CSC/ECE 573 – Internet Protocols

Project #3
Due Date: November 30, 2011

Project Objectives
In this project, you will implement the two types of routing algorithms that are used in the Internet, link-state (LS) and distance-vector (DV).

Link-State (LS) Routing Algorithm
You will implement the LS algorithm described in Chapter 4.5.1 of the textbook to find the shortest paths between all pairs of nodes in a given network, as well as the routing tables at each node. Since the LS algorithm in the book finds shortest paths from a given source node u, you will have to run this algorithm once for each node in the given network.

Distance-Vector (DV) Routing Algorithm
You will implement the DV algorithm described in Chapter 4.5.2 of the textbook to find the shortest path costs between all pairs of nodes in a given network, as well as the routing tables at each node.

Unlike the LS algorithm which is centralized, the DV algorithm is distributed, and its execution at a node is triggered by the receipt of a distance vector from one of the node's neighbors. It would be possible to simulate this distributed execution by creating a number of threads, one for each node, each executing the DV algorithm for one node and passing the distance vector to the threads corresponding to the node's neighbors. If you wish, you may implement the algorithm in this manner. But to keep things simple, you can emulate the execution of the distributed algorithm as follows.

First, you must initialize the table maintained at each node as shown in the leftmost column of Figure 4.30. Then, one node (provided as input from the command line) will "send" its distance vector to its neighbors. Note that the node does not need to actually send the distance vector, since the algorithm is implemented in a centralized manner. Rather, "sending" a distance vector involves copying the node's distance vector to the tables maintained by its neighbors, and changing a flag for those neighbors to indicate that they have "received" a new distance vector. Then, the algorithm enters a loop in which it checks the flags of all nodes, and if they are set, the DV algorithm is executed for that node and (if necessary) the distance vector is "sent" to the node's neighbors. Note that the order in which the DV algorithm is executed for the nodes with the flag set is not important: in a distributed algorithm, nodes would execute the algorithm asynchronously and in any random order. Also note that in this centralized implementation, the algorithm terminates whenever no flag for any node is set, indicating that the algorithm has converged at all nodes. We will not be concerned with changes in the link costs.

Input Graph Instance Format
The graph instances will be supplied as plain ASCII text files, with the format described below. The first line of the graph contains a single number which gives the number N of vertices (nodes) in the graph. The rest of the file is an adjacency list, one line per edge. In each line, there are three numbers, separated by spaces or tabs. The first two numbers are integers between 1 and N and indicate two vertices joined by an edge (i.e., two neighboring nodes). The third number is a real number indicating the cost of the edge.

For example, the file representation of a 4-node ring network would be:

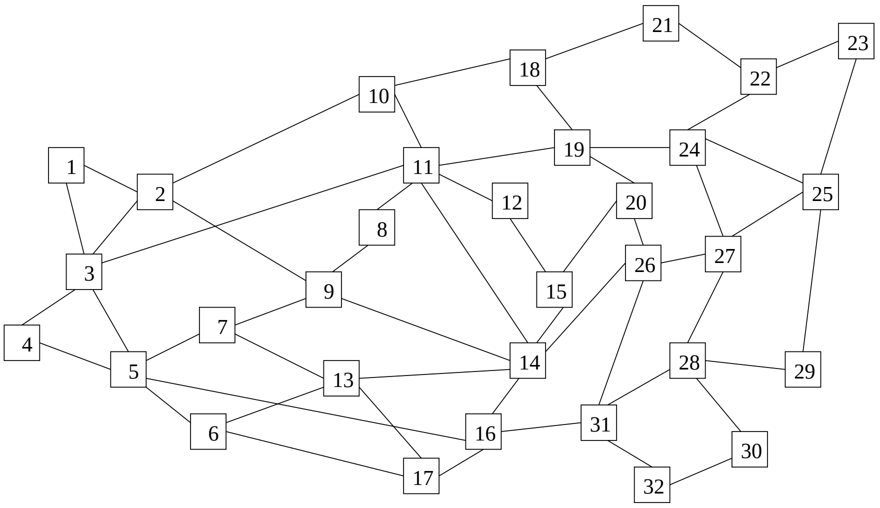

Figure 1: The 32-node network

```
4
1 2 4.0
1 3 3.0
2 4 4.0
3 4 8.0
```

I have provided three input files on which you may run your algorithms:

1. *small-net.txt* is a small network with 10 nodes and 16 edges that you may use to debug your algorithms;
2. *large-net-a.txt* is the 32-node network shown in Figure 1, with unit edge costs; and
3. *large-net-b.txt* is the same 32-node network with edge costs that are a function of the distance between the nodes.

Command Line Arguments
The LS algorithm must be invoked as follows:

```
link_state file-name node1 node2
```

where `file-name` is the name of the file containing the graph representation, and *node1* and *node2* are the numbers corresponding to two nodes in the graph.

The DV algorithm must be invoked as follows:

```
distance_vector initial-node file-name node1 node2
```

where `file-name` is the name of the file containing the graph representation, *initial-node* is an integer corresponding to the node that initially sends its distance vector to its neighbors, as described above, and *node1* and *node2* are the numbers corresponding to two nodes in the graph.

Output
Upon termination, the code you submit must print to the standard output:

- the cost of the least-cost path between nodes *node1* and *node2* provided as command-line arguments, and
- the routing tables of nodes *node1* and *node2*.

Task 1: Running Time of the LS Algorithm
Measure the running time of the LS algorithm for each source node. Plot the measured times for the three different networks provided (the 10-node network and the 32-node network with both sets of edge costs). Is this what you expected? Explain.

Task 1: Iterations of the DV Algorithm
Count the number of iterations that it takes the DV algorithm to converge; this number is the maximum number that any node's distance vector has to be updated before convergence is achieved. Plot this number for each of the three networks provided against the initial node that sends its distance vector to start the algorithm (i.e., for the 10-node network, plot the maximum number of iterations 10 times, each time with a different initial node). Is the convergence time affected by the initial condition? Explain your findings.

Grading

LS algorithm code (compiles and runs correctly)	40 Points
DV algorithm code (compiles and runs correctly)	40 Points
Task 1	10 Points
Task 2	10 Points
	100 Points

Extra Credit: 15 Points
Implement the DV algorithm over UDP and demonstrate it using at least four "nodes" (e.g., peers). Each node will have a DV server to receive the distance vectors from its neighbors, and a DV client that sends the distance vector to the neighbors. For this implementation, the neighboring peers of a node can be manually configured. However, make sure that you complete the basic LS and DV implementation first, before attempting this task.