This report aims to explain the representation for our multigraph, as well as explain the decisions we made and the reasoning behind why we chose to do it this way.

Representation

Our implementation includes three interfaces, MultiGraphADT, Edge, and Node. The MultiGraph class implements the MultiGraphADT, and the BostonMetro creates a new MultiGraph.

MultiGraph provides descriptions for the methods outlined in the MultiGraphADT interface. As well as this, in addition, it has two additional methods responsible for finding the shortest path between two nodes, and getting the path between two nodes with line changes called findRoute and getPath respectively.

Station class implements Node, and Line implements Edge. The method signatures in the interfaces receive their descriptions from the classes that implement them.

The Parser class is responsible for parsing information from a text file. This information is then used to create the subsequent Line and Station objects that are added to the MultiGraph. This MultiGraph is then used by BostonMetro find the desired route from the provided source station to the provided destination station, which are given via user input.

Rationale

The usage of interfaces is justified by the fact that it provides a good level of abstraction. Alongside that it separates the BostonMetro from the MultiGraph. It means that the core functionality of the MultiGraph is retained even if the scenario or requirements are different for the application. It is for this same reason we have Station implement Node and Line implement Edge; Stations and Lines are essentially just nodes and edges in a multigraph so the same functionality can be applied, again with the occasional additional method as required. It also means that BostonMetro can send Stations to the MultiGraph it’s using, as station implements Node meaning Station is (like Node) an accepted parameter for MultiGraph’s methods.