**SP #31– Graph Exercises**

**Purpose:**

This project is intended to help you improve your software development skills with either C++ and the standard C++ library or C# and the .Net framework. It is not trivial, but not too complex either, and its relatively easy to understand what the code needs to do.

Typical applications for a graph class are to record dependency information between code packages, to represent relationships between classes in some set of packages, or to record relationships between people and tasks in a software development process.

Develop a graph class, for modeling directed graphs of vertices with directed edges between some of the vertices. Provide useful member functions, including methods to build, edit, search, and persist. Provide additional graph algorithms for finding strong components, shortest path between two vertices, and longest path.

The C# dictionary or C++ unordered\_map have all of the data handling facilities needed to store graph information. The purpose of the graph class is to wrap one of these data structures to specialize for managing graph data. Note that this project can be implemented in either C# using the .Net Framework or C++ and the standard C++ library. You might wish to implement both to get practice and strengthen your skills in both programming environments.

**Requirements:**

The requirements of this project are to:

1. Implement a class to represent directed graphs, e.g., collections of vertices some of which are joined by directed edges to other vertices. Please use either a .Net Dictionary<V,E> or C++ std::unordered\_map<V,E> to implement the data storage for this class.
2. Build the class with templates (C++) or generics (C#). The template parameters are unspecified classes V and E, where V holds information needed for each vertex, and E holds information needed for each edge.
3. Support search using Depth First Search (DFS) and Breadth First Search (BFS)[[1]](#footnote-1).
4. Find Strong Components using the Tarjan algorithm[[2]](#footnote-2).
5. Finding longest path should be simply a matter of executing DFS and doing a bit of recording of paths, always saving the longest.
6. Support persistence of graph information to and from an XML file.
1. You may wish to look at this demo:

https://ecs.syr.edu/faculty/fawcett/handouts/Coretechnologies/Cpp/Code/TreeWalkDemo/
These demos work for acyclic trees. For graphs you need to add vertex marking so that your search does not travel forever around a dependency cycle. [↑](#footnote-ref-1)
2. https://en.wikipedia.org/wiki/Tarjan's\_strongly\_connected\_components\_algorithm [↑](#footnote-ref-2)