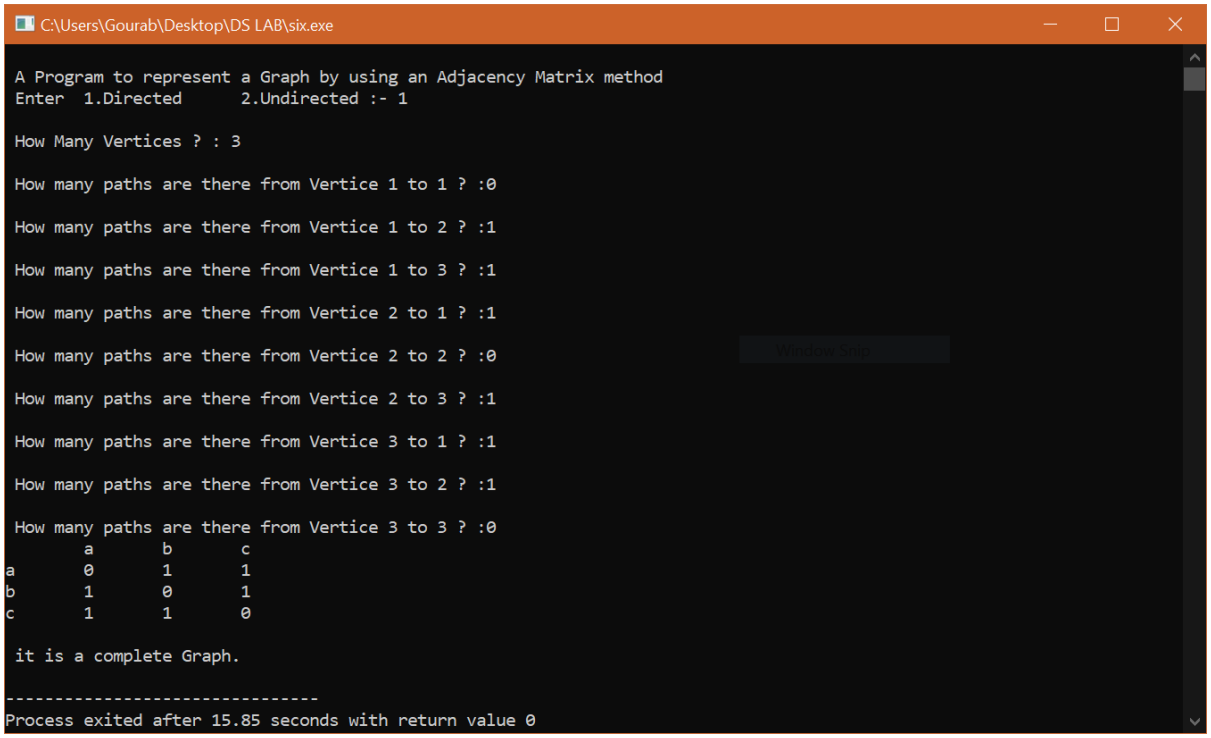
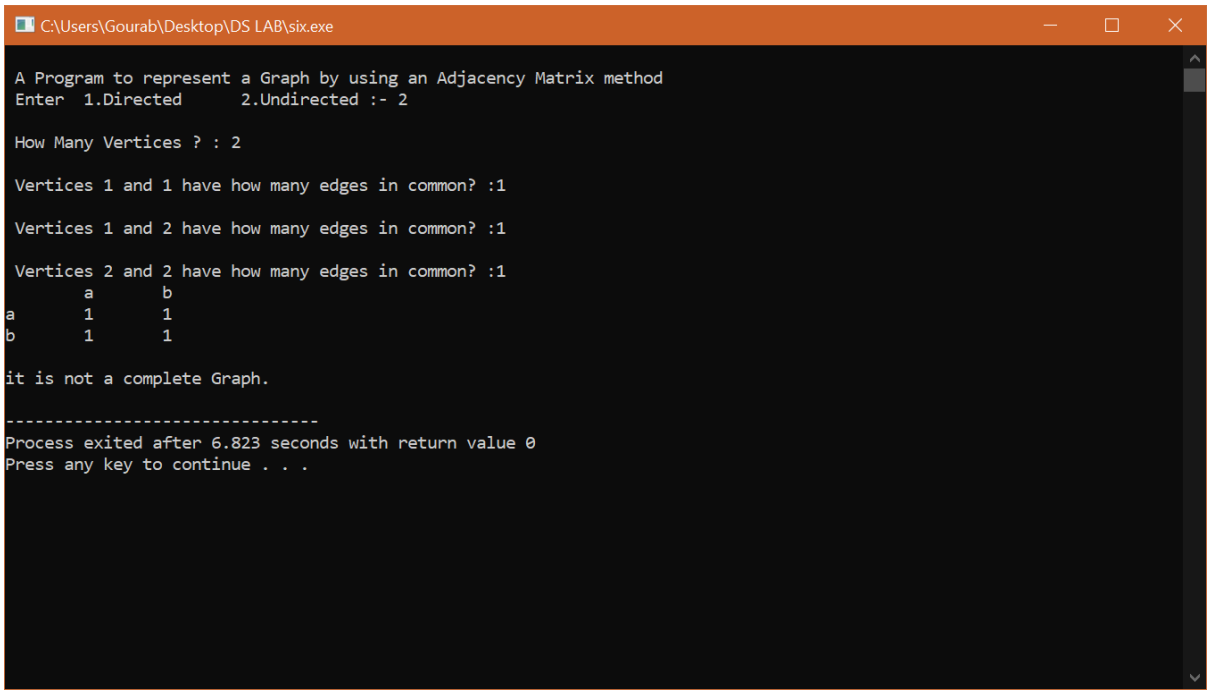
1. Write a Program to represent Graphs using the Adjacency Matrices and check if it is a complete graph.

```
#include<iostream>>
using namespace std;

int read_graph ( int adj_mat[□□][□□],int n )
{
   int i, j;
   int reply;
   for ( i = □ ; i <= n ; i++ )
   {
      for ( j = □ ; j <= n ; j++ )
      {
         cout<<"\n How many paths are there from Vertice "<<i<<" to "<<j<<" ? :";
         cin>>reply;
         adj_mat[i][j] =reply;
          }
   }
   return □;
}

int read_graph□ ( int adj_mat[□□][□□],int n )
{
   int i, j;
   int reply;
   for ( i = □ ; i <= n ; i++ )
   {
      for ( j = □ ; j <= n ; j++ )
      {
         if ( i > j  )
         {
            adj_mat[i][j] = adj_mat[j][i];
                                        continue;
         }
         cout<<"\n Vertices "<<i<<" and "<<j<<" have how many edges in common? :";
         cin>>reply;
         adj_mat[i][j]=reply;
          }
   }
   return □;
}
int dir_graph(int s)
{
   int adj_mat[□□][□□],n,in_deg=□,out_deg=□;
```

```
    bool test=true;
    char c,f[□□];
    cout<<("\n How Many Vertices ? : ");
    cin>>n;
    if (s==□)
            read_graph(adj_mat, n);
    else
            read_graph□(adj_mat, n);
    for (int i=□□;i<(□□+n);i++)
            {       c=(char)i;
                    f[i-□□]=c;
                    cout<<"\t"<<c;
            }
    for (int i = □; i <= n ; i++ )
    {
            cout<<"\n"<<f[i-□]<<"\t";
       for (int j = □ ; j <= n ; j++ )
       {
            int temp=adj_mat[i][j];
            cout<<temp<<"\t";
            if (i==j & temp>=□)
                    test=false;
            if (i!=j & temp==□)
                    test=false;
       }
    }
    if (test==true)
            cout<<"\n \n it is a complete Graph. \n";
    else
            cout<<"\n\nit is not a complete Graph. \n";
    return □;
}
int main()
{
    cout<<("\n A Program to represent a Graph by using an ");
            cout<<("Adjacency Matrix method \n ");
            int s;
            cout<<"Enter    □.Directed       □.Undirected :- ";
            cin>>s;
            dir_graph(s);
            return □;
}
```

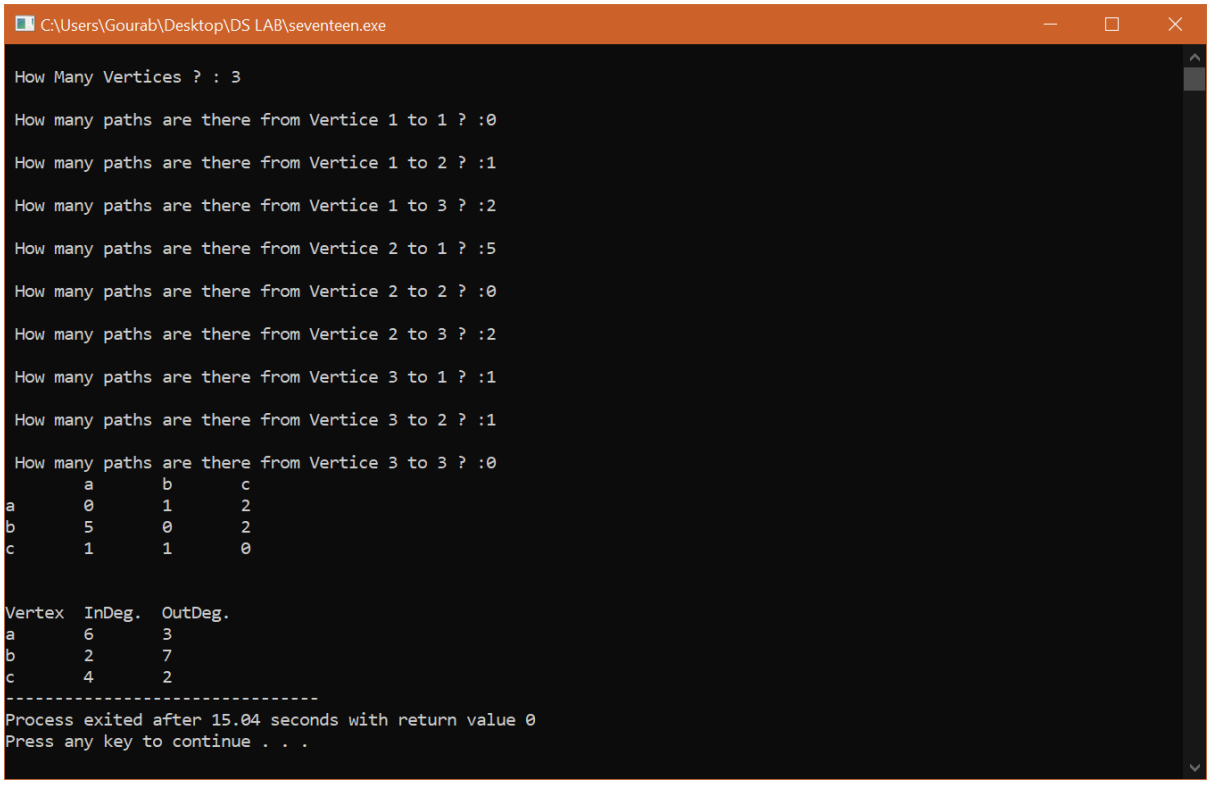

C:\Users\Gourab\Desktop\DS LAB\six.exe
A Program to represent a Graph by using an Adjacency Matrix method
Enter 1.Directed 2.Undirected :- 1
How Many Vertices ? : 3
How many paths are there from Vertice 1 to 1 ? :0
How many paths are there from Vertice 1 to 2 ? :1
How many paths are there from Vertice 1 to 3 ? :1
How many paths are there from Vertice 2 to 1 ? :1
How many paths are there from Vertice 2 to 2 ? :0
How many paths are there from Vertice 2 to 3 ? :1
How many paths are there from Vertice 3 to 1 ? :1
How many paths are there from Vertice 3 to 2 ? :1
How many paths are there from Vertice 3 to 3 ? :0
a b c
a 0 1 1
b 1 0 1
c 1 1 0
it is a complete Graph.
Process exited after 15.85 seconds with return value 0


C:\Users\Gourab\Desktop\DS LAB\six.exe
A Program to represent a Graph by using an Adjacency Matrix method
Enter 1.Directed 2.Undirected :- 2
How Many Vertices ? : 2
Vertices 1 and 1 have how many edges in common? :1
Vertices 1 and 2 have how many edges in common? :1
Vertices 2 and 2 have how many edges in common? :1
a b
a 1 1
b 1 1
it is not a complete Graph.
Process exited after 6.823 seconds with return value 0
Press any key to continue . . .

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

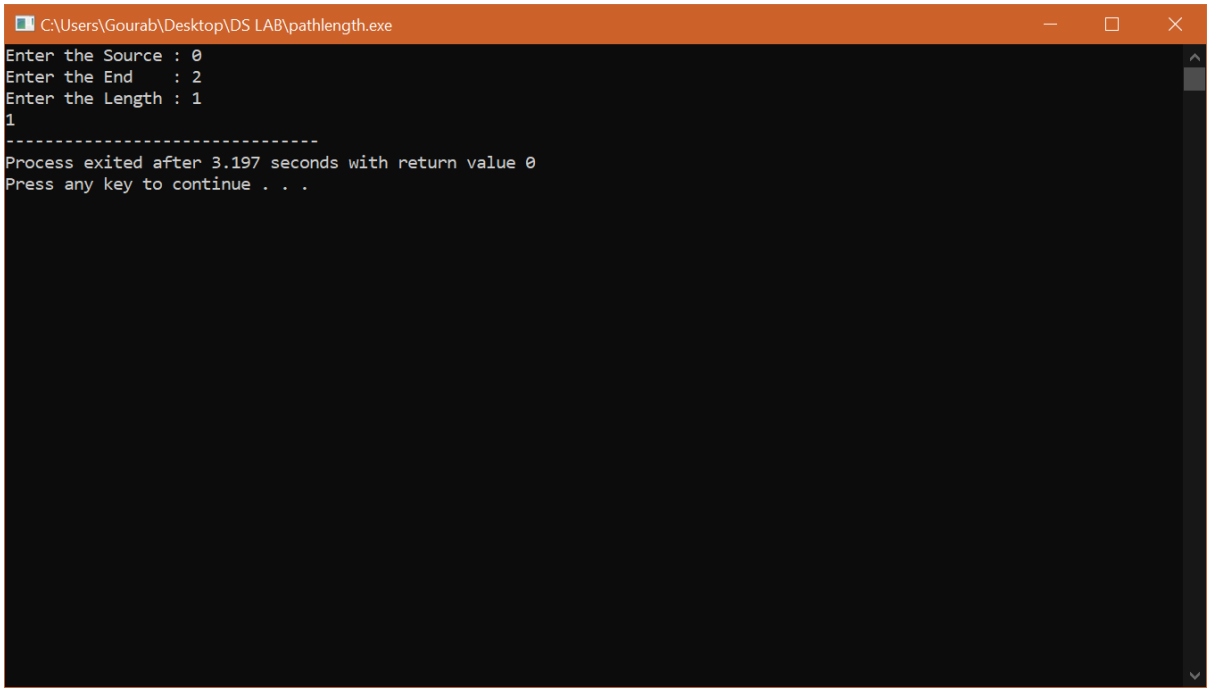
```
#include<iostream>>
using namespace std;

int read_graph ( int adj_mat[50][50], int n )
{
   int i, j;
   int reply;
   for ( i = 1 ; i <= n ; i++ )
   {
      for ( j = 1 ; j <= n ; j++ )
      {
         cout<<"\n How many paths are there from Vertice "<<i<<" to "<<j<<" ? :";
         cin>>reply;
         adj_mat[i][j] =reply;
          }
   }
   return 0;
}

int dir_graph()
{
   int adj_mat[50][50];
   int n,in_deg=0,out_deg=0;
   bool test=true;
   char c,f[50];;
   cout<<("\n How Many Vertices ? : ");
   cin>>n;
   read_graph(adj_mat, n);
   for (int i=97;i<(97+n);i++)
   {
          c=(char)i;
          f[i-97]=c;
                    cout<<"\t"<<c;
          }
   for (int i = 1; i <= n ; i++ )
   {
          cout<<"\n"<<f[i-1]<<"\t";
      for (int j = 1 ; j <= n ; j++ )
      {
          int temp=adj_mat[i][j];
          cout<<temp<<"\t";
                    }
```

```
        }
        int ind[50],outd[50];
        for (int i=1;i<=n;i++)
        {
                int temp1=0,temp2=0;
                for (int z=1;z<=n;z++)
                        {
                                temp1+=adj_mat[i][z];
                        }
                outd[i]=temp1;
                for (int z=1;z<=n;z++)
                        {
                                temp2+=adj_mat[z][i];
                        }
                ind[i]=temp2;
        }
        cout<<endl<<endl<<endl;
        cout<<"Vertex   InDeg.   OutDeg.";
   for (int i=97;i<(97+n);i++)
   {
        c=(char)i;
        cout<<endl;
                cout<<c<<"       "<<ind[i-96]<<" "<<outd[i-96];
        }




   return 0;
}




int main()
{
   cout<<"";
        dir_graph();
        return 0;
}
```


C:\Users\Gourab\Desktop\DS LAB\seventeen.exe
How Many Vertices ? : 3
How many paths are there from Vertice 1 to 1 ? :0
How many paths are there from Vertice 1 to 2 ? :1
How many paths are there from Vertice 1 to 3 ? :2
How many paths are there from Vertice 2 to 1 ? :5
How many paths are there from Vertice 2 to 2 ? :0
How many paths are there from Vertice 2 to 3 ? :2
How many paths are there from Vertice 3 to 1 ? :1
How many paths are there from Vertice 3 to 2 ? :1
How many paths are there from Vertice 3 to 3 ? :0
a b c
a 0 1 2
b 5 0 2
c 1 1 0
Vertex InDeg. OutDeg.
a 6 3
b 2 7
c 4 2
Process exited after 15.04 seconds with return value 0
Press any key to continue . . .

3. 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.

```
#include <iostream>
using namespace std;

#define V 4

int check(int g[][V], int u, int v, int k)
{

	if (k == 0 && u == v)
		return 1;
	if (k == 1 && g[u][v])
		return 1;
	if (k <= 0)
		return 0;

	int count = 0;

	for (int i = 0; i < V; i++)
		if (g[u][i] == 1)
			count += check(g, i, v, k - 1);

	return count;
}

int main()
{
	int g[V][V] = { { 0, 1, 1, 1 },
                                                { 0, 0, 0, 1 },
                                                { 0, 0, 0, 1 },
                                                { 0, 0, 0, 0 } };
	int u = 0, v = 3, k = 2;
	cout<<"Enter the Source : ";
	cin>>u;
	cout<<"Enter the End    : ";
	cin>>v;
	cout<<"Enter the Length : ";
	cin>>k;
	cout << check(g, u, v, k);
	return 0;
}
```


C:\Users\Gourab\Desktop\DS LAB\pathlength.exe
Enter the Source : 0
Enter the End : 2
Enter the Length : 1
1

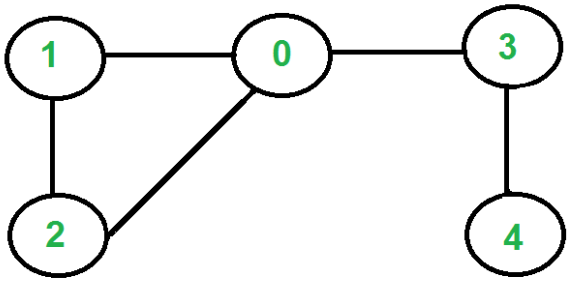
Process exited after 3.197 seconds with return value 0
Press any key to continue . . .

4. 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).

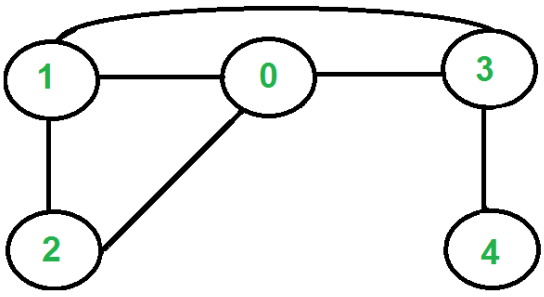
```
#include<iostream>
#include <list>
using namespace std;

class Graph
{
	int V;
	list<int> *adj;
public:
	Graph(int V) {this->V = V; adj = new list<int>[V]; }
	~Graph() { delete [] adj; }
	void addEdge(int v, int w);
	int isEulerian();
	bool isConnected();
	void DFSUtil(int v, bool visited[]);
};
void Graph::addEdge(int v, int w)
{
	adj[v].push_back(w);
	adj[w].push_back(v);
}
void Graph::DFSUtil(int v, bool visited[])
{
	visited[v] = true;
	list<int>::iterator i;
	for (i = adj[v].begin(); i != adj[v].end(); ++i)
		if (!visited[*i])
			DFSUtil(*i, visited);
}
bool Graph::isConnected()
{
	bool visited[V];
	int i;
	for (i = 0; i < V; i++)
		visited[i] = false;
	for (i = 0; i < V; i++)
		if (adj[i].size() != 0)
			break;
```

```
        if (i == V)
                return true;

        DFSUtil(i, visited);
        for (i = 0; i < V; i++)
        if (visited[i] == false && adj[i].size() > 0)
                        return false;
        return true;
}

int Graph::isEulerian()
{

        if (isConnected() == false)
                return 0;
        int odd = 0;
        for (int i = 0; i < V; i++)
                if (adj[i].size() & 1)
                        odd++;
        if (odd > 2)
                return 0;
        return (odd)? 1 : 2;
}
void test(Graph &g)
{
        int res = g.isEulerian();
        if (res == 0)
                cout << "graph is not Eulerian\n";
        else if (res == 1)
                cout << "graph has a Euler path\n";
        else
                cout << "graph has a Euler cycle\n";
}
int main()
{
        // Let us create and test graphs shown in below figures
        Graph g1(5);
        g1.addEdge(1, 0);
        g1.addEdge(0, 2);
        g1.addEdge(2, 1);
        g1.addEdge(0, 3);
        g1.addEdge(3, 4);
        test(g1);
        cout<<endl;
        Graph g2(5);
```

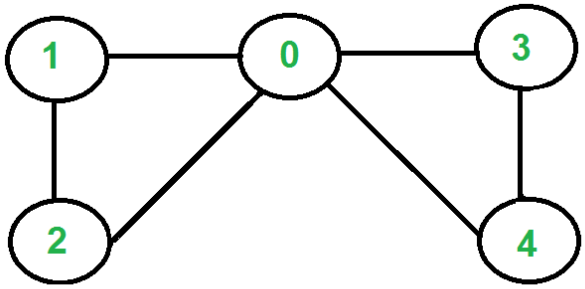
```
    g2.addEdge(1, 0);
    g2.addEdge(0, 2);
    g2.addEdge(2, 1);
    g2.addEdge(0, 3);
    g2.addEdge(3, 4);
    g2.addEdge(4, 0);
    test(g2);
    cout<<endl;
    Graph g3(5);
    g3.addEdge(1, 0);
    g3.addEdge(0, 2);
    g3.addEdge(2, 1);
    g3.addEdge(0, 3);
    g3.addEdge(3, 4);
    g3.addEdge(1, 3);
    test(g3);
    cout<<endl;

    // Let us create a graph with 3 vertices
    // connected in the form of cycle
    Graph g4(3);
    g4.addEdge(0, 1);
    g4.addEdge(1, 2);
    g4.addEdge(2, 0);
    test(g4);
    cout<<endl;

    // Let us create a graph with all veritces
    // with zero degree
    Graph g5(3);
    test(g5);

    return 0;
}
```



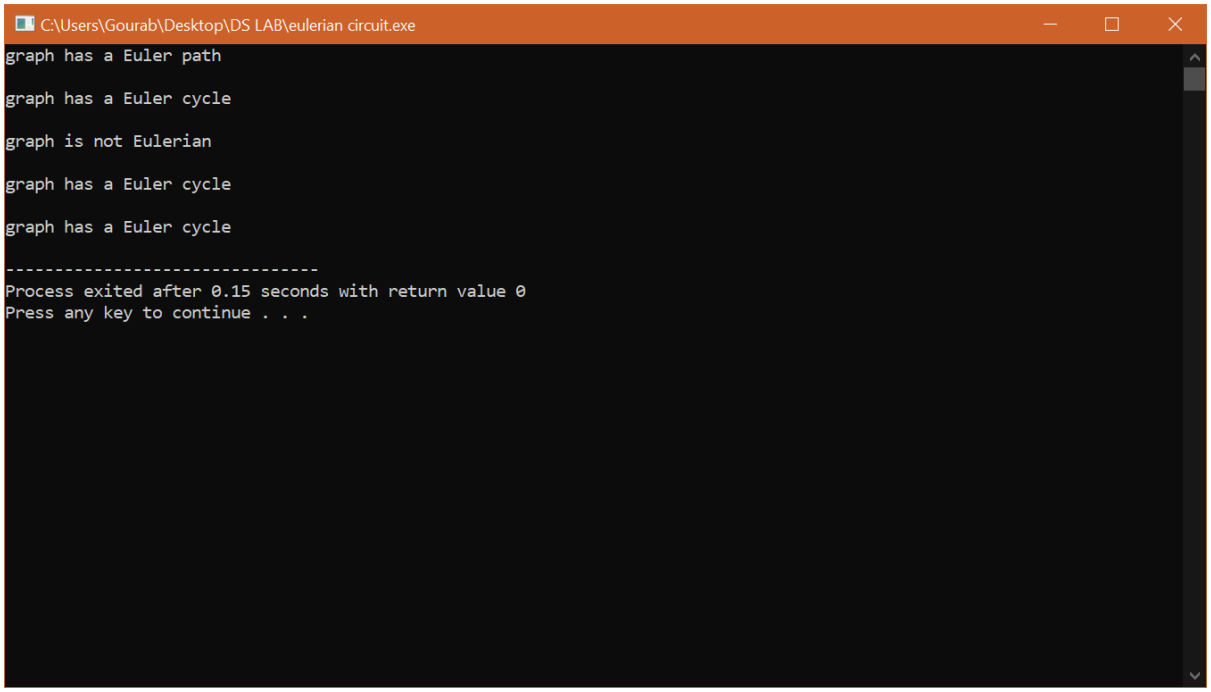
The graph has Eulerian Paths, for example "4 3 0 1 2 0", but no Eulerian Cycle. Note that there are two vertices with odd degree (4 and 0)



The graph is not Eulerian. Note that there are four vertices with odd degree (0, 1, 3 and 4)



The graph has Eulerian Cycles, for example "2 1 0 3 4 0 2" Note that all vertices have even degree


C:\Users\Gourab\Desktop\DS LAB\eulerian circuit.exe
graph has a Euler path
graph has a Euler cycle
graph is not Eulerian
graph has a Euler cycle
graph has a Euler cycle

Process exited after 0.15 seconds with return value 0
Press any key to continue . . .

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

```
#include <bits/stdc++.h>
using namespace std;

int calcNodes(int m, int i)
{
	int result = 0;
	result = i * (m - 1) + 1;
	return result;
}

int main()
{
	int m = 5, i = 2;
	cout<<"Enter the value of m for m-ary tree   : ";
	cin>>m;
	cout<<"Enter the number of internal vertices : ";
	cin>>i;
	cout << "Leaf nodes = " << calcNodes(m, i);
	return 0;
}
```