**CS 51 Final Project Specification**

Robert Chen, Alex Nie, Annie Rak, Evan Yao

Annotations in red.

We will be exploring expected congestion in subway networks using Python.

**Included Files:**

* Graph.py
* Shortest\_Paths.py
* Data\_Structures.py

**Boston MBTA data** <http://www.mbta.com/uploadedfiles/documents/2014%20BLUEBOOK%2014th%20Edition.pdf> We did indeed use this website to obtain the Boston data and had few problems; it supplied most of the info we needed. We printed out the map and put a piece of graph paper on top of it in order to estimate the coordinates of each station. We also looked into quanturb.com to find data about other cities like Paris.

**Main Features:**

* Given a starting and ending station, we will inform the user how congested each possible path they can take will be.
* Display graph of congestion, which takes into account how many shortest paths will require a particular segment.

We did both of these and incorporated all options for our tool into a user interface.

**Object Oriented Design Structure:**

* Graph Representation:
  + *Class*: Graph – General transportation network graph
    - *Abstract Class*: ShortestPathAlg – adds shortest path methods features to superclass Graph.
      * *Class***:** Dijkstra’s
      * *Class***:** A Star

We maintained more or less the same structure, but just incorporated methods calling Dijkstra’s and A-Star directly into Graph, rather than having an abstract class.

* Data Structure for Algorithms
  + *Abstract Class***:** Dijkstra’s Data Structure
    - *Class***:** Priority Queue
    - *Class***:** Binary Heap
    - *Class***:** Fibonacci Heap

Rather than doing a Fibonacci heap, we decided to implement a D-ary heap data structure as well as focus more on doing the A-Star algorithm and running rigorous tests for both our algorithms.

**Overview of Graph Implementation**

* Fields include:
  + Adjacency List
  + Adjacency Matrix
  + Station ID to Station Name dictionary
  + Station Name to Station ID dictionary
  + Number of Stations

We used all these instance variables and more as necessary.

* Constructor:
  + Reads a graph from a txt file of the following format:

SAMPLE INPUT

3 ← Number of Stations

Harvard 40 ← Station Name, # of people entering per minute

Central 20

Kendall 30

Harvard Central 2 ← Each Edge, Time needed

Central Harvard 2

Central Kendall 3

Kendall Central 3

We read from a CSV file instead and change formatting since it was easier to keep track of our data when there were explicit cells in Microsoft Excel. But more or less the same thing.

* Methods:
  + See actual code for documentation

Task Division:

* **Person 1**: Implementation of Graph Data Structure
  + Define methods in *Graph*
  + Look into displaying graph in a more visually appealing way
* **Person 2**: A Star Algorithm
  + Research A Star Algorithm and implement it
  + Define *ShortestPathsAStar*
* **Person 3**: Dijkstra’s Algorithm
  + Implement at least 2 underlying Dijkstra’s data structures such as *Priority* and *Heap*
  + Define methods in *ShortestPathsDijkstra*
* **Person 4**: Data Collection, Miscellaneous
  + Collect Data and format into necessary input structure
  + Test code that others have written
  + Help out with one of the above components

This was indeed the way we originally split the work up for the first few days and then we communicated about how to best adjust how to split up work afterward.

**Timeline**

* Week 1 (by 4/24 at 11:59 PM)
  + Collect data on Boston’s MBTA and convert into input format. Start finding data on other transportation networks too
  + Finish implementation of *Graph* with basic features not involving shortest paths
  + Finish researching Dijkstra’s and start implementing some versions of its underlying data structure
  + **Goal:** Have a working version reading input, creating graph and finding shortest paths between any two stations.
* Week 2 (by 5/1 at 5:00 PM)
  + Create more complicated Dijkstra’s Algorithm Data Structures.
  + Research A Star and implement it
  + Research more visually appealing ways of representing the heat map as well as the graph itself
  + Create presentation video
  + **Goal:** Improve on basic features and hopefully make everything more visually appealing!

We followed this pretty closely. We also added additional features such as “second iteration” method to make edge weights a function of geographical distance + congestion level, getting data for Paris, being able to see effects of adding/deleting edges, and making a user-friendly interface.