CSE 373: Homework 5

Dijkstra's Algorithm

Code due: May 17th, 11:59 PM to Canvas

Writeup due: May 19th: 11:59 PM

Introduction

For this assignment, you will use your graph representation from HW4 to implement Dijkstra's algorithm for finding shortest paths. You do not have to implement Dijkstra'a algorithm exactly as specificied in class, with the exact runtime specified in class, see the description below for more information.

As with HW4, you may use anything in the Java standard collections (or anything else in the standard library) for any part of this assignment. Take a look at the Java API as you are thinking about your solutions. At the very least, look at the Collection and List interfaces to see what operations are allowable on them and what classes implement those interfaces.

1 Provided Files

Copy your old files from HW4 to the new project

The following **new** files are provided:

- `Path.java`: Class with two fields for returning the result of a shortest-path computation. *Do not modify*
- `FindPaths.java`: A client of the graph interface: Needs small additions
- `Edge.java`: Edge Class: You may add to this if you want
- `vertex.txt` and `edge.txt` an example graph in the correct input format

You will also need to add a new method to your MyGraph.java implementation from HW4. You will need to add and implement the additional method `public Path shortestPath(Vertex a, Vertex b)`. You can copy and paste the method header and comments from the skeleton code HW5_MyGraph.java. The name of this file is HW5_MyGraph.java, only so you don't accidentally overwrite your MyGraph.java from HW4 when downloading this file. You only need this file to copy and paste the method header for shortestPath. You should still be implementing a graph called MyGraph and turn in a file at the end called MyGraph.java You do not need to re-implement all of the other methods, you can use your MyGraph implementation from HW4. However, you may modify any of the code you submitted for HW4 in your HW5 MyGraph file.

2 Functionality

In this the assignment, you will use your graph from HW4 to compute shortest paths. The MyGraph class has a method shortestPath you should implement to return the lowest-cost path from its first argument to its second argument.

Return a Path object as follows:

- If there is no path, return null.
- If the start and end vertex are equal, return a path containing one vertex and a cost of 0.
- Otherwise, the path will contain at least two vertices – the start and end vertices and any other vertices along the lowest-cost path. The vertices should be in the order they appear on the path.

Because you know the graph contains no negative-weight edges, Dijkstra's algorithm is what you should implement. **Additional implementation notes:**:

- One convenient way to represent infinity is with Integer.MAX_VALUE.
- You definitely need to be careful to use equals instead of == to compare Vertex objects. The way the FindPaths class works (see below) is to create multiple Vertex objects for the same graph vertex as it reads input files. You may want to refer to your old notes on the equals method from CSE143. Remember that equals lets us compare values (e.g. do two Vertex objects have the same label) as opposed to just checking if two things refer to the exact same object.

The program in FindPaths.java is mostly provided to you. When the program begins execution, it reads two data files and creates a representation of the graph. It then prints out the graph's vertices and edges, which can be helpful for debugging to help ensure that the graph has been read and stored properly. Once the graph has been built, the program loops repeatedly and allows the user to ask shortest-path questions by entering two vertex names. The part you need to add is to take these vertex names, call shortestPath, and print out the result. Your output should be as follows:

- If the start and end vertices are X and Y, first print a line
  `Shortest path from X to Y:`
- If there is no path from the start to end vertex, print exactly one more line
  `does not exist`
- Else print exactly two more lines.
  - On the first additional line, print the path with vertices separated by spaces. For example, you might print
    `X Foo Bar Baz Y`
  - On the second additional line, print the cost of the path (i.e., just a single number).

The FindPaths code expects two input files in a particular format. The names of the files are passed as command-line arguments. The provided files vertex.txt and edge.txt have the right format to serve as one (small) example data set where the vertices are 3-letter airport codes. Here is the file format:

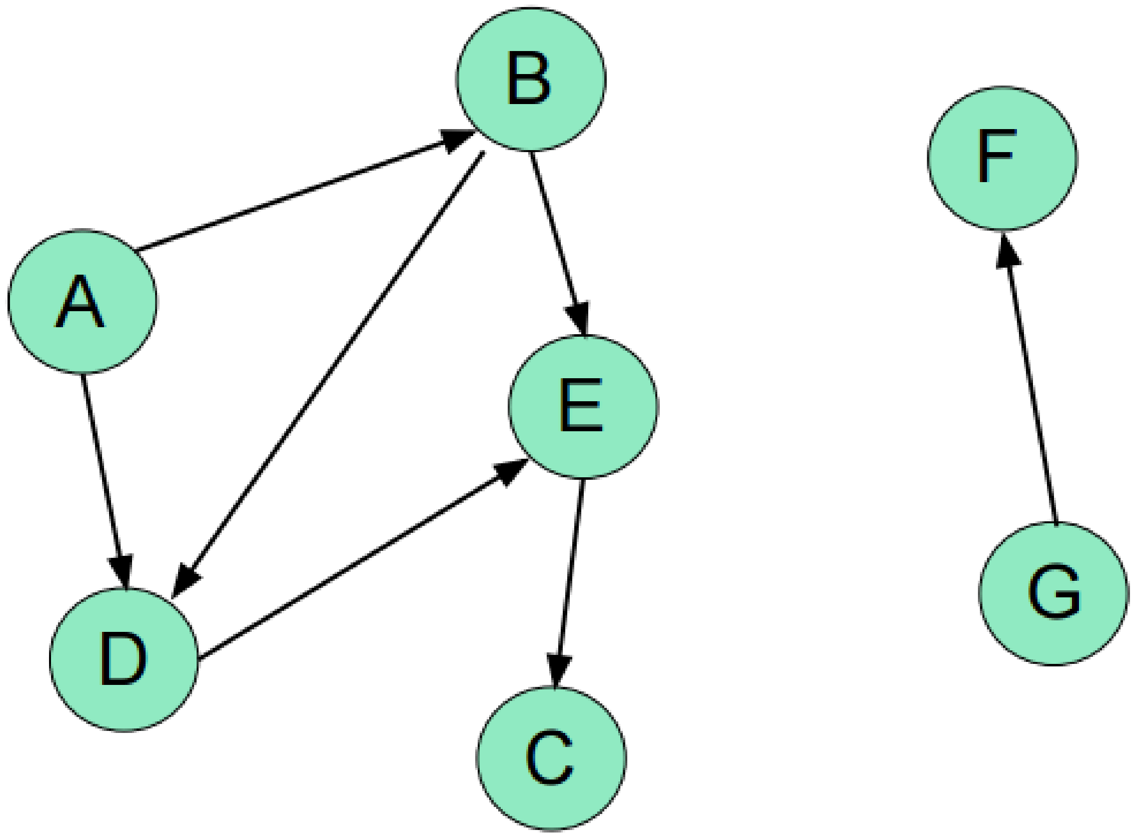
- The file of vertices (the first argument to the program) has one line per vertex and each line contains a string with the name of a vertex
- The file of edges (the second argument to the program) has three lines per directed edge (so lines 1-3 describe the first edge, lines 4-6 describe the second edge, etc.) The first line gives the source vertex. The second line gives the destination vertex. The third line is a string of digits that give the weight of the edge (this line should be converted to a number to be stored in the graph).

Note data files represent directed graphs, so if there is an edge from A to B there may or may not be an edge from B to A. Moreover, if there is an edge from A to B and an edge from B to A, the edges may or may not have the same weight.
Feel free to add additional public functionality that you think would be useful for a client. There might be some redundant code between your public methods; it might help to make some private helper methods to clean up your code.

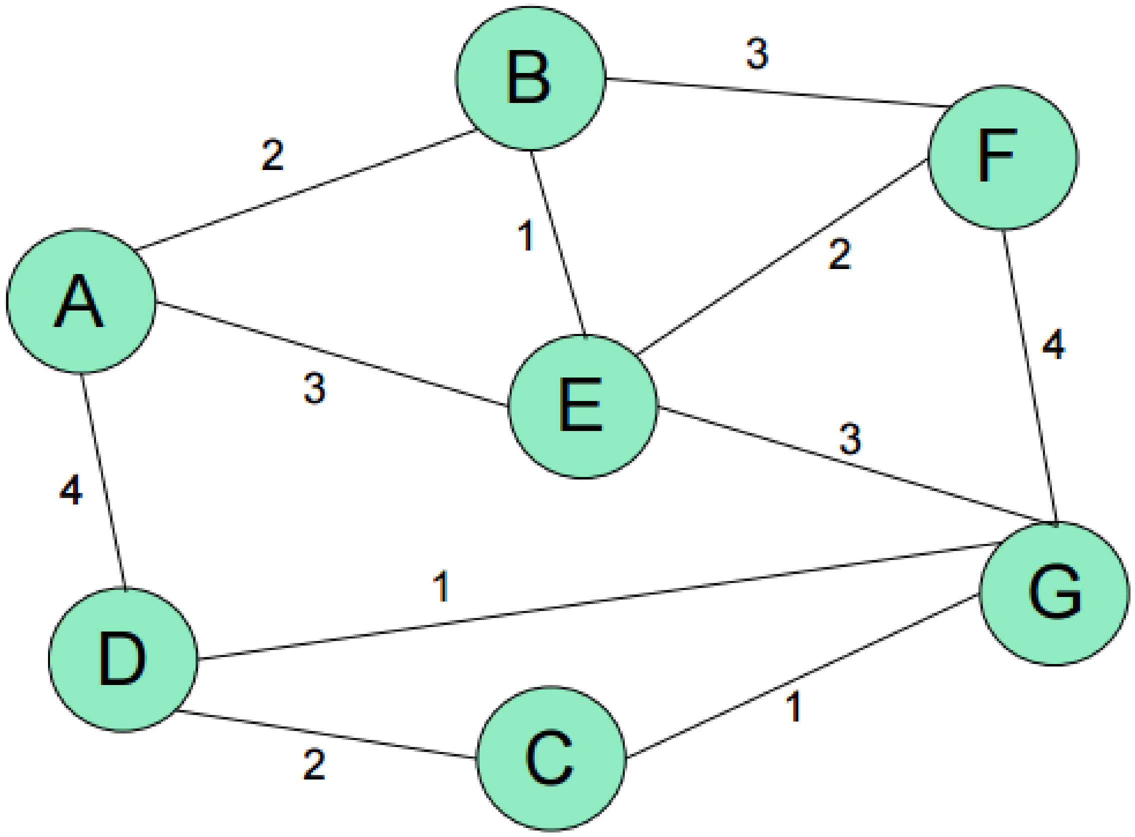
3 Writeup

Answer the following questions thoroughly and completely.

1. **Topological Sort:** Determine a topological ordering for the following directed acyclic graph (DAG). Show your work by comupting the in-degree value of each vertex and keeping track of the vertices with in-degree of 0 in a queue.



2. **Minimum Spanning Trees**: Use the following graph for both of the following minimum spanning tree algorithms.



   (a) Build a minimum spanning tree for the graph using Kruskals algorithm. Number each edge according to when it is entered into the minimum spanning tree. The first edge will get number 1, etc. Show the ordering of the edges you consider by showing the state of the pending set of edges to consider.

   (b) Now build a minimum spanning tree for the graph using Prims algorithm, starting with vertex A. Again, number each edge according to when it is entered into the set of edges. The first edge will get number 1, etc. Show the partial state in a table keeping track of known, cost, and path similar to the lecture slides.

3. **Dijkstra's and Negative Edges:**

   (a) If there is more than one minimum cost path from v to w, will Dijkstras algorithm always find the path with the fewest edges? If not, explain in a few sentences how to modify Dijkstras algorithm so that if there is more than one minimum path from v to w, a path with the fewest edges is chosen. Assume no negative weight edges or negative weight cycles.

(b) Give an example where Dijkstras algorithm gives the wrong answer in the presence of a negative cost edge but no negative-cost cycles. Explain briefly why Dijkstras algorithm fails on your example. The example need not be complex; it is possible to demonstrate the point using as few as 3 vertices.

(c) Suppose you are given a graph that has negative-cost edges but no negative-cost cycles. Consider the following strategy to find shortest paths in this graph: Uniformly add a constant k to the cost of every edge, so that all costs become non-negative, then run Dijkstras algorithm and return that result with the edge costs reverted back to their original values (i.e., with k subtracted).

- Give an example where this technique fails (Dijkstras would not find what is actually the shortest path) and explain why it fails.
- Give a general explanation as to why this technique does not work. Think about your example and why the original least cost path is no longer the least cost path after adding k.

4. Describe how you tested your shortestPath method. Explain any difficulties in implementation you may have experienced and how testing helped find the problem, if applicable

Deliverables

For this assignment, there will be two submissions on Canvas.

- Part 1 consists of a zip of your FindPaths, Edge, Vertex and MyGraph classes.
- Part 2 is the pdf of the writeup. Make sure all questions are answered.