CS 513 Computer Networks
Craig E. Wills
Available: Tuesday, October 28, 2025

WPI, Fall 2025
Network Layer Routing

Introduction

This project is to program adaptive network routing algorithms and be able to test them as changes occur in the network. For the project assume the network is a graph with nodes represented as single characters and links between nodes with costs represented as positive integers. Assume the all links are bidirectional with the cost the same in each direction. A sample network graph is shown in Figure 1 with 8 nodes and 11 edges between them.

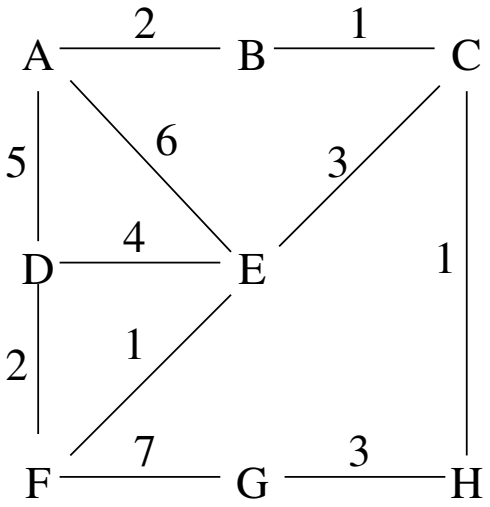

Figure 1: Sample Network Graph

Graph Representation

Your program will work in an interactive manner reading lines from standard input to initially build and subsequently modify a network graph. Input can either be manually entered or stored in a text file and redirected via the command shell as input to your program.

Your program will need to maintain a representation of the nodes and edges in a network graph. Input to add an edge in the graph (or modify an existing one) is specified as "X Y cost" indicating there is a link between node X and node Y (a node is represented as a single capital letter) with the given cost (an integer). Note that specifying an input of "Y X cost" is equivalent. If either node X or Y does not already exist in your graph representation then you will need to add it. If an edge from node X to Y does not already exist then you will need to add one with the given cost. If such an edge already exists in your graph representation then you need to update the cost of the edge.

A special input of "X Y -" is used to indicate that an existing edge between node X and node Y should be removed from the graph representation. If no such existing edge exists then you can print a warning, but otherwise ignore the input.

Using this specification, the following set of 11 input lines can be used to build the network graph shown in Figure 1.

```
A D 5
D F 2
A B 2
A E 6
D E 4
E F 1
F G 7
B C 1
C E 3
C H 1
G H 3
```

Link-State Routing Algorithm

In addition to lines of input for building and modifying a network graph, your program needs to accept commands to compute and show routing tables for different routing algorithms. The first routing algorithm you should provide is the link-state routing algorithm. This algorithm is requested when the command "ls" is specified as input of the form "ls X" indicating the link-state algorithm should be executed to compute and show the routing table for node X. Should node X not exist in your graph representation then you should print an appropriate message and ignore the request.

The output for a valid request should be to show a tuple of (node, via, total cost) for each node that can be reached from node X. For example, for the graph representation in Figure 1, a request of "ls A" should output something like:

```
A - 0
B A 2
C B 3
H C 4
D A 5
E C 6
G H 7
F E 7
```

Note: if more than one equivalent route exists to a node then any one route may be chosen and shown. If a node is not accessible from the specified node then it does not need to be shown. If edge costs are subsequently changed or removed then re-execution of the algorithm for the same node may well result in different paths and costs to nodes in the network. Your program will allow such experimentation.

Distance-Vector Routing Algorithm

Once you have completed your link-state algorithm, you need to use your graph representation to allow for execution of the distance-vector routing algorithm for each node. This algorithm maintains a distance vector for each node, which is initialized with the costs for all directly-connected nodes.

The distance-vector algorithm is different than link-state in that it works in a distributed manner where each node updates its own distance vector based upon updates it receives from its neighbors.

Your program should execute one iteration of the distance-vector algorithm when it receives the "dv" command as input. The form of the command is "dv X" which causes each node in the network graph to "send" its distance vector to each of its neighbors and then each node updates its distance vector accordingly. Once updates are done then your program should print the distance vector for node X. Entering the command "dv X" multiple times shows how the distance vector for node X changes (or does not change) as updates to the distance vectors are propagated in the network.

Your program will also allow you to experiment with your network by seeing how changes in edge costs or the availability of edges translate into routing changes using the distance-vector algorithm. You can also investigate how quickly the distance vectors converge (if they do) based on changes.

Design Decisions

One important issue to address in your design for the project is how to represent the network graph. You can also think of adding other commands and options. For example, you could add another option for the "ls" and "dv" commands that cause your program to show intermediate results during execution of the algorithm.

You may also consider adding commands that show a visualization of the network graph or of the resulting spanning tree.

Grading

Projects will be primarily graded via the final report, although in-person demonstrations may also be used. Projects will be graded based on features and correctness. Correctly implemented projects for the only the link-state algorithm will generally receive grades no higher than in the B range. Correctly implemented projects for both routing algorithms will be eligible for higher grades, particularly when there is a comparison of the algorithms under different scenarios. Variation in grades will depend on the quality of the work, features provided, the analyses that were done and the final report.

Final Comments

This document outlines the project and details on how to design and build it. There are also decisions you need to make. Each of the algorithms are well-known with code available in the textbook, although you probably do not want to use the code verbatim. The Tanenbaum textbook also shows demonstration code. Any code you do use should be documented in your final project report.