BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI
Work Integrated Learning Programmes Division
Data Structures and Algorithms
Lab Sheet -2 Topic: Airline Tickets

General Instructions for Programming :

1. All inputs to the program must be command line arguments.

2. Use standard Java coding conventions. Each class: Graph, Queue, Stack and Main code should be written in separate .java files.

Problem Statement:

a. Create an ADT for an undirected, weighted graph, represented by an adjacency matrix. Use whole numbers for the weights.
b. Write a simple application that simulates an airline's graph of ticket prices. The program should read in vertices and prices and keep an array that maps vertices to the names of the airports. For each route that the airline does not fly, store Zero. Once the graph is set up, the program should accept queries consisting of the names of two airports and either report a cost or report that no flight is available.
c. The (Depth-First Search DFS) or (Breadth First Search BFS) algorithm can be used to traverse all the vertices of a graph

Below is the list of operations the program should be able to perform

1. Add Vertex (Airport / City)
2. Remove Vertex (Airport / City)
3. Add Edge (Source, Destination and Airfare)
4. Remove Edge
5. Display Adjacency Matrix
6. Display Edges
7. BFS
8. DFS
9. Check Flight Availability

Also ensure to include a proper exit sequence from the program.

Function Descriptions:

1) Add Vertex: void addVertex(String s)
Precond: s is the Vertex (airport) as input in the command prompt
Effect : a new vertex is added to the Graph. If the Graph is empty, a new vertex is created at the start

2) Remove Vertex: String deleteVertex(int n)
Precond : All vertices are already added to the Graph
Effect : the program finds the vertex at the entered position and removes it from the list of vertices. It also checks if there are any edges associated with that vertex and resets their weight to Zero. It also

adjusts the weights in the Adjacency Martix to reflect the correct weights for the other edges in the graph.

3) Add Edge: void addEdge(String v,String w, int x)
Precond : All vertices v and w are already added to the Graph
Effect : the program finds the position of each of the vertices v and w and records the weight x against edge for v and w

4) Remove Edge: void removeEdge(String v,String w)
Precond : All vertices are already added to the Graph
Effect : the program finds the position of the vertices in the Graph and resets the corresponding edge weight to Zero.

5) Display Adjacency Matrix: void displayAdj()
Precond: Graph is not empty
Effect: Displays all vertices in a 2 by 2 matrix. Each valid edge is represented by the weight of the edge displayed in the corresponding position in the matrix. Each invalid edge is represented by Zero weight.

6) Display Edges: void displayEdges()
Precond : Graph is not empty
Effect: Displays all combinations of valid edges. For example: if there is a valid edge between A and B, then the program will show both “A->B” and “B->A”

7) BFS: void BFS(String v)
Precond : Graph is not empty
Effect : Display the vertices in the Graph starting with the vertex input by the user and then traversing the graph using BFS.

8) BFS: void DFS(String v)
Precond : Graph is not empty
Effect : Display the vertices in the Graph starting with the vertex input by the user and then traversing the graph using DFS.

9) Check Flight Availability: void flightAvailable(String s,String d)
Precond : Graph is not empty
Effect : Based on the source s and destination d entered, the program displays the weight (cost) of the edge if it is a valid edge. If the edge is not a valid one, a relevant message is displayed.

Input:
The input to the program should be provided real time through the command prompt.
The following input considerations need to be made:

1. Each of the operations should be driven by a menu and sub menu option.

Sample Input for Airline Tickets:

Vertices: A, B, C, D

Adjacency Matrix showing edge weights:

```
  A B C D
A 0 5 9 7
B 5 0 0 6
C 9 0 0 8
D 7 6 8 0
```

Sample Output:

1. Add Vertex

```
1.AddVertex
2.removeVertex
3.AddEdge
4.removeEdge
5.DisplayAdjacentMatrix
6.DisplayEdges
7.BFS
8.DFS
9.Check Flight availability
0.exit()

enter ur choice
1
enter vertex
A
```

2. Add Edge

```
1.AddVertex
2.removeVertex
3.AddEdge
4.removeEdge
5.DisplayAdjacentMatrix
6.DisplayEdges
7.BFS
8.DFS
9.Check Flight availability
0.exit()
```

```
enter ur choice
3
enter 2 vertices and weight for creating edge
A C 9
```

3. Display Adjacency Matrix

```
1.AddVertex
2.removeVertex
3.AddEdge
4.removeEdge
5.DisplayAdjacentMatrix
6.DisplayEdges
7.BFS
8.DFS
9.Check Flight availability
0.exit()


enter ur choice
5
 A B C D
A 0 5 9 7
B 5 0 0 6
C 9 0 0 8
D 7 6 8 0
```

4. Display Edges

```
1.AddVertex
2.removeVertex
3.AddEdge
4.removeEdge
5.DisplayAdjacentMatrix
6.DisplayEdges
7.BFS
8.DFS
9.Check Flight availability
0.exit()

enter ur choice
6
A->B
A->C
A->D
B->A
B->D
C->A
C->D
D->A
D->B
D->C
```

5. Check Flight Availability

```
1.AddVertex
2.removeVertex
3.AddEdge
4.removeEdge
5.DisplayAdjacentMatrix
6.DisplayEdges
7.BFS
8.DFS
9.Check Flight availability
0.exit()

enter ur choice
9
enter souce and destination
A D
Flight Price is    7
```

Deliverables:

1. graph.java – should contain the basic graph definition and all functions mentioned above.
2. GraphClass.java – should contain the main function.
3. Queue.java – should contain the basic queue definition and basic functions required to perform BFS
4. Stack.java – should contain the basic stack definition and basic functions required to perform DFS.

Required Text Book Reading:

1) Section 6.1, 6.2, 6.3, 6.4 of Algorithm Design by Michael Goodrich and Roberto Tamassia