Project for Week 1: Implementing basic graphs for real transportation data

Before you begin: Getting help

Before you begin, don't forget that you should **use the discussion forums to get help anytime you are stuck on this assignment**. Also, please visit the forums to help your fellow learners by contributing answers to their questions. We're all in this together!

If at any point this week you feel like you are stuck or need some more hints, check out the support videos for the week. They not only provide you with some hints for completing the methods below, but also discuss a cool trick for computing the 2-hop neighbors of a vertex using matrix multiplication!

We also have included an FAQ for this assignment as a reading immediately following this assignment, so check out this document for more help if you get stuck.

Assignment Overview

In this project, you will implement the Graph ADT and compare graphs that come from real transportation data mapping intersections, roads, and flights. Your goals in this assignment are to see how real life data can be modeled by graphs, and how implementation choices affect the performance of the algorithms. You'll also explore how different data properties impact the structure of the graph, both by computing the degree sequence of the graph and by visualizing the graph structure using a tool we provide.

Getting Set Up

Before you begin this assignment, make sure you have the starter code and that you check Part 2 in the setup guide to make sure the starter code has not changed since you downloaded it. If there have been any changes, follow the instructions in the setup guide for updating your code base before you begin.

Also make sure you have read through the starter code and front end orientation guide so you are familiar with what is included with the starter code.

Open the starter code for this week by expanding the basicgraph folder to see the package basicgraph. You will add code to the files Graph.java, GraphAdjMatrix.java and GraphAdjList.java .

Before you begin coding, make sure you know the definitions of a graph and the two implementation strategies for nodes and edges discussed in the core videos: adjacency matrices and adjacency lists.

Assignment and Submission Details

Part 1: Implement the degreeSequence method in the file Graph.java

1. Examine the provided code in Graph.java, GraphAdjMatrix.java and GraphAdjList.java. Make sure you understand how GraphAdjMatrix and GraphAdjList provide two concrete implementations of the abstract notion of a graph, defined in Graph.java.

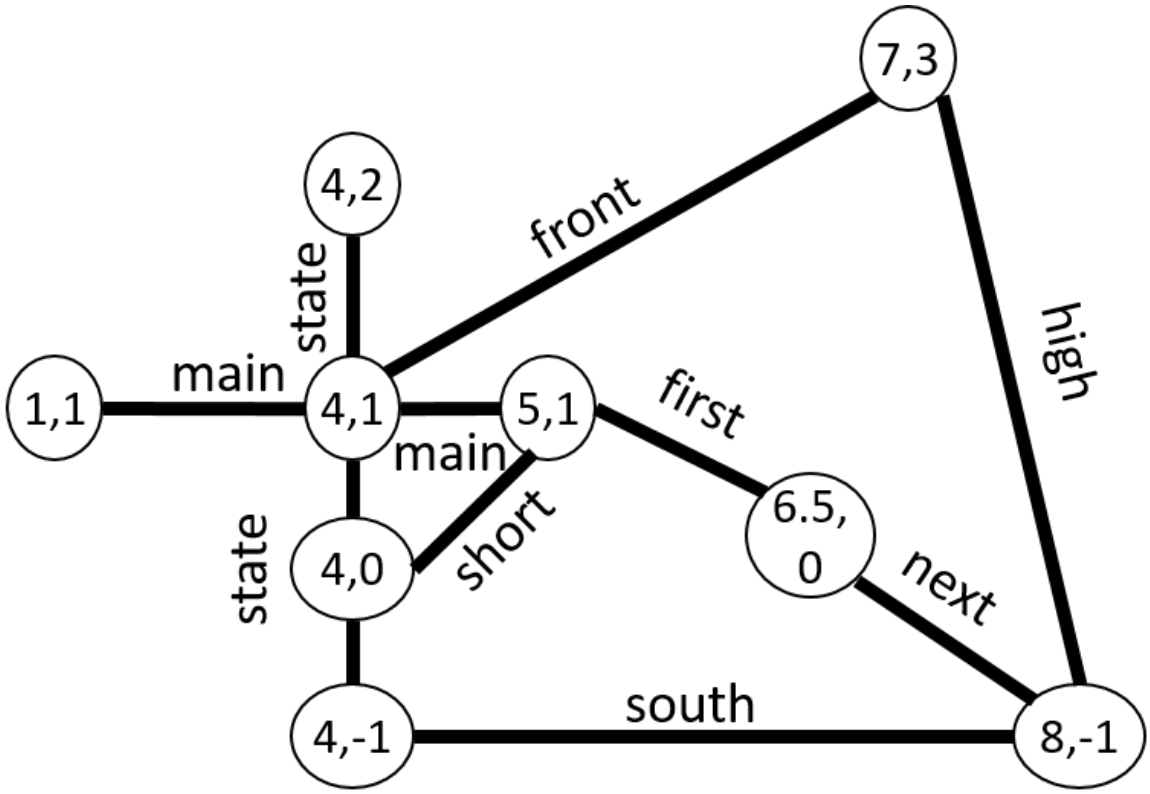
2. Implement the degreeSequence method in the Graph class. Use the comments in the code and the description in the videos to understand what this method should do.

You will find the following two provided methods very helpful for this part:

- getNeighbors(int v) -- returns a list containing the integer indices of vertices appearing as the end point of edges starting at v. Note: if there are multiple edges between v and one of its neighbors, then that neighbor appears multiple times in this list.
- getInNeighbors(int v) -- similar to above except returning list of indices of vertices which are the start points of edges that end at v.

Note that these methods are implemented in Graph's subclasses, and can be called from the Graph class.

3. Test your implementation on fake data. Our main method in Graph.java provides a skeleton for testing the methods you wrote in this section. Use and add to this method to do thorough testing of your methods. The starter code and front end orientation guide gives you an overview of the data provided for testing. Start with the file data/testdata/simpletest.map which contains simulated road data, representing the following (fake) street map:



All edges in the above map are considered 2-way. That is, for each line in the map, there are actually two edges in the graph, one in each direction.

You can use this very simple map for testing, and you can also look at it to see how the map files are set up. You can make a copy of this file and modify it to modify the map you are testing with.

4. (Optional) Make predictions about the degree-sequence of real-world data. Before you run your code on the real-world road intersections and flight graph data we provide, make some predictions:

1. What will be the maximum degree of the road intersections graph?
2. What will be the maximum degree of the nonstop flights graph?
3. Which of these graphs will be more regular?

5. (Optional, but very interesting!) Confirm or deny your predictions by running your code on the real-world files described below. To see a visualization of the graphs

- data/maps/ucsd.map: A small region of the streets near the UCSD campus. This is real-world data

There are also several other real-world road data files in the data/maps folder. You probably want to stick with the ones that are labeled small until you're reasonably sure things are working correctly.

- data/airports/routesUA.dat: A list of the non-stop routes that United Airlines files.
- Custom data of your choosing (optional): If you like, you can generate custom raw-map files for any part of the world that you are interested in. Follow the instructions at the end of the starter code and front end orientation guide. Make sure to generate the.intersections file also. You'll use it in Part 3 below.

6. (Optional, but fun) Skip ahead to the optional Part 3 of this assignment to **visualize this real-world data as a graph**. See more information in Part 3 below.

Part 2: Implement the getDistance2 method in both GraphAdjList and GraphAdjMatrix implementations

Follow the method header comments to implement the method getDistance2 in *both* the GraphAdjList and GraphAdjMatrix classes.

Also, make sure you are consistent with our test cases/examples in the main method. In particular, if there are multiple two-hop paths to get to a vertex, then that vertex should be listed multiple times.

Hints:

- For the list representation you'll use a hard-coded search to implement `getDistance2(int v)`. In other words, first find the neighbors of v, and then expand them out for the two-hop cities.
- For the matrix representation, `getDistance2(int v)` can be implemented with matrix multiplication (square the matrix then read the non-zero entries from the appropriate row). The support video guides you through some more of the details. However, implementing this method using general matrix multiplication is not required as long as your method works correctly.

What and How to Submit (Parts 1 and 2)

1.Create a zip file containing only the files Graph.java, GraphAdjList.java and GraphAdjMatrix.java. Upload this zip file to **BOTH** parts 1 and 2. When you submit, we'll run some tests to make sure your code is correct - if it's not, we'll let you know.

As with all Coursera custom graders, it takes a few minutes. While the grader is running, you might see a 0 for your score. Do not be alarmed. This just means the grader is still running. When it completes the page will refresh.

If you found you had errors, and you can't figure them out, we've provided the Java code we use for testing your files. DegreeGrader in the basicgraph package is our part 1 grader, and the graph files it uses can be found in data/graders/mod1. GraphGrader in the basicgraph package is our part 2 grader, and the graph files it uses can be found in data/graders/mod1.

Part 3 (Optional, nothing to submit): Visualizing the road data

You can confirm your work on this assignment with the visualization tool we're providing. Note that this file works only for the files in the data/intersections folder, or any custom .intersections files you have created.

To run the tool, download and open the HTML file linked from the online version of these instructions.

You can load data into this graph viewer using the following process:

1. Open the file containing the intersection data you want to visualize. NOTE: This must be a .intersections file and not the raw .map files. Copy everything in the file and paste it into the window in the graph viewer window.
2. Click "Import" to load the data.

Play around with the graph visualization tool for different data. What would the graph structure of the intersections of a busy downtown city look like? How about the intersections around a major highway?