CS310
Project 4

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.

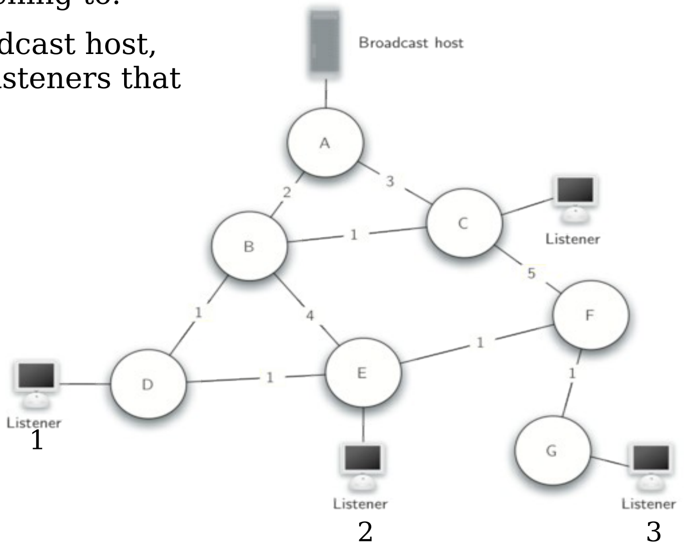
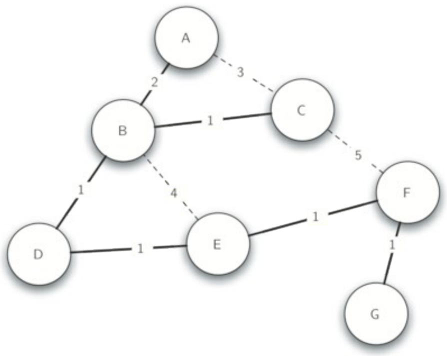


Figure 1. *A weighted connected graph.*

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 and the weight is a non-negative integer value.

The program will read from the file whose name is passed as a command line argument, and then construct the graph. Next, the user is allowed to enter commands for sending messages on the network. The user can choose the sending method **broadcast**, **MST**, or **direct** interactively from the keyboard. Each command consists of the method, the message to send, and the source node. For the broadcast method, there is an additional TTL value that must be provided.

To enable this, you will need to complete the following functions in the provided file `network.py`:

`broadcast(src, graph, ttl)` uses a depth-first approach to send messages from the source to all nodes in the network. It adds nodes to its stack as it encounters edges leading to them, except if the distance (not the cost) from the source to that node is greater than the TTL value that was given. It is recommended that you use the Python `deque` class for your stack.

`direct(src, graph)` uses Djikstra's single source shortest path algorithm to find the shortest path from the source to each other node in the network. It is recommended that you use the included `heap` data structure, as it has a method for decreasing keys when a lower cost path is found.

`mst(src, graph)` constructs a minimum spanning tree rooted at the source node using Prim's algorithm, and then uses that spanning tree to find paths to each other node. These paths may not be the lowest cost paths, but they will require the least repeated network traffic, as described above. It is recommended that you use the included `heap` data structure when building the spanning tree, as it has a method for decreasing keys when a better edge cost is found.

The return value for each of these methods should be a dictionary which stores, for each node name, the name of the node previous to it in the path leading back to the source node. The stored value should be `None` for the source node and any node for which a path could not be found.

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

Submission:

- You need to submit only the file **`network.py`**.
- Make sure your submission runs successfully on the CS server using the `batch.run` file and the sample input files provided to you. Programs that do not run with the provided start files will have points deducted and risk receiving a zero score.
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
# CS310 Assignment 4
# Name: XXX
# Date: XXX
# Collaboration Declaration: assistance received from TA, etc.
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features or design requirements.
- 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 your program thoroughly 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.

Broadcast	
correct paths	10
correct approach	15
Direct	
correct paths	10
correct approach	15
MST	
correct paths	10
correct approach	15