PRACTICAL-1

Q1) Write a Program to represent Graphs using the Adjacency Matrices and check if it is a complete graph.

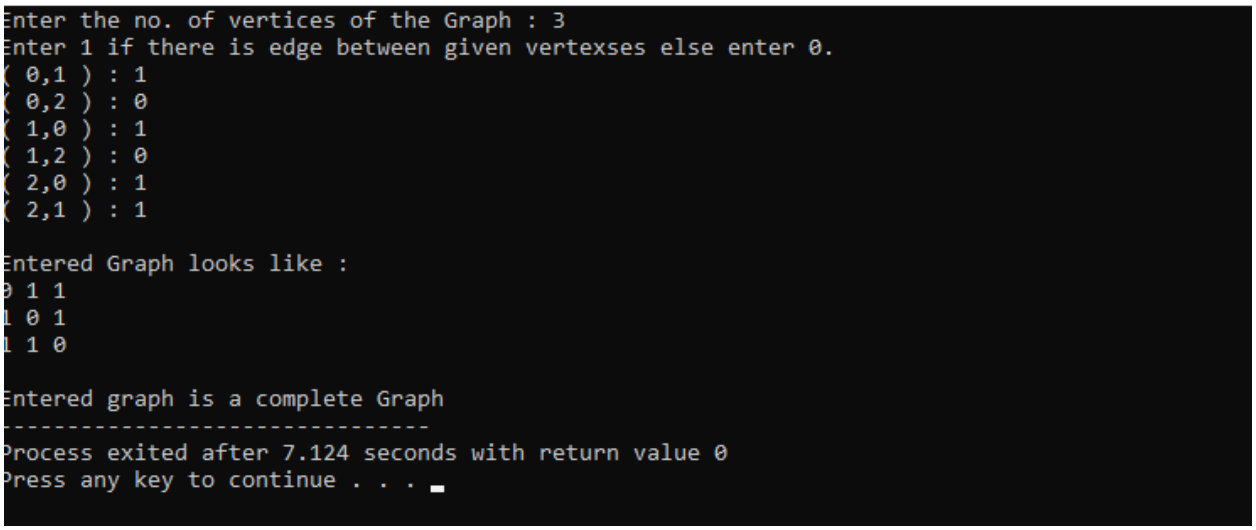
CODE:

```
#include<iostream>

using namespace std;

int vertArr[20][20]; //the adjacency matrix initially 0

int count = 0;


void displayMatrix(int v) {

  int i, j;

  for(i = 0; i < v; i++) {

    for(j = 0; j < v; j++) {

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

    }

    cout << endl;

  }

}


void add_edge(int u, int v) { //function to add edge into the matrix

  vertArr[u][v] = 1;

  vertArr[v][u] = 1;

}


main(int argc, char* argv[]) {
```

```
int v,flag=0,flag2=0;
int rolt=0,rol=0,col=0;
int no=0;
cout<<"Enter the no. of vertices of the Graph : ";
cin>>v;
cout<<"Enter 1 if there is edge between given vertexses else enter 0."<<endl;
while(flag<v*v)
{
        if(rol!=col)
        {
        cout<<"( "<<rol<<","<<col<<" ) : ";
        cin>>flag2;}
        if(flag2==1)
        {
                add_edge(rol,col);
                no++;
        }
        flag++;
        flag2=0;
        col++;
        rolt++;
        if(rolt==v)
        {   rol++;
                rolt=0;
```

```
        }
        if(col==v)
        {col=0;}
}
cout<<endl<<"Entered Graph looks like :"<<endl;
displayMatrix(v);
cout<<endl;
if(no=((v*(v-1))/2))
{
        cout<<"Entered graph is a complete Graph";
}
else{
        cout<<"Entered graph is not a complete Graph";
}
}
```

OUTPUT:



Q2) Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.

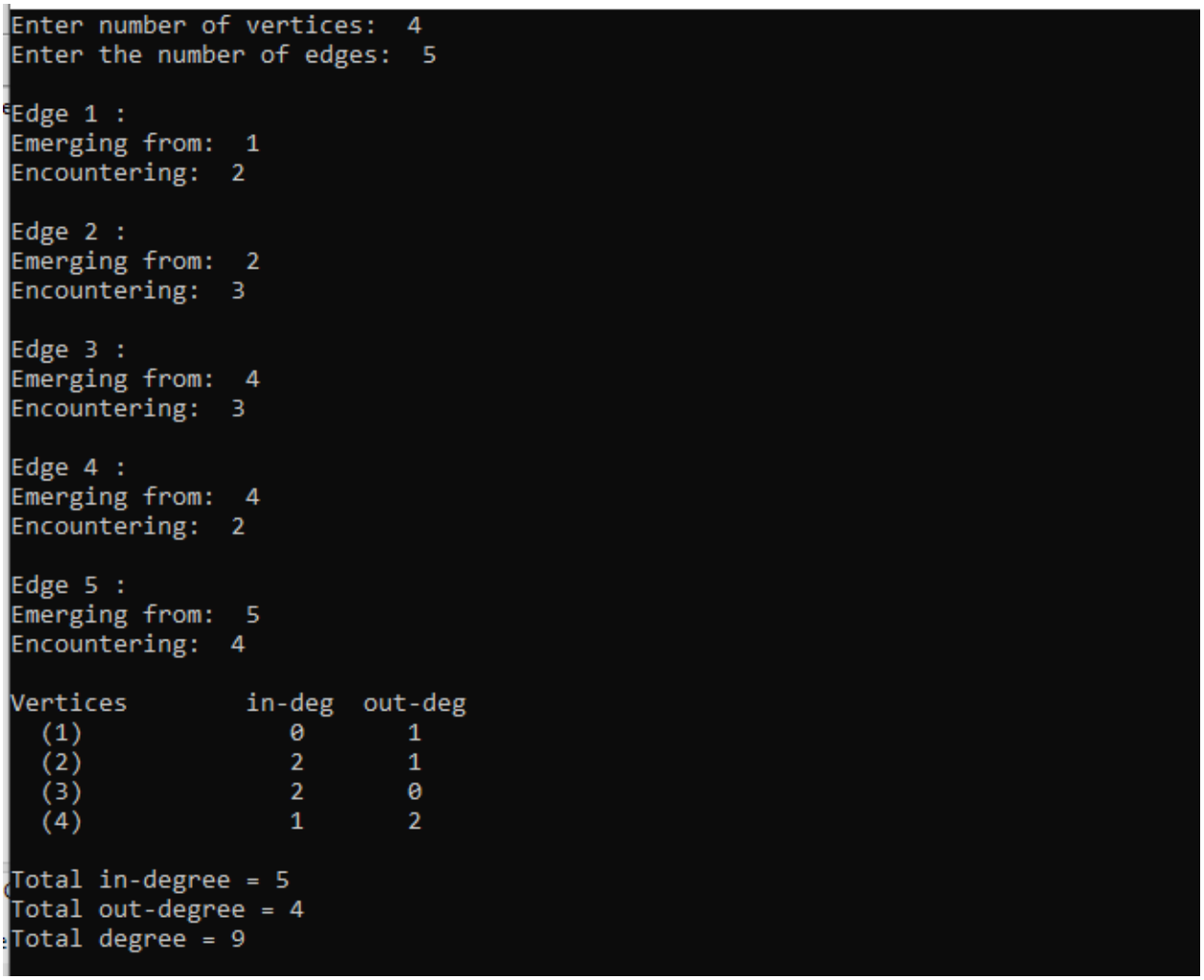
CODE:

```
#include <iostream>

#include <vector>

using namespace std;


int main()

{

   int vertices, edges, vert1, vert2, indeg, outdeg;

   indeg = outdeg = 0;

   int row = 1;

   cout << "Enter number of vertices:  ";

   cin >> vertices;

   vector<int> indegree(vertices, 0);

   vector<int> outdegree(vertices, 0);


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

   {

      outdegree[i] = indegree[i] = 0;

   }


   cout << "Enter the number of edges:  ";

   cin >> edges;

   cout << endl;
```

```
for (int i = 0; i < edges; i++)
{
   cout << "Edge " << i + 1 << " :  " << endl;
   cout << "Emerging from:  ";
   cin >> vert1;
   cout << "Encountering:  ";
   cin >> vert2;
   cout << endl;
   indegree[vert2 - 1]++;
   outdegree[vert1 - 1]++;
}

for (int i = 0; i < vertices; i++)
{
   indeg += indegree[i];
   outdeg += outdegree[i];
}

cout << "Vertices\tin-deg\tout-deg" << endl;
for (int i = 0; i < vertices; i++)
{
   cout << "  (" << i + 1 << ")  \t"
        << "          " << indegree[i] << '\t' << "   " << outdegree[i] << endl;
}
```

```
    cout << endl;


    cout << "Total in-degree = " << indeg << endl;
    cout << "Total out-degree = " << outdeg << endl;
    cout << "Total degree = " << (indeg + outdeg) << endl;
}
```

OUTPUT:



Q3) . **Given a graph G, Write a Program to find the number of paths of length n between the source and destination entered by the user.**

CODE:

```
#include <iostream>

#include <conio.h>

#include <vector>

using namespace std;

int vertices, edges;

vector<vector<int>> graph;

void graphinput()

{

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

   cin >> vertices;

   cout << "Enter the number of edges: ";

   cin >> edges;


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

   {

      graph.push_back(vector<int>());

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

      {

         graph[i].push_back(0);

      }

   }

   cout << "Enter connected vertices two at a time(those joined by edges): " << endl;

   int vert1, vert2;
```

```
    for (int i = 0; i < edges; i++)
    {
        cin >> vert1 >> vert2;
        graph[vert1 - 1][vert2 - 1] = 1;
        graph[vert2 - 1][vert1 - 1] = 1;
    }
}
int checkTotPath(int startNode, int goalNode, int pathLen)
{
    if (pathLen == 0 && startNode == goalNode)
    {
        return 1;
    }
    else if (pathLen == 1 && graph[startNode][goalNode])
    {
        return 1;
    }
    else if (pathLen <= 0)
    {
        return 0;
    }
    int totPath = 0;
    for (int i = 0; i < 5; i++)
        if (graph[startNode][i] == 1)
        {
```

```
            totPath += checkTotPath(i, goalNode, pathLen - 1);
        }


    return totPath;
}
int main()
{
    graphinput();
    int startNode;
    int goalNode;
    int pathLen;
    cout << "Enter the starting node:-";
    cin >> startNode;
    cout << "Enter the goal node:-";
    cin >> goalNode;
    cout << "Enter the length of path:-";
    cin >> pathLen;


    cout << checkTotPath(startNode, goalNode, pathLen);
    return 0;
}
```

Q4) Given an adjacency matrix of a graph, write a program to check whether a given set of vertices {v1,v2,v3.....,vk} forms an Euler path / Euler Circuit (for circuit assume vk=v1).

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int vertices, edges;
vector<vector<int>> graph;
void graphinput()
{
    cout << "Enter the number of vertices: ";
    cin >> vertices;
    cout << "Enter the number of edges: ";
    cin >> edges;

    for (int i = 0; i < vertices; i++)
    {
        graph.push_back(vector<int>());
        for (int j = 0; j < vertices; j++)
        {
            graph[i].push_back(0);
        }
    }
    cout << "Enter connected vertices two at a time(those joined by edges): " << endl;
    int vert1, vert2;
    for (int i = 0; i < edges; i++)
    {
```

```
        cin >> vert1 >> vert2;
        graph[vert1 - 1][vert2 - 1] = 1;
        graph[vert2 - 1][vert1 - 1] = 1;
    }
}
bool isconnected()
{
    bool f = 0;
    vector<int> visiting;
    vector<int> visited;
    visiting.push_back(2);
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 6; j++)
        {
            if (graph[visiting[0]][j] == 1 && !count(visiting.begin(), visiting.end(), j) && !count(visited.begin(), visited.end(), j))
            {
                visiting.push_back(j);
            }
        }

        if (!visiting.empty())
        {
            visited.push_back(visiting[0]);
            visiting.erase(visiting.begin());
```

```
        }
        else
        {
            break;
        }
    }
    for (int i = 0; i < 6; i++)
    {
        if (!count(visited.begin(), visited.end(), i))
        {
            f = 1;
            break;
        }
    }
    return f;
}
int isEuler()
{
    int i, j, k;
    vector<int> degree(vertices, 0);
    int odddegree = 0;
    for (i = 0; i < vertices; i++)
    {
        for (j = 0; j < vertices; j++)
        {
```

```
            if (graph[i][j] == 1)
                degree[i]++;
        }
    }
    for (i = 0; i < vertices; i++)
    {
        if (degree[i] % 2 != 0)
            odddegree++;
    }
    return odddegree;
}
int main()
{
    graphinput();
    if (isconnected())
    {
        cout << "Graph is disconnected.";
        return 0;
    }
    switch (isEuler())
    {
    case 0:
        cout << "The graph has Euler Path as well as circuit.";
        break;
    case 2:
```

```
        cout << "The graph has Euler Path but no circuit.";
        break;
    default:
        cout << "The graph has neither Euler Path nor a Euler circuit.";
        break;
    }
    return 0;
}
```

Q5) Given a full m-ary tree with i internal vertices, Write a Program to find the number of leaf nodes.

CODE:

```
#include <iostream>

using namespace std;

int main()

{

   int m;

   int ivert;

   cout << "Enter the m value for m-ary tree: ";

   cin >> m;

   cout << "Enter the number of internal vertices: ";

   cin >> ivert;

   cout << "The number of leaf nodes is: " << (ivert * (m - 1)) + 1;

}
```

OUTPUT: