COMP 282 Project 2

NOTE: THE ASSIGNMENTS WILL BE GRADED USING AUTOMATED SCRIPTS SO PLEASE FOLLOW THE SUBMISSION GUIDELINES PROVIDED. PLEASE GO THROUGH THIS ENTIRE DOCUMENT BEFORE CODING AND SUBMITTING THE ASSIGNMENT.

Problem: Complete the code to find the SHORTEST PATH from a given origin vertex in Graph G(V, E) to every other vertex V in graph G.

Use the classes and interface provided in Project2.zip


Project2.zip

The Project2.zip contains Graph.java, GraphInterface.java, Test.java and input.txt

ADD YOUR CODE to below methods in Graph.java

- public List<List<Edge>> createWeightedGraph(List<V> vertices, int[][] edges) //TODO
- public Tree getShortestPath(V sourceVertex,List<V> vertices,List<List<Edge>> neighbors) //TODO

DO NOT MODIFY ANY OTHER METHODS AND FILES.

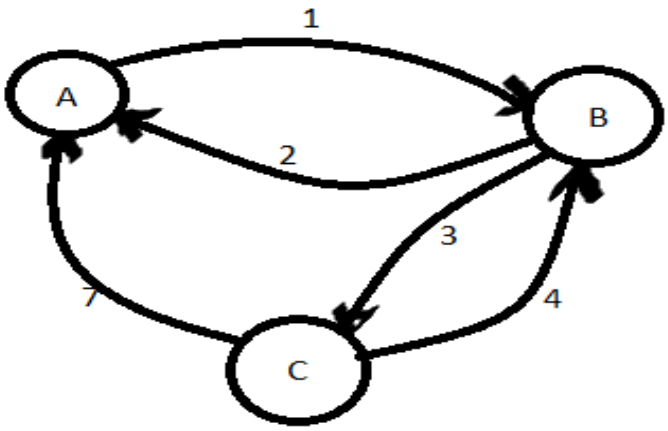
Sample input provided in input.txt file. All inputs will have a connecting path between every pair of vertices and the connecting path may consist of one or more edges.

```
EXPLANATION in COMMENTS FOR INPUT in input.txt file BELOW

3 5 3       //NUMBER_OF_VERTICES   NUMBER_OF_EDGES   NUMBER_OF_ORIGINS
A           //NAME_OF_VERTEX_WITH_INDEX_0
B           //NAME_OF_VERTEX_WITH_INDEX_1
C           //NAME_OF_VERTEX_WITH_INDEX_2
0 1 1       //EDGE FROM VERTEX_A (INDEX_0) TO VERTEX_B (INDEX_1) and WEIGHT_1
1 2 3       //EDGE FROM VERTEX_B (INDEX_1) TO VERTEX_C (INDEX_2) and WEIGHT_3
2 0 7       //EDGE FROM VERTEX_C (INDEX_2) TO VERTEX_A (INDEX_0) and WEIGHT_7
2 1 4       //EDGE FROM VERTEX_C (INDEX_2) TO VERTEX_B (INDEX_1) and WEIGHT_4
1 0 2       //EDGE FROM VERTEX_B (INDEX_1) TO VERTEX_A (INDEX_0) and WEIGHT_2
A           //ORIGIN IS A
B           //ORIGIN IS B
C           //ORIGIN IS C
```

NOTE: The inner Class Edge and Tree defined in Graph.java use index to refer vertices. In the example above Vertex A is referred by index 0 and similarly Vertex B by index 1 and Vertex C by index 2.

The graph G(V, E) for the above input in input.txt file is as below:



The output for the above input in input.txt file should be as below:

```
All shortest paths from A are:
A path from A to A: A (cost: 0.0)
A path from A to B: A B (cost: 1.0)
A path from A to C: A B C (cost: 4.0)

All shortest paths from B are:
A path from B to A: B A (cost: 2.0)
A path from B to B: B (cost: 0.0)
A path from B to C: B C (cost: 3.0)

All shortest paths from C are:
A path from C to A: C B A (cost: 6.0)
A path from C to B: C B (cost: 4.0)
A path from C to C: C (cost: 0.0)
```

Since the input had 3 origins A, B, C the output shows shortest paths to every other node from all the three origins A,B,C. DO NOT WRITE YOUR OWN METHOD TO DISPLAY THE OUTPUT.

Hints:

1) The adjacency list returned by the createWeightedGraph method in Graph.java for above input in input.txt file should be as below

Vertex 0 i.e. A	Edge 0 to 1 weight 1	
Vertex 1 i.e. B	Edge 1 to 0 weight 2	Edge 1 to 2 weight 3
Vertex 2 i.e. c	Edge 2 to 0 weight	Edge 2 to 1

	7	weight 4

The above adjacency list is returned in an object of Arraylist<Arraylist<Edge>> from createWeightedGraph method.

2) The Tree returned from method getShortestPath in Graph.java for above input in input.txt file should be as below
    a. When Origin is A the shortest distance tree is as below:
        i. Root = 0

        ii. Parent[]=

Value (Parent of Vertex)	-1	0	1
Index (Vertex)	0	1	2

        iii. Cost[]=

Value (cost of path from root to Vertex)	0	1	4
Index (Vertex)	0	1	2

    b. When Origin is B the shortest distance tree is as below:
        i. Root = 1

        ii. Parent[]=

Value (Parent of Vertex)	1	-1	1
Index (Vertex)	0	1	2

        iii. Cost[]=

Value (cost of path from root to Vertex)	2	0	3
Index (Vertex)	0	1	2

    c. When Origin is C the shortest distance tree is as below:
        i. Root = 2

        ii. Parent[]=

Value (Parent of Vertex)	1	2	-1
Index (Vertex)	0	1	2

        iii. Cost[]=

Value (cost of path from root to Vertex)	6	4	0
Index (Vertex)	0	1	2

For each of the above origin case the tree is returned as an object of Tree class in Graph.java.

NOTE: The code should be tested using Test.java class provided in project2.java. **DO NOT WRITE ANY METHOD TO PRINT THE OUTPUT.** Test.java will call the appropriate method to print the output.

ECLIPSE IDE: If you are using any IDE the input file should be read from path "src/input.txt". If you are using java command from a terminal or command prompt read file using only file name "input.txt".

SUBMISSION DETAILS: Assignment will be graded using automated scripts so please make sure you follow the below-mentioned submission guidelines

1) DO NOT USE '/* */' for comments, use '//' instead.
2) Create a new folder for your group. Example: For Group 1, create a directory/folder called **Group1**. DO NOT USE ANY SPACES OR SPECIAL CHARACTERS FOR THE DIRECTORY NAME.
3) Copy all the four files Graph.java (Complete with your code), GraphInterface.java, Test.java and input.txt in your group directory.
4) DO NOT ALTER THE INPUT METHOD FOR FILE IN TEST.JAVA FILE. If you do then make sure you set the input file path in Test.java is "input.txt" as the assignment will be graded through the terminal. i.e. below code should be used to read the input file in Test.java

   ```
   File file = new File("input.txt")
   ```

5) **Java version 8** should be used for compiling and executing the code.
6) Your code should now compile from command prompt or terminal using: **javac *.java**
7) Run your code using **java Test** and check the output in terminal or command prompt
8) If everything is working fine, zip the directory for your group created in step 1. Example for group 1 a zip archive Group1.zip will be created.
9) Make sure your zip folder contains a single folder with all four files Graph.java (Complete with your code), GraphInterface.java, Test.java and input.txt files inside the folder. Example: For Group 1, the hierarchy should be like the following

   Group1.zip/Group1/(all 4 files)

   DO NOT KEEP EXTRA FOLDERS OR PUT ALL FILES DIRECTLY INSIDE ZIP FOLDER. IT SHOULD BE EXACTLY AS MENTIONED ABOVE.

10) Finally, submit Group1.zip archive in canvas.