ANC Assessed Exercise

Robert Allison 1102085a

Program Overview:

The main design of the program starts with a graph. This graph consists of a set of nodes, representing the routers in a network, and a set of undirected edges with a weight, representing the links and costs thereof between them. Both of these sets are stored in lists and are accessible by the graph.
Nodes store a label, a routing table, a list of their direct neighbours with the associated cost, and an initially empty list to store the vectors they receive. Edges consist of two node references and an integer weight between them.
A wrapper main class handles user input from the command line using flags, and handles building the graph from an input file of nodes and edges. The user passes this input files and any commands to the wrapped class, and it displays the appropriate output by manipulating the graph.
Each Node's routing table stores each entry as a row, and the entire table is discarded when the newest data is being processed. During each exchange, each Node slices the cost column of this table, and attaches a reference to itself, forming a 'vector'. It then sends this vector to each of its neighbours.
After the graph has instructed each node to send vectors to their neighbours, it then tells each node to calculate a new routing table based on all the vectors it has. Each node stores this data temporarily in a container, with a reference node as a key, and a list of all obtained costs and links as the value (e.g. N1:{<N2,3>, <N3,1>} means that the node has routes to N1 through N2 and N3, with costs 3 and 1 respectively), and it then checks all of these entries for each Node, and finds the lowest cost path to update its table with.
Theoretically, with split horizon on, a node will check each entry when calculating its vector, and if it has learned that route from the neighbour it is currently trying to send to, it will omit that entry from its vector. This should help ease slow convergence and avoid count-to-infinity.

Data Structures:
DVGraph: A simple graph object with nodes and edges
DVNode: An abstract graph node representing a router
DVEdge: A simple edge object representing a link cost between nodes in the graph
DVNeighbour: A neighbour node object with node reference and cost to reach
RoutingTable: An object representing a routing table, with row entries
RTRow: A row in the routing table, contains Node, Cost, and Next Link entries
Cvector: Simple Cost Vector object. Stores reference to creator node, and a list of costs