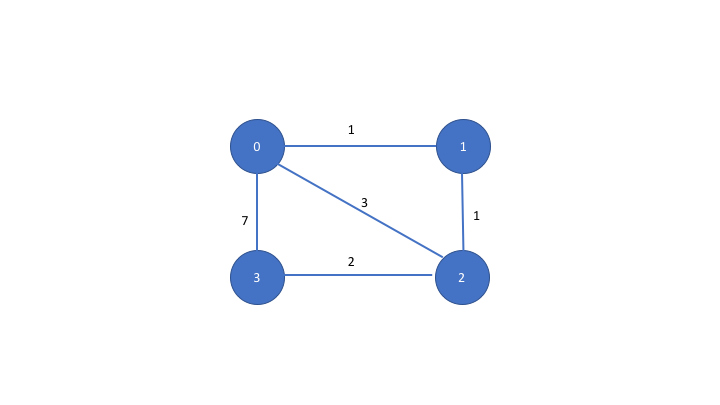
**CS356 Programming Assignment**

This project will be done in phases. The final phase will require four instances of a program running across a network that emulate the operation of a distance-vector routing protocol. Given a network with initial link costs, the output will print out the initial routing table of Router0, and all updates and resulting changes to that routing table based on information from neighboring routers.

**Network Topology**

Network Topology showing four nodes or routers (Router0, Router1, Router2, Router3) with link costs to other routers.



**Implementation Options**

You may use C, C++, Java or Python. CS majors are strongly encouraged to use C or C++ to gain “native” experience with the socket API.

**Computer Systems**

The following computers can be used for running the programs:

* afsaccess1.njit.edu through afsaccess4.njit.edu

**Avoid** using afsconnect1.njit.edu or afsconnect2.njit.edu. Students have reported issues with communications not working between these machines.

**Phase 1 Description**

In this phase, you will implement code for only 2 routers: Router0 and Router1. The code will exchange the initial routing tables between client and server only (processing of the router updates will be done in later phases). Router0 will act as a “client” and Router1 as a “server”. Router0 and Router1 must run on different machines and communicate over the network.

Router0 and Router1 start with an initial routing table based on the topology above. Each router starts off knowing only the link cost of each of its directly connected interfaces. For example, the initial link costs of Router0 to directly connected routers will look as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Destination Routers** | | | |
| **Source Router** | **Router0** | **Router1** | **Router2** | **Router3** |
| Router0 | 0 | 1 | 3 | 7 |

Router1’s initial table can be determined similarly by looking at its directly connected interfaces in the network topology above. Note: It is possible that the link cost is unknown since there may not be a directly connected interface to all routers. Hence, there must be a way in your program to represent an “unknown” link cost.

In Phase 1, you will design a protocol to enable Router0 and Router1 to request and advertise routing updates as follows over the UDP transport protocol.

The protocol message should have a **header** and a **body**. The header will include the following fields: version number, message type (e.g. request or response) and an identifier (to match requests and response messages). The body of the message will include a routing update (which in Phase 1 is just the initial known costs from the source router to all destination routers as shown in the table above). The protocol should be designed to allow a router to **request** a routing update from another router (“pull” model). and the ability for a router to **advertise** its routing update to another router **asynchronously** (“push” model).

Phase 1 includes the following functionality on the Client:

1. Client (Router0) prints its own initial routing table
2. Client (Router0) requests a routing update from the Server (Router1). Router1 responds with its initial table. Client prints Router1’s response.
3. Client (Router 0) advertises its routing update to the Server (Router1) asynchronously
4. Please include “**trace” output** on the screen prior to the client sending requests and after receiving responses so that it is clearer what the client is doing e.g. Prior to sending the routing table request to the server, the client may print a statement as follows prior to sending a request to the server to get the server’s routing table, and after receiving a response:  
   *“Sending routing table request to R1 at [<IP address>, <port>] “*

…..

*“Received routing table response from R1 at [<IP address>, <port>]”*

Phase 1 includes the following functionality on the Server:

1. Server (Router1) prints its own initial routing table
2. Server (Router1) waits for at least two messages from Client ( Router 0). This message may be either a request for a routing update from the server, or it may be an asynchronous update (the information in the message header should allow the Server to distinguish between these two cases).
3. Server (Router1) prints the update it received from Router0
4. Please include **“trace” output** analogous to that for the client specified above.

Phase 1 Deliverables:

You will copy your source files to a special directory for testing, as well as submit paper documentation.

In your AFS user directory, each of you should see a symbolic link to a directory of the form:

**s17.cs.356.[002, 004, 102]**

depending on what section you belong in.

The source files for client and server needs to be copied to this directory along with a text file called “README” that describes the command lines necessary to compile and run your program with all necessary arguments. The program will be tested on the afsaccess\* machines, so you need to make sure the program runs there. If you have hard-coded the IP address and port of your server, specify this in the README file. If you have specified these as arguments to your program, please indicate this clearly in the README file.

In summary, please include the following in the above directory:

1. README file explaining the following
   1. Command line to compile server program
   2. Command line to run server program (including arguments, if any)
   3. Command line to compile client program
   4. Command line to run client program (including arguments, if any)
2. Source flle for server
3. Source file for client

Also, please print out the program files and hand in in class on the due date plus the following:

1. Short description of your protocol including message format, message types and message flow
2. Code listing of Client and Server
3. Screenshot of expected output, including
   1. client and server routing tables
   2. trace messages specified in the description above
4. List References consulted when designing and building your program (if any). Clearly indicate what code is yours versus based on others.

**Phase 2 Description**

In this phase, you will run all 4 routers in the given network topology. You will convert your program such that each router can run as peers of one another i.e. any router can both send and receive routing updates. Each router receives routing updates (distance vectors) from **neighboring** routers in the network topology, runs the Bellman-Ford algorithm to determine any updates to its distance vector, and sends its distance vector to its neighbors if changed. The routers continue to exchange information until no more changes need to be made.

Phase 2 includes the following functionality on each Router:

1. Router prints its own initial routing table (distance vector)
2. Router sends its own initial distance vector to neighboring routers in topology
3. Router waits for distance vectors from other neighboring routers
4. Router updates its own distance vector as necessary
5. Router sends its updated distance vector when necessary
6. Continue until converged to shortest paths
7. Please include **“trace” output** analogous to Phase 1 that indicates
   1. Initial distance vector
   2. When a router sends its distance vector and to whom
   3. When a router receives a distance vector and from whom
   4. Updated distance vectors (its own distance vector and distance vectors received from others)

Phase 2 Deliverables

As for Phase 1, you will copy source code to the shared directory in Phase 1 and include a README file describing how to compile and run your program. Please choose different filenames for Phase 2 files so they can be distinguished from Phase 1 files. **Do not overwrite** Phase 1 files unless told otherwise.

Also, please hand in in class on the due date the following:

1. Code listing of Router code
2. Screenshot of expected output, including trace messages specified in the description above, that illustrate algorithm converging to shortest path
3. List References consulted when designing and building your program (if any). Clearly indicate what code is yours versus based on others.

**Academic Integrity:** The [University Code of Academic Integrity](https://www5.njit.edu/doss/sites/doss/files/University%20Policy%20on%20Academic%20Integrity.pdf) will be upheld, and any violations will be brought to the immediate attention of the Dean of Students.

Remember, the program must be your own work. You must give credit where your answers are based on help from others. For example, It is OK to reference socket programming tutorials in textbooks and elsewhere to research socket API details and usage, as long as you give credit, and it is clear what is your own work versus others.