Josphat Magutt

07/08/2012

**Train Routes Problem Readme**

***Design***

* The Train Routes application solves two major problems: finding the distance of a given route and finding the routes between two cities that satisfy certain conditions. Naturally, these problems lend themselves to graph processing algorithms
* The graph is represented as a hash map; its entry keys are vertices representing the cities, and the values of entry is a list of weighted directed edges. I keep track of edges that point to and from each vertex: in total there are 5 objects:
  + Edge-weighted directed edge
  + Edge-weighted directed graph
  + Trains BFS (breadth-first search)
  + Trains SP (shortest path)
  + Integer indexed Minimum Priority Queue (Note: from a previous implementation that I’ve used before in class)
* Problems 1 – 5 are similar, and I use the following approach to find the length of a path:
  + Split the path into pairs of adjacent vertices
  + Look up the weights of edges formed by each pair from the graph
  + Sum up all weights
* Problems 6, 7, 10
  + Use modified BFS to traverse all possible paths, exiting when the desired conditions are met
* Problems 8 and 0
  + Use Dijkstra’s shortest path algorithm. Modifications include computing the shortest path between a vertex and itself

***Assumptions***

* The graph has no negative weights
* The graph is sparse to justify the use of BFS and Dijskstra’s algorithm
* All distances are integer valued

***How to Run***

* Place the folder “Sample Code” in an appropriate directory
* Command line compile as:
  + javac TrainsClient.java
* Command line run as:
  + java TrainsClient

then follow the prompts (namely, enter the file name – e.g. input.txt, hit Enter; followed by question number – e.g 1, hit Enter; followed by the arguments – e.g. A-E-D, hit Enter)

Each question requires different inputs, and those should be entered from the command line as well.

The application solves each problem one at a time.