CS308 Multigraph ADT

The multigraph will consist of 3 classes (multigraph, MGNode, MGEdge) that represent the core elements that are present in a graph. Each class will define a set of behaviours specified by interfaces representing the abstract data type multigraph, i.e. these classes implement the interfaces. Once multigraph functionality has been defined, the Boston metro system will be built in another class (BostonMetro) that will use an implementation of a multigraph interface to represent the subway map. It will pass the necessary data associated with the subway map into the multigraph where it will be handled away from user interaction. The BostonMetro class will make use of a file reader class that parses the text file with subway data, to tokenise each line to process all information in an iterative manner.

The proposed architecture for implementing the graph is as follows:

An instance of multigraph holds an ArrayList of node interface implementation objects, so can handle all implementations of nodes, and each node has an ArrayList of ArrayLists of edge implementations, so each node has a reference to every node of the graph and can have any number of edges to each (as required by a multigraph). When nodes are added to the graph, the list of nodes and the list of list of edges within each node instance, will synchronise in size to ensure the index of the ArrayLists can be used to properly identify nodes for searching and connecting by edges. These edges are directional.

The graph will provide functionality to (according to the specification as an interface) add nodes, add directional edges, return the number of nodes in the graph, and search for the shortest path from one node to another, returning a list of integers that describe the path. Likewise, the node implementation will allow edges to be added, and check, if any, how many edges exists between itself and another node. Finally the edge implementation provides functionality to set and get its label. The instantiation of this object is the marker for an edge existing.

The algorithm for searching for the shortest path between two nodes will be implemented using the breadth-first-search method. It will operate as follows:

The starting node is given as its ID/name, as is the end node. The multigraph will then look at the instance of node specified by that information, and check its neighbours specified by its edges in its list to see if they match the specified destination node. If not, the graph then does the same check for every node at that depth, one at a time, moving only one layer deeper after each neighbour has been checked. It does this until all possible paths have been checked. Nodes to search first will chosen based on how many edges each has, the one with the most will be checked first and it will go in descending order form there. The possible routes will be tracked using stacks to push and pop currently visited nodes and previously visited nodes until it finds the intended destination. Multiple stacks will be required as there can be more than one path.