Western University

CS 2210B - Data Structures and Algorithms
Fall 2021-22, Instructor: Dr. Saad Bin Ahmed

Assignment 5 (Programming)

Total Marks: 100
Due Date: April 09, 2022

1 Graphs and Shortest Paths

For this assignment, you will develop a graph representation and use it to implement Dijkstra's algorithm for finding shortest paths. Unlike previous assignments, you will use some classes in the Java standard libraries, gaining valuable experience reading documentation and understanding provided APIs.

Different for this assignment: 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[1] as you are thinking about your solutions. At the very least, look at the Collection[2] and List[3] interfaces to see what operations are allowable on them and what classes implement those interfaces.

2 Working with a Partner

You are strongly encouraged, but not required, to work with a partner. You can use the Discussion Board to find a partner, and you can just write the name of both partners at the top of the files when you turn it in. No more than two students may form a team. You may divide the work however you wish, but place the names of both partners at the top of all files. Team members will receive the same project grade unless there is an extreme circumstance. Beyond working with a partner, all the usual collaboration policies apply.

If you work on a team, you should:

1. Turn in only ***ONE*** submission including per team.
2. Work together and make sure you both understand the answers to all questions.
3. Understand how your partner's code is structured and how it works.
4. Test all of your code together to be sure it properly integrates.

3 Provided Files

Download these files into a new directory:

[1] https://docs.oracle.com/javase/7/docs/api/
[2] https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html
[3] https://docs.oracle.com/javase/7/docs/api/java/util/List.html

- Graph.java – Graph interface. Do not modify.
- Vertex.java – Vertex class
- Edge.java – Edge class
- MyGraph.java – Implementation of the Graph interface: you will need to fill in code here
- 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
- vertex.txt and edge.txt – an example graph in the correct input format

4 (30 Marks) Programming - Part 1

In this part of the assignment, you will implement a graph representation that you will use in Part 2. Add code to the provided-but-incomplete MyGraph class to implement the Graph interface. Do not change the arguments to the constructor of MyGraph and do not add other constructors. Otherwise, you are free to add things to the Vertex, Edge, and MyGraph classes, but please do not remove code already there and do not modify Graph.java. You may also create other classes if you find it helpful.

As always, your code should be correct (implement a graph) and efficient (in particular, good asymptotic complexity for the requested operations), so choose a good graph representation for computing shortest paths in Part 2.

We will also grade your graph representation on how well it protects its abstraction from bad clients. In particular this means:

- The constructor should check that the arguments make sense and throw an appropriate exception otherwise. You can define your own exceptions by making classes like the exception classes we have provided you in previous homeworks. Here is what we mean by make sense:
  - The edges should involve only vertices with labels that are in the vertices of the graph. That is, there should be no edge from or to a vertex labeled A if there is no vertex with label A.
  - Edge weights should not be negative.
  - Do not throw an exception if the collection of vertices has repeats in it: If two vertices in the collection have the same label, just ignore the second one encountered as redundant information.
  - Do throw an exception if the collection of edges has the same directed edge more than once with a different weight. Remember in a directed graph an edge from A to B is not the same as an edge from B to A. Do not throw an exception if an edge appears redundantly with the same weight; just ignore the redundant edge information.

Other useful information:

- The Vertex and Edge classes have already defined an appropriate equals method (and, therefore, as we discussed in class, they also define hashCode appropriately). If you need to decide if two Vertex objects are "the same", you probably want to use the equals method and not ==.

- As you are debugging your program, you may find it useful to print out your data structures. There are toString methods for Edge and Vertex. Remember that things like ArrayLists and Sets can also be printed.

5 (30 Marks) Programming - Part 2

In this part of the assignment, you will use your graph from Part 1 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.
- Using a priority queue is above-and-beyond. You are not required to use a priority queue for this assignment. Feel free to use any structure you would like to keep track of distances and then search it to find the one with the smallest distance that is also unknown.
- 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.

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. (Do not print a period, that is just ending the sentence.) 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 formate 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.

6 (20 Marks) Testing

Unlike Assignment 4, here you are supposed to write test cases for your programs, be sure to test your solutions thoroughly and to turn in your testing code. Part of the grading will involve thorough testing including any difficult cases. Name your test file as TestGraph.java

7 What to Turn In

If you work with a partner, only one partner should submit the files. Be sure to list both partners' names in your files.

Turn-in: You should turn in the following files electronically, named as follows:

- Vertex.java
- Edge.java
- MyGraph.java
- FindPaths.java
- Any additional Java files needed, if any.
- TestGraph.java

Do not turn in Graph.java, Path.java, vertex.txt and edge.txt. You must not change the Graph.java and Path.java. Your implementations must work with the code as provided to you.

8 (20 Marks) Coding Style

Your mark will be based partly on your coding style.

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and white space should be used to improve readability.
- No variable declarations should appear outside methods ("instance variables") unless they contain data which is to be maintained in the object from call to call.

9 Important Note

- To be eligible for full marks, your programming assignment must run on the departmental computing equipment. You may develop assignments on your home computer, but you must allow for the amount of time it will take to get the final programs working on Computer Science's machines. All programming assignments must be submitted through OWL.

- The late penalty for programming assignment is [2.5i] (2.5 to the i-th power, rounded to the nearest integer), where $i > 0$ is the number of days you are late. So if you hand in your assignment 1 day late, you will be penalized 3%, a delay of 2 days will decrease your grade by 6%, 3 days is penalized 16% and 4 days takes 39% off your grade. You cannot be more than 4 days late.