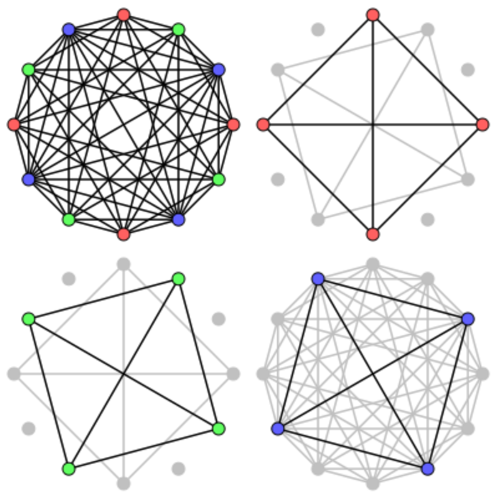
Data Structures
COMP 2100
Fall 2022



Project 3: Graphic Violence

Due by Friday, December 2 at 11:59pm

Introduction

An army of bellowing, enraged traveling salesmen are descending on Otterbein University! Their mission: To find tours that will allow them to visit all of their clients in the shortest amount of time. They don't care if the problem is computationally infeasible to solve! They are willing to wait for a brute force solution. And, if you don't give them a brute force solution, they are prepared to use brute force on you!

Overview

Your job is to write a class that can load an (undirected) graph from a file. Then, your class should support a number of options, all accessed through a text-driven menu. Your class should be called Graphs and stored in Graphs.java. You are permitted to create other helper classes if you think it would be useful. A separate class to hold the representation of a graph is not a bad idea.

Let's list the things you're going to need to be able to do.

1. **Loading the Graph**
   You will need to open the file specified by the user and read in a representation of an undirected graph.

2. **Display a Menu**
   Display a menu the user can make choices from. Each choice will either give information about the currently loaded graph or transform it in some way.

3. **Is Connected**
   You will need to determine whether or not the graph is connected.

4. **Minimum Spanning Tree**
   If the graph is connected, you will need to find a minimum spanning tree for the graph and print an error otherwise.

5. **Shortest Path**
   Prompt the user for a node. Then, print the lengths of the shortest paths from that node to all other nodes and the actual paths as well.

6. **Is Metric**
   You will need to determine whether or not the graph is metric.

7. **Make Metric**
   A graph that is not metric can be made metric by making it a complete graph and adjusting all of its edges so that every direct edge gives the shortest path between the two connected nodes.

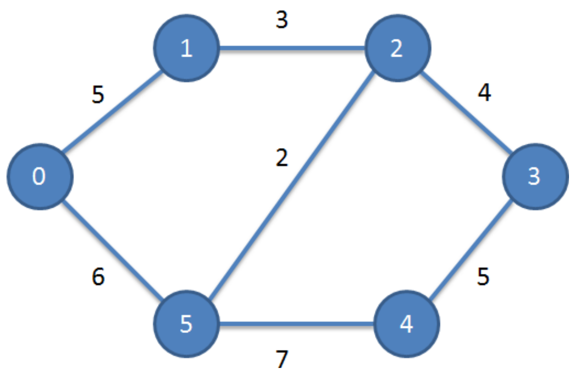
8. **Traveling Salesman Problem**
   Use brute force to try all possible tours starting at node 0. Print the length of the shortest one and the sequence of nodes

9. **Approximate TSP**
   If the graph is metric, use the doubled MST method of finding a 2-approximation to TSP. Otherwise, return an error. Once you have read all the tokens into a queue or similar data structure, it is simple to print out each value with a single space between them.

Loading the Graph

When your program starts, you must prompt the user for a graph file. This graph file will start with a number saying the number of nodes in the graph. Then, each line after that will correspond to a particular node. Each such line will start with a number specifying the number of edges connected to a node. The rest of the line will give a series of pairs of numbers. The first number in the pair gives the number of node that the current node is connected to. The second gives the length of the edge connecting them.

We will use the following graph for many of our examples.



The input file for this graph is as follows.

```
6
2 1 5 5 6
2 0 5 2 3
3 1 3 3 4 5 2
2 2 4 4 5
2 3 5 5 7
3 0 6 2 2 4 7
```

Display a Menu

After the graph has been loaded, display a menu the user can make choices from. Each choice will either give information about the currently loaded graph or transform it in some way.

The choices for the menu are:

1. Is Connected
2. Minimum Spanning Tree
3. Shortest Path
4. Is Metric
5. Make Metric
6. Traveling Salesman Problem
7. Approximate TSP
8. Quit

The user should be able to pick any sequence of choices repeatedly until finally selecting Quit (8).

Sample output for the menu is as follows.

```
1. Is Connected
2. Minimum Spanning Tree
3. Shortest Path
4. Is Metric
5. Make Metric
6. Traveling Salesman Problem
7. Approximate TSP
8. Quit

Make your choice (1 - 8):
```

Choices

Is Connected

If the user selects choice 1, you will need to determine whether or not the graph is connected. If the graph is connected, print `Graph is connected.` Otherwise print `Graph is not connected.`

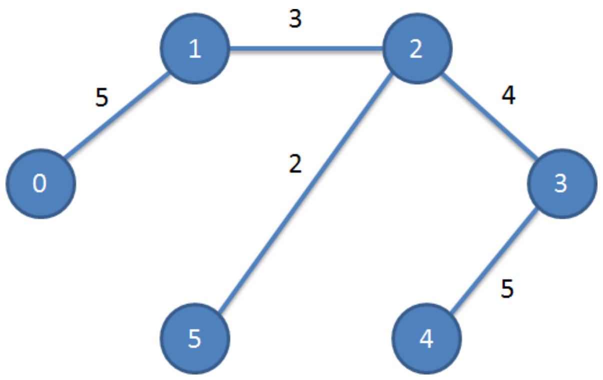
Output for the sample graph is as follows.

```
Graph is connected.
```

Minimum Spanning Tree

If the graph is connected, you will need to find a minimum spanning tree for the graph. Otherwise, print `Error: Graph is not connected.` Print the MST in the same general format used for the graph input file.

The MST for the sample graph is as follows.



Output for the sample graph is as follows.

```
6
1 1 5
2 0 5 2 3
3 1 3 3 4 5 2
2 2 4 4 5
1 3 5
1 2 2
```

Shortest Path

Prompt the user for a node. Then, print the lengths of the shortest paths from that node to all other nodes in parentheses. For the starting node, print `(0)`. For unreachable nodes, print `(Infinity)`. After the length of each shortest path, print a tab and then the nodes visited in the path itself, separated by arrows. Note that you will need to keep a back pointer giving the previous node in the path as well as the best distance found so far in order to produce each shortest path.

If a user entered node 0 for this choice, the output for the sample graph would be as follows. User input is shown in **red**.

```
From which node would you like to find the shortest paths (0 - 5): 0
0: (0)      0
1: (5)      0 -> 1
2: (8)      0 -> 1 -> 2
3: (12)     0 -> 1 -> 2 -> 3
4: (13)     0 -> 5 -> 4
5: (6)      0 -> 5
```

Is Metric

You will need to determine whether or not the graph is metric. There are two conditions to be a metric graph:

1. The graph must be completely connected
2. The every edge in the graph must be obey the triangle inequality

There are three possible outputs. If the graph is not completely connected, print `Graph is not metric: Graph is not completely connected.` If the graph is completely connected but its edges do not obey the triangle inequality, print `Graph is not metric: Edges do not obey the triangle inequality.`

Output for the sample graph would be as follows:

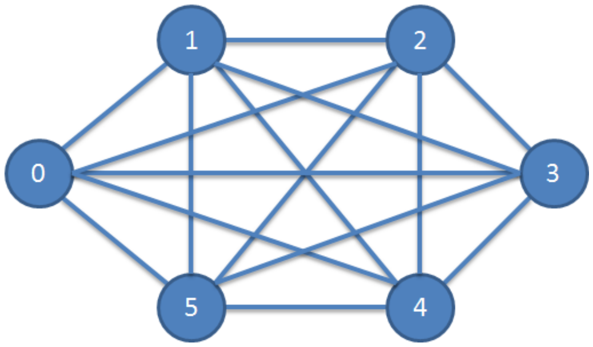
```
Graph is not metric:  Graph is not completely connected.
```

Make Metric

A graph that is not metric can be made metric by making it a complete graph and adjusting all of its edges so that every direct edge gives the shortest path between the two connected nodes.

This choice is the only choice that changes the structure of the graph. If the graph is connected, you must add edges between every node that does not already have an edge. Then, you must set every direct edge to have the cost of the shortest path between those edges. Afterwards, print the graph data as if it were a graph input file. If the graph is not connected, print `Error: Graph is not connected.`

For the sample graph, the resulting graph is below. Edge weights have been omitted for clarity, but they are listed in the sample output.



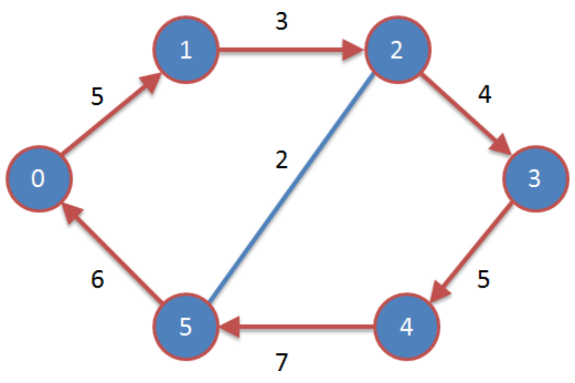
```
6
5 1 5 2 8 3 12 4 13 5 6
5 0 5 2 3 3 7 4 12 5 5
5 0 8 1 3 3 4 4 9 5 2
5 0 12 1 7 2 4 4 5 5 6
5 0 13 1 12 2 9 3 5 5 7
5 0 6 1 5 2 2 3 6 4 7
```

Traveling Salesman Problem

Use brute force to try all possible tours starting at node 0. Print the length of the shortest one and the sequence of nodes it takes.

There are two possible errors. If the graph is not connected, print `Error: Graph is not connected.` Remember that a TSP path can only cross an edge once. If there is no path that returns to the starting node without re-crossing an edge, print `Error: Graph has no tour.`

The sample graph has the following (obvious) TSP tour:



For the sample graph, output is below. The length of the tour is given first, followed by a colon, followed by the sequence of nodes, separated by arrows.

```
30: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 0
```

Warning: If you try this on a large problem instance, you will be waiting for the Universe to grow cold. If you execute it efficiently, you should be able to run TSP on an instance with around 30 edges. Graph structure makes a big difference.

Approximate TSP

If the graph is metric, we can approximate the best TSP tour with the following steps:

1. Find a minimum spanning tree for the graph
2. Do a depth-first search of the minimum spanning tree, starting at the first node
3. The DFS will return an array giving each node a number corresponding to the order in which it was visited. Convert this information to a tour ordering where the first visited node is first, the second visited node is second, the third visited node is third, and so on.
4. Total up the edge weights between each node and the one that follows it in the tour. Then, add in the edge weight from the last node back to the first node to complete the tour.

From the approximation analysis perspective, this algorithm works is because we can imagine that the edges of the MST are doubled and we are simply going backwards and forwards along all of its edges.

Then, because the graph is metric, we can shortcut directly to the next node in the order instead of backtracking along the MST. Because the MST cannot be longer than the best TSP tour, and shortcutting past nodes in a metric graph will not increase the distance, the resulting tour is no more than twice the optimum TSP tour.

Other decent explanations of this algorithm are given here and here.

If the graph is not metric, print `Error: Graph is not metric.`

For our unchanged sample graph, the output would be `Error: Graph is not metric.` However, if we had modified the graph so that it is metric, the result would be the same as the brute force TSP. In this case, the approximation gets the optimal tour.

Note: You can compare your results against the example on this webpage. An applet there will also do this approximation. You can compare your results, but it does not allow you to enter nodes at specific locations. So, you'll have to approximate your graphs in the Euclidean plane to use it. Search around. There are probably other TSP applets out there.

Hints

- Start early. Try to work on the pieces in the order they are given to you. There is a natural dependence in many cases. For example, making the final TSP approximation is dependent on MST and making the graph metric. Making the graph metric is dependent on knowing whether or not the graph is connected.
- You are allowed to implement the graph with an adjacency matrix or with adjacency lists. It's up to you.
- Think before you code. 1 minute of design == 10 minutes of coding == 100 minutes of debugging.
- You may use the Java Collection Framework if you need containers. Very little there is likely to help you much. You are (naturally) forbidden from using graph libraries such as JGraph and JGraphT.

Submit

Zip up all the .java files you use to solve this problem from your src directory and email this zip file as an attachment to dstucki@otterbein.edu. All work must be submitted by Friday, December 2, 2020 at 11:59 p.m.

Only the team leader should turn in the final program. I must be able to compile your program with the command javac Graphs.java and run with the command java Graphs.

All work must be done within assigned teams. You may discuss general concepts with your classmates, but it is never acceptable for you to look at another team's code. Please refer to the course policies if you have any questions about academic integrity. If you have trouble with the assignment, I am always available for assistance.

Grading

Your grade will be determined by the following categories:

Category	Weight
Loading the Graph	5%
Display a Menu	5%
Is Connected	10%
Minimum Spanning Tree	10%
Shortest Path	10%
Is Metric	5%
Make Metric	5%
Traveling Salesman Problem	20%
Approximate TSP	20%
Style and Comments	10%

Note: Submissions which do not compile will automatically score zero points.

Under no circumstances should any member of one group look at the code written by another group. Tools will be used to detect code similarity automatically.