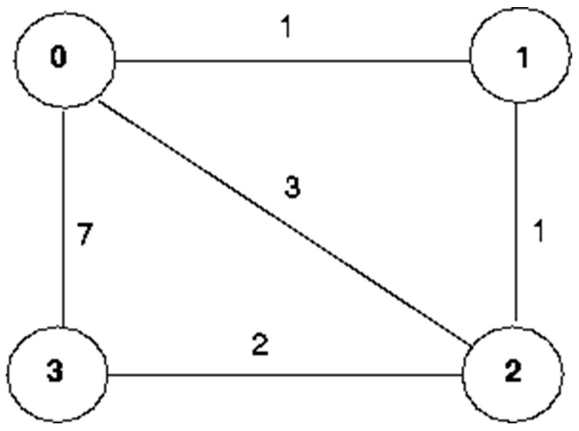
Overview:

In this third programming assignment, you will be writing a ``distributed'' set of procedures that implement a distributed asynchronous distance vector routing for the network shown below.



Tools Used -

In Windows (MinGW/gcc compiler) was used to compile and execute the code in ***C***.

Algorithm Used: Distance Vector Routing Algorithm

The basic assumption of this algorithm is that each node knows its address and the cost to reach each of its directly connected neighbors and each node can exchange information only between nodes that are its immediate neighbors. This algorithm is asynchronous, iterative and self terminates when there is no more information to exchange between neighbors.

This algorithm uses Bellman-Ford distributed route computation to define and update the cost information on its distance table. This involves the following,

- Each node x maintains the cost of direct links: c(x,v) = cost for direct link from x to v
- Each node x maintains distance vector Dx = [Dx(y): y ϵ N], where Dx(y) = estimate of least cost from x to y.
- Node x maintains its neighbors' distance vectors: For each neighbor v, x maintains Dv = [Dv(y): y ϵ N]
- If a node is not a connected neighbor of node x, then the cost to reach that node will be considered as infinity during initialization.
- Each node v periodically sends Dv to its neighbors and neighbors update their own distance vectors. The Bellman-Ford equation to update its own distance vector as follows,

Dx(y) ← minv{c(x,v) + Dv(y)} for each node y ∊ N

Implementation:

Given:

- The prototype versions of the node files (node0.c, node1.c, node2.c, node3.c) containing declarations of some structs pertaining to packet and distance table, and a function that prints the data in distance table.
- An emulator file 'prog3.c' which simulates nodes and medium through which the packet exchange between nodes happen.

Node Initialization Method:

A node initialization routine (rtinit0(), rtinit1(), rtinit2(), rtinit3()) is implemented in each of the node files. In all the nodes this routine does the following functionality,

- Initializes the distance table with the direct costs of links between the node and its connected neighbors. The cost to all the nodes that are not connected neighbors of a node in focus is set as 999.
- Creates a distance vector that contains the cost information from the node to other nodes present in network.
- Creates a packet containing distance vector information along with the source and destination node information.
- Sends the created packet to all the connected neighboring nodes.
- Prints the distance table of the node.

Method to Fetch Minimum cost:

Each node has a method that finds the minimum link cost from that node to all other destination nodes in the network. This information is stored in an array and is part of the packet that gets exchanged between nodes.

Method to Update Distance Table:

Each node has an update method (rtupdate0(), rtupdate1(), rtupdate2(), rtupdate3()) that gets invoked by the emulator when the arrival packet destination id is same as that node's id. The minimum cost values that are present in the arriving packet of node 'x' are the current shortest link cost from node 'x' to all other nodes in network. These values are used to update the distance table of the node. As a result of this update if any of its own minimum cost to other nodes in network changes then, these changes are bundled in a routing packet and is sent to all its connected neighbors. In the end the method prints the distance table information of the node.

The packet exchange between nodes continue iteratively until all the packets in the medium are exhausted and the distance table in each node at this point will contain the minimum possible link cost from the node to all other nodes in the network.