Implementing Distance Vector Routing Protocol

All tangible works (e.g. code and report) handed in must be original. Duplicate or very similar assignments will receive a failing grade. Also the defined actions will be taken according to the rules of the department, college, and university.

In this assignment, you have a choice (option) to work in a team of TWO. Both team members should turn-in the assignment. And the report must include contributor's name(s). For team works, no name no marks.

Assignment requirements:

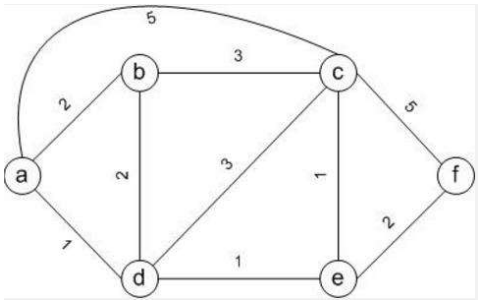
In this project, you will implement the distance vector routing protocol. Your program will be running either at several different machines (preferred) or in a single machine (simulation). At each router, the input to your program is the set of directly attached links and their costs. Note that the program at each host doesn't know the complete network topology. Your routing program at each router should report the cost and the next hop for the shortest paths to all other routers in the network.

Instead of implementing the exact distance vector routing protocol described in the textbook, you are asked to implement a variation of the protocol. In this protocol, each host sends out the routing information to its neighbors at a certain frequency (once every 15 seconds), regardless whether the information has changed since the last announcement. This strategy improves the robustness of the protocol. For instance, a lost message will be automatically recovered by later messages. In this strategy, typically a host re-computes its distance vector and routing table right before sending out the routing information to its neighbors.

Since we don't have real network routers to test the routing algorithm, your programs will run on your laptop or desktop. However, if your routing program runs well on a set of desktop machines, it will also likely to work on real network routers. As specified in the distance vector protocols, your routing program will exchange the routing information with directly connected neighbors. Real routing protocols use UDP for such exchanges. Under such a scheme, each host should listen at a UDP port for incoming routing messages.

Input format:

Your program at `router x` should take a text file as input, which describes the set of directly attached links and their costs. The first line of the text file is a single number, which stands for the number of directly attached links. All subsequent lines in the input file are in the format of `"y cost"`, which stands for a link between `node x` and `node y` with cost `cost`. Note that `cost` can be a floating point number. The two fields are separated by a single space in each line of the input file. Let's look at an example with the network topology shown in the following figure.



The input files at all hosts will look like the following: a.dat, b.dat, c.dat, d.dat, e.dat, and f.dat. Be aware that we might use different network topologies in our testing.

Example for a.dat

```
3
b 2.0
c 5.0
d 1.0
```

Output format:

Your program should produce a terminal output each time it sends out the routing information to its neighbors (i.e., once every 15 seconds). Each such output should include an incremental output number (1 for the first output, 2 for the second output, etc.). Each output should also include the cost and the next hop for the shortest paths to all other network nodes. For instance, one of the terminal outputs at `node a` may look like the following.

```
> output number 4
shortest path a-b: the next hop is b and the cost is 2.0
shortest path a-c: the next hop is d and the cost is 3.0
shortest path a-d: the next hop is d and the cost is 1.0
shortest path a-e: the next hop is d and the cost is 2.0
shortest path a-f: the next hop is d and the cost is 4.0
```

You must produce the terminal output from the first time the host sends out its routing information to its neighbors. As expected, the first a few outputs often contain immature routing information.

Startup and termination:

It is possible that some hosts may start earlier than their neighbors. As a result, you might send the routing information to a neighbor which has not run yet. You should not worry about this since your routing program at each host will repeatedly send the routing information to its neighbors and a slow-starting neighbor will eventually get the information. You program does not need to terminate. It should keep running and outputting the routing information until being killed.

A Challenge - Link cost change:

Your implementation should also be able to handle link cost changes. More specifically, to emulate the cost change of a link, the input files for the two nodes attached to the link should be updated. During your testing, you might not be able to update the two files simultaneously. But this is fine --- in practice the two attached nodes may not simultaneously detect a link cost change either. Due to potential link cost change, your program at each host should **re-load** the link input file each time it is about to re-compute its distance vector and routing table (typically right before sending out the routing information to its neighbors).

Although normally a link cost change can be handled quickly, the distance vector algorithm may suffer from the recursive-update problem. In class we discussed a solution to handle the routing loops. You should implement this solution and test it. Note that you implementation only needs to support link cost changes, not network topology changes.

Turn-in:

1. You are asked to turn in your source files, and a makefile if needed. Any programming language (e.g. C++, Java, C#, Python) can be used. But your program should be able to take two parameters (listening UDP port number and the input file that describes directly attached links) on startup. Use your judgement for port numbers and make the necessary connections that is convenient to you.
2. The report should contain a description of your design and implementation strategies. It should describe how to run your program (the exact parameter format, etc.).
3. Direct questions to TA - Akhil Reddy Kallam (akallam1@uncc.edu).

Grading:

1. 60%: a working program that correctly computes the shortest paths from all-node startup and prints out the results in a reasonable time frame.
2. 20%: correctly re-computes the shortest paths from link cost changes and prints out the results in a reasonable time frame (no loops).
3. 10%: quickly handle the recursive-update problem that only involves routing loops.
4. 10%: a well formatted report in pdf, clarity of your source code and completeness of your comments.
5. Late turn-ins will be accepted for up to three days, with 10% penalty for each late day. No turn-ins more than three-day late will be accepted.

Reference: adapted from online problem set.