EE450 Programming Assignment 2

Fall 2019

Due Date: Sunday November 17th, 2019 11:59 PM (Midnight)

Hard Deadline

(Strictly Enforced: the submission website will automatically close on the deadline)

The objective of this assignment is to help you get familiar with Link-state Routing. This assignment is worth 3% of your overall grade in this course. **It is an individual assignment and no collaborations are allowed. Any cheating will result in an automatic F in the course (not just in the assignment).**

If you have any doubts/questions, email TA your questions or come by TA's office hours.**You can ask TAs any question about the content of the project but TAs has the right to reject your request for debugging.**

Problem Statement:

This project is to building routing table given that the node(router) has already got the complete set of link-state. As an example shown in figure 1, the node 0 has already acquired the "map" from link-state advertisement. The router will make use of Dijkstra's algorithm to build its routing table(destination, cost, next hop).

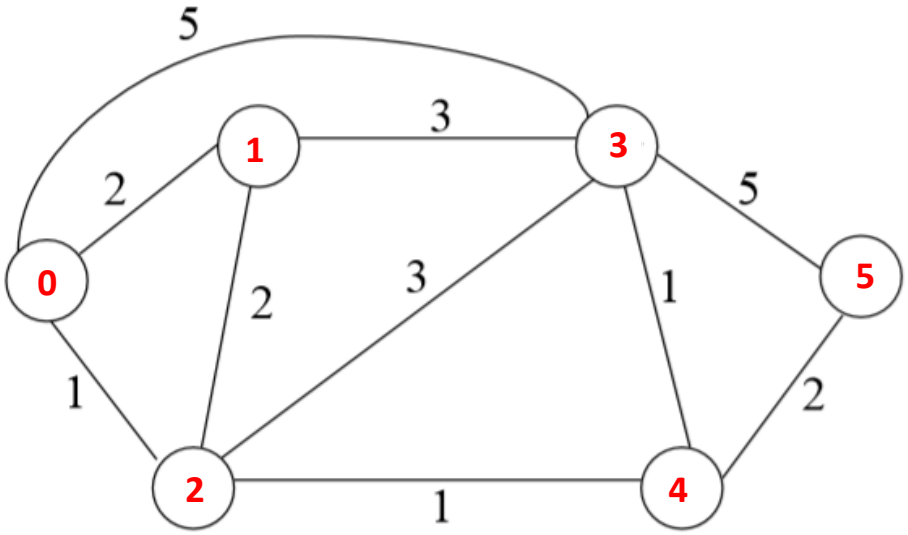

Figure 1. Link state map acquired by node 0

To implement the Dijkstra's algorithm, the graph of link state will be represented as adjacency list in Adj.txt file. The first line of the file indicates the number of nodes. Starting at the second line, each row represents an edge in the graph as <fromNode> <toNode> <distance>. For instance, "0 1 2" represents the edge from node 0 to node 1 whose distance is 2. The node 0 indicates the source node.

After implementation of Dijkstra's algorithm, a routing table of node 0(required in this

project) will be print out as <Destination> <Cost> <Next hop>.

Example Output:

```
Routing table(<Destination> <Cost> <Next hop>):
1 2 1
2 1 2
3 3 2
4 2 2
5 4 2
```

Assumptions:

If you need to have more code files than the ones that are mentioned here, please use meaningful names and all small letters and mention them all in your README file.

Bonus point:

This part worth 20% extra point. To get the bonus point, you are required to output the minimum path tree. For instance, for the example above, the minimum path tree should be the following,

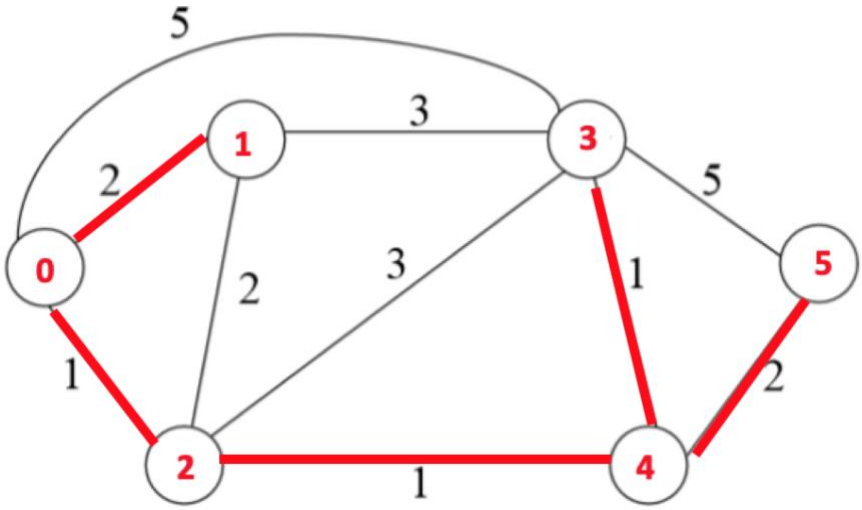

Figure 2. Minimum path tree for source node 0

It could be represented in multiple ways, one possible way is show it by edge.

```
Minimum path tree(represented by edge):
0 1
0 2
2 4
4 3
4 5
```

The other way is show it by path:

```
Minimum path tree(represented by path):
0 1
0 2
0 2 4
0 2 4 3
0 2 4 5
```

It should be noted that the minimum path tree(result of Dijkstra's) doesn't have to the same as minimum spanning tree(result of Prim's).
For example

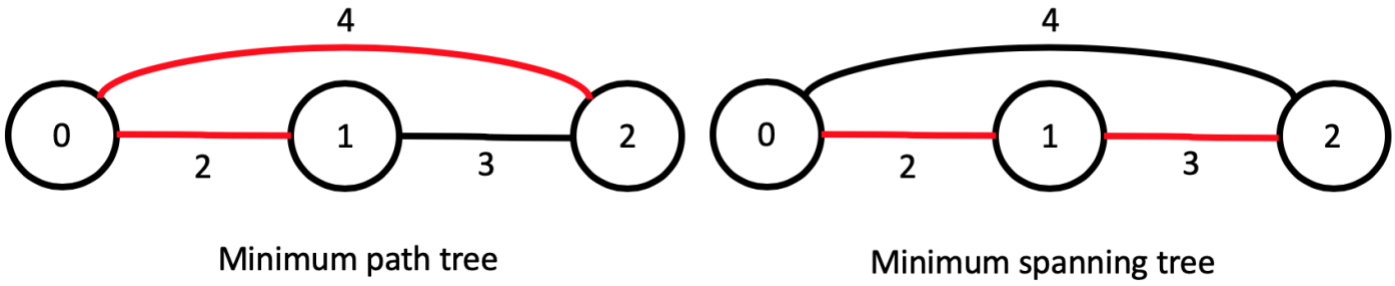

Figure 3. Minimum path tree and minimum spanning tree

Requirements:

1. Your programs should terminate itself after all done.
2. All the naming conventions and the on-screen messages must conform to the previously mentioned rules.
3. All the on-screen messages must conform exactly to the project description. You should not add anymore on-screen messages. If you need to do so for the debugging purposes, you must comment out all of the extra messages before you submit your project.

Programming platform and environment:

1. All your submitted code **MUST** work well on the provided virtual machine Ubuntu.

2. All submissions will only be graded on the provided Ubuntu. TAs won't make any updates or changes to the virtual machine. It's your responsibility to make sure your code working well on the provided Ubuntu. "It works well on my machine" is not an excuse and we don't care.

3. Your submission MUST have a Makefile. Please follow the requirements in the following "Submission Rules" section.

Programming languages and compilers:

You must use only C/C++ on UNIX.

You can use a unix text editor like emacs to type your code and then use compilers such as `g++` (for C++) and `gcc` (for C) that are already installed on Ubuntu to compile your code. You must use the following commands and switches to compile `yourfile.c` or yourfile.cpp. It will make an executable by the name of "`yourfileoutput`".

```
gcc -o yourfileoutput yourfile.c
g++ -o yourfileoutput yourfile.cpp
```

Do NOT forget the mandatory naming conventions mentioned before!

Submission Rules:

1. Along with your code files, include a **README file and a Makefile**. In the README file write
   a. Your **Full Name** as given in the class list
   b. Your Student ID
   c. What you have done in the assignment.
   d. What your code files are and what each one of them does. (Please do not repeat the project description, just name your code files and briefly mention what they do).
   g. Any idiosyncrasy of your project. It should say under what conditions the project fails, if any.
   h. Reused Code: Did you use code from anywhere for your project? If not, say so. If so, say what functions and where they're from. (Also identify this with a comment in the source code.)

Submissions WITHOUT README AND Makefile WILL BE SUBJECT TO A SERIOUS PENALTY

Makefile tutorial:
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

About the Makefile: makefile should support following function:

`make all`	Compiles **your Dijskatra algorithm and run.**

TA will first compile all codes using **`make all`** and run each program with the

corresponding makefile command.

2. Compress all your files including the README file into a single “tar ball” and call it: **ee450_PA2_yourUSCusername.tar.gz**. Please make sure that your name matches the one in the class list. Here are the instructions:

   a.
   On your VM, go to the directory which has all your project files. Remove all executable and other unnecessary files. **Only include the required source code files, Makefile and the README file**. Now run the following commands:

   b.
   >> `tar cvf` **ee450_PA2_yourUSCusername.tar** *
   >> `gzip` **ee450_PA2_yourUSCusername.tar**
   Now, you will find a file named “ee450_PA2_yourUSCusername.tar.gz” in the same directory. Please notice there is a star(*) at the end of first command.

   c.
   Do NOT include anything not required in your tar.gz file. Do NOT use subfolders.
   Any compressed format other than .tar.gz will NOT be graded!

3. Please take into account all kinds of possible technical issues and do expect a huge traffic on the DEN website very close to the deadline which may render your submission or even access to DEN unsuccessful.

4. Please DO NOT wait till the last 5 minutes to upload and submit because some technical issues might happen and you will miss the deadline. And a kind suggestion, if you still get some bugs one hour before the deadline, please make a submission first to make sure you will get some points for your hard work!

5. After receiving the confirmation email, please confirm your submission by downloading and compiling it on your machine. If the outcome is not what you expected, try to resubmit and confirm again. We will only grade what you submitted even though it’s corrupted.

Grading Criteria:

Notice: We will only grade what is already done by the program instead of what will be done.

Your project is graded for 100 ponts and your grade will depend on the following:

1. Correct functionality.

2. Inline comments in your code. This is important as this will help in understanding what you have done.

3. Whether your programs work as you say they would in the README file.

4. **Complete README file: 10 out of 100;**

5. **Working Makefile: 10 out of 100;**

6. **Read in adjacency list: 20 out of 100;**

7. **Correct routing table output: 60 out of 100(1 error will get 10 points deducted);**

8. Your code will not be altered in any way for grading purposes and however it will be tested with different inputs. Your TA runs your project as is, according to the project description and your README file and then check whether it works correctly or not. If your README is not consistent with the description, we will follow the description.

Cautionary Words:

In view of what is a recurring complaint near the end of a project, we want to make it clear that the target platform on which the project is supposed to run is *the provided* ***Ubuntu (16.04)***. It is strongly recommended that students develop their code on this virtual machine. In case students wish to develop their programs on their personal machines, possibly running other operating systems, they are expected to deal with technical and incompatibility issues (on their own) to ensure that the final project compiles and runs on the requested virtual machine. If you do development on your own machine, please leave at least three days to make it work on Ubuntu. It might take much longer than you expect because of some incompatibility issues.

Academic Integrity:

All students are expected to write all their code on their own.

Copying code from friends is called **plagiarism** not **collaboration** and will result in an F for the entire course. Any libraries or pieces of code that you use and you did not write must be listed in your README file. All programs will be compared with automated tools to detect similarities; examples of code copying will get an F for the course. **IF YOU HAVE ANY QUESTIONS ABOUT WHAT IS OR ISN'T ALLOWED ABOUT PLAGIARISM, TALK TO THE TA.** "I didn't know" is not an excuse.