**CSCI 385 Algorithms and Data Structures**

**Homework #6 –There is no tree like a Minimal Spanning Tree**

**Due: Friday December 7, 2018 6:00 pm**

**(100 pts)**

**Extra Credit (60 pts)**

1. Create a *Graph* class that uses an adjacency list representation of the graph. In many graph algorithms, it is often necessary to label graph vertices with strings. In this case, you would need to create an additional structure to store the string associated with the vertex. The vertex name is a characteristic of the vertex and an array can be easily used to map the name to a vertex number. Incorporate a way to allow strings to be used as vertex labels in your *Graph* class. Build a graph to use to test your *Graph* class and then traverse the graph and print it out. To build the graph, the client should be able to input the number of nodes in the graph, provide the “string” label for each node, and indicate what nodes have edges between them. Include the source code, a “drawing” of the graph you are using to test your class, and the output produced from this test. Does your representation work for both directed and undirected graphs? (15 pts)
2. In class, we discussed two graph traversal methods. These methods differ in that they impose an ordering in which the nodes are visited. What are these two methods? Explain how each impose an ordering on the nodes that are visited during traversal. Add one these of these traversal methods to your graph implementation from problem #1 and provide a mechanism to test the implementation. Include the source code, a “drawing” of the graph you are testing and output from these tests. (25 pts)
3. As part of your Graph class, implement Prim’s algorithm for finding a minimal spanning tree of a graph G = (V, E). Provide a mechanism to test the implementation. Include the source code, a “drawing” of the graph you are testing and output from these tests. (30 pts)
4. (**Extra Credit**) Also as part of your Graph class, implement Kruskal’s algorithm for finding a minimal spanning tree of a graph G = (V, E). Provide a mechanism to test the implementation. Include the source code, a “drawing” of the graph you are testing and output from these tests. (40 pts)
5. (**Extra Credit**) Implement Dijkstra’s algorithm for finding a minimum cost path from a specified source node to a specified destination node as part of your graph class in Problem #1. Your implementation must include a priority queue. (20 pts)
6. Answer the following questions about minimal spanning trees:
7. State the “cut property” for knowing when to include an edge in a minimal spanning tree. Using the “cut property” provide an argument that Prim’s does indeed produce a minimal spanning tree of a graph G=(V,E). (10 pts)
8. State the “cycle property” for knowing when *not* to include an edge in a minimal spanning tree. What algorithm utilizes the “cycle property” to show it produces a minimal spanning tree? Explain this algorithm. (10 pts)
9. (2A) Will both Prim’s and Kruskal’s algorithms *always* produce the same MST of G? Either prove that it does or show a counterexample. (5 pts)
10. MSTs have been studied for many years (Prim’s is ~1930 and Kruskal’s is ~1956) and have been proven to produce correct results. Let’s say we wanted to find the ***maximum spanning tree*** of a graph G. How might we do this by preserving the use of the algorithms exactly as they exist? (5 pts)

You should submit the following ***all*** ***in one file*** labeled ***Assign6*** and placed into the D2L dropbox:

* Source code for the graph class and test clients – separated by question
* Image of the graphs used to test your implementation
* Test client output
* Answers to the questions provided