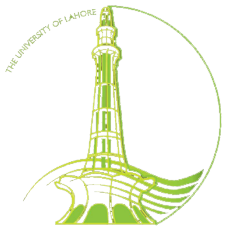

THE UNIVERSITY OF LAHORE
ISLAMABAD CAMPUS

Data Structure and Algorithms

Lab Report

Name:	Muhammad Arslan Ishaq
Registration #:	CSU-F16-112
Lab Report #:	07
Dated:	21-04-2018
Submitted To:	Mr. Usman Ahmed

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

Experiment # 8
Data Structure - Graph

Objective
The objectives of this lab session are to understand the basic of Graphs in Data Structure.

1 Theory

A set of items connected by edges. Each item is called a vertex or node. Formally, a graph is a set of vertices and a binary relation between vertices, adjacency.

Graph
Graph is a non linear data structure, it contains a set of points known as nodes (or vertices) and set of linkes known as edges (or Arcs) which connets the vertices. A graph is defined as follows... Graph is a collection of vertices and arcs which connects vertices in the graph Graph is a collection of nodes and edges which connects nodes in the graph Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.

Example
The following is a graph with 5 vertices and 6 edges. This graph G can be defined as G = (V , E) Where V = A,B,C,D,E and E = (A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D).

Graph Terminology:-
We use the following terms in graph data structure...

Weighted Graph:-
A weighted graph is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive)

Undirected Graph:-
A graph with only undirected edges is said to be undirected graph.

Directed Graph:-
A graph with only directed edges is said to be directed graph.

2 Lab Task

Write a C++ program to implement all the above described algorithms and display the following menu and ask the user for the desired operation.
The program should have the option for reusing it after you have completed the desired task.

2.1 Program

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

// To add an edge
void addEdge(vector <pair<int, int> > adj[], int u,
                                                    int
{
        adj[u].push_back(make_pair(v, wt));
        adj[v].push_back(make_pair(u, wt));
}

// Print adjacency list representaion ot graph
void printGraph(vector<pair<int,int> > adj[], int V)
{
        int v, w;
        for (int u = 0; u < V; u++)
        {
```

```
                cout << "Node " << u << " makes an edge with \n";
                for (auto it = adj[u].begin(); it!=adj[u].end(); it++)
                {
                        v = it->first;
                        w = it->second;
                        cout << "\tNode " << v << " with edge weight ="
                                << w << "\n";
                }
                cout << "\n";
        }
}

// Driver code
int main()
{
        int V = 5;
        vector<pair<int, int> > adj[V];
        addEdge(adj, 0, 1, 10);
        addEdge(adj, 0, 4, 20);
        addEdge(adj, 1, 2, 30);
        addEdge(adj, 1, 3, 40);
        addEdge(adj, 1, 4, 50);
        addEdge(adj, 2, 3, 60);
        addEdge(adj, 3, 4, 70);
        printGraph(adj, V);
        return 0;
}
```

3 Conclusion

Well to form it in a proper object oriented way i would make a class of edge which would contain the nodes it connects and its weight, another class of node which contains a list of edges(going in or coming out from that node). Lastly i would make a class of graph which would contain list of nodes it connects. Its a most rudimentary implementation details of implementation would vary depending on what queries are made. I have added a method for breadth-first search just to illustrate.