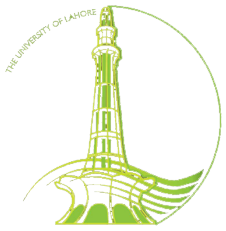


THE UNIVERSITY OF LAHORE
ISLAMABAD CAMPUS

# Data Structuers and algorithms (CS09203)

## Lab Report

| | |
|---|---|
| Name: | Irtaza Kashir Raja |
| Registration #: | SEU-F16-127 |
| Lab Report #: | 08 |
| Dated: | 16-04-2018 |
| Submitted To: | Mr. Usman Ahmed |

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

# Experiment # 8
# Graph and its representationsl

**Objective**
The objective of this session is to show the representation of graphs using C++.

**Software Tool**
1. Code Blocks with GCC compiler.

## 1 Theory

Graph is a data structure that consists of following two components: 1. A finite set of vertices also called as nodes. 2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Adjacency Matrix: Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

## 2 Task

### 2.1 Task 1

Write a C++ code using functions for the following operations.

1. Adding Edges to the Undirected Graph. 2. Adding Edges to the Directed Graph . 3. Adding Edges to the Weighted Graph.

### 2.2 Procedure: Task 1

```
#include <cstdio>
#include <vector>
#include <list>
#include <utility>
#include<iostream>
#include<iomanip>
using namespace std;
class Graph {
private:
      bool** adjMatrix;
      int numVertices;
public:
      Graph(int numVertices) {
            this->numVertices = numVertices;
            adjMatrix = new bool*[numVertices];
            for (int i = 0; i < numVertices; i++) {
                  adjMatrix[i] = new bool[numVertices];
                  for (int j = 0; j < numVertices; j++)
                        adjMatrix[i][j] = false;
            }
    }

      void addEdge(int i, int j) {
                  adjMatrix[i][j] = true;
                  adjMatrix[j][i] = true;
    }

      void removeEdge(int i, int j) {
                  adjMatrix[i][j] = false;
                  adjMatrix[j][i] = false;
    }

      bool isEdge(int i, int j) {
                  return adjMatrix[i][j];
    }

    void toString() {
```

```
        for (int i = 0; i < numVertices; i++) {
                    cout << i << " : ";
                    for (int j = 0; j < numVertices; j++)
                          cout << adjMatrix[i][j] << " ";
                    cout << "\n";
        }

    }

    ~Graph() {
            for (int i = 0; i < numVertices; i++)
                  delete[] adjMatrix[i];
            delete[] adjMatrix;
    }
};

void PrintMat(int **mat, int n)
{
        int i, j;

        cout<<"\n\n"<<setw(4)<<"";
        for(i = 0; i < n; i++)
                cout<<setw(3)<<"("<<i+1<<")";
        cout<<"\n\n";

        // Print 1 if the corresponding vertexes are connected otherwise 0
        for(i = 0; i < n; i++)
        {
                cout<<setw(3)<<"("<<i+1<<")";
                for(j = 0; j < n; j++)
                {
                        cout<<setw(4)<<mat[i][j];
                }
                cout<<"\n\n";
        }
}


int main(){
        char w;
```

```
        line:
        cout<<"press a for Adding Edges to the Undirected Graph\n";
        cout<<"\npress b for Adding Edges to the Directed Graph \n";
        cout<<"\npress c for Adding Edges to the Weighted Graph\n";
        cout<<"\npress d to Exit\n ";
        cout<<"\nEntre your choice\n";
        cin>>w;
        switch(w){
                case 'a':
                {
                                int a,b,c=0,y=0;
              cout<<"how many vertics"<<endl;
        cin>>c;
        Graph g(c);
           while(y!=c){

                    cout<<"add edges "<<endl;
        cin>>a>>b;
        g.addEdge(a,b);

        y++;

           }
        g.toString();

           break;
}
           case'b':

                int i, v, e, j, v1, v2;

        cout<<"Enter the number of vertexes of the graph: ";
        cin>>v;

        int **graph;
        graph = new int*[v];

        for(i = 0; i < v; i++)
        {
                graph[i] = new int[v];
```

```
                for(j = 0; j < v; j++)graph[i][j] = 0;
        }

        cout<<"\nEnter the number of edges of the graph: ";
        cin>>e;

        for(i = 0; i < e; i++)
        {
                cout<<"\nEnter the vertex pair for edge "<<i+1;
                cout<<"\nV(1): ";
                cin>>v1;
                cout<<"V(2): ";
                cin>>v2;

                graph[v1-1][v2-1] = 1;
                graph[v2-1][v1-1] = 1;
        }

        PrintMat(graph, v);
    break;

        case 'c':
    {
            int vertices, edges, v1, v2, weight;

 cout<<"Enter the Number of Vertices -\n";

 cin>>vertices;
cout<<vertices;
cout<<"Enter the Number of Edges -\n";

 cin>>edges;

 vector< list< pair<int, int> > > adjacencyList(vertices + 1);

 cout<<"Enter the Edges V1 -> V2, of weight W\n";

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

cin>> v1, v2, weight;
```

```
 adjacencyList[v1].push_back(make_pair(v2, weight));

 }

 cout<<"\nThe Adjacency List-\n";


 for (int i = 1; i < adjacencyList.size(); ++i) {

 cout<<"adjacencyList ", i;

 list< pair<int, int> >::iterator itr = adjacencyList[i].begin();

 while (itr != adjacencyList[i].end()) {

 cout<< (*itr).first, (*itr).second;

 ++itr;

 }

cout<<"\n";

 }break;

}
goto line;
   case 'd':
        exit;
}

 return 0;}
```

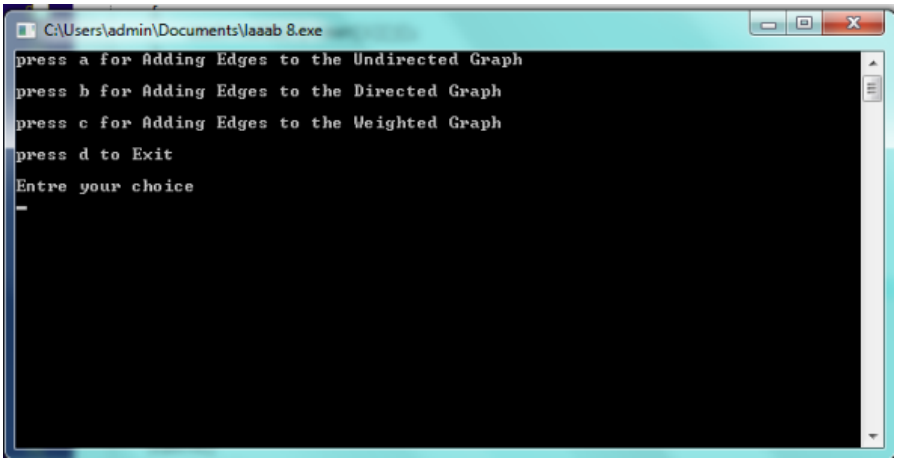


Figure 1: output