CSCI 2110 – Data Structures in Java
Assignment A5

The goal of this assignment is to practice on *Graphs*.

You are given an implementation of graphs, based on adjacency list, in the following java files:

1. Vertex.java: class for each vertex
2. Neighbor.java: class that keeps a vertex number as a neighbor of another vertex
3. DirGraph.java: class for a directed graph
4. UndirGraph.java: subclass of DirGraph for an undirected graph

Based on the classes provided:

a) (10 points) By sub-classing the classes of Neighbor and UndirGraph implement a WeightedUndirGraph class for a weighted undirected graph. Override the inherited methods for this purpose. Weights are positive integer values assigned to the edges.

b) (10 points) Draw two weighted undirected connected graphs by mspaint, and save them as .jpg files. Each graph should have at least eight nodes and fifteen edges. Weights should be positive. Submit the images along with your codes.

c) (10 points) For each graph of part b, build the respective weighted graph by using the java classes. Declare a main method in class *TestGraph* and build the graphs one by one. Provide enough comments for this part so markers can understand how the graphs are created.

d) (15 points) Implement DFS algorithm and add it as a public method to UndirGraph.java. It takes a starting vertex as input and outputs a sequence of vertices visited. You must implement DFS by using a **stack**. You do not need to mark edges. You probably need to add a field to Vertex.java. Feel free to add a field. For each graph built in part c, take a starting vertex and print the output of DFS. If you implement DFS by recursion, you will not get any point.

e) (15 points) Implement BFS algorithm and add it as a public method to DirGraph.java. It takes a starting vertex as input and outputs a sequence of vertices. You must implement BFS by using a **queue**. You do not need to mark edges. You probably need to add a field to Vertex.java. For each graph built in part c, take a starting vertex and print the output of BFS. If you implement BFS by recursion, you will not get any point.

f) (20 points) Implement the Dijkstra's algorithm and add it to the UndirGraph class as a public method. It takes a starting vertex S and prints out the length of shortest path from S to each vertex of the graph. For each graph built in part c, choose a starting vertex and call the Dijkstra's method.
The output should look like the following:
S -> B: 3
S -> A: 2
S -> V: 4
…

g) (20 points) Implement the Kruskal's algorithm and add it to the UndirGraph class as a public method. It prints out the vertices of the minimum spanning tree of the graph. For each graph built in part c, call the Kruskal's method.

**Your main method of TestGraph should look like the following:*

Creating the first graph by calling the required methods

Calling its DFS

// output

Calling its BFS

// output

Calling Dijkstra (S)

// output

Calling Kruskal

// output

Creating the second graph by calling the required methods

Calling its DFS

// output

Calling its BFS

// output

Calling Dijkstra (S)

// output

Calling Kruskal

// output