Assignment 9
Due: Dec 7, 2023 at 11:59PM

1 Assignment 9 Specification

1.1 Implementation

The skeleton code at `https://classroom.github.com/a/q-HokpwN` is the start of a program to generate minimum spanning trees on an Euclidean graph. The input file is a list of x/y coordinates for verticies in the graph, followed by an epsilon distance. The graph is an *implicit* graph – the edges are not listed in the input file, but exist between any two vertices that are closer than the epsilon value. The 'Euclidean' descriptor means that distance (and an edge's weight) is measured in a direct line between the two coordinates. Every vertex will fall into the unit square, $[0, 1] \times [0, 1]$, so an epsilon of $\sqrt{2}$ or larger means that the graph is fully connected. This gives a mechanism for reducing the number of edges in a dense graph (but leads to an unconnected graph if epsilon is too small).

For this assignment, implement the **FindMST** method (and any helper methods you want) in **A9.java**, using whichever minimal-spanning-tree algorithm you prefer. You should not delete any code from the **Graph**, **Vertex**, or **Edge** classes. You can include additional methods or fields in any of those three classes, depending on what your extra requirements are for your graph. If you create a new class file, remember to add and commit it to your repository.

Depending on which algorithm you pick, you will need either a Union-Find or Min-Heap data structure. You may implement this structure yourself or rely on a Java library class.

Note that Kruskal's algorithm requires the edges to be sorted from smallest to largest. You can sort the graph's array of edges however you want, though I would recommend using the built-in method **Arrays.sort**. The **Edge** class already has the **compareTo** method implemented.

The input argument to **FindMST** is a graph with all of its vertices, but none of its edges. **You are responsible for creating edges for the graph.** The object you return should be a graph that you create, which contains all of the vertices of the input graph and only the edges included in the MST of the input graph. Look through the other classes to see what fields and methods are available for you to use. You can modify the original graph in any way you want.

1.2 Testing

For this assignment, JUnit tests are provided in the **edu.wit.cs.comp2350.tests** package. The tests check if the output has the correct number of edges and total edge sum. For grading this assignment, I will use some different vertex lists in addition to the ones supplied. Assume that all vertices fall within the unit square and epsilon is large enough that the graph is connected. Write some of your own tests to test your code more robustly.

A **GraphMaker** class is included, which will create a graph of Picasso's bull, along with the MST edges that your algorithm returns. If you are interested in visualizing the MST you generate for different graphs, you can change the points file on line 29 of **GraphMaker.java**.

2 Grading

Spanning Tree: 60%

Minimal Spanning Tree: 40%