CNT4007C Computer Network Fundamentals, Spring 2015

Programming Assignment 3

--- babak.ap@ufl.edu

Date assigned:	Friday April 3, 2015
Date due:	Friday April 17, 2015 (7:00 am EST)
	NO LATE submission will be accepted for grading!
How to submit:	Please submit through SAKAI.
Questions:	Come to office hour or contact TA (babak.ap@ufl.edu).

Introduction

In this programming assignment, we are going to implement the Link-State Routing Algorithm, known as Dijkstra's algorithm. The detail of this algorithm is in Section 4.5 of your textbook (6th edition), so please make sure you understand the material clearly before starting this project. We also follow the terms defined in the textbook, such as D( ), p( ) and N'.

Notice that this assignment is a simulation of the routing algorithm and is implemented on a stand-alone computer, so it does not require socket programming. But keep in mind that we test your program under CISE Linux machines such as thunder, storm.

Problem Outline

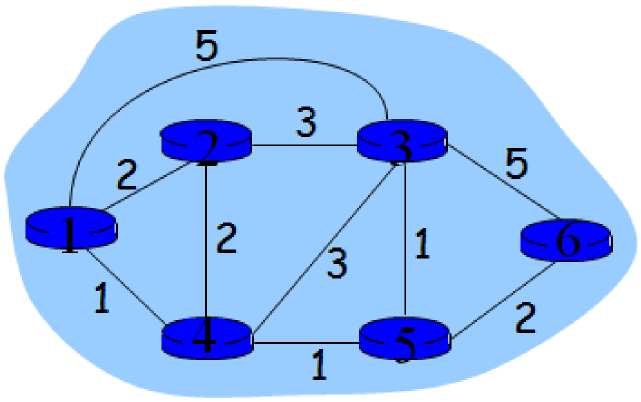
This assignment requires one single program that will read the "network.txt" as the network set-up and runs the Dijkstra's algorithm to get the routing table from node 1 (just like the node u in the textbook example) to all other nodes.

Network Graph

The network will have many nodes labeled 1, 2, 3, … and the links between those nodes will each have an associated weight (or cost, or distance). The network scenario we will use in the project is given in the file "network.txt".

Following is a simple example. However, a **larger network** will be used when testing your program.

```
0,2,5,1,N,N.
2,0,3,2,N,N.
5,3,0,3,1,5.
1,2,3,0,1,N.
N,N,1,1,0,2.
N,N,5,N,2,0.
EOF.
```



This specifies the exact network in the figure. To be more specific, each line (except the last line) describes a corresponding node and the weight associated with each of the link from this node to all other nodes. If there is no direct connection between two nodes, the weight is N (infinity). As we can see, the weight from a node to itself is always 0, otherwise the weight is a positive integer or N. For example, the first line describes the node 1 in the figure and its links to node 2, node 3 and node 4, with weight 2, 5, 1 respectively.

Notice that each weight is separated by a comma ‘,’ and the end of each line is a period ‘.’. Additionally, the file is finished with “EOF”, specifying the end-of-file.

Note that in grading we will use a **larger network** (20+ nodes). See appendix for example.

Algorithm and Output

The Dijkstra’s Algorithm is described in Section 4.5.1 in the textbook. Basically in each iteration we find a node with the least weight and update its route, until eventually all the nodes are included in our routing table (like Table 4.3 in the book). Please read the book carefully to understand the algorithm.

Here what we need to do is to find the route from node 1 to all other nodes, i.e. node 2, 3, … And the program should print out each step just like Table 4.3, except that we use node number 1, 2, 3… not u, v, w… And when a tied weight happens in your algorithm, select the node with smaller node number. The **output format** is:

```
-----------------------------------------------------------------------------------------------------
Step        N’              D(2),p(2)       D(3),v(3)       D(4),v(4)       …
-----------------------------------------------------------------------------------------------------
0           1               2, 1            5, 1            1, 1            …
-----------------------------------------------------------------------------------------------------
1           1, 4            2, 1            4, 4                            …
-----------------------------------------------------------------------------------------------------
…
-----------------------------------------------------------------------------------------------------
5           1,2,3,4,5,6
-----------------------------------------------------------------------------------------------------
```

Make sure you print out the header and **each step** separated by dashed lines! That is the major way for us to tell the correctness of your program.

To test your program, you can copy the network file example I mentioned above, and the output should look like Table 4.3. However, a **larger network** will be used in grading.

Running the System

The program should be able to run on CISE machines, read the “network.txt” file, perform the Dijkstra’s algorithm and print out all the steps in detail. After that, the program will exit.

Here is an example on how to run the system:

```
[C] ./linkstate  network.txt
[Java] java linkstate network.txt
```

Important Note

• The network file name should be command line input, do NOT hard code it.
• Print out clearly all the steps, **including** the header and the dashed lines.
• Test your program with **large network**, run it on CISE Linux machines.

Note on Run-away Processes for the Graceful Termination:

Your program should terminate gracefully. While testing your programs, run-away processes might exist. However, these run-away processes should be killed. Please check frequently if there are any remaining processes after termination. CISE department has a policy regarding this issue and your access to the department machines might be restricted if you do not clean these processes properly.

Some useful Linux/Unix commands:

To check your running processes: ps -u <your-username>
To kill a process: kill -9 pid
To kill all Java processes: killall java
To check processes on remote hosts: ssh <host-name> ps -u <your-username>
To clean Java: ssh <host-name> killall java

Report

Submitted report file should be named “report.pdf” and include the following:

• Your personal information: Full name, UF ID, and Email
• How to compile and run your code under which environment.
• Description of your code structure.
• Show some of the execution results.
  - Discuss the results you got with your program.
• Explain about the bugs, missing items and limitations if there is any.
  - Honesty is valued here.

Submission Guidelines:

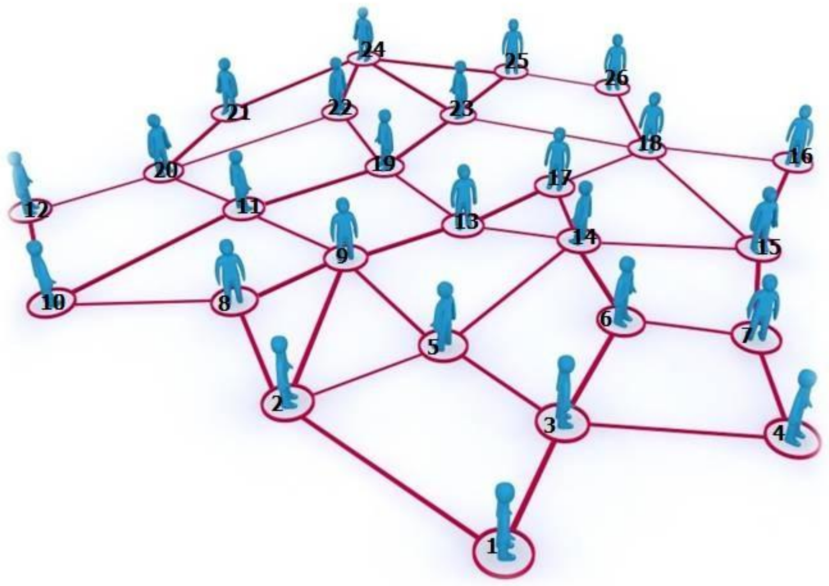
1. The source code files should be named “linkstate.c/java”.
2. Only submit C/Java code files and report, do not include .obj/.class or executable.
3. Include “makefile” if you have one. (Not required)
4. Include the report in .pdf format. Name it “report.pdf”.
5. Zip all your files into a packet: Firstname_Lastname_ID.zip
6. Submit to SAKAI before deadline.

Grading Criteria:

Correct Implementation	50%
Output	20%
Report / Code Style	30%
Total:	100%

Appendix

Here is the proposed network scenario for grading. It might subject to change.



```
0,3,2,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
3,0,N,N,5,N,N,5,3,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
2,N,0,1,4,5,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
N,N,1,0,N,N,2,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
N,5,4,N,0,N,N,N,4,N,N,N,N,7,N,N,N,N,N,N,N,N,N,N,N,N.
N,N,5,N,N,0,2,N,N,N,N,N,N,6,N,N,N,N,N,N,N,N,N,N,N,N.
N,N,N,2,N,2,0,N,N,N,N,N,N,N,5,N,N,N,N,N,N,N,N,N,N,N.
N,5,N,N,N,N,N,0,7,1,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
N,3,N,N,4,N,N,7,0,N,2,N,6,N,N,N,N,N,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,N,1,N,0,9,3,N,N,N,N,N,N,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,2,9,0,N,N,N,N,N,N,N,6,4,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,N,3,N,0,N,N,N,N,N,N,N,5,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,6,N,N,N,0,4,N,N,3,N,5,N,N,N,N,N,N,N.
N,N,N,N,7,6,N,N,N,N,N,N,4,0,8,N,7,N,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,5,N,N,N,N,N,N,8,0,2,N,6,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,2,0,N,3,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,3,7,N,N,0,8,N,N,N,N,N,N,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,6,3,8,0,N,N,N,N,5,N,N,3.
N,N,N,N,N,N,N,N,N,N,6,N,5,N,N,N,N,N,0,N,N,8,2,N,N,N.
N,N,N,N,N,N,N,N,N,N,4,5,N,N,N,N,N,N,N,0,4,7,N,N,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,4,0,N,N,9,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,8,7,N,0,N,9,N,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,5,2,N,N,N,0,4,5,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,9,9,4,0,2,N.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,5,2,0,9.
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,3,N,N,N,N,N,N,9,0.
EOF.
```