**Final Project: Graphs**

**Description**: Create a program that will read a graph from files passed in from the command line. The files passed in will have an adjacency matrix for either an undirected weighted graph or a directed unweighted graph. Those are the only two cases you for which you need to account.

**Requirements**:

- Your program will begin by opening a single graph file passed in from the command prompt. The files given to you are graph01.dat and graph02.dat. You can assume the files have no errors or strange configurations, and that the file structure is constant, but you **cannot** assume the file is one type of graph or the other. Your program must determine the kind of graph from the contents of the file itself.
- You must also create your own graph03.dat from the graph in figure 20-36 (see the graph theory assignment for the figure).
- For all **three** graph files, demonstrate you can create a fully functional and self-contained graph object with all the following methods:
    - Breadth-First traversal
        - Output the graph to the console in breadth-first fashion. This method may print.
        - You must demonstrate in automated fashion that the BFT can start from any node.
    - Depth-First traversal
        - Output the graph to the console in depth-first fashion. This method may print.
        - You must demonstrate in automated fashion that the DFT can start from any node.
    - **BONUS**: Point to Point Traversal
        - Demonstrate you can attempt a traversal from any point to any point.
        - Output the cost of the traversal if it's a weighted graph.
        - Output an error if the path cannot be found (i.e. if the start or end is disconnected)
        - You do not need to find the optimal path, or an efficient path, only a possible or not possible path.
        - You can use BFT or DFT.
        - **This is worth up to 5% added to your Test 1 score.**
    - Adjacency Matrix:
        - Output the adjacency matrix **to a file**.
        - Your output should look exactly like the input files. In other words, you should be able to recreate the input files from your object without the original input file.
        - Name your output files different than the input files.
    - Manipulation Methods: Implement the manipulation methods listed here.
        - Add a vertex. This may create a disconnected graph, account for that case.
        - Remove a vertex. This may create a disconnected graph, account for that case.
        - Add an edge between any two distinct edges (if one does not already exist, error case if it does).
        - Remove an edge. This may create a disconnected graph, account for that case.
    - Informational Methods: Implement all the informational methods listed here.
        - Get number of vertices.
        - Get number of edges.
        - Is Connected. This will return true or false..
        - List Disconnected: List the vertices (if any) that are not connected.

- You must start a code repository properly with only .ccp, .h files (optional readme and/or make file, but no IDE project files or other “stray” files) and use proper commits throughout your coding (starting from the first file and the first lines of code). You must fully demonstrate proper version control using regular and intelligent commits.
- Demonstrate all standards, processes, procedures, proper architecture, best practice, and proper testing discussed throughout the semester.

Think of this assignment as the computer science version of a term paper. You are simply being given a topic and some basic I/O guidelines. Your goal is to demonstrate all the skills and best practices that have been discussed throughout the semester. **This assignment must be submitted on or before the due date. It is a ONE time submission. You must be present for the last class, show your repository, and take a quiz on this assignment to get credit. Think of this as your “final exam.”**