**Homework 4: Social Networks**

**Design Choices:**

* Base data structure
  + I chose to build the social network on top of an Adjacency Map Graph because it allows for the fastest lookup time (constant). I did this because speed is more important than space for Internet applications, as the host can always purchase more memory, but the user does not want to wait while a profile is queried from the network.
  + I first wanted to outline the basic structure of the Adjacency Map Graph. The graph is composed of a collection of Vertex and Edge objects that are stored in an ordered list. Each Vertex and Edge has an instance variable specifying its position in that list, so it can be indexed into in constant time. Vertices hold low-level personal identifying information, and Edges hold low-level connections. This information is represented by the hypothetical instance variable of type PERSONAL\_DATA\_t. For simplicity, it is assumed that the interface for this data structure is defined elsewhere.
* Multiple levels of objects
  + I chose to use multiple levels of objects to simplify code and leverage the advantages of polymorphism. Instead of designing the social network from the ground up, it was conceptually easier (and more consistent with the principles of information hiding) to have the network be an implementation of a simple data structure. Each vertex encapsulates a person, and the social network does not need to know how the innards work, only work with the external interface.
* Member Functions
  + I tried to pick member functions that would be useful such as querying people you might know (friends of friends), done simply by iterating through the friends (opposite vertices of incident edges) of your friends. I also wanted to look up a profile based on a string (done via hash ma[), and query people who have similar jobs (assuming there is some external database containing a similar graph connecting jobs that the function could access).

**Issues and Improvements:**

* Multiple names
  + My implementation is problematic for a real implementation of a social network, as it cannot handle multiple profiles with the same name. As it stands, identical names would be handled by a validation function that checks if a name is already in the list, and throw an exception if it is not.
  + **Potential Solution:** Instead of storing vertices as values in the name2vertex and vertex2profile hash maps, store a list of vertices. This way, the key “John Doe” would map a list of all vertices with that name, and then you would iterate through that list in order to identify the correct profile based on user I/O or other parameters.