CS 260
Programming Assignment 8

For this lab, you will create two different graph classes: one for a non-weighted directed graph and one for a weighted non-directed graph. The first will be used to create a connectivity table and the second will be used to create a min-cost spanning tree.

This lab should follow the course coding requirements and be split into multiple files as per your chosen language. There is a driver provided for this lab, you should test each part of your code as it is developed.

Connectivity Table

Create a **non-weighted directed** graph class and use it to display a connectivity table. To work with the driver, the class should be named DGraph and it needs the following methods: addNode, addEdge, breadthFirst, and connectTable. A depthFirst traversal is optional.

Optional methods to display that table may also be implemented. The driver has commented out calls to such methods using the names listNodes, displayEdges, and displayMatrix.

Once the class is working properly, create a new method named connectTable that returns a string. This method should show what nodes can be reached from node in turn. The output should appear as shown below.

```
A: C D E
B: D
C:
D: A E
E: B
```

The first letter output indicates the starting node and the letters after the colon show those nodes which can be reached from that starting node.

In Moodle, there are documents describing how to implement a directed graph and connectivity tables. You should base your solution off the provided pseudo code algorithms in these documents.

Minimum Cost Spanning Tree

Create a **weighted non-directed** graph class and use it to display a minimum cost spanning tree. To work with the driver, the class should be named WGraph. It will need the same set of methods as above, with the addition of a method named minCostTree, that is passed a character for the starting node and returns a string. The minCostTree method should generate output as shown below.

```
A-B B-C A-D D-E C-F B-G
```

This algorithm requires implementation of a variation on a priority queue that stores edges and ranks them based on their weights. It must be searchable and allow removing edges other than the minimal one. There is an algorithm in Moodle that describes such a priority queue. The optimal solution is to use an unsorted array or list that is searched to find the smallest edge and has an intelligent addEdge method.

This work by Jim Bailey is licensed under a Creative Commons Attribution 4.0 International License.