Spring 2018 CS313 Project

Implementation of a GUI to work with Graphs.

Reminder: This is a pass/fail assignment, you must pass it to pass the course. You do not have to do a perfect job to pass the assignment, but you should submit something that compiles and creates a GUI that performs some of the required tasks. The surest way to fail this assignment is to use code written by somebody else (in or outside of the class) or to share code with another student. I will use an automated system that detects similarity between different pieces of code.

The final deadline for this project is Wednesday 5/16/2018. However there are preliminary milestones that you should try to achieve. You can submit preliminary versions of the project to check that you have achieved them. The deadlines for these milestones are as follows:

- Phase 1: Due 04/23/18 code to show a Gui screen with required buttons on which the help button works and pressing one radio button releases all of the other radio buttons.
- Phase 2: Due 05/02/18 The Gui now stores and displays a graph in response to user mouse clicks on the screen.
- Phase 3: Due 05/09/18 The Gui now has options to enter and store weights for edges of the graph.
- Phase 4: Due 05/16/18 Either the shortest path or minimal spanning tree option is implemented and works. This is the version that will receive a grade.

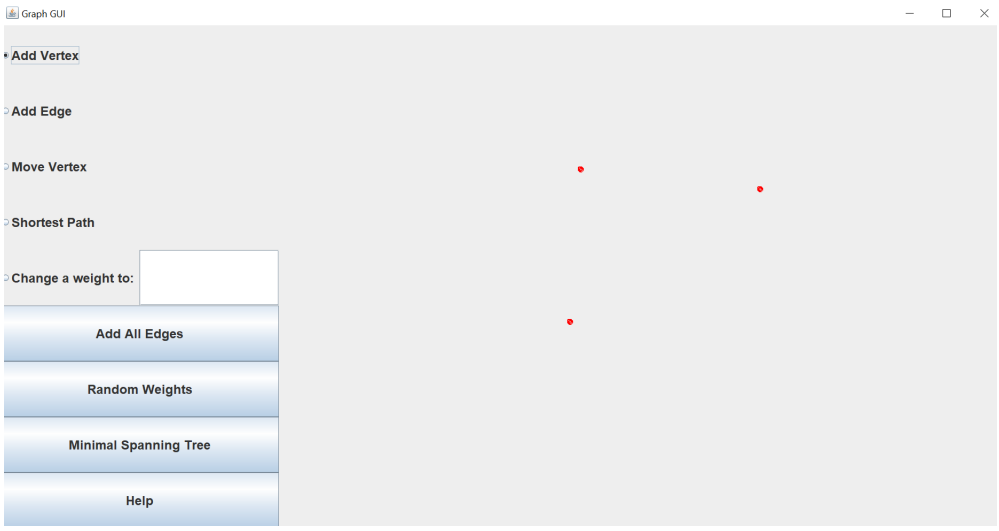
I advise you to try to submit work for the preliminary phases. This is optional, but it is a way for me to give you feedback about whether you are on track with the project.

When you submit any version of your project (preliminary or final) you should submit only one file named GGxxxx.java where xxxx is changed to be your last name. For example, if your name was Waxman you would submit a file called GGWaxman.java.

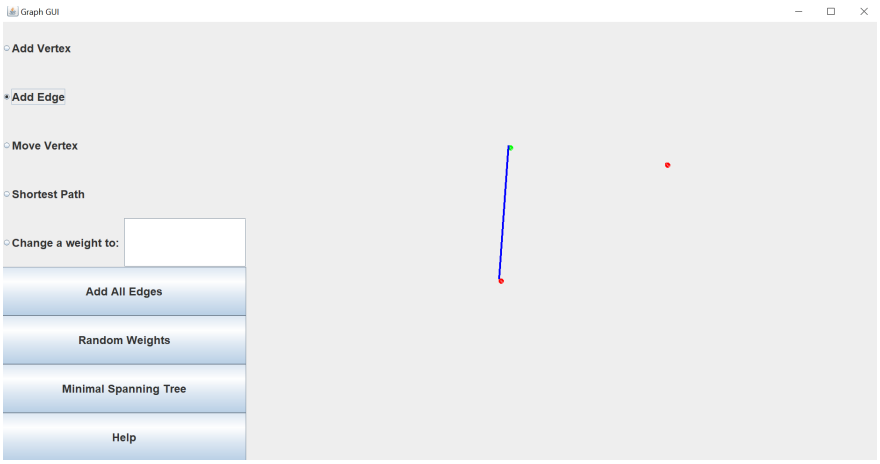
The main class in this file must be a public class GGWaxman but no other classes that you use can be public (because only one public class is allowed in a file). Although this is not an ideal way to organize code it simplifies my task in grading. I will not accept any other type of submission either for the final version or the earlier milestones. Submit the homework to me through blackboard. Late work will not be considered for any phase of the project.

The gui that you make should offer the features shown here. This sreenshot has been made before any graph has been entered. Choosing the *Help* button should pop up another window with instructions about how to run your gui.

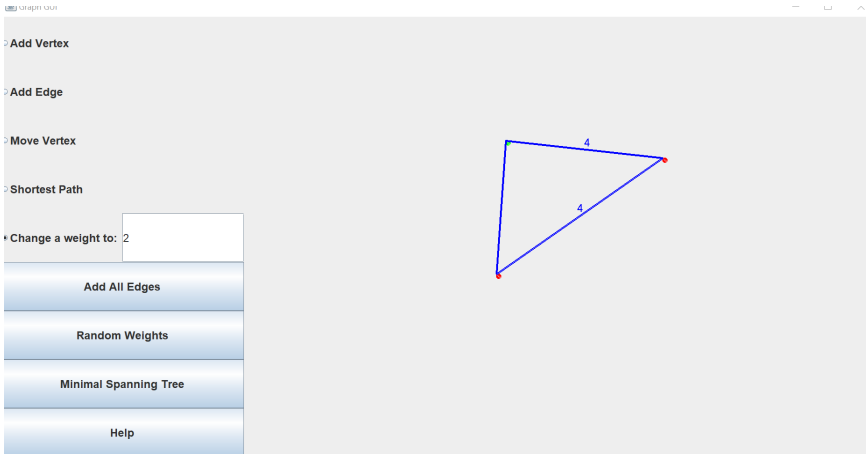
In order to pick vertices, the user selects the radio button marked *Add Vertices* and selects positions of vertices in the right half of the gui by clicking the mouse. Each mouse click generates a vertex of the graph and these vertices are marked in red as they are added.



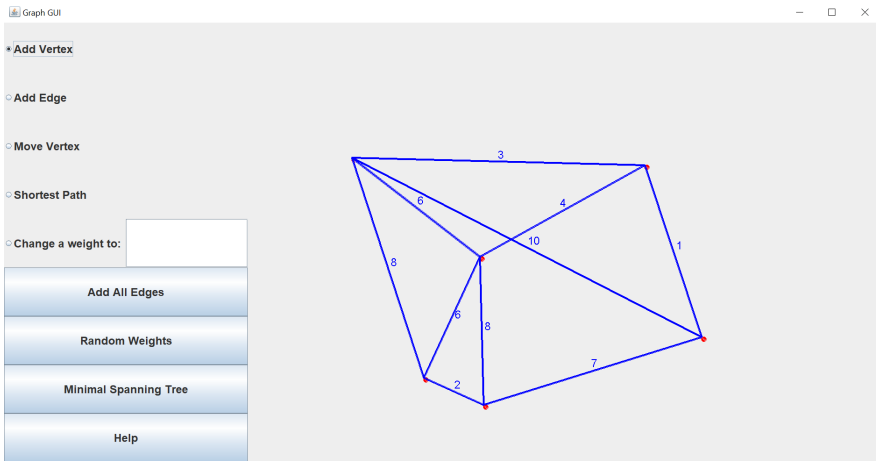
The user selects the radio button marked *Add Edges* to add edges. An Edge is made by clicking on the two vertices that specify its ends. There is no way for the user to guarantee a perfectly placed click on an existing vertex so you must allow for reasonably close clicks as well. After the first end vertex of an edge is selected it is highlighted in green until the second end has been selected too at which time the edge is added to the graph and drawn on the gui in blue.



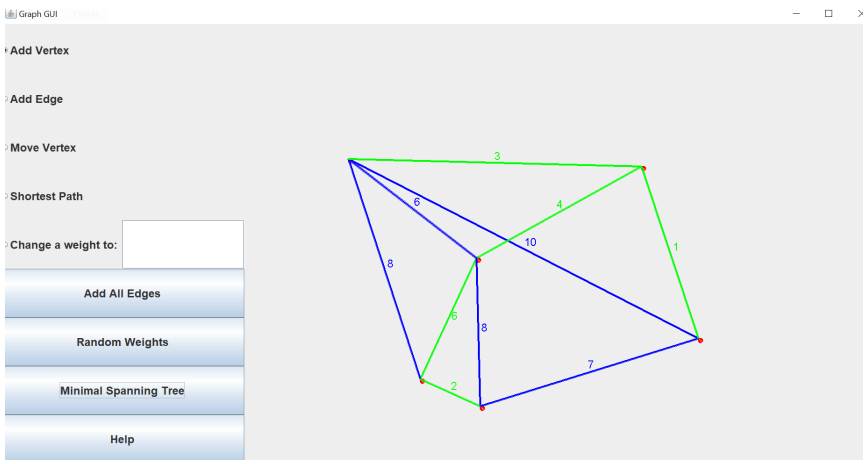
After edges have been entered they are assigned weights. If we imagine edges as roads linking cities, the weights might signify the cost of building a road. Before choosing the option to change a weight (or add a weight) to an edge a value for the weight is entered in the text box. Then edges are selected as usual by clicking on their two vertices and the value is the text box is used as a weight.



The buttons marked *Add All Edges* and *Random Weights* are shortcuts that add in all possible edges between selected vertices and choose random weights for edges in a graph. Once the graph and edge weights have been entered, the running gui might look as follows.



A minimal spanning tree corresponds to a network of roads that link all cities and cost as little as possible to build in the case where the graph represents a network of roads linking cities (as described earlier). Pressing the button *Minimal Spanning Tree* causes the gui to calculate a minimal spanning tree and display it in green.



In a similar way a shortest path between two vertices requires the selection of the vertices and then displays the result as a sequence of edges shown in green.

Your Gui can be written either with swing or with Java FX. Although the first steps just involve Gui code, later steps will require work with a Graph data structure. This is the topic of Chapter 14 of the text which you should read. Section 14.2 explains various plans for storing Graph data. Any of these plans will work well for this project, for example you might choose to use the simplest of the plans, the Adjacency list as described in Section 14.2.2 on page 620. To find shortest paths you should apply Dijkstra's algorithm that is described in Section 14.6.2 on page 653, To find minimal spanning trees you should apply Prim's algorithm as described in Section 14.7.1 on page 664.

You should probably plan to use a number of classes in your project. In addition to your main class GGxxxx which has the job of displaying the gui on screen, you might consider having a class Vertex, a class Edge, a class Graph, a class for the Help screen, listener classes for your Gui and a class that represents the panel of the Gui in which the graph is displayed.

The following code fragment could help you get started with an implementation that uses swing classes.

```
public class GGxxxx extends JFrame {

   GraphPicturePanel picture;  // you would have to make this class
                               // to implement behavior of the picture

   public static void main(String args[]) {
      try {
         UIManager.setLookAndFeel(UIManager
               .getCrossPlatformLookAndFeelClassName());
      } catch (ClassNotFoundException | InstantiationException
            | IllegalAccessException | UnsupportedLookAndFeelException e) {
      }
      new GraphGui();
   }

   public GraphGui() {
      super("Graph GUI");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // code to layout gui components omitted
   }

   public void paint(Graphics g) {   // method to be sure the
                                     // picture gets redrwan as it is modified
   // if you want to use this plan your GraphPicturePanel
   // needs a method with title line:
   //    public void paintComponent(Graphics g) {
      super.paint(g);
      picture.repaint();
      setVisible(true);
```

```
    }

    // other gui code omitted
}
```

You will also need a class that implements an `ActionListener` to respond to user selections of buttons and a class that extends `MouseAdapter` to detect mouse clicks in the picture panel.