Assignment 4

Data Structure

Classes used:

* AdjacencyMapGraph<V, E>
  + Vertex<V>
  + Edge<E>
* HashSet
* LinkedQueue<E>
* LinkedStack<E>
* ProbeHashMap<>

Attributes

* vertexList, stores all of the vertices in order from 0 to 375, making it very easy to access a specific station if the station number is known.
* parisMetro stores the created AdjacencyMapGraph of the Paris metro system read from metro.txt.
* takes a filename as parameter to read and create a AdjacencyMapGraph out of it.

Methods

* Graph<Integer, Integer> getGraph(): returns the attribute parisMetro
* Vertex[] getVertexList(): returns the list of vertices
* Void readMetro(String filename): uses a BufferREader to read the inputted file. First, it goes through each station and creates a Vertex for each station number, then storing the Vertex in the vertexList. Next after the ‘$’ in the metro.txt file inserts the first two vertices on the line into the AdjacencyMapGraph from the newly created vertexList. Then the reader moves to the edge weighting that then connects the two vertices. If the edge weighting is -1 we assumed a constant walking time of 90 seconds.
* LinkedQueue<Vertex<Integer>> sameLine(Vertex<Integer> v): returns a LinkedQueue that consists of all station numbers on the line of the inputted vertex. Uses Edge[] originalEdges to get the two, or one if the inputted vertex is the end of a line, outgoing edges of the inputted vertex that are NOT weighted as 90. This means that we will only be accessing vertices on the line, not vertices that switch lines. Method uses a HashSet to store the visited vertices. This helps with traversing because we can make sure we do not go to a vertex we have already visited by using !visited.contains(parisMetro.opposite(nextStation, e)), where visited is the HashSet.
* LinkedStack<Integer> shortestTimeToDestination(Vertex<Integer> src, Vertex<Integer> dest): We copied the method shortestPathLengths() from the GraphAlgorithms class in the Net folder from lab 9. Modifications were made to stop the method once the destination vertex is reached. Using a ProbeHashMap, previousVisit, will store the current vertex as the key and the previous vertex as the value. Once the destination vertex is reached the method runs a loop that iterates through the previous vertices from the destination and stops once it reaches the source. During this loop the previous vertex is pushed into the stack making it so that it pops out in order from source station to destination station. Also pushed onto the top of the stack is the shortest time. This makes it easily available in the main method when outputting the results of the test.
* Void closeLine(Vertex<Integer> v): this method runs v through sameLine() and stores the LinkedQueue in line. Then loops through the vertices and removes them from the graph.
* Void printLine(Vertex<Integer> v): prints the line that v is on in a legible format.
* Void printStack(LinkedStack<Integer> stack): prints parameter stack in a legible format.

Output Examples:

