CSCI 4964 Programming Assignment 3

Distance Vector Simulation

Due Date: Tuesday, August 16th, 11:59:59 PM

Overview

In this assignment you will be implementing a simplified simulation of Distance Vector routing. Your output should match the provided output as closely as possible, but whitespace does not need to match. I used columns of width 10 for our spacing, but you can try to make an alternate easily readable layout. Your program should run quickly, but you do not need to be extremely concerned with efficiency. 999999 is used as infinity, you can assume that no shortest path will ever be length 999999 or larger.

You may use C++ or Python 3. My solution is in Python 3, however due to the amount of sorting involved, you may find a language such as C++ attractive due to STL maps. Your code should have good comments and be well organized. You should not use any external libraries such as Boost or NetworkX.

The following files are provided, along with *_output* versions containing sample output:
pa3_DVBook.txt - *DV sample from p387, constructed before initialization*
pa3_DVBook_moreoutput.txt - *DV sample from p387, constructed after initialization*
pa3_DVBook_countinf_nopoison.txt - *DV sample from p389, no reverse poisoning*
pa3_DVBook_countinf_poison.txt - *DV sample from p389, reverse poisoning*

SUBMISSION

Submit your source code, at least two different interesting test cases that use 5 or more nodes, the output for your test cases, and a README.pdf containing any notes about your code, how long you took to do the assignment, and a description of your tests.

GRADING

The grade breakdown will be as follows:

6 points: Comments and organization
12 points: DV Implementation (no poisoning)
10 points: DV w/ Reverse Poisoning Implementation
10 points: Correct output
12 points: README.pdf and Test Cases

(continued on next page)

ARGUMENTS

Your program should take 2 arguments, an input file to read commands from, and an output file to write output to.

COMMAND FILE HEADERS

These keywords will always appear at the top of the input file

MODE=[DV,DVPoison]
DV = Distance Vector with no poisoning
DVPoison = Distance Vector with reverse poisoning
During these calculations, if there is a tie for shortest path, sort the names of the routers that tied and pick the first one from the sorted list.

PRINT_TABLE=[ALWAYS,ONCHANGE]
ALWAYS = Print all tables after every command
ONCHANGE = Print only tables that changed after a command

PRINT_MESSAGES=[YES,NO]
YES = Print "Message sent: (source,destination)" every time a message is sent
NO = Do not print anything extra when messages are sent

SIMULATION COMMANDS

ADD_ROUTER label
Adds a new router with the name "label". Names are unique.

ADD_EDGE label1 label2 weight
Adds an undirected edge between Router label1 and Router label2 and gives the edge a cost of "weight". Do not add duplicate edges.

CHANGE_EDGE label1 label2 new_weight
Updates an existing edge between Router label1 and Router label2 to have a cost of "new_weight"

INITIALIZATION_DONE
Because we need to construct an initial graph, until your program sees this command it should not update any routing tables. When this command is read in your program should do any per-router intialization and then do the communication and first calculation of routing tables. Your program should then output the DV table the same way as described in the "ENDING OUTPUT" section. After this command, any changes to the state of an edge or router should be dealt with immediately. You will only see this command once per file.

(continued on next page)

UPDATE RULES

Distance vector is normally run asynchronously, however since we do not want to deal with a distributed/-parallel program, we will artifically cause some synchronization. Note that this creates unrealistic behavior, but is still a reasonable approximation of DV. The DV psuedocode for change_edge (add_edge and initialization_done are similarly) is shown below. Sorting is done using the standard lexicographical string sorting on node IDs.

```
S = empty set
For the changed edge (u,v)
  Update u's table, if it changed add u to S
  Update v's table, if it changed add v to S
While S is not empty
  Sort S
  S' = empty
  for r in S
     remove r from S
     for n in neighbors of r
        send r's vector to n and n will recalculate immediately
        if n was updated and is not in S, add n to S'
  S = S'
  Print changed routing tables
```

ENDING OUTPUT

At the end, every router's DV table should be printed.

For DV this is a matrix where every column is one destination, and every row is one source. Each cell in the matrix is lowest cost from a source to a destination. See Figure 5.6 in the text. The columns and rows should be sorted by label.

Print the tables sorted by router name (i.e. Router A's table, then B's table, etc.) To express an infinite distance, you can use 999999.

At the very bottom, print the total number of messages sent between all routers.