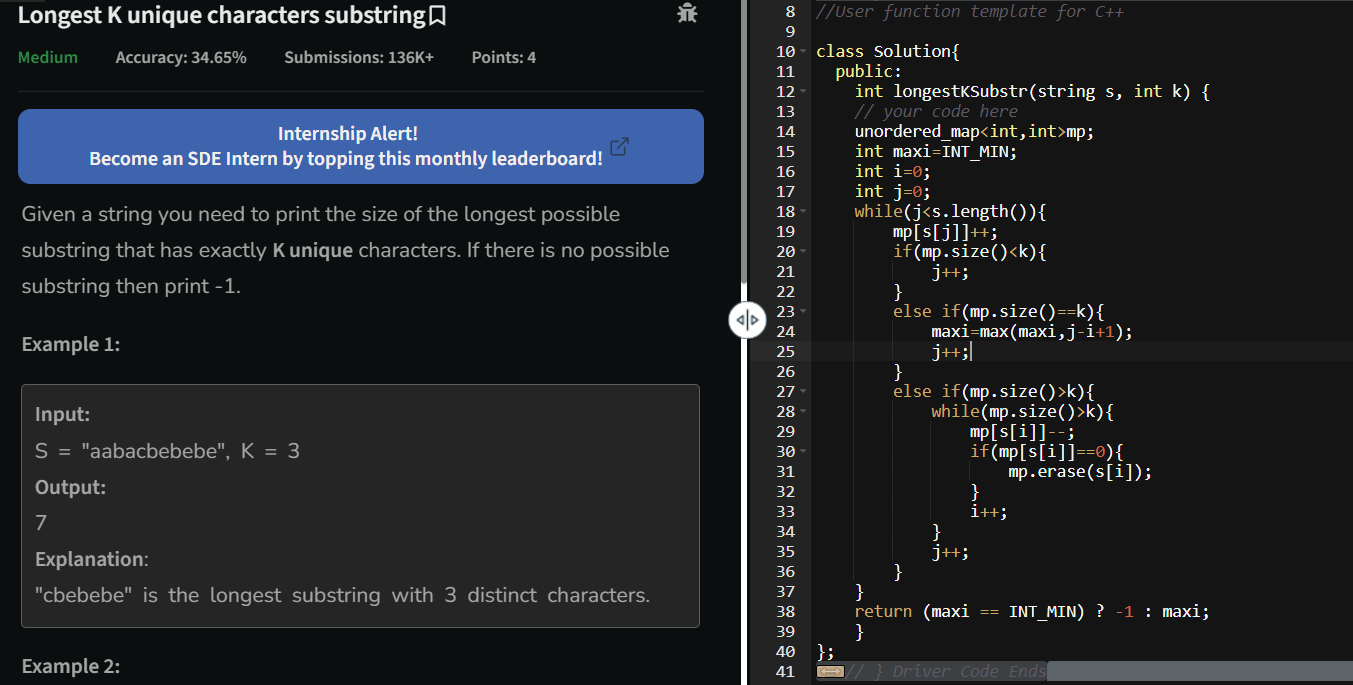
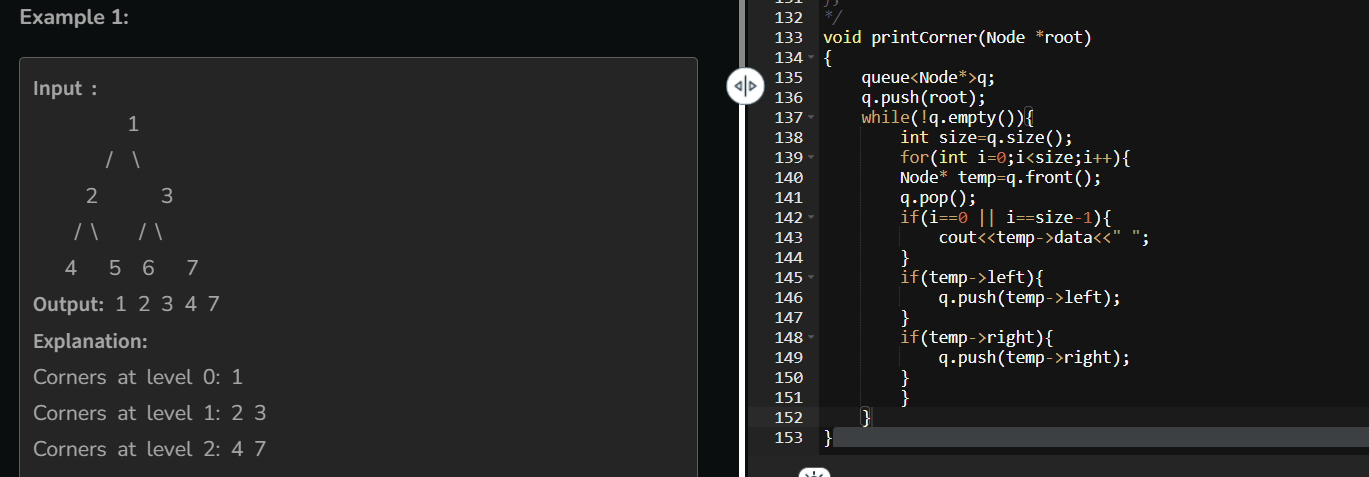
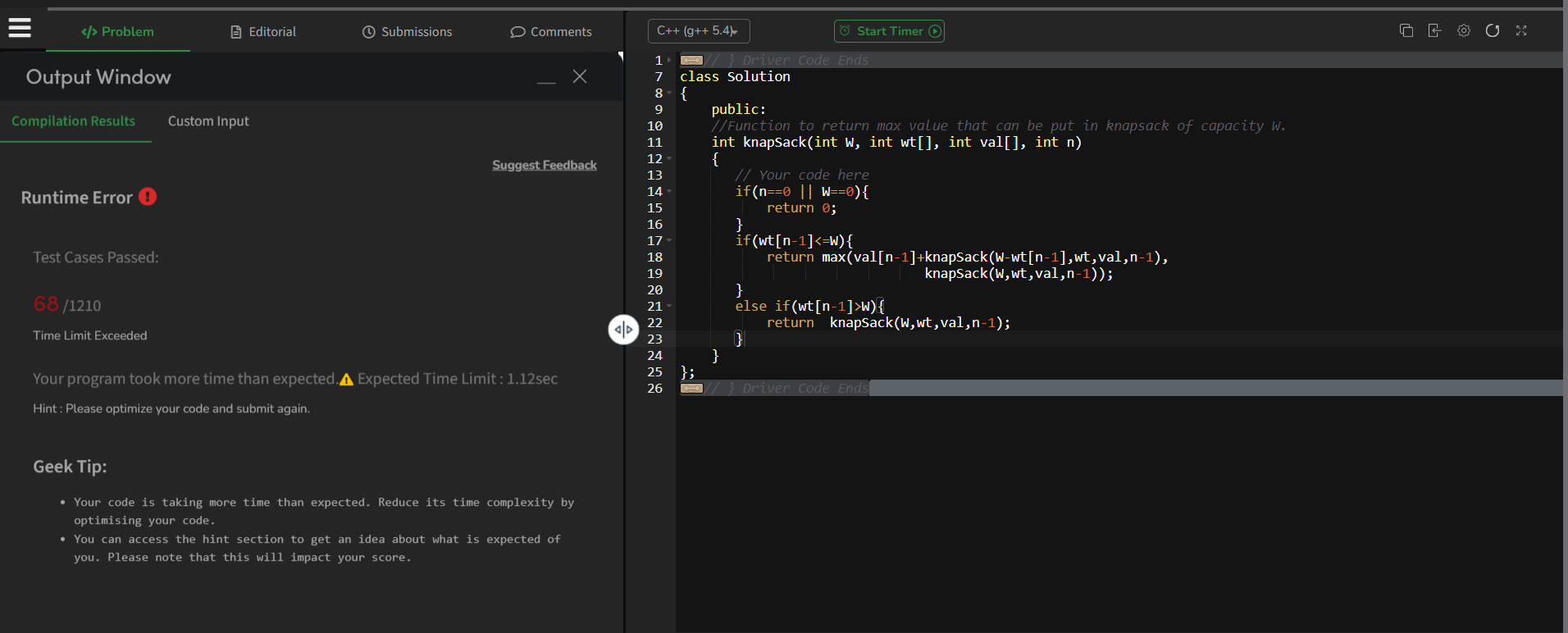
**Longest K unique Characters Substring:**



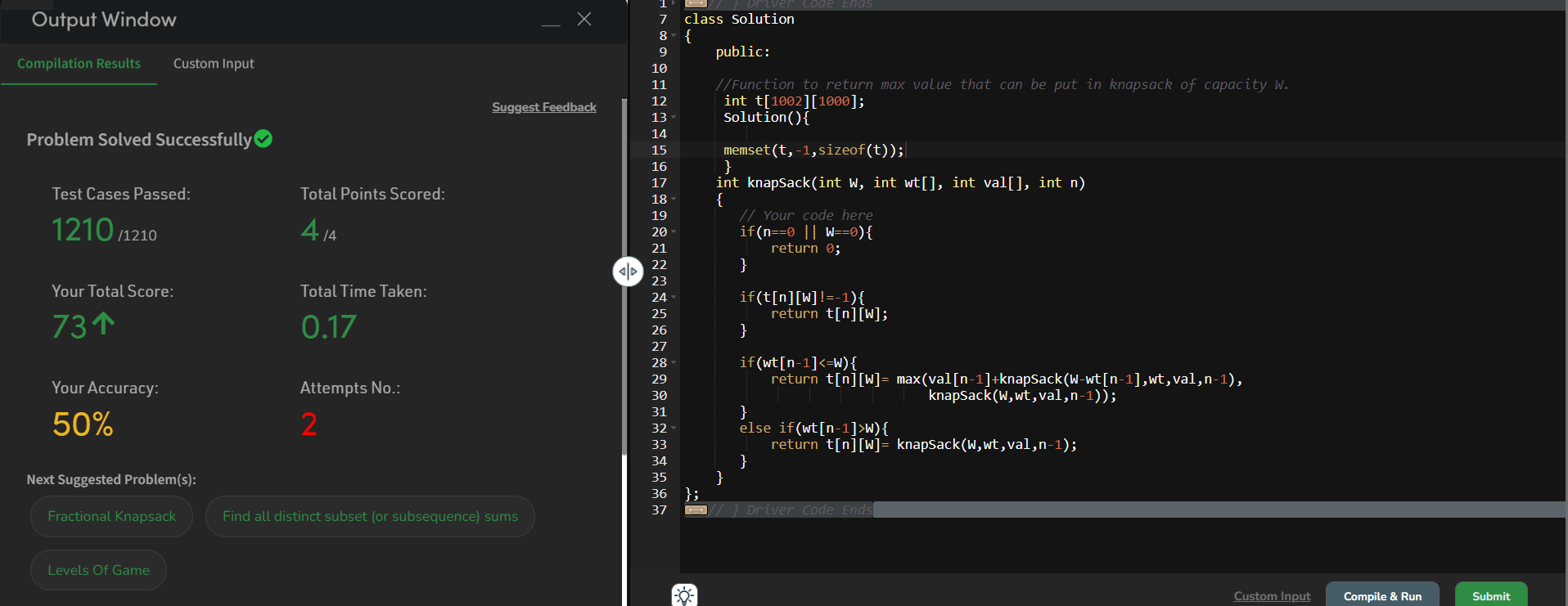
**Print corners of A Binary Tree**

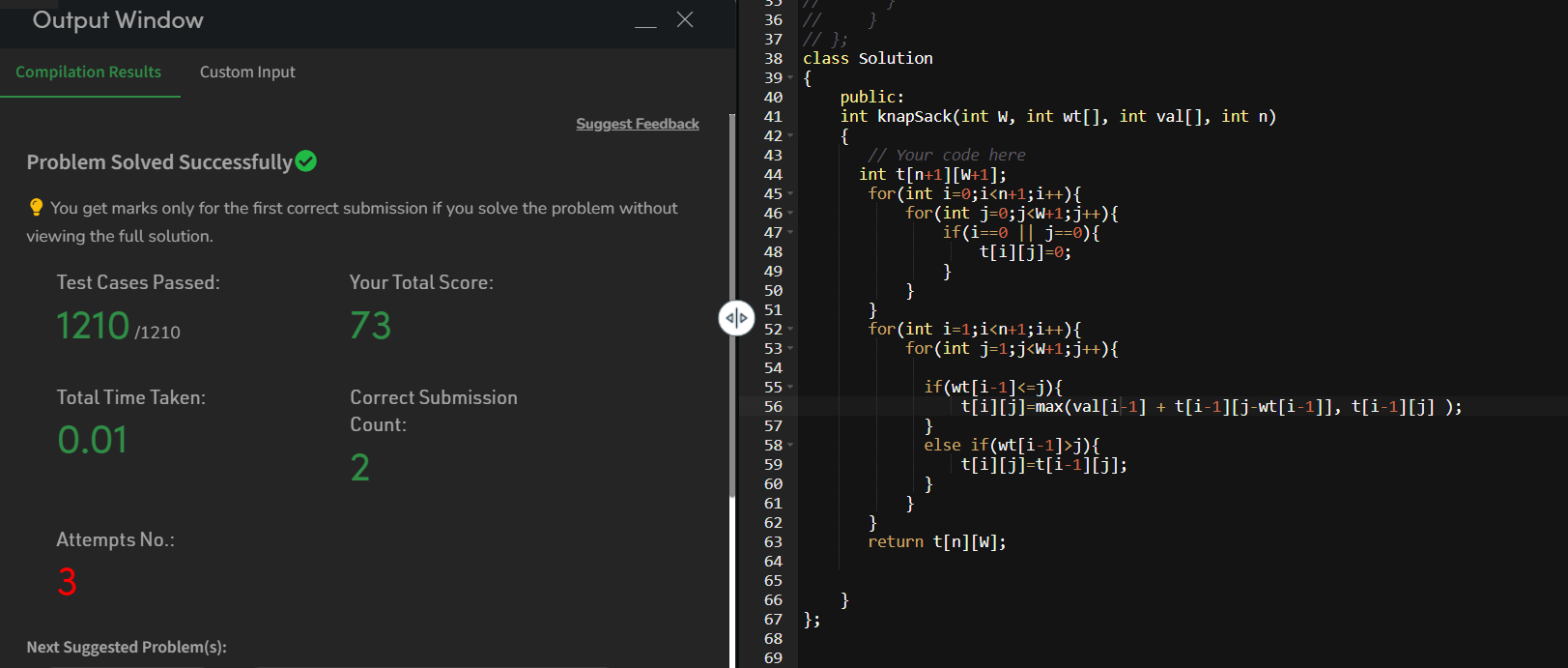


**0 -1 Knapsack**

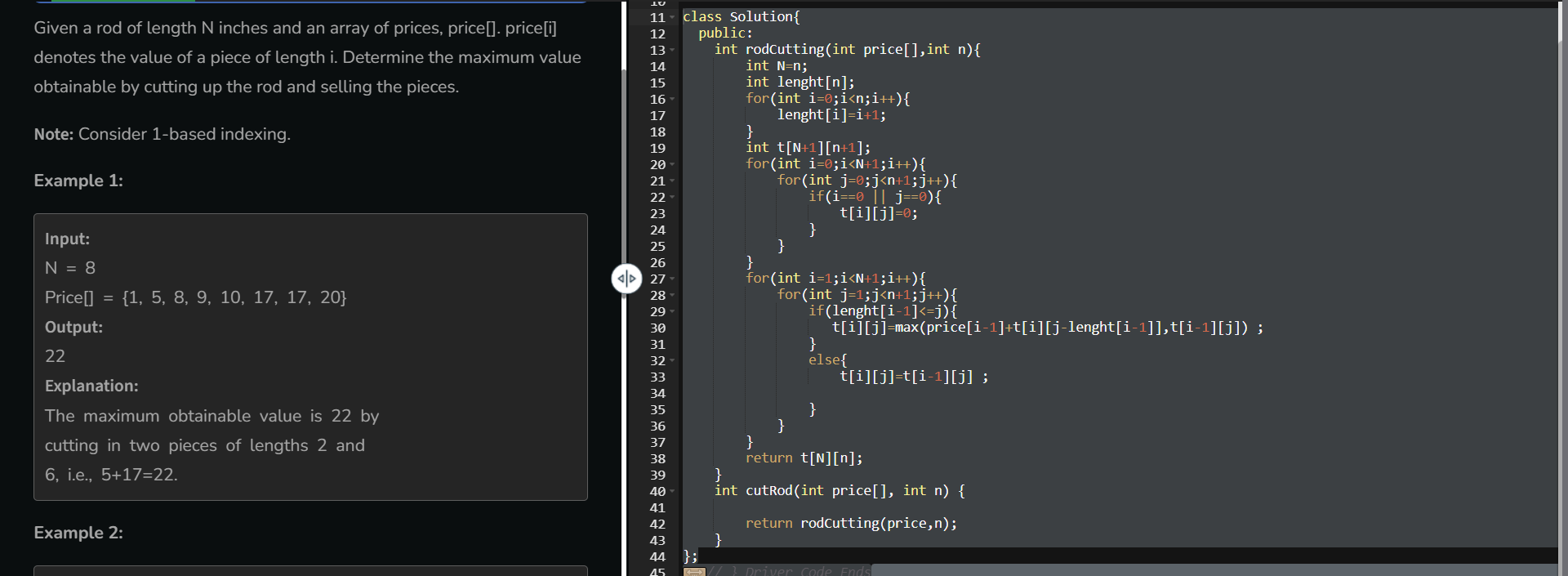
****

**Memoization**

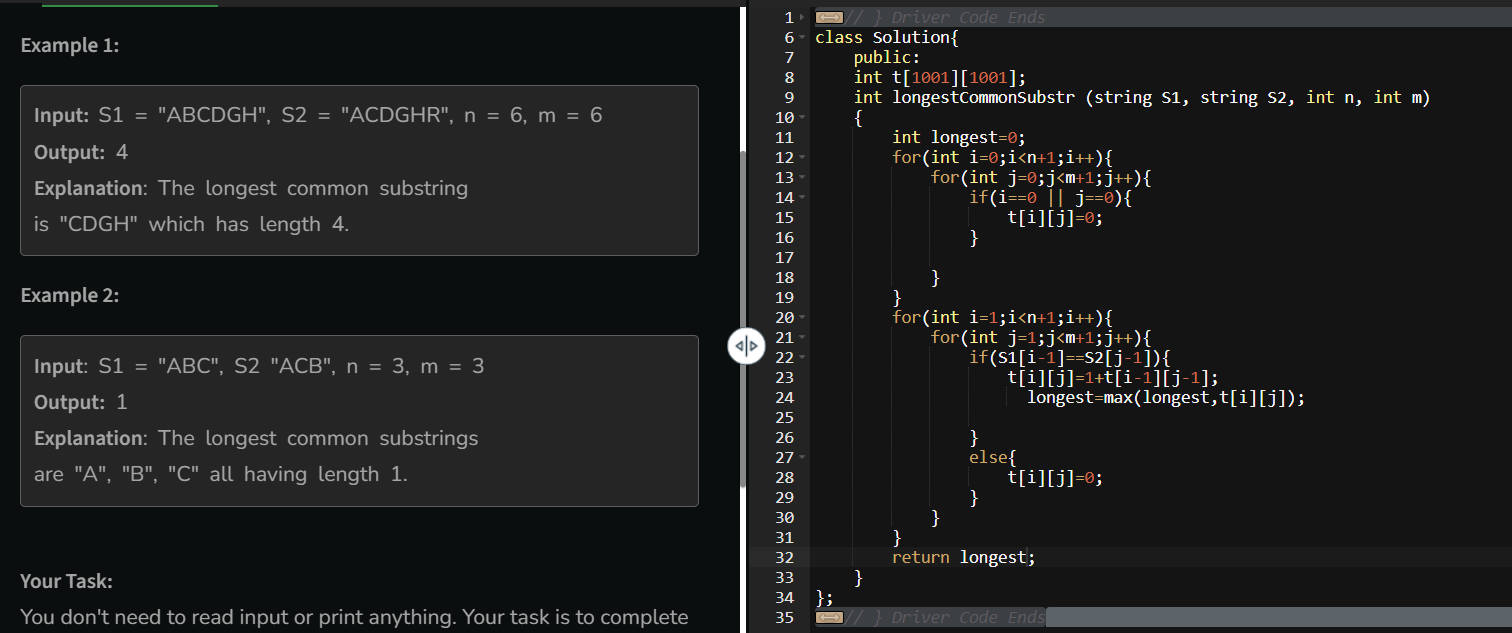
****

**Top Down**

**Rod Cutting Unbounded Knapsack**

****

**Longest Common Substr:**

****

**Palindromic Partition fully optimized**

class Solution{

public:

    int t[501][501];

    bool palindrome(string str,int i,int j){

        if(i>=j || i==j){

            return true;

        }

        while(i<j){

            if(str[i]!=str[j]){

                return false;

            }

            i++;

            j--;

        }

        return true;

    }

    int solve(string str,int i,int j){

        int mini=INT\_MAX;

        if(i>=j){

            return 0;

        }

        if(palindrome(str,i,j)){

            return t[i][j]=0;

        }

        if(t[i][j]!=-1){

            return t[i][j];

        }

        for(int k=i;k<=j-1;k++){

            // int temp=solve(str,i,k)+solve(str,k+1,j) +1;

            int left,right;

            if(t[i][k]!=-1){

                left=t[i][k];

            }

            else{

                left=solve(str,i,k);

                t[i][k]=left;

            }

            if(t[k+1][j]!=-1){

                right=t[k+1][j];

            }

            else{

                right=solve(str,k+1,j);

                t[k+1][j]=right;

            }

            int temp= 1+left+right;

            mini=min(mini,temp);

        }

        return t[i][j]=mini;

    }

    int palindromicPartition(string str)

    {

        // code here

        int i=0;

        int j=str.length()-1;

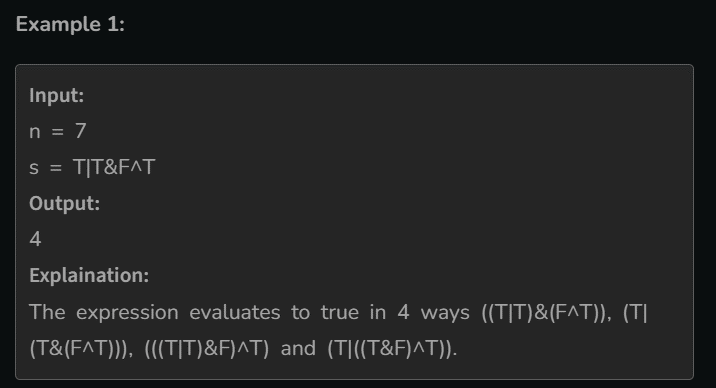
        memset(t,-1,sizeof(t));

        return solve(str,i,j);

    }

};

**Boolean Parenthesization**



**Recursive code With TLE**

class Solution{

public:

    int solve(string s,int i,int j, bool isTrue){

        if(i>j)return false;

        if(i==j){

            if(isTrue==true){

                return s[i]=='T';

            }else{

                return s[i]=='F';

            }

        }

        int ans=0;

        for(int k=i+1;k<=j-1;k+=2){

            int LT=solve(s,i,k-1,true);

            int LF=solve(s,i,k-1,false);

            int RT=solve(s,k+1,j,true);

            int RF=solve(s,k+1,j,false);

            if(s[k]=='&'){

                if(isTrue==true){

                    ans=ans + LT\*RT;

                }

                else{

                    ans=ans+ LF\*RF + LT\*RF + LF\*RT;

                }

            }

            if(s[k]=='|'){

                if(isTrue==true){

                    ans=ans + LT\*RT + LT\*RF + LF\*RT;

                }

                else{

                    ans=ans+ LF\*RF ;

                }

            }

            if(s[k]=='^'){

                if(isTrue==true){

                    ans=ans + LT\*RF + LF\*RT;

                }

                else{

                    ans=ans+ LF\*RF + LT\*RT ;

                }

            }

        }

        return ans;

    }

    int countWays(int n, string s){

        int i=0;

        int j=s.length()-1;

        // bool isTrue=true;

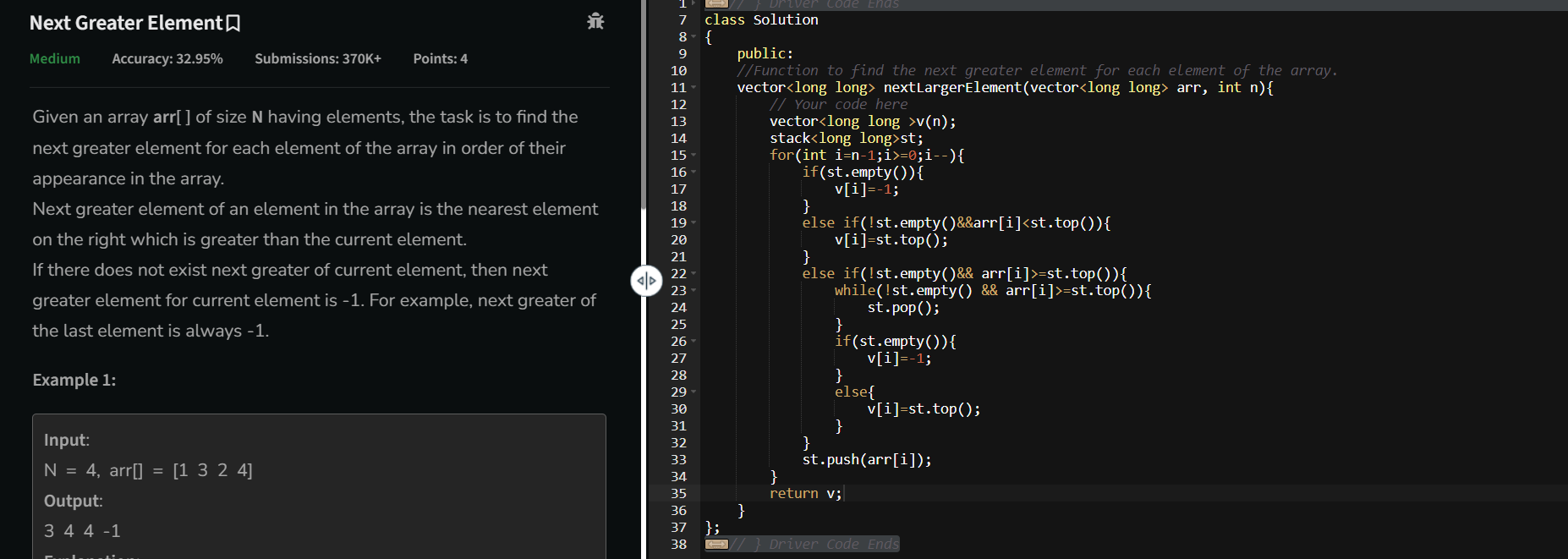
        return solve(s,i,j,true);

    }

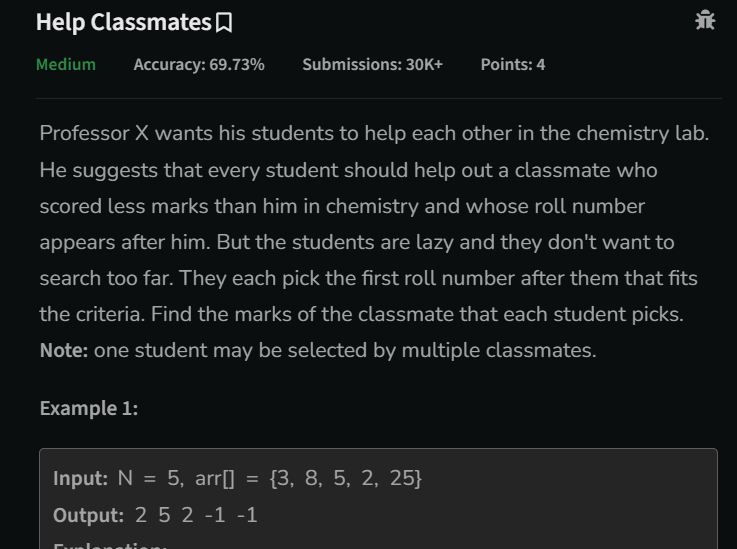
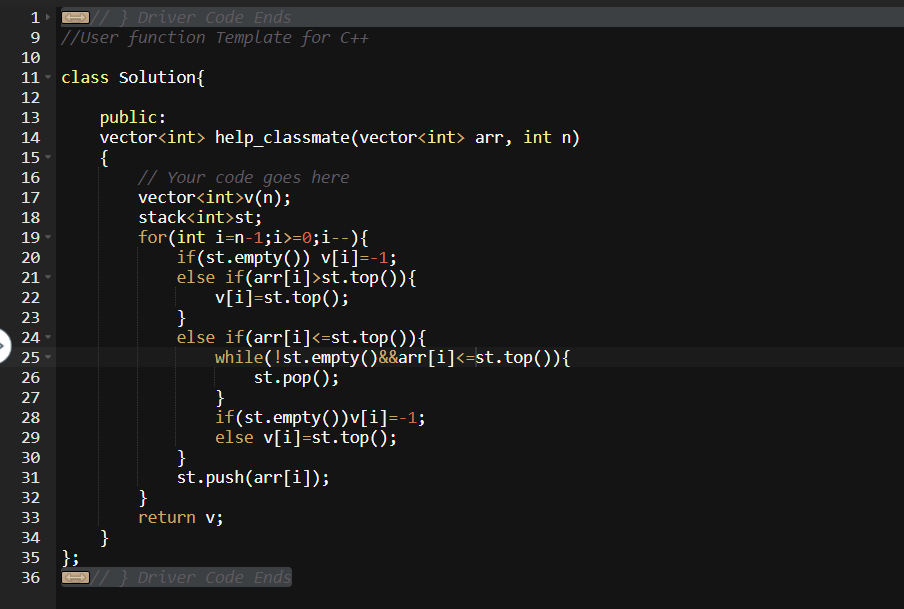
};

|  |  |
| --- | --- |
|  |  |

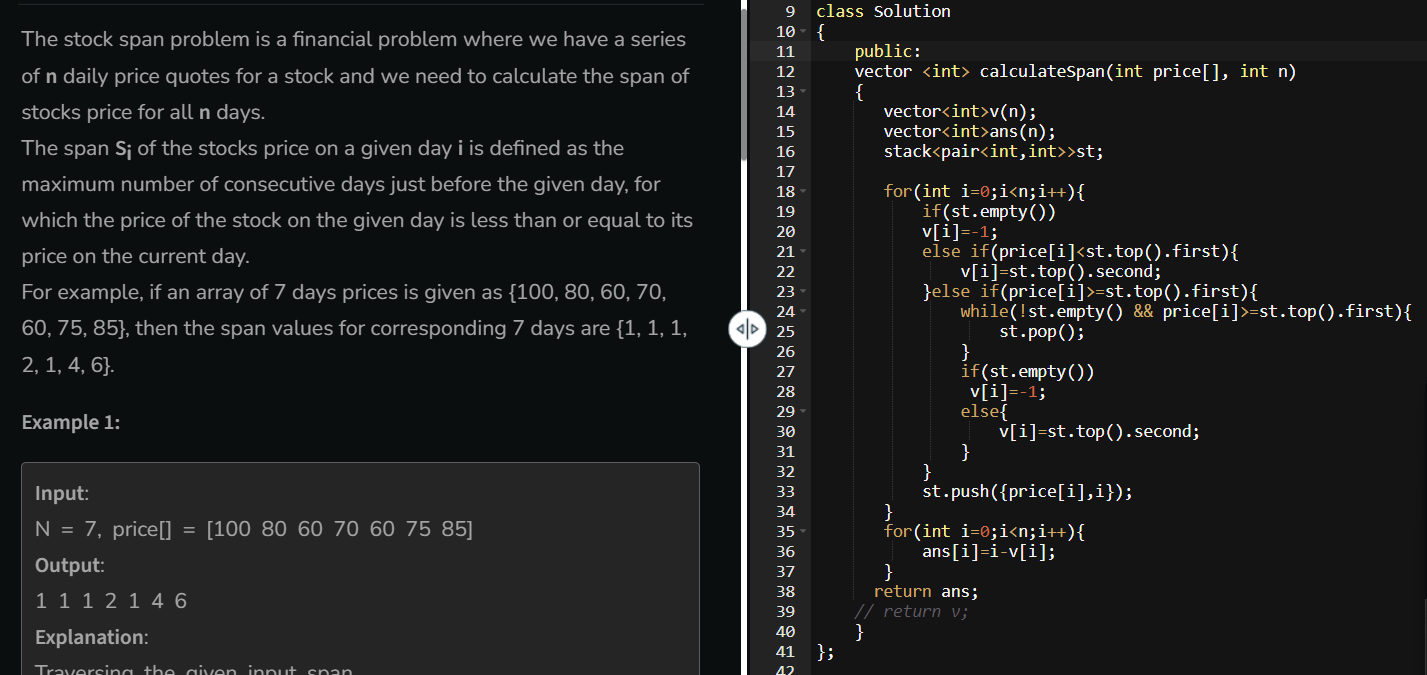
**Next greater Element**

****

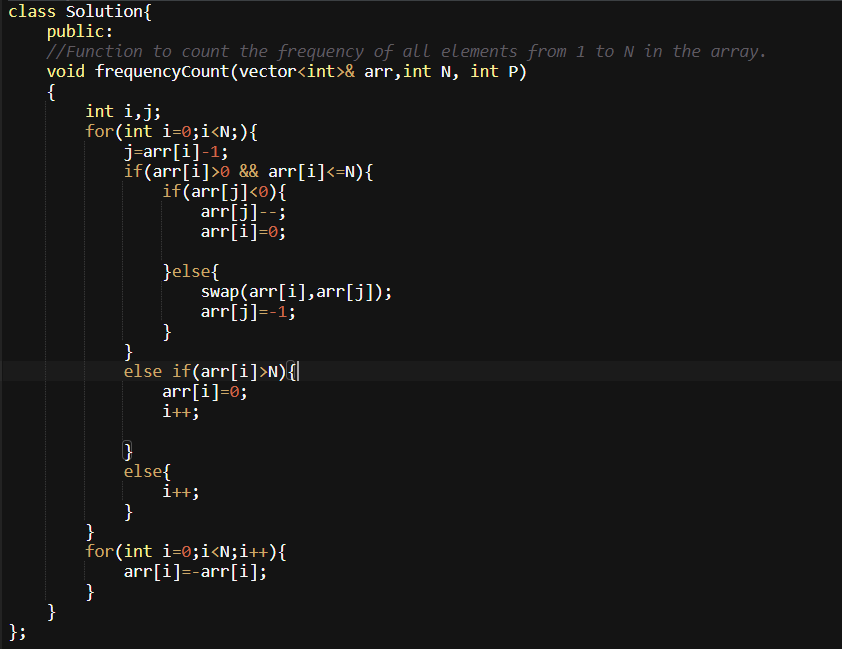
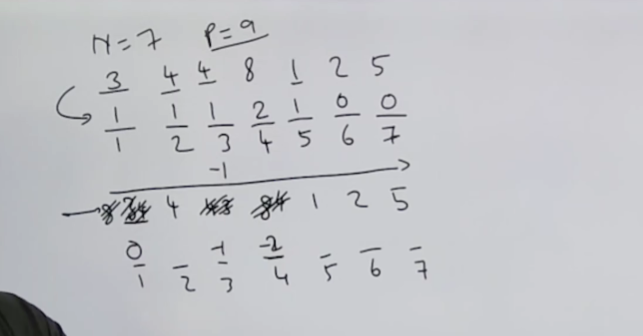
**Nearest Smaller to right**

****

**Stock span problem / variation of nearest greater left**

****

**Frequencies of Limited Range Array Elements**

****

**Insertion in Singly LL**

SinglyLinkedListNode\* insertatHead(SinglyLinkedListNode\* llist, int data){

     auto temp=new  SinglyLinkedListNode(data);

     temp->next=llist;

     llist=temp;

     return llist;

 }

 SinglyLinkedListNode\* insertAttail(SinglyLinkedListNode\* llist, int data){

     auto n=new  SinglyLinkedListNode(data);

     if(llist==NULL){

         llist=n;

         return llist;

     }

     auto temp=llist;

     while(temp->next!=NULL){

         temp=temp->next;

     }

     temp->next=n;

     return llist;

 }

SinglyLinkedListNode\* insertNodeAtPosition(SinglyLinkedListNode\* llist, int data, int position) {

    if(position==1){

       auto head= insertatHead(llist,data);

       return head;

    }

    int count=1;

    auto temp=llist;

    while(count<position){

        temp=temp->next;

        count++;

    }

    if(temp->next==NULL){

        return insertAttail(llist, data);

    }

    SinglyLinkedListNode\* nodetoinsert=new SinglyLinkedListNode(data);

    nodetoinsert->next=temp->next;

    temp->next=nodetoinsert;

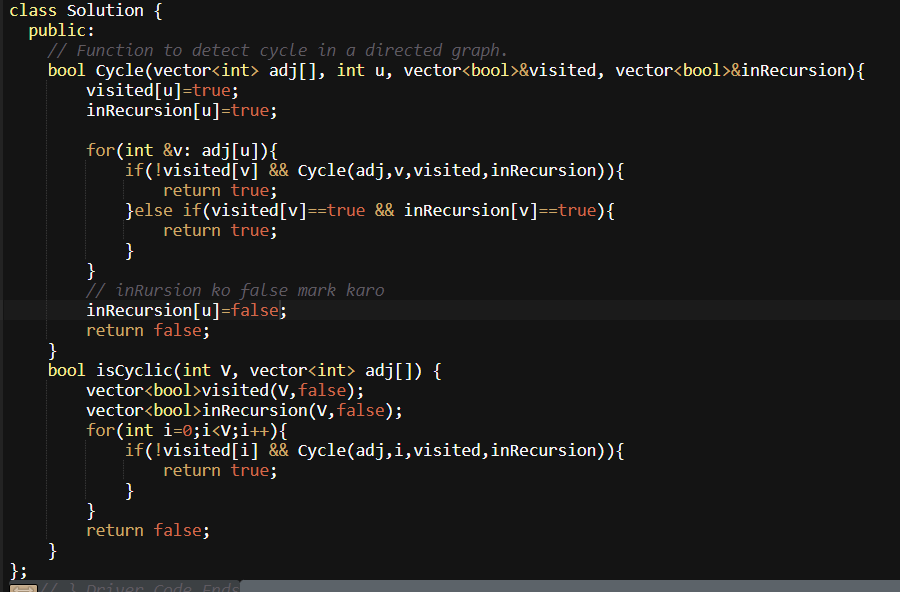
    return llist;

|  |  |
| --- | --- |
|  |  |

**Topological Sort**

|  |
| --- |
|  |

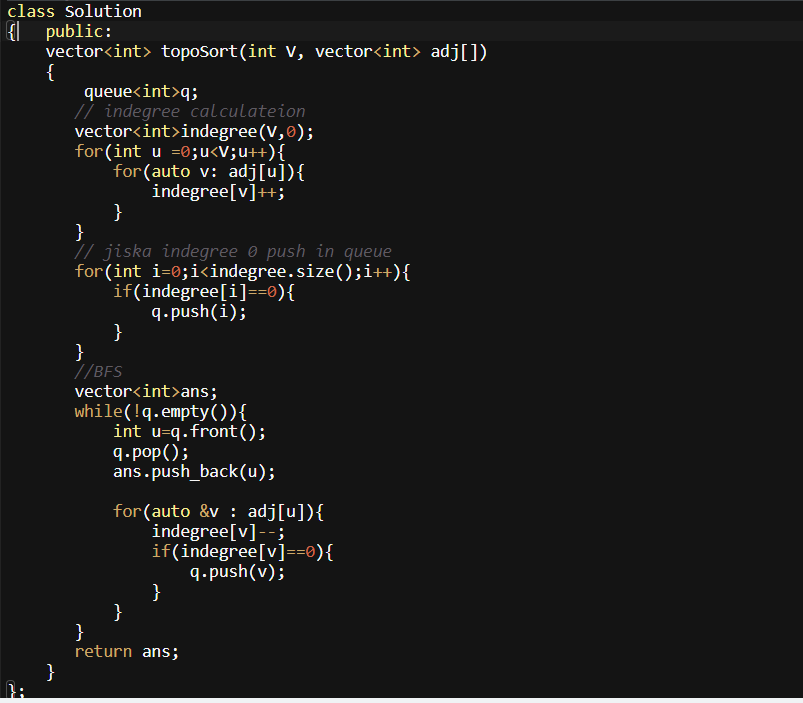
**Cycle Detection in Directed Graph**

****

**Using Kahn,s algo**

1🡪indegree nikalo

2🡪jiska indegree 0 push in que

3-🡪BFS and indegree minus karte raho corresponding u ke ****

**Cycle Detect in graph using Khan’s Algo (BFS)**

class Solution {

  public:

    // Function to detect cycle in a directed graph.

    bool isCyclic(int V, vector<int> adj[]) {

        // Using KAHN's Algo

        // 1.indegree nakalo

        // 2. jiski indegree 0 push in queue

        // track a count variable

        // 3. BFS maro --> indegree -- karo aur jiski 0 ho jaye queue me push karo

        // if count == No. of Vertext--->no cycle else cycle

    // --------------------------------------------

    queue<int>q;

    vector<int>indegree(V,0);

    for(int u=0;u<V;u++){

        for(int v:adj[u]){

            indegree[v]++;

        }

    }

    int count=0;

    for(int i=0;i<indegree.size();i++){

        if(indegree[i]==0){

            q.push(i);

            count++;

        }

    }

    while(!q.empty()){

        int u=q.front();

        q.pop();

        for(int v:adj[u]){

            indegree[v]--;

            if(indegree[v]==0){

                q.push(v);

                count++;

            }

        }

    }

    if(count==V)return false;

    else return true;

    }

};

|  |  |
| --- | --- |
| [**547. Number of Provinces**](https://leetcode.com/problems/number-of-provinces/)  **https://leetcode.com/problems/number-of-provinces/description/** | class Solution {  public:      int n=0;      void DFS( unordered\_map<int,vector<int>>&adj,int u,vector<bool>&visited){          visited[u]=true;          for(int &v: adj[u]){              if(!visited[v]){                  DFS(adj,v,visited);              }          }      }      int findCircleNum(vector<vector<int>>& isConnected) {          n=isConnected.size();          vector<bool>visited(n,false);          unordered\_map<int,vector<int>>adj;          for(int i=0;i<n;i++){              for(int j=0;j<n;j++){                  if(isConnected[i][j]==1){                      adj[i].push\_back(j);                      adj[j].push\_back(i);                  }              }          }          int count=0;          for(int i=0;i<n;i++){            if(!visited[i]){              DFS(adj,i,visited);              // sirf yaha par count++ hoga baki same hai DFS              count++;            }          }          return count;      }  }; |

|  |  |
| --- | --- |
|  | class Solution {  public:      bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {          unordered\_map<int,vector<int>>adj;          int n=numCourses;          vector<int>indegree(n,0);          for(auto v:prerequisites){              int a=v[0];              int b=v[1];              adj[b].push\_back(a); // b---->a [a,b]              indegree[a]++;          }          queue<int>q;          int count=0;          for(int i=0;i<indegree.size();i++){              if(indegree[i]==0){                  q.push(i);                  count++;              }          }          while(!q.empty()){              int u=q.front();              q.pop();              for(auto v:adj[u]){                  indegree[v]--;                if(indegree[v]==0){                  q.push(v);                  count++;              }              }          }          if(count==n)return true; // means no cycle so it can be finished          else return false;      }  }; |

|  |  |
| --- | --- |
| * **Topological sort pucha hai basically** * **Adj list pprepare kro according to preq,** * **Then calculate indegree** * **Kahn, algo to detect cycle in ADG**  1. **Indegree 0 hai jiski push in que** 2. **Count++** 3. **Push in ans(jiski indegree 0)** 4. **While(!q.empty)** 5. **Front nakalo aur uske adj list me indegree-- karo** 6. **If indegree ==0 then** 7. **Queue me push kro** 8. **Count++** 9. **Ans me push karo**  * **If count==numCourses return ans** * **That is your topological sort** | class Solution {  public:      vector<int>topologicalSort(unordered\_map<int,vector<int>>&adj,int numCourses,vector<int>&indegree){          queue<int>q;          int count=0;          vector<int>ans;          for(int i=0;i<indegree.size();i++){              if(indegree[i]==0){                  q.push(i);                  count++;                  ans.push\_back(i);              }          }          while(!q.empty()){              int u=q.front();              q.pop();              for(auto v:adj[u]){                  indegree[v]--;                  if(indegree[v]==0){                      q.push(v);                      count++;                      ans.push\_back(v);                  }              }          }          if(count==numCourses)return ans;          else return {};      }      vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) {          unordered\_map<int,vector<int>>adj;          vector<int>indegree(numCourses,0);          for(auto &v:prerequisites ){              int a=v[0];              int b=v[1];              // {a,b}              // b--->a  matlab indegree a kka increase hoga              adj[b].push\_back(a);              indegree[a]++;          }          return topologicalSort(adj,numCourses,indegree);      }  }; |

|  |  |
| --- | --- |
| 1. sort the intervals 2. take first interval in temp vector 3. process rest intervals 4. if overlapping happens:  * ----> first element of temp-->min(first element of temp, first elemnt of just processed interval) * ---->second element of temp-->max(second element of temp, second elemnt of just processed interval) * else : * ---->push temp in ans vector  1. push last interval 2. return ans | class Solution {  public:  static auto comp(vector<int>&a,vector<int>&b){      return a[0]<b[0];  }      vector<vector<int>> merge(vector<vector<int>>& intervals) {          int n=intervals.size();          vector<vector<int>>ans;          vector<int>temp;          // sort the intervals on first element based      //     auto comp=[](const vector<int>&a,const vector<int>&b){      //     return a[0]<b[0];      // };          sort(intervals.begin(),intervals.end(),comp);          // take first vector in temp          temp=intervals[0];          // travese on rest          for(int i=1;i<n;i++){              // overlapping happens              if(temp[1]>=intervals[i][0]){                  temp[0]=min(temp[0],intervals[i][0]);                  temp[1]=max(temp[1],intervals[i][1]);              }              else{                  // push in ans;                  ans.emplace\_back(temp);                  temp=intervals[i];              }          }          // push last interval          ans.emplace\_back(temp);          return ans;      }  }; |

|  |  |
| --- | --- |
|  | class Solution {  public:      int minSwaps(vector<int>& nums) {          int n=nums.size();          vector<int>temp(2\*n);          for(int i=0;i<2\*n;i++){              temp[i]=nums[i%n]; //to access first element after last(circular access)          }          int total1s=0; // represent window size          for(int i=0;i<n;i++){              if(nums[i]==1){                  total1s++;              }          }          int currOnes=0;          int maxcount=0;          int i=0;          int j=0;          while(j<2\*n){              if(temp[j]==1){                  currOnes++;              }              if(j-i+1>total1s){                  currOnes-=temp[i];  // remove o or from count if 0 no effect else count dec by 1                  i++;              }              j++;              maxcount=max(maxcount,currOnes);          }          return total1s-maxcount;      }  }; |

|  |  |
| --- | --- |
|  | class Solution {    public:      pair<Node \*, Node \*> splitList(struct Node \*head) {            Node\* slow= head;          Node\* fast=head->next;          while(fast!=head and fast->next!=head){              slow=slow->next;              fast=fast->next->next;          }            Node\* head1=head;          Node\*head2=slow->next;          slow->next=head1;            Node\*curr=head2;          while(curr->next!=head){              curr=curr->next;          }          curr->next=head2;            return {head1,head2};      }  }; |

|  |  |
| --- | --- |
|  | class Solution {    public:      int sum(Node\* root){        if(root==NULL){            return 0;        }          return root->data + sum(root->left) + sum(root->right);    }        bool isSumTree(Node\* root) {          // if root is null or leaf node return true          if(root==nullptr ||( root->left==NULL and root->right==NULL)){              return true;          }          int leftsum=0;          int rightsum=0;             leftsum=sum(root->left);           rightsum=sum(root->right);            if(root->data==(leftsum+rightsum) && isSumTree(root->left) && isSumTree(root->right)){              return true;          }          return false;      }  }; |

|  |  |
| --- | --- |
|  | class Solution {    public:    void solve(vector<int> &arr,int sum,vector<int>&ans,int index){        if(index==arr.size()){            ans.push\_back(sum);            return;        }      // exclusion      solve(arr,sum,ans,index+1);      // inclusion      sum+=arr[index];      solve(arr,sum,ans,index+1);      // backtrack      sum-=arr[index];    }      vector<int> subsetSums(vector<int> arr, int n) {          // Write Your Code here           vector<int>ans;          //  vector<int>output;           int sum=0;           solve(arr,sum,ans,0);           return ans;      }  }; |