KATHMANDU UNIVERSITY

DHULIKHEL, KAVRE

School of Engineering

Lab Report No: 05

Data Structures and Algorithm (COMP 202)

Submitted By:

Aayush Pokhrel

Roll No:42

Group: CE

Level: 2nd Year/1st Sem

Submitted To

Dr. Rajani Chulyadyo

Department of Computer Science And Engineering

Objective: Implement graph data structure with the following operations:

(a) isEmpty(): Returns true if the graph is empty, and false otherwise

(b) isDirected(): Returns true if the graph is directed, and false otherwise

(c) addVertex(newVertex): Inserts a new vertex to the graph

(d) addEdge(vertex1, vertex2): Adds an edge from vertex1 to vertex2

(e) removeVertex(vertexToRemove): Remove a vertex from the graph

(f) removeEdge(vertex1, vertex2): Remove an edge from the graph

(g) numVertices(): Returns the number of vertices in the graph

(h) numEdges(): Returns the number of edges in the graph

(i) indegree(vertex): Returns the indegree of a vertex

(j) outdegree(vertex): Returns the outdegree of a vertex

(k) degree(vertex): Returns the degree of a vertex

(l) neighbours(vertex): Returns the neighbours of a vertex

(m) neighbour(vertex1, vertex2): Returns true if vertex2 is a neighbour of vertex1.

Description

A graph is a collection of nodes, called vertices, and line segments, called arcs or edges, that connect pairs of nodes.

There are variety of ways to represent graphs

- Two common ways:
- Adjacency matrix
- Incidence matrix
- Adjacency list

We have Used Adjacency List In Our Program to represent the graph data structure.

Main.cpp Code:

```
int main()
{
    bool direction = true;
    Graph graph(direction);

    graph.addVertice('A');
    graph.addVertice('B');
    graph.addVertice('C');
    graph.addVertice('D');
    graph.addVertice('E');

    graph.addEdge('A', 'B');
    graph.addEdge('A', 'D');
    graph.addEdge('A', 'E');

    graph.addEdge('B', 'A');
    graph.addEdge('B', 'D');
    graph.addEdge('B', 'C');

    graph.addEdge('C', 'D');
    graph.addEdge('D', 'A');

    graph.addEdge('E', 'C');
    graph.addEdge('E', 'A');

    graph.neighbours('A');
    graph.neighbours('B');

    cout << "Removing Edge A and D" << endl;
    graph.removeEdge('A', 'D');
    graph.neighbours('A');

    cout << "The outdegree of A is " << graph.outdegree('A');
    cout << "Number of Vertice: " << graph.numVertices() << endl;
    cout << "Is B and A neighbour: " << graph.neighbour('A', 'B') << endl;

    // Print the indegree and degree of vertex 'A'
    cout << "Indegree of vertex A: " << graph.indegree('A') << endl;
    cout << "Degree of vertex A: " << graph.degree('A') << endl;

    // Remove vertex 'B' and print the updated graph information

    graph.removeVertice('B');
    cout << "Number of vertices after removing B: " << graph.numVertices() <<
endl;
```

```
    cout << "Number of edges after removing B: " << graph.numEdges() << endl;

    return 0;
}
```

Output Of The Above Problem:

```
The neighbour of A are :
B
D
E
The neighbour of B are :
A
D
C
The outdegree of A is 3
Number of Vertice: 5
Is B and A neighbour: 1
Indegree of vertex A: 3
Degree of vertex A: 6
Number of vertices before removing B: 5
Number of vertices after removing B: 4
Number of edges after removing B: 6
```

Github Link: [1014Aayush/CE_2021_42_Graph (github.com)]