Python 要求

This is an individual project so collaboration or sharing of code is not allowed. The University policy on plagiarism and collaboration will be strictly enforced. **Limited internet references may be used (you can't submit a project you download from the internet), but any source you use more than 10 lines of code from should be documented in your report.** Some examples of acceptable code snippets include: code for how to use TCP sockets in Python, code for how to make containers thread-safe, code for how to create threads, etc.

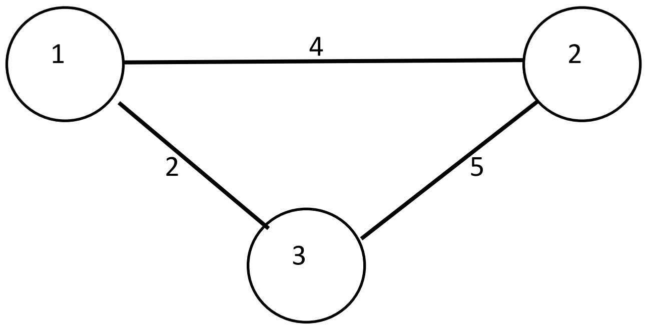
The goal of this project is to Implement link state and distance vector routing algorithms. To make this simpler, it is broken up into several parts. These algorithms work on weighted graphs, so as input to your project, you will be given a file describing the graph with the following format:

```
Node <X>
<connected node>  <cost>
…
```

For example,

```
Node 1
2         4
3         2
Node 2
3         5
```

Is the graph:



Note that because edges are undirected, and the costs are bidirectional, the links are only specified once.

1. (10 points) Read in the graph configuration file.

2. (30 points) Implement the Dijkstra’s algorithm (link state). Because this algorithm requires global knowledge, you can implement it just for one node. You should output the solution to a file. This file can be any format you like, but document it in your report.
3. (50 points) Implement the Bellman-Ford distance vector algorithm. Because this algorithm is distributed, you need to do the following:
    a. For each node in the graph, you need to spawn a separate process. (programmatically)
    b. Each process needs to make a TCP connection to the neighbors specified in the graph file. For example, Node 1 would connect to 2 and 3, and Node 2 would connect to Node 3. Since TCP is bidirectional, only one connection is required.
    c. Each process should start with *only* local information, meaning the costs to send to each neighbor node. It’s OK to read in the whole file, but you can’t use information from the file that isn’t for your node.
    d. You should create a protocol to exchange distance vector information, as required by Bellman-Ford.
    e. Nodes should asynchronously exchange distance vector information as needed until the algorithm converges.
    f. Since costs are fixed, you don’t have to worry about poison reverse or anything other than the basic algorithm.
    g. After convergence, each node should output their local DV table to a file (once file per node). Again this can have any format, but you should document this in your report.

4. (10 points) A report detailing what you implemented, how to build and run your code, any known bugs or limitations and any internet sources you used more than 10 lines of code from.

Suggestion

First off, to clarify the assignment, you need to take the name of the graph definition file as a command line parameter. Don't use prompts and ask the user to type in the file name. This is very important. We're going to test these programmatically, so if you do something like "Please enter the file name" your program will fail all test cases and you'll get a zero. It should be run as "myProgram nodeFile.txt". I'm attaching and example nodefile.txt to this message. Note that there are tabs between the node # and the weight, though most libraries will take care of this for you pretty easily, since a tab is whitespace.

There are many ways to do project 2 part 2, but I thought I'd share some thoughts on how to get started. In part 2, you need to spawn separate processes and exchange information between them. To do this, I'd suggest the following approach:

1. In Python, you can spawn a new process with the subprocess library https://docs.python.org/3/library/subprocess.html#module-subprocess

2. One of the things you can do with this library is pass in command line arguments to the programs you run.

3. You need to establish conventions on who starts processes, how processes choose listening ports, and who connects to who. For example you have 3 nodes in your graph. Who starts the three processes? What 3 ports will they use? And for a connection, will node 1 connect to node 2 or node two connect to node 1? Connections are bidirectional, so you shouldn't make two connections.

4. My suggestion is that you make the syntax for starting a command "myProgram <nodeFile name> <nodeNumber>", with the last parameter being optional. Then, when the program starts, check the number of parameters. If there is no node number, then this is node 1. Then have node 1 read the nodeFile and start the other nodes, but passing in the node number to the other nodes. For example you would start the second node as "myProgram nodeFile.txt 2".

5. Establish a convention for what ports nodes listen on. All the processes run on the same machine so an easy one would be something like "Node N's listening port is 1000+N". This means node 1 listens on 1001, node 2 listens on 1002, etc etc.

6. Establish a convention for who connects to who. The simplest is the smaller number in a pair connects to the larger number node. For example, is there is a connection from node 2 to node 3, then node 2 initiates a TCP connection to node 3. This follows the file format, and makes things simpler. A node can simply read its part of the file and connect to whatever nodes are in its section.

7. Realize that you may need to wait a bit to connect until everything is started up. For example if you start node 2 and immediately try to connect to node 3, node 3 may not be running yet. You either need to handle this failure and retry, or just sleep a couple of seconds and wait for everything to start up. Remember to start your listened before you sleep or it's pointless...

8. When you connect to a node, it won't know what your node number is. Probably the first thing you should send is some fixed length string or integer and tell the node who you are. For example, node 2 connects to node 3, and you send "2" as the first message so node 3 knows this connection is from node 2. If you make this fixed length, it's easier. For example 4 bytes.