Hunter Noey
November 1st, 2019
CS 4310

Design Pattern

To store the data for the Routing Table, I implemented a dynamically-allocated 2D-Array that consisted of vectorRoutingTable struct objects. The vectorRoutingTable struct contains members for the distance connecting the two nodes (determined by their indexes in the array, left serving as source and right serving as destination) nextHop which is used by the first index of the array given it serves as where the packet is being sent from, and isNeighbor, a Boolean that determines if the nodes are direct neighbors or not. To determine the size of the array, we iterate through the file and use the highest number+1 from the given nodes, as adding 1 ensures we can account for 0. After getting the size of the array, we reset the position in the file and obtain the data to store into our array, initializing the given paths and nextHops for the given edge as well as setting isNeighbor to true if the provided nodes are neighbors. Once we have collected our initial data, I initialize the cost of non-neighbors to 999999 (our number for infinity) to compute paths easier via the isNeighbor member.

For calculating the shortest path between nodes, I chose to use the Bellman-Ford Algorithm that replaces a path if a cheaper one is found. In terms of the calculating the cost, the paths are calculated by using a n^4 for loop: the first for is for the amount of rounds provided, the second serving as the source the packet comes from, the third serving as the destination and the last is used for the communication between the paths (an alternative path). When the for loop executes, the nodes compare valid paths and see which one is the shortest, and if a shorter path is found we update the cost of the array element as well as giving it a new hop. We then set a flag to true, which increments the total amount of rounds used within the end of the round loop assuming it returns true.