CS515

Assignment 12

The purpose of this assignment is to give you practice with graphs and graph algorithms.

Consider a problem that online game designers and Internet radio providers face: How can you efficiently transfer a piece of information to everyone who may be listening. This is important in real-time gaming so that all the players know the very latest position of every other player; this is important for Internet radio so that all the listeners that are tuned in are getting all the data they need to reconstruct the song they are listening to.

Figure 1 illustrates a network with a broadcast host, routers (represented by nodes A-G), and listeners that are connected to some routers. The weight of each connection represents the cost of sending a message using this link.

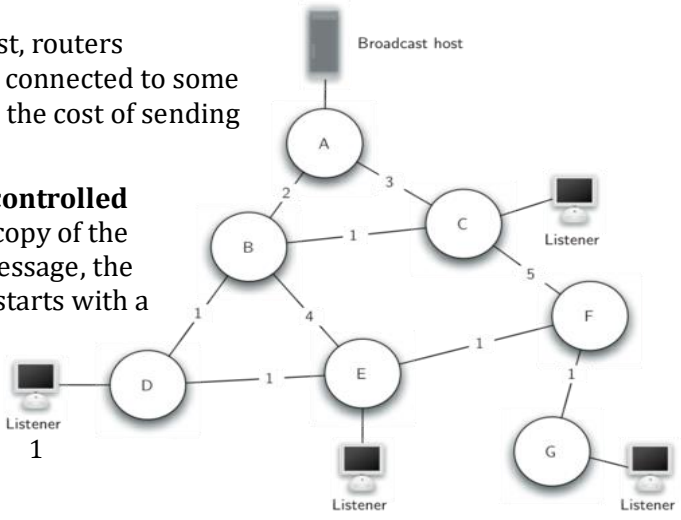

Figure 1. *A weighted connected graph.*

One approach is a brute force approach called **uncontrolled flooding.** For the broadcast host to send a single copy of the broadcast message and let the routers relay the message, the flooding strategy works as follows: each message starts with a time to live (`ttl`) value set to some number greater than or equal to the number of edges between the broadcast host and its most distant listener. Each router gets a copy of the message and passes the message on to all of its neighboring routers. When the message is passed on the `ttl` is decreased. Each router continues to send copies of the message to all its neighbors until the `ttl` value reaches 0. It is obvious that the uncontrolled flooding approach generates many unnecessary messages.

Another approach is for the broadcasting host to keep a list of all of the listeners and send individual messages to each. Using this approach on the graph shown in Figure 1, four copies of every message would be sent using the least cost path (shortest path) to the four listeners. All messages from the broadcaster will go through router A, so A sees all four copies of every message. Similarly, routers B and D would see three copies of every message since routers B and D are on the shortest paths from the broadcasting host to listeners 1, 2 and 3. This approach still generates lots of traffic in the network.

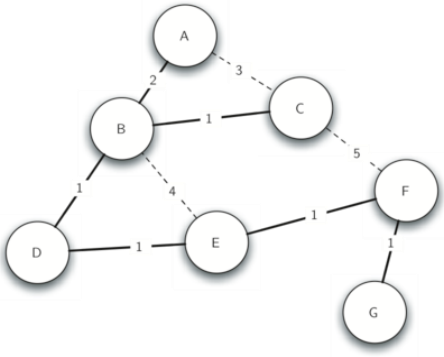

Figure 2. *A minimal spanning tree.*

A better solution to this problem is to construct a **minimal spanning tree (MST)**. Figure 2 shows a simplified version of the broadcast graph in Figure 1.

The highlighted edges form a minimal spanning tree for the graph. Now to solve our broadcast problem, the broadcast host simply sends a single copy of the broadcast message into the network. Each router forwards the message to any neighbor that is part of the spanning tree, excluding the neighbor that just sent it the message.

In this example, A forwards the message to B. B forwards the message to D and C. D forwards the message to E, which forwards it to F, which forwards it to G. No router sees more than one copy of any message, and all the listeners that are interested see a copy of the message.

In this assignment, you are asked to implement a program to simulate a network. The program first constructs a graph from an input file. The file name will be the one and only command-line argument passed to the program. The input file stores a *weighted, undirected, connected* graph. The file contains entries formatted as follows:

<name1> <name2> <weight>

Each 3-tuple denotes a link between the nodes <name1> and <name2> that has a weight of <weight>. The name of the node is an ASCII string (the label may be longer than one character) and the weight is a non-negative integer value.

The starter code contains an executable that can be run as

```
%mynetwork data0
```

to demonstrate a sample run with the sample graph in Figures 1 and 2.

The program will read from the file whose name is passed as a command line argument, and then construct the graph and its minimal spanning tree. Next, the user is presented a prompt to query the graph from a starting node (which is the first node of the first tuple of the input file). The starting node is displayed at the prompt as:

<start>#

The user can then work with simulated network commands **ping**, **ssh**, and **exit** interactively from the keyboard. When the user issues the **ping** command to communicate to a target node, the program uses the MST to generate a stack of nodes that construct the path from the start node to the target node (shown➔).

```
<A># ping B
From: A
  To: B
<A># ping F
From: A
  To: B
  To: D
  To: E
  To: F
<A># ping E
From: A
  To: B
  To: D
  To: E
```

The **ssh** command allows the user to change the current node. For example, by calling ssh to B at current node A, the current node is changed to B. Then the ping commands that follow will print out paths using B as the new starting node.

```
<A># ssh B
<B># ping E
From: B
  To: D
  To: E
```

The user may continue to move to other nodes using ssh, and they are also allowed to ssh to the same node. The program must keep track of the user's ssh history, so they are able to exit from the previous ssh using the **exit** command. For example, suppose they have moved from A to B and then from B to E. The first exit disconnects from E and returns to B. The second exit disconnects from B and returns to A (the current run's starting node), and the last exit terminates the program.

```
<B># ssh E
<E># ping A
 From: E
   To: D
   To: B
   To: A
<E># exit
<B># exit
<A># exit
%          (program ends)
```

The program displays an error message if ping or ssh commands are issued to a non-existing node in the graph (as illustrated below)

```
<A># ping Z
Cannot find node.  Available nodes are:
  A
  B
  C
  D
  E
  F
  G
```

```
<A># ssh Z
Cannot find node.  Available nodes are:
  A
  B
  C
  D
  E
  F
  G
```

You may assume that the input file and commands issued at the program prompt are all in proper valid formats.

Requirements:

- The MST must be built using **Kruskal's minimal spanning tree algorithm.**
- You need to implement your own **disjoint set** data structure (See Lab 10) that will be used in Kruskal's algorithm.

Submission:

- You need to submit **all** the files (header and source) required to compile your program. You must submit a makefile that will make an executable target named **`mynetwork`**. A sample makefile is provided in the starter code.

```
CC = g++
DEBUG = -g
EXECUTABLE = mynetwork
SOURCES = mynetwork.cpp

all: $(EXECUTABLE)

$(EXECUTABLE): $(SOURCES)
	$(CC) $(DEBUG) $< -o $@

clean:
	rm -f $(EXECUTABLE)
```

http://stackoverflow.com/questions/3220277/what-do-the-makefile-symbols-and-mean

- Make sure your submission compiles successfully on Agate and Mimir when **`make`** is called. Programs that produce compile time errors or warnings will receive a zero mark.
- You should also fill out the README file and submit it as well. Submit all your files, including your makefile and README file to the appropriate assignment on Mimir.
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**     CS515 Assignment 12
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- To successfully pass the provided sample tests is not an indication of a potential good grade; to fail one or more of these tests is an indication of a potential bad grade.
- You must test thoroughly your program with your own test data/cases to ensure all the requirements are fulfilled. We will use additional test data/cases other than the sample tests to grade your program.
- Here is a tentative grading scheme. Valgrind checks are not required for this assignment.

directions	100
test run 1.1 (Ping)	10
test run 1.2 (Ping)	10
test run 1.3 (Ping)	10
test run 1.4 (Ping)	10
test run 2.1 (Ping, SSH and Exit)	15
test run 2.2 (Ping, SSH and Exit)	15
test run 2.3 (Ping, SSH and Exit)	15
test run 2.4 (Ping, SSH and Exit)	15