**Problem No: 01**

**Topic: Graph**

**Problem Title:**

Listing the edges of a graph and giving the number of each edge appears from a given adjacent matrices.

**Objectives:**

To list the edges of a graph.

**Source Code:**

#include <bits/stdc++.h>

using namespace std;

int main(){

int n, i, j;

cout << "Number of vertices: ";

cin >> n;

int adjM[n+1][n+1]={0};

cout << "Enter elements for adjacent matrices:\n";

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

for(j=1; j<=n; j++)

cin >> adjM[i][j];

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

for(j=1; j<=n; j++){

if(adjM[i][j] != 0){

if(i!=j)

printf("Edge(%d,%d)/(%d,%d) = %d\n",i,j,j,i,adjM[i][j]);

else

printf("Edge(%d,%d) = %d\n",i,j,adjM[i][j]);

}

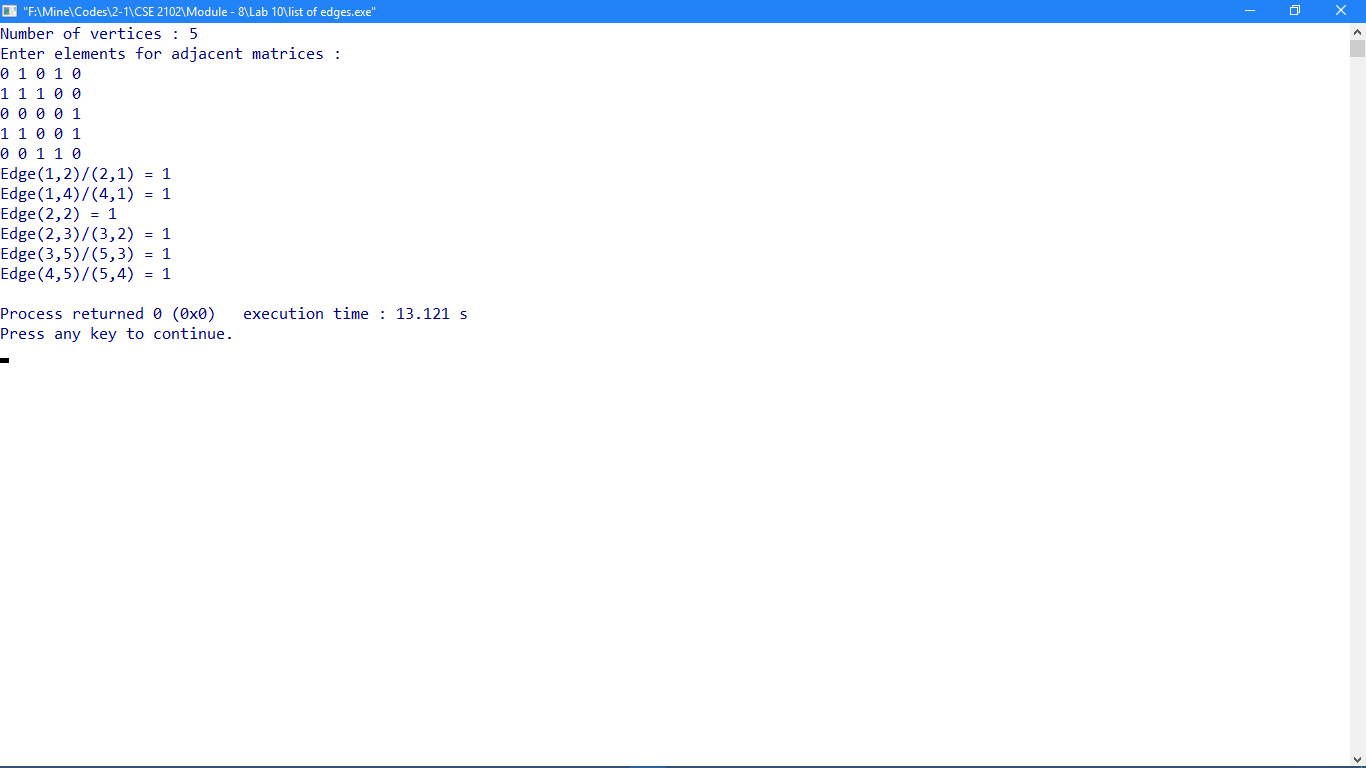
adjM[i][j] = 0; adjM[j][i] = 0;

}

}

}

**Output:**



**Problem No: 02**

**Topic: Graph**

**Problem Title:**

Given the vertex pairs associated to the edges of an undirected graph and the number of times each edge appears, construct an incidence matrix for the graph.

**Objectives:**

To construct an incidence matrix for the graph.

**Source Code:**

#include <iostream>

#include <vector>

using namespace std;

void addEdge(vector<int> graph[], int u, int v);

int main()

{

int n, e;

cout << "Enter no. of vertex / node: ";

cin >> n;

cout << "Enter no. of edges: ";

cin >> e;

cout << endl;

int m[n][e] = {};

vector <int> graph[n];

int u, v;

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

cout << "Enter the start and end vertex of edge: ";

cin >> u;

cin >> v;

addEdge(graph, u, v);

m[u][i]++;

m[v][i]++;

if(m[u][i] > 1)

m[u][i] = 1;

if(m[v][i] > 1)

m[v][i] = 1;

}

cout << endl;

cout << "Incidence Matrix for Undirected Graph: \n";

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

for(int j = 0; j < e; j++)

cout << m[i][j] << " ";

cout << endl;

}

return 0;

}

void addEdge(vector<int> graph[], int u, int v)

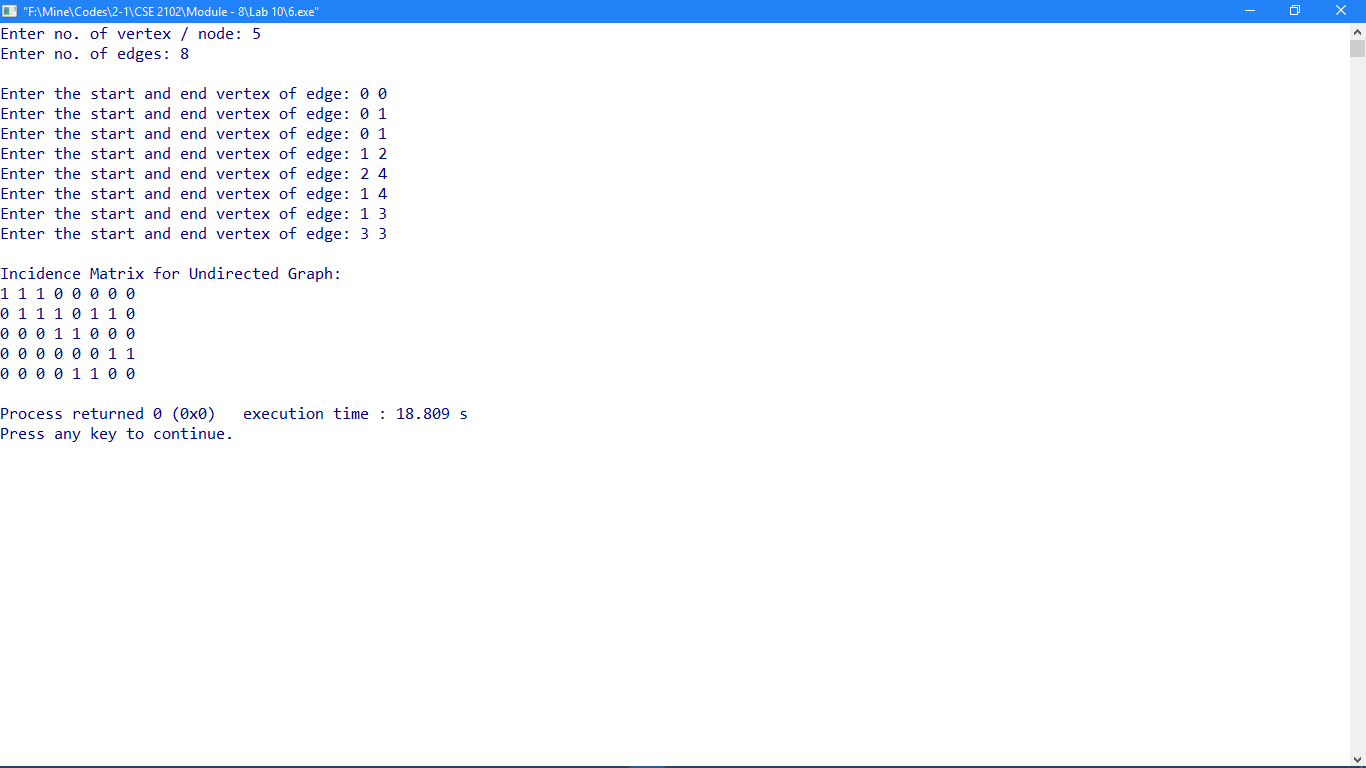
{

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

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

}

**Output:**



**Problem No: 03**

**Topic: Graph**

**Problem Title:**

Given the list of edges of a simple graph, determine whether the graph is bipartite.

**Objectives:**

To check whether the graph is bipartite.

**Source Code:**

#include <iostream>

#include <queue>

#define V 4

using namespace std;

bool isBipartite(int G[][V], int src);

int main()

{

int G[][V] = {{0, 1, 0, 1}, {1, 0, 1, 0},{0, 1, 0, 1}, {1, 0, 1, 0}};

isBipartite(G, 0) ? cout << "Yes" : cout << "No";

return 0;

}

bool isBipartite(int G[][V], int src)

{

int colorArr[V];

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

colorArr[i] = -1;

colorArr[src] = 1;

queue <int> q;

q.push(src);

while (!q.empty()) {

int u = q.front();

q.pop();

if (G[u][u] == 1)

return false;

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

if (G[u][v] && colorArr[v] == -1) {

colorArr[v] = 1 - colorArr[u];

q.push(v);

}

else if (G[u][v] && colorArr[v] == colorArr[u])

return false;

}

}

return true;

}

**Output:**

