WEEK-6:

/\* Ques 1. Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)

\*/

#include<bits/stdc++.h>

using namespace std;

void checkDFS(vector<int>adj[],vector<int>&vis, int s) {

vis[s]=1;

for(auto i:adj[s]) {

if(vis[i]==0)

checkDFS(adj,vis,i);

}

}

int main() {

int V,E;

cout<<"Enter number of vertices ";

cin>>V;

cout<<"Enter number of edges ";

cin>>E;

vector<int>adj[V];

cout<<"Enter graph edges ";

for(int i=0;i<E;i++) {

int u,v;

cin>>u>>v;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

// adjacency list is ready till now....

vector<int> vis(V,0); //visited array

cout<<"Enetr source and destinaton ";

int s,d;

cin>>s>>d;

checkDFS(adj,vis,s);

if(vis[d]==1)

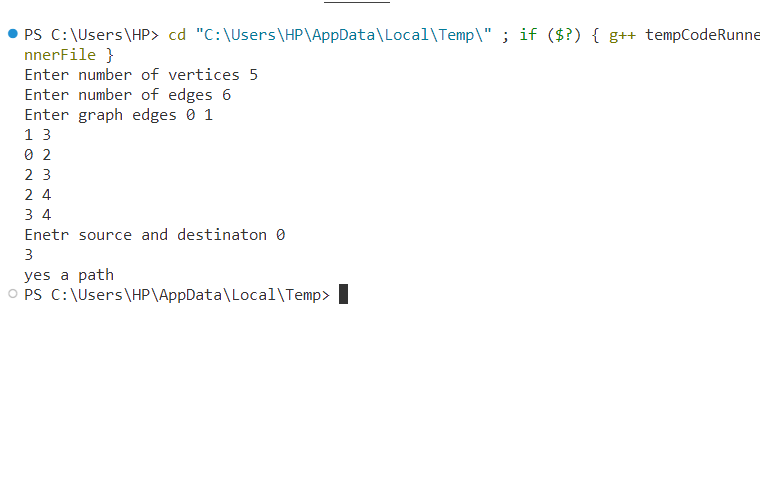
cout<<"yes a path";

else

cout<<"Not a path";

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)

\*/

#include<iostream>

#include<bits/stdc++.h>

#include<vector>

using namespace std;

bool dfs(int node,int col,int color[],vector<int>adj[]) {

color[node]=col;

for(auto it:adj[node]) {

if(color[it]==-1) {

if(dfs(it,!col,color,adj)==false)

return false;

}

else if(color[it]==col)

return false;

}

return true;

}

bool bipartite (int n, vector<int>adj[]) {

int color[n];

for(int i=0;i<n;i++)

color[i]=-1;

for(int i=0;i<n;i++) {

if(color[i]==-1) {

if(dfs(i,0,color,adj)==false)

return false;

}

}

return true;

}

int main() {

int n,m;

cout<<"enter the number of vertices";

cin>>n;

cout<<"enter the number of edges";

cin>>m;

vector<int>adj[n+1];

for(int i=0;i<m;i++) {

int u, v;

cin>>u>>v;

adj[u].push\_back(v);

adj[v].push\_back(u);

}

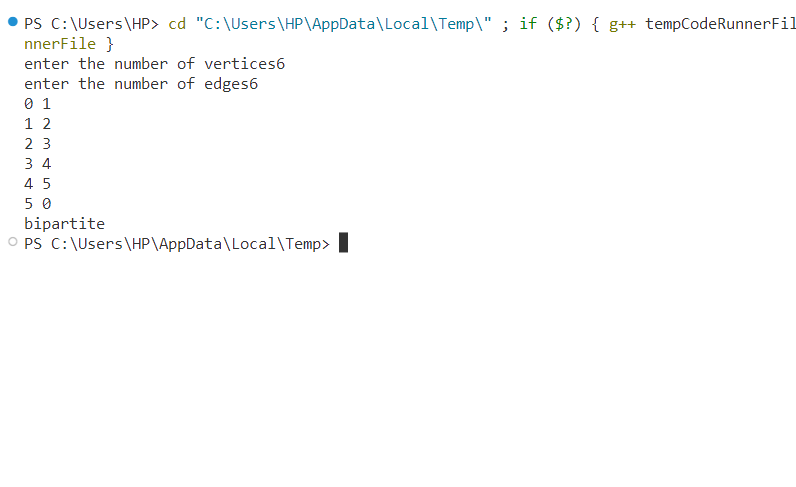
if(bipartite(n,adj)==false)

cout<<"not bipartite";

else cout<<"bipartite";

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given a directed graph, design an algorithm and implement it using a program to find whether a cycle exists in the graph or not.

\*/

#include<bits/stdc++.h>

using namespace std;

int findp(int u, vector<int>&parent){

    if(u== parent[u])

        return u;

    else{

        int res= findp(parent[u],parent);

        parent[u]=res;

        return res;

    }

}

void union1(int u,int v, vector<int>&parent) {

    int up= findp(u,parent);

    int vp= findp(v,parent);

    parent[up]=vp;

}

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

    vector<int>parent(V);

    for(int i=0;i<V;i++)

        parent[i]=i;

    vector<pair<int,int>>edges;

    for(int i=0;i<V;i++) {

        for(auto j:adj[i]) {

            if(j>i)

                edges.push\_back({i,j});

        }

    }

    for(int i=0;i<edges.size();i++) {

        int u=edges[i].first;

        int v=edges[i].second;

        if(findp(u,parent)== findp(v,parent))

            return 1;

        else

            union1(u,v,parent);

    }

    return 0;

}

int main(){

    int V,E;

    cout<<"Enter number of vertices ";

    cin>>V;

    cout<<"Enter number of edges ";

    cin>>E;

    vector<int>adj[V];

    cout<<"Enter graph edges ";

    for(int i=0;i<E;i++) {

        int u,v;

        cin>>u>>v;

        adj[u].push\_back(v);

        adj[v].push\_back(u);

    }

    // adjacency list is ready till now....

    int ans= isCycle(adj,V);

    if(ans==0)

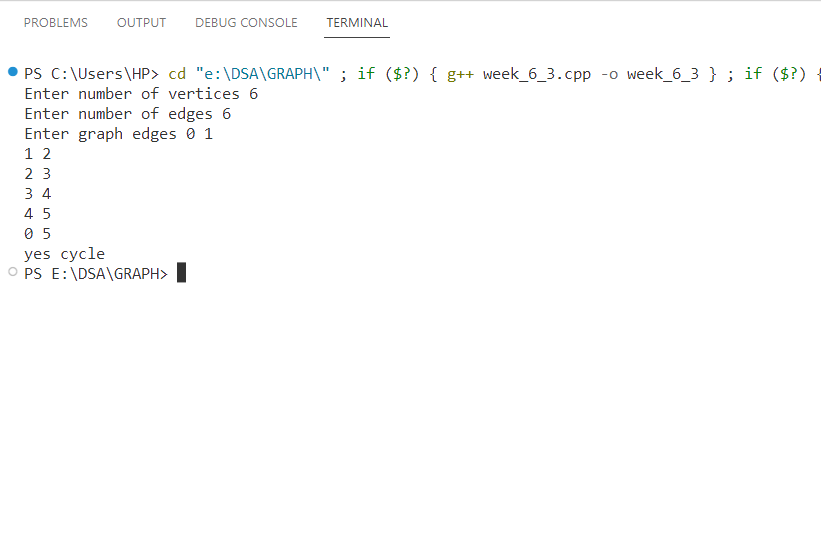
        cout<<"no cycle ";

    else

        cout<<"yes cycle";

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 7:

/\* Ques 1. After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay’s house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends’ location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend’s location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra’s algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay’s results? (Hint: Print shortest path and distance from friends’ location to Akshay’s house)

\*/

#include<bits/stdc++.h>

using namespace std;

vector<int> dijkstra(vector<pair<int,int>>G[], int V, int s) {

    vector<int>dis(V,INT\_MAX);

    dis[s]=0;

    priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;

    pq.push({0,s});

    while(!pq.empty()) {

        int wt = pq.top().first;

        int u= pq.top().second;

        pq.pop();

        for(auto i: G[u]) {

            if(wt+i.second < dis[i.first]) {

                dis[i.first]=wt+i.second;

                pq.push({dis[i.first],i.first});

            }

        }

    }

    return dis;

}

int main() {

    int V,E;

    cout<<"Enter number of vertices ";

    cin>>V;

    cout<<"Enter number of edges ";

    cin>>E;

    vector<pair<int,int>>G[V];

    //Here we will be storing the graph in edge list form along with weights.

    // u  v  wt

    // 0  1  7 like that

    cout<<"Enter u to v with weight ";

    for(int i=0;i<E;i++)  {

        int u,v,wt;

        cin>>u>>v>>wt;

        G[u].push\_back({v,wt});

    }

    // Graph has been inserted.

    cout<<"Enter the source node ";

    int s;

    cin>>s;

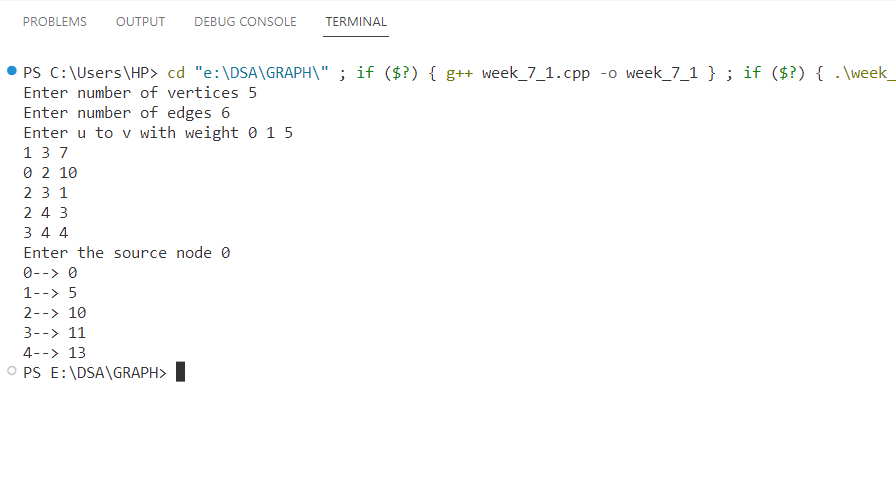
    vector<int>ans= dijkstra(G,V,s);

    for(int i=0;i<V;i++)

        cout<<i<<"--> "<<ans[i]<<endl;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm

\*/

#include<bits/stdc++.h>

using namespace std;

vector<int> bellmanFord(vector<pair<int,int>>G[], int V, int E, int s) {

    vector<int>dis(V,INT\_MAX);

    dis[s]=0;

    for(int i=1;i<V;i++) {

        for(int j=0;j<E;j++) {

            int u=j;

            for(auto it: G[u]) {

                int v=it.first;

                int wt= it.second;

                if(dis[u]!=INT\_MAX && dis[v]> dis[u]+wt)

                    dis[v]=dis[u]+wt;

            }

        }

    }

    for(int i=0;i<E;i++) {

        int u=i;

        for(auto it : G[i]) {

            int v= it.first;

            int wt= it.second;

            if(dis[u]!=INT\_MAX && dis[v]> dis[u]+wt) {

                cout<<"Neagtive cycle detected";

                return {-1};

            }

        }

    }

    return dis;

}

int main() {

    int V, E;

    cout<<"enter number of vertices ";

    cin>>V;

    cout<<"Enter number of edges ";

    cin>>E;

    cout<<"Enter u to v with weight "<<endl;

    vector<pair<int,int>>G[V];

for(int i=0;i<E;i++) {

        int u,v,wt;

        cin>>u>>v>>wt;

        G[u].push\_back({v,wt});

    }

    cout<<"enter source ";

    int s;

    cin>>s;

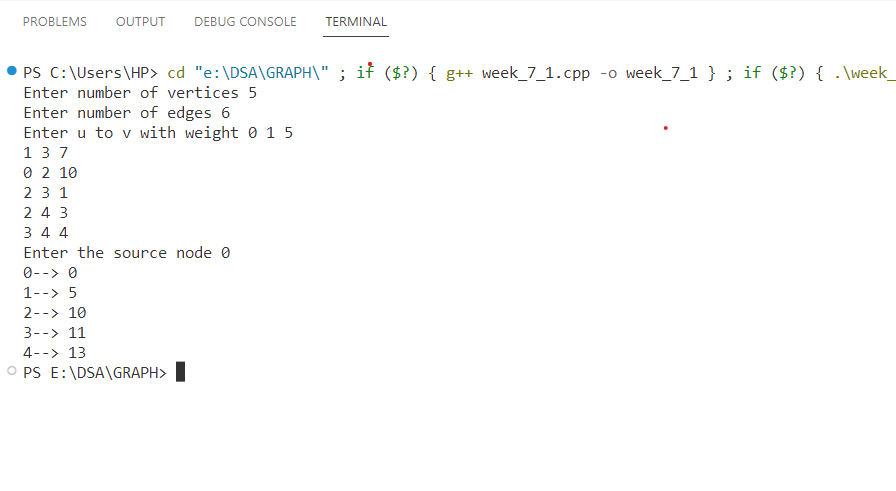
    vector<int>ans= bellmanFord(G,V,E,s);

    for(int i=0;i<ans.size();i++)

        cout<<i<<"-->"<<ans[i]<<endl;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



\*/ Ques3. Given a directed graph with two vertices (source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.

\*/

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int SPWKE(vector<vector<int>>& graph, int source, int destination, int k) {

int V = graph.size();

vector<vector<int>> dp(V, vector<int>(k + 1, INT\_MAX));

dp[source][0] = 0;

for (int edge = 1; edge <= k; edge++) {

for (int u = 0; u < V; u++) {

for (int v = 0; v < V; v++) {

if (graph[u][v] != INT\_MAX && dp[u][edge - 1] != INT\_MAX)

dp[v][edge] = min(dp[v][edge], dp[u][edge - 1] + graph[u][v]);

}

}

}

if (dp[destination][k] != INT\_MAX)

return dp[destination][k];

else

return -1;

}

int main() {

int V, source, destination, k;

cout << "Enter the number of vertices in the graph: ";

cin >> V;

vector<vector<int>> graph(V, vector<int>(V));

cout << "Enter the adjacency matrix for the graph:" << endl;

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

cin >> graph[i][j];

}

}

cout << "Enter the source vertex: "; cin >> source;

cout << "Enter the destination vertex: "; cin >> destination;

cout << "Enter the value of k: "; cin >> k;

int shortestPathWeight = SPWKE (graph, source - 1, destination - 1, k);

if (shortestPathWeight != -1)

cout << "Weight of shortest path from (" << source << "," << destination << ") with "

<< k << " edges: " << shortestPathWeight << endl;

else

cout << "No path of length " << k << " is available." << endl;

return 0;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***

Enter the adjacency matrix for the graph:

0 10 3 2

0 0 0 7

0 0 0 6

0 0 0 0

Enter the source vertex: 1

Enter the destination vertex: 4

Enter the value of k: 2

Weight of shortest path from (1,4) with 2 edges: 2

WEEK- 8:

/\* Ques 1. Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)

\*/

#include<iostream>

#include<bits/stdc++.h>

#include<vector>

using namespace std;

int prims (int v,vector<vector<int>>adj[]) {

    vector<int>vis(v,0);

    priority\_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>>pq;

    pq.push({0,0});

    int sum=0;

    while(!pq.empty()) {

        auto it=pq.top();

        pq.pop();

        int node=it.second;

        int wt=it.first;

        if(vis[node]==1)continue;

        vis[node]=1;

        sum+=wt;

        for(auto it:adj[node]) {

            int adjnode=it[0];

            int edgwt=it[1];

            if(!vis[adjnode])

                pq.push({edgwt,adjnode});

        }

    }

    return sum;

}

int main() {

    int n,m;

    cout<<"enter the number of vertices";

    cin>>n;

    cout<<"enter the number of edges";

    cin>>m;

    vector<vector<int>>adj[n+1];

    for(int i=0;i<m;i++) {

        int u,v,wt;

        cin>>u>>v>>wt;

        adj[u].push\_back({v,wt});

        adj[v].push\_back({u,wt});

    }

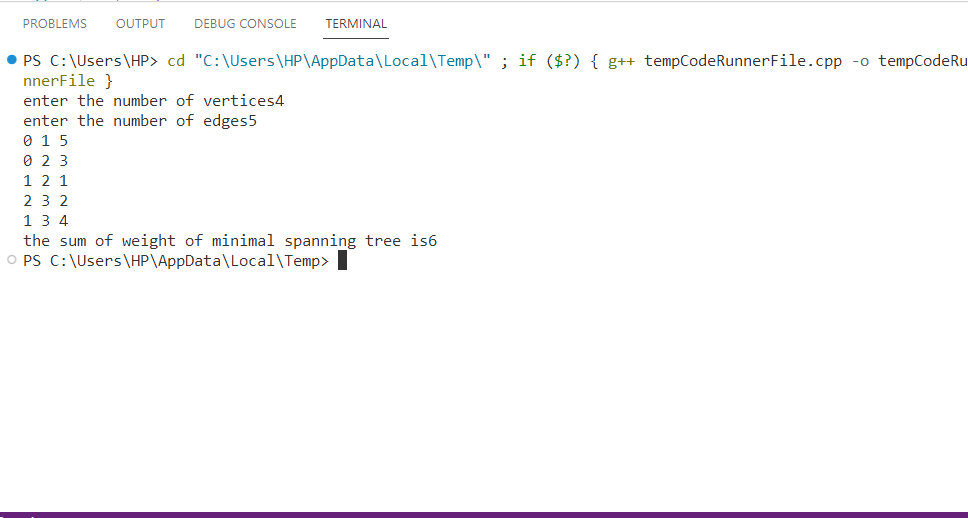
    cout<<"the sum of weight of minimal spanning tree is";

    int s=prims(n,adj);

    cout<<s;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques2. Implement the previous problem using Kruskal's algorithm.

\*/

#include <bits/stdc++.h>

using namespace std;

bool sortcol(const vector<int> &g1, const vector<int> &g2) {

    return g1[0] < g2[0];

}

int findP(int u, vector<int> &parent) {

    if (u == parent[u])

        return u;

    else {

        int res = findP(parent[u], parent);

        parent[u] = res;

        return res;

    }

}

void union1(int u, int v, vector<int> &parent) {

    int u\_par = findP(u, parent);

    int v\_par = findP(v, parent);

    if (u\_par == v\_par)

        return;

    parent[u\_par] = v\_par;

}

void addedge(vector<vector<int>> &G, int w, int u, int v) {

    vector<int> t;

    t.push\_back(w);

    t.push\_back(u);

    t.push\_back(v);

    G.push\_back(t);

}

int mst(vector<vector<int>> &G, int V) {

    vector<int> parent(V);

    for (int i = 0; i < V; i++)

        parent[i] = i;

    vector<vector<int>> ans;

    sort(G.begin(), G.end(), sortcol);

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

        int u = G[i][1];

        int v = G[i][2];

        if (findP(u, parent) != findP(v, parent)) {

            ans.push\_back(G[i]);

            union1(u, v, parent);

        }

    }

    int sum = 0;

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

        cout << ans[i][0] << " " << ans[i][1] << " " << ans[i][2] << endl;

        sum += ans[i][0];

    }

    return sum;

}

int main() {

    vector<vector<int>> G;

    addedge(G, 4, 0, 1);

    addedge(G, 8, 0, 7);

    addedge(G, 11, 1, 7);

    addedge(G, 8, 1, 2);

    addedge(G, 1, 7, 6);

    addedge(G, 7, 7, 8);

    addedge(G, 2, 2, 8);

    addedge(G, 6, 8, 6);

    addedge(G, 2, 6, 5);

    addedge(G, 7, 2, 3);

    addedge(G, 4, 2, 5);

    addedge(G, 14, 3, 5);

    addedge(G, 9, 3, 4);

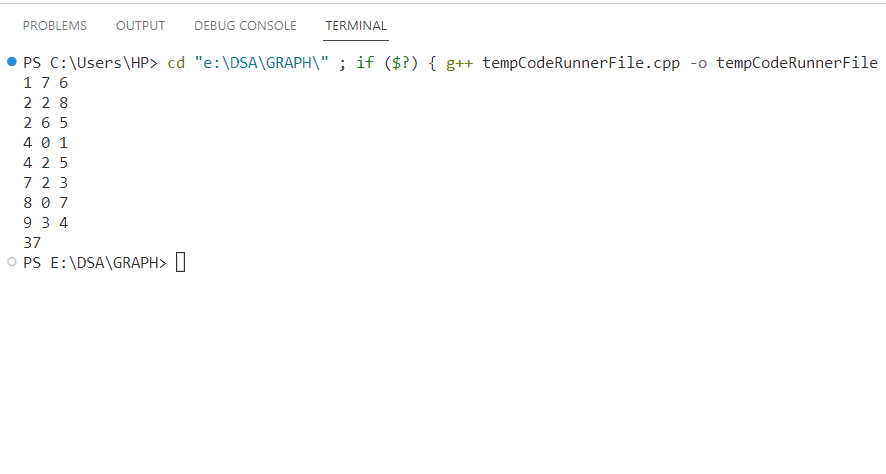
    addedge(G, 10, 5, 4);

    int t = mst(G, 9);

    cout << t;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

\*/

#include<bits/stdc++.h>

using namespace std;

bool sortcol(const vector<int>&v1, const vector<int>&v2){

    return v1[2]>v2[2];

}

int findp(int u, vector<int>&parent){

    if(u==parent[u])

        return u;

    else {

        int res= findp(parent[u],parent);

        parent[u]=res;

        return res;

    }

}

void union1(int u, int v,vector<int>&parent){

    int up=findp(u,parent);

    int vp=findp(v,parent);

    if(up==vp)

        return;

    parent[up]=vp;

}

int sum=0;

int maximum(vector<vector<int>>&G, int V, int E){

    vector<int>parent(V);

    for(int i=0;i<V;i++)

        parent[i]=i;

    sort(G.begin(),G.end(),sortcol);

    for(int i=0;i<E;i++) {

        int u=G[i][0];

        int v=G[i][1];

        int wt=G[i][2];

        if(findp(u,parent)!= findp(v,parent)) {

            union1(u,v,parent);

            sum+=wt;

        }

    }

    return sum;

}

int main(){

    int V,E;

    cout<<"Enter number of verices ";

    cin>>V;

    cout<<"Enter number of edges ";

    cin>>E;

    cout<<"Enter u to v with weight ";

    vector<vector<int>>G;

    for(int i=0;i<E;i++) {

        int u,v,wt;

        cin>>u>>v>>wt;

        G.push\_back({u,v,wt});

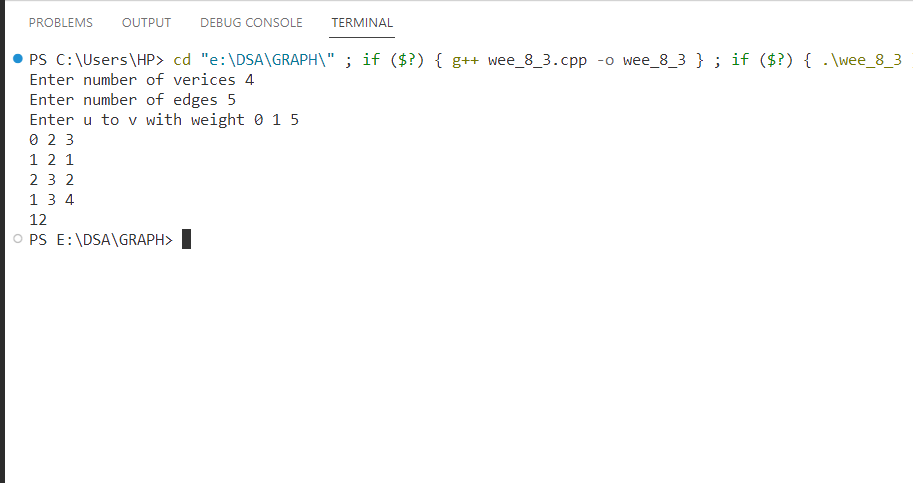
    }

    int sum= maximum(G,V,E);

    cout<<sum;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 9:

/\* Ques 1. Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm.

\*/

#include<bits/stdc++.h>

using namespace std;

vector<vector<int>> floydWarshal(vector<vector<int>>&G, int v){

    for(int k=0;k<v;k++)

        for(int i=0;i<v;i++)

            for(int j=0;j<v;j++)

                if(G[i][k]!=INT\_MAX && G[k][j]!=INT\_MAX && G[i][j]> G[i][k]+G[k][j])

                    G[i][j]= G[i][k]+G[k][j];

return G;

}

int main(){

    int v,e;

    cout<<"Enter number of vertices ";

    cin>>v;

    cout<<"enter number of edges ";

    cin>>e;

    cout<<"Enter u to v with weight ";

    vector<vector<int>>G(v,vector<int>(v,INT\_MAX));

    for(int i=0;i<e;i++)  {

        int u,v,w;

        cin>>u>>v>>w;

        G[u][v]=w;

        //G[v][u]=w;

    }

    for(int i=0;i<v;i++)

        G[i][i]=0;

    G= floydWarshal(G,v);

    for(int i=0;i<v;i++){

        for(int j=0;j<v;j++)

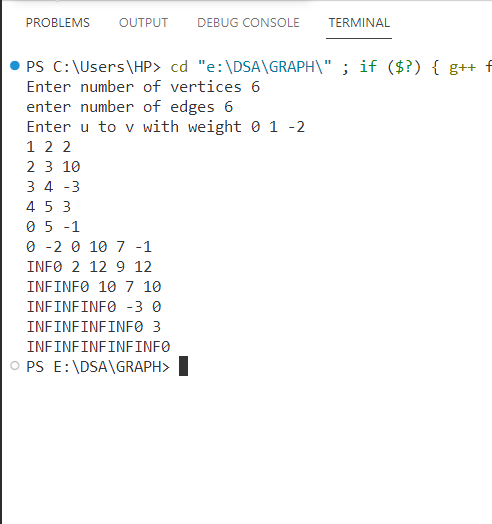
            cout<<G[i][j]<<" ";

        cout<<endl;

    }

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight w and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction xi of item i, where 0 ≤xi≤ 1.

\*/

#include<bits/stdc++.h>

using namespace std;

bool sortcol(const vector<double>&v1,const vector<double>&v2){

    return v1[2]>v2[2];

}

double knapsack(vector<vector<double>>item, int cap) {

    double ans=0;

    sort(item.begin(),item.end(),sortcol);

    for(int i=0;i<item.size();i++) {

        int v=item[i][0];

        int w=item[i][1];

        if(cap>= w) {

            ans+= v; cap-=w;

        }else {

            ans += cap\* item[i][2]; cap=0;

            break;

        }

    }

    return ans;

}

int main(){

    int n;

    cout<<"Enter the number of items "; cin>>n;

    vector<vector<double>>item(n);

    cout<<"Enter value with weight ";

  for(int i=0;i<n;i++) {

        double v,w;

        cin>>v>>w;

        item[i].push\_back(v);

        item[i].push\_back(w);

        item[i].push\_back(v/w);

    }

    int cap;

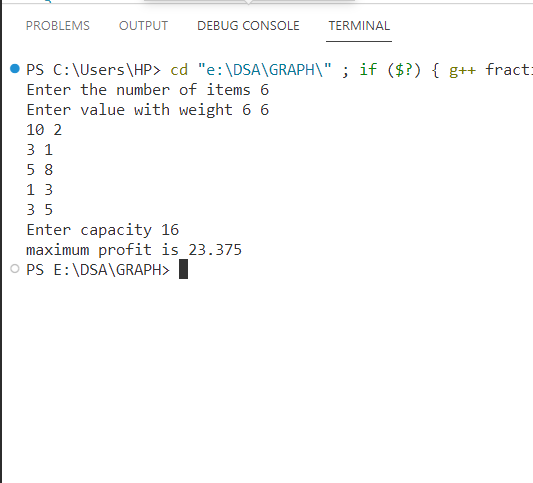
    cout<<"Enter capacity ";

    cin>>cap;

    cout<<"maximum profit is "<<knapsack(item,cap);

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given an array of elements. Assume arr[i] represents the size of file i. Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes m and n, computation cost of merging them is O(m+n). (Hint: use greedy approach)

\*/

#include<bits/stdc++.h>

using namespace std;

int main(){

    int n;

    cout<<"enter n";

    cin>>n;

    cout<<"enter array elememts ";

    vector<int>v;

    for(int i=0;i<n;i++){

        int t;

        cin>>t;

        v.push\_back(t);

    }

    sort(v.begin(),v.end());

    int ans= v[0]+v[1];

    int t=ans;

    for(int i=2;i<n;i++) {

        t=t+v[i];

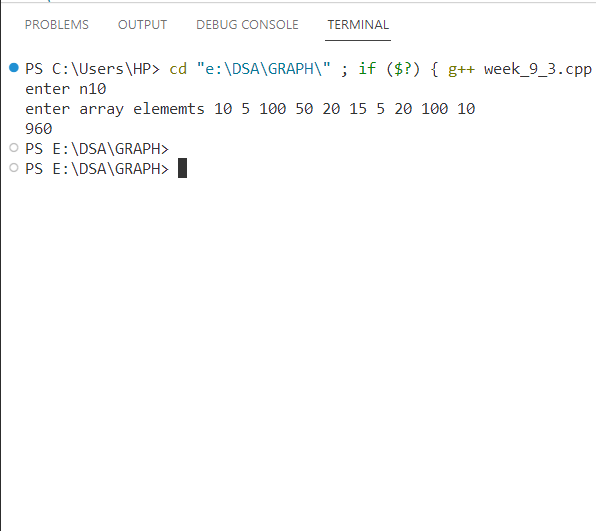
        ans+=t;

    }

    cout<<ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 10:

/\* Ques 1. Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.

\*/

#include <bits/stdc++.h>

using namespace std;

bool sortcol(const vector<int> &v1, const vector<int> &v2){

    return v1[2] < v2[2];

}

int activity(vector<pair<int, int>> act, int n){

    vector<vector<int>> G;

    for (int i = 0; i < n; i++)

        G.push\_back({i + 1, act[i].first, act[i].second});

    sort(G.begin(), G.end(), sortcol);

    int c = 1, end = G[0][2];

    cout << G[0][0] << " ,";

    for (int i = 1; i < n; i++){

        if (G[i][1] < end) continue;

        else{

            cout << G[i][0] << " ,";

            c++; end = G[i][2];

        }

    }

    return c;

}

int main(){

    int n;

    cout << "Enter number of aCTIVITIES ";

    cin >> n;

    vector<pair<int, int>> act;

    for (int i = 0; i < n; i++){

        int s, e;

        cin >> s >> e;

        act.push\_back({s, e});

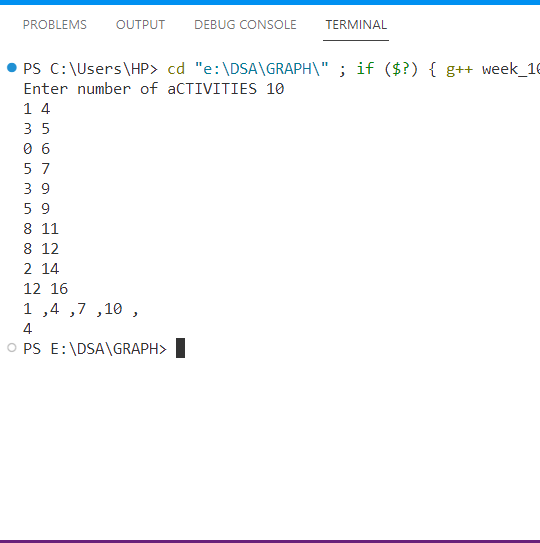
    }

    int num = activity(act, n);

    cout << endl<< num;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.

\*/

#include<bits/stdc++.h>

using namespace std;

bool sortcol(const vector<int>&v1 , const vector<int>&v2){

    return v1[2]>v2[2];

}

int job\_seq(vector<pair<int,int>>&job,int j){

    vector<vector<int>>v;

    for(int i=0;i<job.size();i++)

        v.push\_back({i,job[i].first,job[i].second});

    //sort(job.begin(),job.end(),sortcol);

    sort(v.begin(),v.end(),sortcol);

    int max=v[0][1];

    for(int i=0;i<v.size();i++){

        if(max< v[i][1])

            max= v[i][1];

    }

    vector<int>ans(max,-1);

    for(int i=0;i<v.size();i++){

        int j=v[i][0];

        int d= v[i][1]-1;

        int p= v[i][2];

        while(d>=0) {

            if(ans[d]==-1) {

                ans[d]=j;

                break;

            }

            d--;

        }

    }

    int sum=0;

    for(int i=0;i<ans.size();i++){

        if(ans[i]!=-1){

            sum+= job[ans[i]].second;

          cout<<ans[i]<<" ";

        }

    }

    return sum;

}

int main(){

    int j;

    cout<<"enter the number of jobs ";

    cin>>j;

    vector<pair<int,int>>job;

    cout<<"Enter the deadline and profit ";

    for(int i=0;i<j;i++) {

        int d,p;

        cin>>d>>p;

        job.push\_back({d,p});

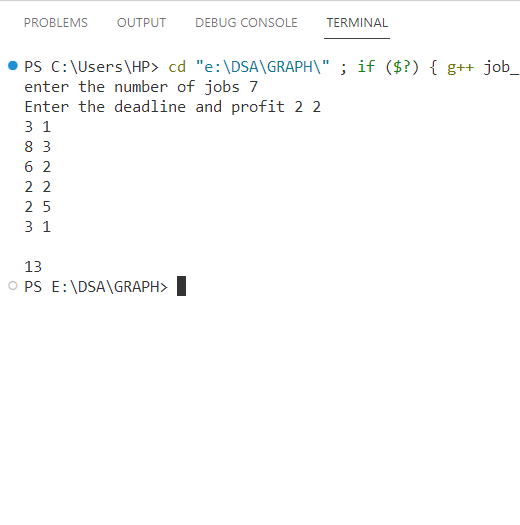
    }

    int maximum= job\_seq(job,j);

    cout<<endl<<maximum;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than n/2 times, where n is the size of array

\*/

#include <bits/stdc++.h>

using namespace std;

int main(){

    int n;

    cout << "Enter size of array ";

    cin >> n;

    vector<int> arr;

    for (int i = 0; i < n; i++)  {

        int x;

        cin>>x;

        arr.push\_back(x);

    }

    sort(arr.begin(),arr.end());

    if(arr[n/2]==arr[0])

        cout<<"yes";

    else

        cout<<"NO";

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 11:

/\* Ques 1. Given a sequence of matrices, write an algorithm to find most efficient way to multiply these matrices together. To find the optimal solution, you need to find the order in which these matrices should be multiplied.

\*/

#include <bits/stdc++.h>

using namespace std;

int MCM(vector<int>arr, int i,int j){

    if(i==j)    return 0;

    int mini=INT\_MAX;

    for(int k=i;k<j;k++) {

        int c= arr[i-1]\*arr[k]\*arr[j] + MCM(arr,i,k)+MCM(arr,k+1,j);

        mini= min(mini,c);

    }

    return mini;

}

int main(){

    int n;

    cout<<"enetr number of matrices ";

    cin>>n;

    vector<int>matrix(n+1);

    cout<<"Enter m \* n for each matrices ";

    for(int i=0;i<n;i++) {

        int m,n;

        cin>>m>>n;

        matrix[i]=m;

        matrix[i+1]=n;

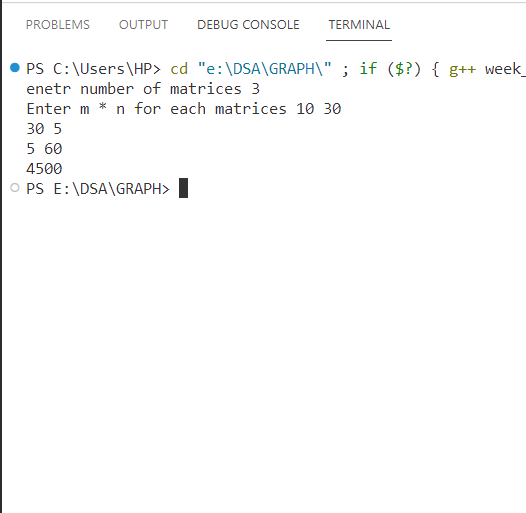
    }

    int ans= MCM(matrix,1,n);

    cout<<ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.

\*/

#include <bits/stdc++.h>

using namespace std;

int combo(vector<int> &arr, int i, int t){

    if (i == 0) {

        if (t % arr[i] == 0)

            return 1;

        else

            return 0;

    }

    int nt = combo(arr, i - 1, t);

    int take = 0;

    if (arr[i] <= t)

        take = combo(arr, i, t - arr[i]);

    return take + nt;

}

int main({

    int n;

    cout << "Enter number of coins ";

    cin >> n;

    cout << "Enter value of each coin ";

    vector<int> coin;

    for (int i = 0; i < n; i++){

        int x;

        cin >> x;

        coin.push\_back(x);

    }

    int t;

    cout << "Enetr target sum ";

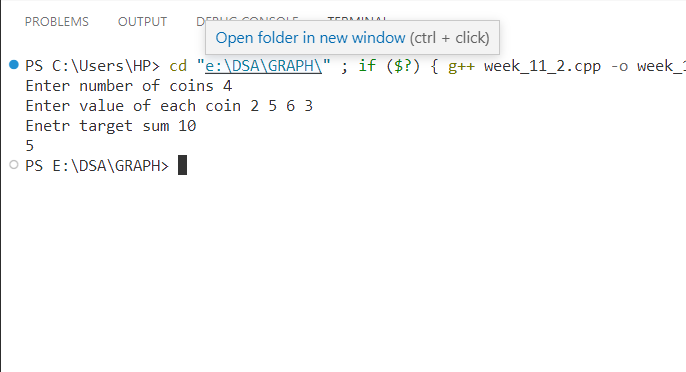
    cin >> t;

    int ans = combo(coin, n - 1, t);

    cout << ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.

\*/

#include<bits/stdc++.h>

using namespace std;

bool part(vector<int>&v, int n, int sum){

    if(sum==v[n])

        return true;

    if(n==0)  {

        if(v[n]==sum)

            return true;

        else

            return false;

    }

    bool nt= part(v,n-1,sum);

    if(nt==true)

        return true;

    if(v[n]<sum)

        return part(v,n-1, sum-v[n]);

}

int main(){

    int n;

    cout<<"enter size ";

    cin>>n;

    cout<<"enter element ";

    vector<int>v;

    for(int i=0;i<n;i++){

        int x;

        cin>>x;

        v.push\_back(x);

    }

    int sum=0;

    for(int i=0;i<n;i++)

        sum+= v[i];

    if(sum%2==1)

        cout<<"NO";

    else  {

        bool t= part(v,n-1,sum/2);

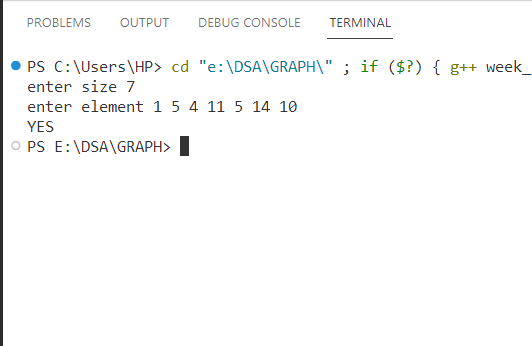
        if(t) cout<<"YES";

        else cout<<"NO";

    }

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 12:

/\* Ques 1. Given two sequences, Design an algorithm and implement it using a program to find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

\*/

#include<bits/stdc++.h>

using namespace std;

int lcm(string a, int m, string b, int n){

    if(m==-1 || n==-1)

        return 0;

    if(a[m]==b[n])

        return 1+ lcm(a,m-1,b,n-1);

    return max(lcm(a,m-1,b,n),lcm(a,m,b,n-1));

}

int main(){

    string a,b;

    cout<<"enter string 1 ";

    cin>>a;

    cout<<"Enter string 2 ";

    cin>>b;

    int m= a.size();

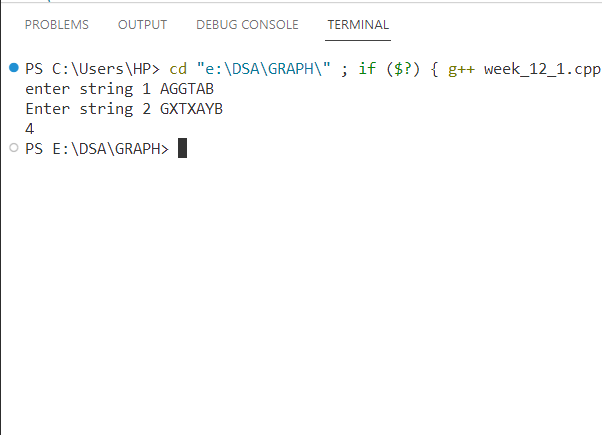
    int n= b.size();

    int ans= lcm(a,m-1,b,n-1);

    cout<<ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given a knapsack of maximum capacity w. N items are provided, each having its own value and weight. Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight <= w and has maximum value. Here, you cannot break an item i.e. either pick the complete item or don't pick it. (0-1 property).

\*/

#include<bits/stdc++.h>

using namespace std;

int knap01(vector<pair<int,int>>item, int n, int cap){

    if(n==0)  {

        if(item[n].first<= cap)

            return item[n].second;

        else return 0;

    }

    if(cap==0)

        return 0;

    int nt= knap01(item,n-1,cap);

    int t=0;

    if(item[n].first<=cap)

        t=item[n].second+knap01(item,n-1,cap-item[n].first);

    return max(nt,t);

}

int main(){

    int n;

    cout<<"enter number of items ";

    cin>>n;

    vector<pair<int,int>>item;  // weight and value

    for(int i=0;i<n;i++) {

        int w,v;

        cin>>w>>v;

        item.push\_back({w,v});

    }

    int cap;

    cout<<"Enter capacity of knapsack ";

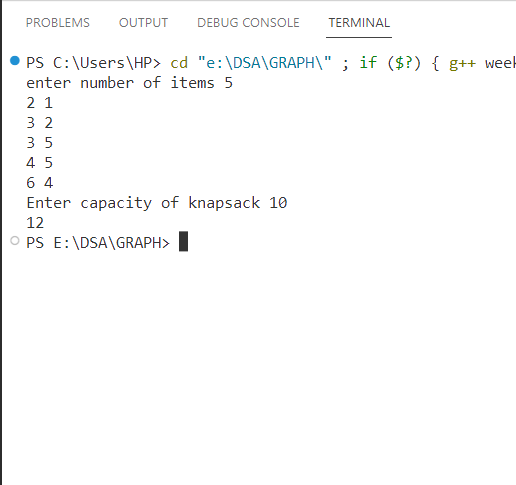
    cin>>cap;

    int ans= knap01(item,n-1,cap);

    cout<<ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given a string of characters, design an algorithm and implement it using a program to print all possible permutations of the string in lexicographic order.

\*/

#include <bits/stdc++.h>

using namespace std;

void permut(int ind, string &s){

    if (ind == s.size()){

        cout << s << endl;

        return;

    }

    for (int i = ind; i < s.size(); i++) {

        swap(s[i], s[ind]);

        permut(ind + 1, s);

        swap(s[i], s[ind]);

    }

}

int main(){

    string s;

    cout << "enetr the string ";

    cin >> s;

    int l = s.size();

    // vector<int>vis(l,0);

    // string t;

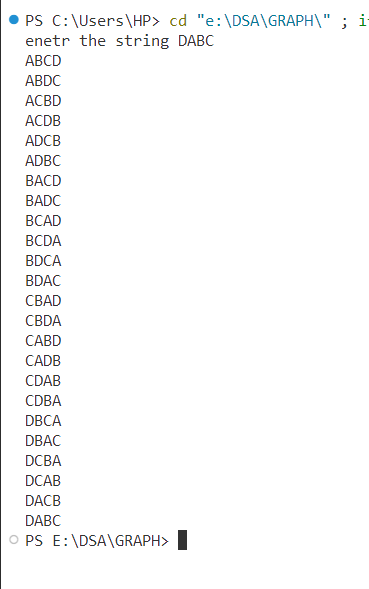
    sort(s.begin(), s.end());

    // permut(s,vis,t);

    permut(0, s);

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 13:

/\* Ques 1. Given an array of characters, you have to find distinct characters from this array. Design an algorithm and implement it using a program to solve this problem using hashing. (Time Complexity = O(n))

\*/

#include <bits/stdc++.h>

using namespace std;

int main(){

    int n;

    cout<<"enter number of char ";

    cin>>n;

    cout<<"enetr characters ";

    vector<char>arr;

    for(int i=0;i<n;i++)  {

        char x;

        cin>>x;

        arr.push\_back(x);

    }

    vector<int>ans(26,0);

    for(int i=0;i<n;i++) {

        int t=(int)arr[i];

        ans[t-97]++;

    }

    for(int i=0;i<26;i++) {

        if(ans[i]!=0) {

            char x= (char)(97+i);

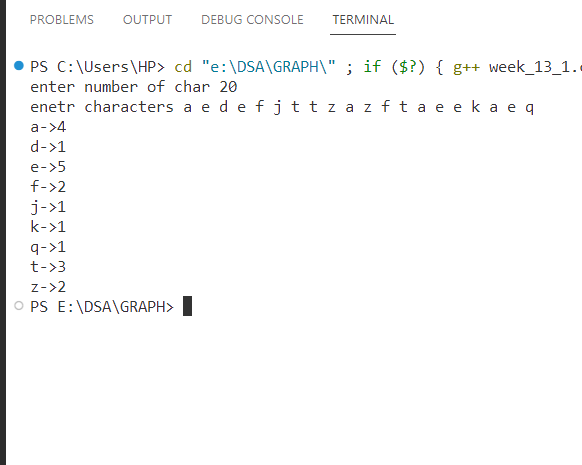
        cout<<x<<"->"<<ans[i]<<endl;

        }

    }

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 2. Given an array of integers of size n, design an algorithm and write a program to check whether this array contains duplicate within a small window of size k < n.

\*/

#include <bits/stdc++.h>

using namespace std;

bool check(vector<int>arr, int n, int k){

    unordered\_set<int>s;

    for(int i=0;i<n;i++)    {

        if(s.count(arr[i])==1)

            return true;

        s.insert(arr[i]);

        if(i>=k-1)

            s.erase(arr[i-k+1]);

    }

    return false;

}

int main(){

    int n;

    cout<<"netr size";

    cin>>n;

    vector<int>arr;

    cout<<"elementys ";

    for(int i=0;i<n;i++){

        int x;

        cin>>x;

        arr.push\_back(x);

    }

    int k;

    cout<<"enetr k ";

    cin>>k;

    bool ans= check(arr, n,k);

    if(ans)

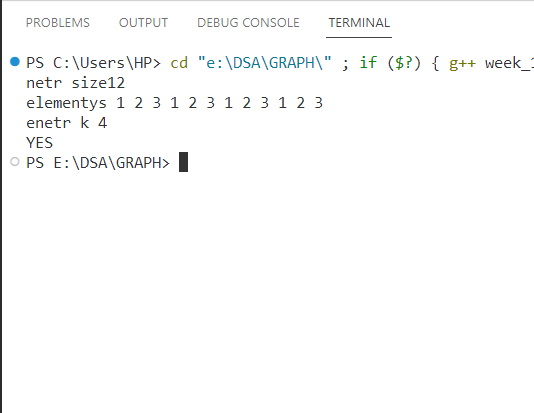
        cout<<"YES";

    else

        cout<<"NPO";

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



/\* Ques 3. Given an array of nonnegative integers, Design an algorithm and implement it using a program to find two pairs (a,b) and (c,d) such that a\*b = c\*d, where a, b, c and d are distinct elements of array.

\*/

#include <bits/stdc++.h>

using namespace std;

void permut(vector<int>&arr, int n) {

    map<int,pair<int,int>>m;

    ///auto it=m;

    int i,j,t;

    for(i=0;i<n;i++)    {

        for(j=i+1;j<n;j++)        {

            t=arr[i]\*arr[j];

            if(m.find(t)==m.end())

                m[t]={i,j};

            else {

                cout<<arr[i]<<" "<<arr[j]<<endl;

                pair<int,int>x=m[t];

                cout<<arr[x.first]<<" "<<arr[x.second]<<endl;

                return;

            }

        }

    }

}

int main(){

    int n;

    cout<<"netr size";

    cin>>n;

    vector<int>arr;

    cout<<"elementys ";

    for(int i=0;i<n;i++) {

        int x;

        cin>>x;

        arr.push\_back(x);

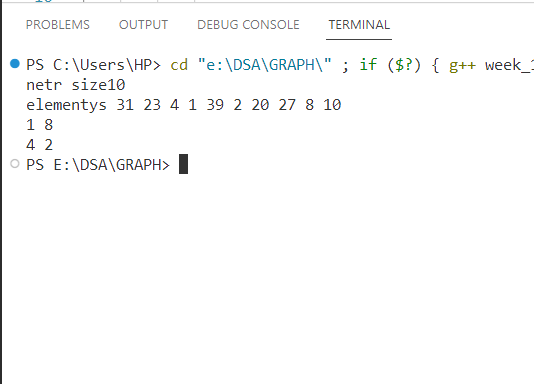
    }

    permut(arr,n);

}

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***



WEEK- 14:

/\* Ques 1. Given a number n, write an algorithm and a program to find nth ugly number. Ugly numbers are those numbers whose only prime factors are 2, 3 or 5. The sequence 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24,..... is sequence of ugly numbers.

\*/

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int getNthUglyNumber(int n) {

vector<int> ugly(n);

ugly[0] = 1;

int i2 = 0, i3 = 0, i5 = 0;

int nextMultipleOf2 = 2, nextMultipleOf3 = 3, nextMultipleOf5 = 5;

for (int i = 1; i < n; i++) {

int nextUN = min(nextMultipleOf2, min(nextMultipleOf3, nextMultipleOf5));

ugly[i] = nextUN;

if (nextUN== nextMultipleOf2) {

i2++; nextMultipleOf2 = ugly[i2] \* 2;

}

if (nextUN == nextMultipleOf3) {

i3++; nextMultipleOf3 = ugly[i3] \* 3;

}

if (nextUN == nextMultipleOf5) {

i5++; nextMultipleOf5 = ugly[i5] \* 5;

}

}

return ugly[n - 1];

}

int main() {

int T;

cout << "Enter the number of test cases: ";

cin >> T;

for (int i = 0; i < T; i++) {

int n;

cout << "Enter the value of n for test case " << i + 1 << ": ";

cin >> n;

int nthUglyNumber = getNthUglyNumber(n);

cout << "The " << n << "th ugly number is: " << nthUglyNumber << endl;

}

return 0;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***

Enter the number of test cases: 3

Enter the value of n for test case 1: 11

The 11th ugly number is: 15

Enter the value of n for test case 2: 8

The 8th ugly number is: 9

Enter the value of n for test case 3: 15

The 15th ugly number is: 24

/\* Ques 2. Given a directed graph, write an algorithm and a program to find mother vertex in a graph. A mother vertex is a vertex v such that there exists a path from v to all other vertices of the graph.

\*/

#include <bits/stdc++.h>

using namespace std;

vector<int> bfs(vector<int> adj[], int v, int s) {

    queue<int> q;

    vector<int> vis(v, 0);

    vis[s] = 1;

    q.push(s);

    while (!q.empty()){

        int u = q.front();

        q.pop();

        for (auto it : adj[u]){

            if (vis[it] == 0) {

                vis[it] = 1;

                q.push(it);

            }

        }

    }

    return vis;

}

int mother(vector<int> adj[], int v, int e){

    for (int i = 0; i < v; i++) {

        vector<int> vis(v, 0);

        vis = bfs(adj, v, i);

        int j;

        for (j = 0; j < v; j++)

if (vis[j] == 0)

                break;

        if (j == v)

            return i;

    }

    return -1;

}

int main() {

    int v, e;

    cout << "Enter number of vertices ";

    cin >> v;

    cout << "enetr number of edges ";

    cin >> e;

    cout << "enter graph u to v";

    vector<int> adj[v];

    for (int i = 0; i < v; i++){

        int u, x;

        cin >> u >> x;

        adj[u].push\_back(x);

    }

    int ans = mother(adj, v, e);

    cout << ans;

}

**\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\***

