && seat[i]<=4)lplace=false;
            if(seat[i]>=3 && seat[i]<=6)mplace=false;
            if(seat[i]>=5 && seat[i]<=8)rplace=false;
        }
    }
    int maxNumberOfFamilies(int n, vector<vector<int>>& reservedSeats) {
        map<int,vector<int>>booked;
        int res=0;
        for(int i=0;i<reservedSeats.size();i++){
            booked[reservedSeats[i][0]-1].push_back(reservedSeats[i][1]-1);
        }
        int b=booked.size();
        res=(n-b)*2;

        for(auto it=booked.begin();it!=booked.end();it++){
            bool lplace=true,rplace=true,mplace=true;
            isPlace(it->second,lplace,rplace,mplace);
            if(mplace && lplace && rplace)res+=2;
            else if(!lplace && !rplace && mplace)res+=1;
            else if(lplace || mplace || rplace)res+=1;


        }
        return res;

    }
};
```

Question 10. RABBITS IN FOREST (GREEDY APPROACH)