Building and usage:

To build just type make in the directory. The executable will be called “prog”.
Pass the text file name like so: “./prog verts.txt”.

Generic lists:

My code uses many lists for different kinds of data. To help reduce the need for so many list structs I created a generic list struct. Insertion and removal can be done with generic methods, but searching still requires a dedicated method for each data type.

A list struct has field for the head and tail.
The head and tail are list_entry structs, which contain a void pointer to the data they store.
It is the responsibility of the user to know what the pointer should be cast to.

Finding subgraphs:

To find subgraphs I start with the first vertex loaded from the file. Each of this vertex’s connections is visited, and their connections in turn are visited, and so on. Each time a connection a vertex is reached this way it is added to the first subgraph.

I then find the next vertex which was not added to the first subgraph. The second subgraph is then created with this vertex, its connections, and its connections’ connections and so on.

This process is repeated until every vertex has been added to a subgraph. Some vertices may be in a subgraph that only has themselves as an entry.

Finding the distance between node paris in a subgraph:

Every permutation of pairs is sent to the *find_distance_between_vertices* method, which recursively goes through the connections looking for the shortest path between the provided vertices.

To avoid infinite loops and going down connections that have already been checked, a list of visited connections is kept and passed to the next recursive call.

Example files:

I have included two sample vertex files. “verts1.txt” is the same as the graph described in the assignment, while “verts2.txt” has multiple subgraphs.