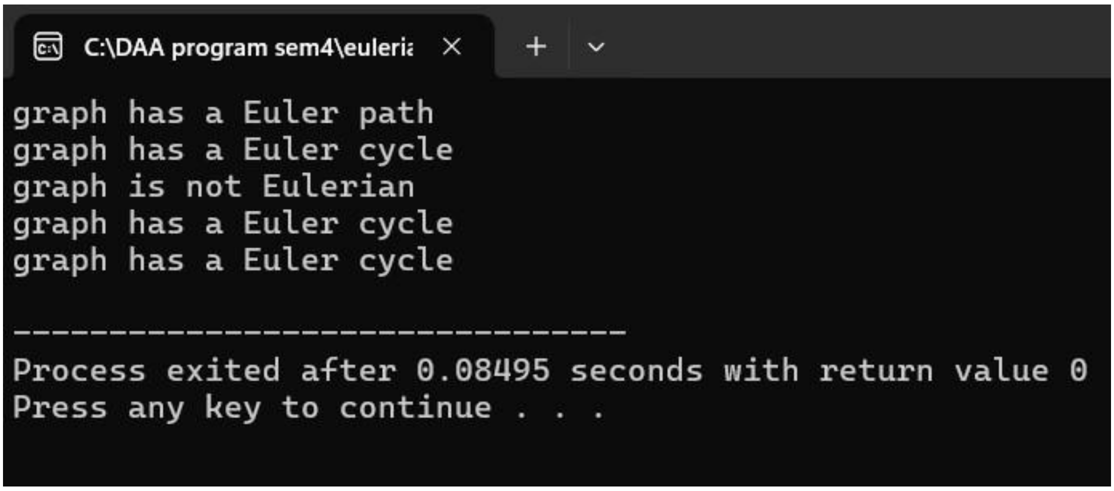
Name: Ashmit Thawait
Roll No: 102203790
Group: 2CO17

Lab Assignment 5

1. Write a program to Eulerian path and circuit, given an undirected/directed graph.

```
#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;
}

/* The function returns one of the following values
0 If graph is not Eulerian
```

```
1 If graph has an Euler path (Semi-Eulerian) 2 If graph has an Euler Circuit (Eulerian) */ int
   Graph::isEulerian()
{
 if (isConnected() == false)  return 0;

 // Count vertices with odd degree
 int odd = 0;
 for (int i = 0; i < V; i++)
          if (adj[i].size() & 1)
                    odd++;

 // If count is more than 2, then graph is not Eulerian
 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()
{
 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);

 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);

 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);

 Graph g4(3);  g4.addEdge(0, 1);
g4.addEdge(1, 2);  g4.addEdge(2, 0);
 test(g4);

 Graph g5(3);
 test(g5);

 return 0;
}
```

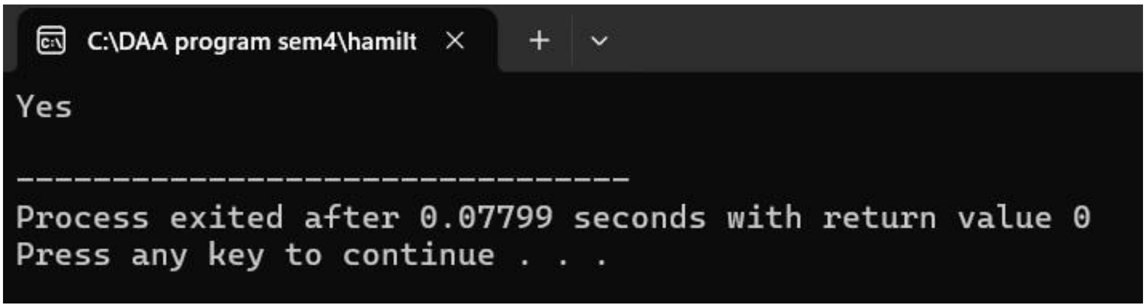


```
C:\DAA program sem4\euleri

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 0.08495 seconds with return value 0
Press any key to continue . . .
```

2. Given an adjacency matrix representation of an undirected graph consisting of N vertices, write a program to find whether the graph contains a Hamiltonian Path or not. If found to be true, then print “Yes”. Otherwise, print “No”.

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

const int MAXN = 10;
bool isSafe(int node, int graph[MAXN][MAXN], int path[], int pos) {
  if (graph[path[pos - 1]][node] == 0) {
    return false;
  }
  for (int i = 0; i < pos; i++) {      if (path[i] ==
node) {
      return false;
    }
  }
  return true;
}

bool hamiltonianPathHelper(int graph[MAXN][MAXN], int path[], int pos, int n) {    if (pos == n) {
    return true;
  }
  for (int node = 1; node < n; node++) {
    if (isSafe(node, graph, path, pos)) {          path[pos] = node;
      if (hamiltonianPathHelper(graph, path, pos + 1, n)) {              return true;
      }

      path[pos] = -1;
    }
  }
  return false;
}
bool hasHamiltonianPath(int graph[MAXN][MAXN], int n) {    int path[MAXN];
  memset(path, -1, sizeof(path));

  for (int start = 0; start < n; start++) {       path[0] = start;
    if (hamiltonianPathHelper(graph, path, 1, n)) {          return true;
    }
  }
```

```
    return false;
}

int main() {
    int graph[MAXN][MAXN] = {
        {0, 1, 1, 0, 0},
        {1, 0, 1, 1, 0},
        {1, 1, 0, 1, 1},
        {0, 1, 1, 0, 1},
        {0, 0, 1, 1, 0}
    };

    int n = 5;

    if (hasHamiltonianPath(graph, n)) {
        cout << "Yes" <<endl;
    } else {
        cout << "No" <<endl;
    }

    return 0;
}
```



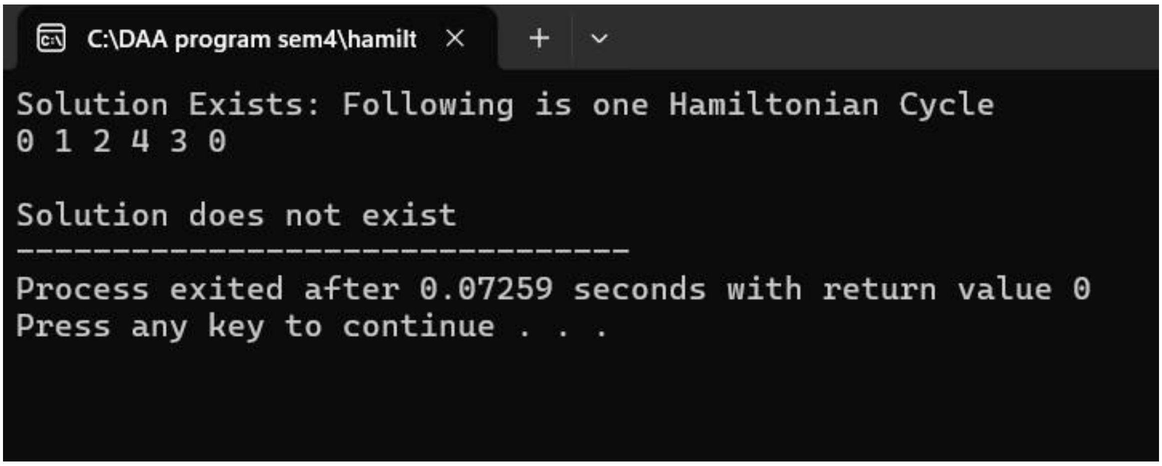
3. Write a program for finding the Hamiltonian Cycle or Hamiltonian Circuit in a graph using backtracking

```
#include <bits/stdc++.h>

using namespace std;

#define V 5


void printSolution(int path[]);


bool isSafe(int v, bool graph[V][V],

                        int path[], int pos)

{


        if (graph [path[pos - 1]][ v ] == 0)

                return false;



        for (int i = 0; i < pos; i++)
```

```
                if (path[i] == v)
                        return false;
        return true;
}

bool hamCycleUtil(bool graph[V][V],
                                int path[], int pos)
{

        if (pos == V)
        {
                if (graph[path[pos - 1]][path[0]] == 1)
                        return true;
                else
                        return false;
        }
        for (int v = 1; v < V; v++)
        {
                if (isSafe(v, graph, path, pos))
                {
                        path[pos] = v;

                        if (hamCycleUtil (graph, path, pos + 1) == true)
                                return true;
                        path[pos] = -1;
                }
        }

        return false;
}

bool hamCycle(bool graph[V][V])
{
        int *path = new int[V];     for (int i = 0; i <
V; i++)
                path[i] = -1;
```

```
        path[0] = 0;
        if (hamCycleUtil(graph, path, 1) == false )
        {
                cout << "\nSolution does not exist";
                return false;
        }

        printSolution(path);        return true;
}
void printSolution(int path[])
{
        cout << "Solution Exists:"
                        " Following is one Hamiltonian Cycle \n";
        for (int i = 0; i < V; i++)
                cout << path[i] << " ";

        cout << path[0] << " ";     cout << endl;
}

int main()
{

        bool graph1[V][V] = {{0, 1, 0, 1, 0},
                                        {1, 0, 1, 1, 1},
                                        {0, 1, 0, 0, 1},
                                        {1, 1, 0, 0, 1},
                                        {0, 1, 1, 1, 0}};

        hamCycle(graph1);

        bool graph2[V][V] = {{0, 1, 0, 1, 0},
                                        {1, 0, 1, 1, 1},
                                        {0, 1, 0, 0, 1},
                                        {1, 1, 0, 0, 0},
                                        {0, 1, 1, 0, 0}};
```

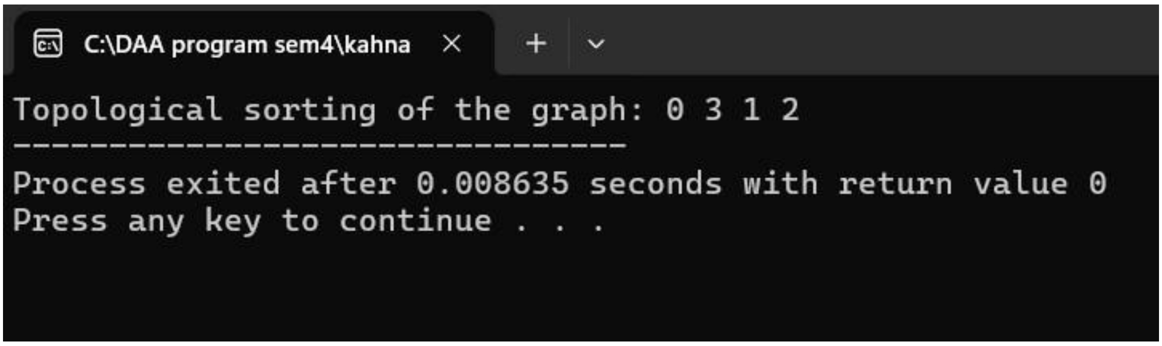
```
        hamCycle(graph2);

        return 0;
}
```



4. Topological sort using Kahn algo and DFS

```
#include <bits/stdc++.h>

using namespace std;

 vector<int> topologicalSort(vector<vector<int> & adj, int V)

{

        vector<int> indegree(V);

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

                for (auto it : adj[i]) {

                        indegree[it]++;

                }

        }

        queue<int> q;

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

                if (indegree[i] == 0) {

                        q.push(i);

                }

        }

        vector<int> result;          while (!q.empty())

{                int node = q.front();

                q.pop();
```

```
                result.push_back(node);                 for
(auto it : adj[node]) {
                        indegree[it]--;
        if (indegree[it] == 0)
                                q.push(it);
                }
        }
        if (result.size() != V) {               cout << "Graph contains
cycle!" << endl;
                return {};
        }

        return result;
}

int main()
{
        int n = 4;
        vector<vector<int> > edges
                = { { 0, 1 }, { 1, 2 }, { 3, 1 }, { 3, 2 } };
        vector<vector<int> > adj(n);            for (auto i : edges) {
                adj[i[0]].push_back(i[1]);
        }
        cout << "Topological sorting of the graph: ";
        vector<int> result = topologicalSort(adj, n);   for (auto i : result)
{
                cout << i << " ";
        }
        return 0;
}
```



5. Write a program to implement Ford-Fulkerson algorithm for Maximum Flow Problem

```
#include <iostream>

#include <limits.h>

#include <queue>

#include <string.h>

using namespace std;

#define V 6


bool bfs(int rGraph[V][V], int s, int t, int parent[])

{

bool visited[V];

memset(visited, 0, sizeof(visited));

queue<int> q;

q.push(s);

visited[s] = true;

parent[s] = -1;

while (!q.empty()) {

        int u = q.front();

        q.pop();


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

                if (visited[v] == false && rGraph[u][v] > 0) {

                        if (v == t) {

                                parent[v] = u;

                                return true;

                        }

                        q.push(v);

                        parent[v] = u;

                        visited[v] = true;

                }

        }

}

return false;

}

int fordFulkerson(int graph[V][V], int s, int t)

{
```

```
int u, v;

int rGraph[V][V];

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

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

                rGraph[u][v] = graph[u][v];


int parent[V];

int max_flow = 0;

while (bfs(rGraph, s, t, parent)) {

        int path_flow = INT_MAX;

        for (v = t; v != s; v = parent[v]) {

                u = parent[v];

                path_flow = min(path_flow, rGraph[u][v]);

        }

        for (v = t; v != s; v = parent[v]) {

                u = parent[v];

                rGraph[u][v] -= path_flow;

                rGraph[v][u] += path_flow;

        }

        max_flow += path_flow;

}

return max_flow;

}

int main()

{

int graph[V][V] = { { 0, 16, 13, 0, 0, 0 }, { 0, 0, 10, 12, 0, 0 },

                { 0, 4, 0, 0, 14, 0 }, { 0, 0, 9, 0, 0, 20 },

                { 0, 0, 0, 7, 0, 4 }, { 0, 0, 0, 0, 0, 0 } };


cout << "Maximum flow: " << fordFulkerson(graph, 0, 5);

return 0;

}
```