Assignment Objectives:
- Implement the graph ADT using a dynamic two-dimensional array.
- Implement Dijkstra's Algorithm for determining shortest paths from a designated vertex.
- Implement Prim's Algorithm for determining a minimum spanning tree.

In this assignment, you shall experimentally compare the effectiveness of Dijkstra's Algorithm and Prim's Algorithm at doing each other's jobs. Recall that Dijkstra's Algorithm determines the shortest path from a given vertex in a connected, weighted, undirected graph to every other vertex in the graph. Consequently, Dijkstra's Algorithm yields a spanning tree of the graph, but that spanning tree may not have minimum cost. Prim's Algorithm determines a minimum spanning tree for a graph, but it may not yield the cheapest path from any vertex to the rest of the vertices in the graph.

For example, for the graph in Figure 1 below, Dijkstra's Algorithm yields the spanning tree in Figure 2 (when vertex 1 is the source), while Prim's Algorithm yields the spanning tree in Figure 3. Note that the cost of the spanning tree generated by Dijkstra's Algorithm is 13, while the minimum spanning tree generated by Prim's Algorithm costs 11. However, the route from vertex 1 to vertex 8 costs 10 with Prim's Algorithm, while the route with those endpoints only costs 5 with Dijkstra's Algorithm.

Your assignment is to write a program that queries the user for the name of a file containing information about a connected, weighted, undirected graph, and then runs Dijkstra's Algorithm (with a user-specified source) and Prim's Algorithm for that graph. The output should consist of a comparison of how well the two algorithms perform each other's tasks. A sample session is demonstrated at right, using the example in the figures above. The data file for this graph can be found on the course website.

To implement this program, you'll need a `Graph` class (defined in `graph.h` and implemented in `graph.cpp`) with the following members:

- A two-dimensional adjacency matrix of integer cost values, and an integer indicating the number of vertices. The adjacency matrix should be dynamic, i.e., a dynamic array of dynamic arrays, as demonstrated below.

```
typedef int* IntArrayPtr; // Defined within graph.h

IntArrayPtr *adjacencyMatrix; // Graph data member
int numberOfVertices; // Graph data member

adjacencyMatrix = new IntArrayPtr[numberOfVertices]; // Dynamic creation of array
for (int k = 0; k < numberOfVertices; k++) // once numberOfVertices has
    adjacencyMatrix[k] = new int[numberOfVertices]; // been initialized in Graph.cpp
```
• Default, copy, and initializing constructors (the initializing constructor should have a parameterized size). For this assignment, the initializing constructor should use a cost value of 0 to indicate the absence of an edge.

• Setting and accessor functions for the number of vertices, and for a particular edge’s cost in the adjacency matrix. Note that the undirected nature of the graph means that the adjacency matrix is symmetric, i.e., the (i,j) slot value always equals the (j,i) slot value.

• A public member function to perform Dijkstra’s Algorithm on the Graph (returning a duplicate Graph, restricted to the edges between vertices and their Dijkstra predecessors). Note that the climits library contains an INT_MAX constant that will be useful in determining lowest-cost paths.

• A public member function to perform Prim’s Algorithm on the Graph (returning a duplicate Graph, restricted to the minimum spanning tree edges determined by Prim).

• A public member function to calculate the cost on either the Dijkstra result or the Prim result from a designated vertex to each other vertex in the Graph.

• A public member function to determine the total cost of all edges on either the Dijkstra result or the Prim result.

For the graph shown in Figure 1, using vertex 1 as the source, the Dijkstra’s Algorithm steps would include:

```
Original Adjacency Matrix
3 2 1 0 0 0 0
2 3 1 0 0 0 0
1 0 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
Initial Adjacency Matrix
3 2 1 0 0 0 0
2 3 1 0 0 0 0
1 0 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
Final Adjacency Matrix
```

Similarly, for the graph shown in Figure 1, Prim’s Algorithm would perform the following steps in order to compute a minimum spanning tree:

```
Original Adjacency Matrix
3 2 1 0 0 0 0
2 3 1 0 0 0 0
1 0 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
Initial Adjacency Matrix
3 2 1 0 0 0 0
2 3 1 0 0 0 0
1 0 1 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
```

```
Connected
```

```
Connected
```

Include explanatory comments that document the purpose of each file and function, and make certain that your program exhibits modularity, modifiability, and readability.

Zip-compress your entire program folder and place it on your dropbox by the deadline mentioned above.

---

You must write your own code on this assignment, with adequate explanatory documentation. Obtaining code assistance from any outside source is considered academic misconduct. The only person permitted to see your code prior to the assignment deadline is the instructor.