DUE: 27 Nov 2023

Project #3— Kruskal's Algorithm for MST

Problem Statement: The problem at hand is as follows:

> *Given a weighted undirected graph. We want to find a subtree of this graph which connects all vertices (i.e. it is a spanning tree) and has the least weight (i.e. the sum of weights of all the edges is minimum) of all possible spanning trees. This spanning tree is called a minimum spanning tree.*

This is programming exercise 29.1 in your text. The text introduced Prim's algorithm for finding a minimum spanning tree. Kruskal's algorithm is another well-known algorithm for finding a minimum spanning tree (MST). The algorithm repeatedly finds a minimum weight edge and adds it to the tree if it does not cause a cycle. The process ends when all vertices are in the tree. For this project your task is to design and implement an algorithm for finding an MST using Kruskal's algorithm.

Recall that a minimum spanning tree is a spanning tree of a connected, undirected graph. It connects all the vertices with the lowest total cost (weights for its edges). Here is how Kruskal's algorithm works:

```
sort all edges in graph  G  in order of their increasing weights;
repeat V-1 times    // as MST contains  V-1  edges
{
    select the next edge with minimum weight from graph  G ;

    if (no cycle is formed by adding the edge in MST, i.e., the edge connects two
            different connected components in MST)
        add the edge to MST;
}
```

Described by Joseph Bernard Kruskal, Jr. in 1956, the algorithm works as follows:

> *Kruskal's algorithm initially places all the nodes of the original graph isolated from each other, to form a forest of single node trees, and then gradually merges these trees, combining at each iteration any two of all the trees with some edge of the original graph. Before the execution of the algorithm, all edges are sorted by weight (in non-decreasing order). Then begins the process of unification: pick all edges from the first to the last (in sorted order), and if the ends of the currently picked edge belong to different subtrees, these subtrees are combined, and the edge is added to the answer. After iterating through all the edges, all the vertices will belong to the same sub-tree, and we will get the answer.*

For better performance it is most often implement with disjoint set union (DSU), often also called **Union Find** because of its two main operations. This data structure provides

the following capabilities. We are given several elements, each of which is a separate set. A DSU will have an operation to combine any two sets, and it will be able to tell in which set a specific element is. The classical version also introduces a third operation; it can create a set from a new element.

Thus the basic interface of this data structure consists of only three operations:

- **make_set(v)** - creates a new set consisting of the new element **v**
- **union_sets(a, b)** - merges the two specified sets (the set in which the element **a** is located, and the set in which the element **b** is located)
- **find_set(v)** - returns the representative (also called leader) of the set that contains the element **v**. This representative is an element of its corresponding set. It is selected in each set by the data structure itself (and can change over time, namely after **union_sets** calls). This representative can be used to check if two elements are part of the same set or not. **a** and **b** are exactly in the same set, if **find_set(a) == find_set(b)**. Otherwise they are in different sets.

Program Requirements. Your programs must contain/do/use the following:

1. Interface Graph<V>, classes AbstractGraph<V>, WeightedGraph<V> with inner classes MST and ShortestPathTree and all methods as presented in your text, WeightedEdge that extends AbstractGraphs.Edge class and implements the comparable<WeightedEdge> interface.
2. Create a WeightedEdgesCities.txt file from which to read the number of vertices (first line) and for each line read thereafter, add to an ArrayList of WeightedEdges. The WeightedEdgesCities.txt file should look like this:

```
12
0, 1, 807 | 0, 3, 1331 | 0, 5, 2097
1, 2, 381 | 1, 3, 1267
2, 3, 1015 | 2, 4, 1663 | 2, 10, 1435
3, 4, 599 | 3, 5, 1003
4, 5, 533 | 4, 7, 1260 | 4, 8, 864 | 4, 10, 496
5, 6, 983 | 5, 7, 787
6, 7, 214
7, 8, 888
8, 9, 661 | 8, 10, 781 | 8, 11, 810
9, 11, 1187
10, 11, 239
```

   Note that each line represents a set of edges. For example, the first line should create the edges {0, 1, 807}, {0, 3, 1331}, and {0, 5, 2097} representing the edges Seattle->San Francisco with distance (weight) 807 miles, Seattle->Denver with distance (weight) 1331 miles, and Seattle->Chicago with distance (weight) 2097 miles.

3. Additional WeightedEdges.txt files should be included for selecting New from File menu. Here is one you might want to consider:

```
7
0, 1, 7 | 0, 3, 5
1, 2, 8 | 1, 3, 9 | 1, 4, 7
2, 4, 5
3, 4, 15 | 3, 5, 6
4, 5, 8 | 4, 6, 9
5, 6, 11
```

Note that each line represents a set of edges. For example, the first line should create the edges {0, 1, 807}, {0, 3, 1331}, and {0, 5, 2097} representing the edges Seattle->San Francisco with distance (weight) 807 miles, Seattle->Denver with distance (weight) 1331 miles, and Seattle->Chicago with distance (weight) 2097 miles.

4. A crucial class for this project is the DisjointSetClass, a class used in Kruskal's algorithm for creating a MST. It should contain:
   - 4.1. Private instance variable `totalWeight` (double) and `parent` (a HashMap)
   - 4.2. Default constructor.
   - 4.3. `getTotalWeight`—getter method to return the total weight of the MST.
   - 4.4. `public void makeSet(int numberOfVertices)`—creates numberOfVertices disjoint sets (one for each vertex).
   - 4.5. `private int find(int k)`—recursive method to find the root of the set in which every element k belongs.
   - 4.6. `private void union(int a, int b)`—perform union of two subsets.
   - 4.7. `public static List<WeightedEdge> kruskalAlgorithm(List<WeightedEdge> edges, int numberOfVertices)`—method to construct MST using Kruskal's algorithm.
5. The project should start with a Splash Screen that closes itself after so many seconds and it should contain a meaningful and detailed About Utility type JDialog form activated from the Help menu.
6. There is no pre-designed GUI for this project--you have to design one with appropriate GUI controls (JButtons , JMenuItems, display controls, etc.). You may use Lab 6 as an example for your GUI construction but don't limit yourself to that look. Use good design principles emphasizing functionality.
7. Appropriate classes and data structures.
8. Javadocs, description of the program, and comments everywhere.
9. Menus that synchronize with corresponding buttons and with at least the following menu choices:
   - 9.1. File with Clear, New (to open new file), Print Form, Save (to save the shortest path between the two cities), and Quit menu items.
   - 9.2. Help with About menu item for an About JDialog form.
10. Print the form in its entirety. Use the provided PrintUtilities class.
11. The forms should not be resizable or maximizable (although one should be able to minimize the main form).
12. A free image of graph for icon of the form.

13. ToolTip text for the buttons and menus at least, explaining their function.
14. Provide image of the graph for each text file representation of the graph.

EXTRA CREDIT (Optional) There is numerous ways to obtain extra credit for this project:

1. Modify the application so that representation of the edges is done with via adjacency list of edge object. Define the WeightedEdge class as in listing 29.1, as a derived class from the AbstractGraph.Edge class. Then define the edges as a list of WeightedEdge objects as follows:
   List<List<WeightedEdge>> list = new java.util.ArrayList<>();
2. Construct and display per user request a MST for the cities graph.
3. Define and use a City class with additional fields such as state, population, area (square miles), and so on.
4. Show a graph for the shortest path.
5. Provide an option to the user to find all shortest paths from a chosen city to all others.

You might find these partial comments worthwhile—each counts as a -1 point.

```
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
'Comments by the prof:
'Great effort.  Here are suggestions for improvement:
'1. Use JavaDoc comments throughout the program, not just at the top heading.
'2. An image of a cities map would enhance the look of the form.
'3. Did not implement printing of form--it is required.
'4. Make the Display button default--pressing the Enter should fire it.
'5. Make the display JTextArea is not editable.
'6. Validate input--do not allow illegal input.
'7. It is good style to tab code inside a method.
'8. Use the required classes and the Graph interface.
'9. Declare all constants as final.
'10. Read values from external files using JFileChooser and FileReader class.
'11. Need to use WeightedGraph class.
'12. Name the classes and project appropriately.
'13. Not using the AbstractGraph class.
'14. Follow the Java naming convention for naming variables, methods and classes.
'15. Read the cities data from an external file --include package in path.
'16. Include a separate Validation class for the project.
'17. Enable menu choices that synchronize with buttons' functionality.
'18. Display the path between the two cities and the total distance (weight).
'19. Missing required and meaningful menus.
'20. Disable maximization or resizing of all forms.
'21. Add a meaningful and detailed About JDialog form which describes the project.
'22. Add a Splash screen that starts the project, displayed in center.
'23. Add icon to the main form.
'24. Not finding the shortest path correctly.
'25. Total weight is incorrect.
'26. Provide ToolTip texts for at least each button to explain their functionality.
'27. Give title to all forms.
'28. Save the results using a FileWriter class.
'29. Program crashes on empty.
'30. Incomplete.
'31. Late.
'
'The ones that apply to your project are:
'6, 16, 19, 26.
'
```

```
'26+2=28/30
'~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```