Documentation for Distance Vector Routing

Abhishek Prashant Chandurkar - BT22CSE104

1 How to Run the Code

To run the Distance Vector Routing (DVR) code, follow the steps below:

- Extract the contents of the ZIP file.
- Open the `code` folder, which contains the script `BT22CSE104-dvr.py`.

To run the code, use the following command format in the terminal or command prompt:

```
python BT22CSE104-dvr.py <path to topology file>
```

For example, for test case 3, use the following command:

```
python BT22CSE104-dvr.py 'c:\Users\HP\Bellman-Ford DV\Code\BT22CSE104-dvr.py'
'c:\Users\HP\Bellman-Ford DV\Test Cases\Multiple Equal Cost
Paths\Multiple Equal Cost Paths.txt'
```

Ensure that you provide the correct path to the topology file after the script name. The topology file contains the network configuration that the code will use to simulate the Distance Vector algorithm.

2 Dynamic Vector Routing Simulation Code Explanation

2.1 Overview

This Python script implements a Distance Vector Routing (DVR) protocol simulation for a network of routers. Each router operates as an independent thread and communicates routing tables with its neighbors to find the shortest path to all destinations. The key functionalities and structure of the code are explained below.

2.2 Key Components

2.2.1 Router Class

The `Router` class models a router in the network. Each instance of the class represents a router with its routing table, neighbor relationships, and logging mechanism. The main components of this class are:

- **Attributes:**
  - `name`: Unique identifier for the router.
  - `neighbors`: A dictionary mapping neighbors to the cost of the link connecting them.
  - `routing_table`: A dictionary containing the cost of reaching each router, initialized with infinity except for itself (cost 0).
  - `shared_queue`: A thread-safe queue used for inter-router communication.
  - `convergence_flag`: A shared flag to indicate whether the router has converged.
  - `log_file`: A file to log the router's activity.
- **Methods:**
  - `run`: The main execution loop for the router thread. It iteratively updates the routing table and broadcasts its updates to neighbors until the network converges.
  - `broadcast_table`: Sends the current routing table to all neighbors.
  - `update_routing_table`: Processes updates received from neighbors and modifies the routing table based on the Distance Vector Algorithm.
  - `log`: Records messages to the log file, along with timestamps.
  - `log_routing_table`: Writes the current state of the routing table to the log.

2.2.2 Network Topology Parsing

The `parse_input` function reads the network topology from a file. It returns the number of routers, router names, and the links between them, which are represented as a list of tuples containing two router names and the link cost.

2.2.3 Network Creation

The `create_network` function constructs the network graph by:

- Creating a dictionary of neighbors for each router.
- Initializing a `Router` object for each router.
- Establishing a shared queue and convergence flag for all routers.

2.2.4 Simulation Execution

The `main` function coordinates the simulation by:

1. Reading the topology file specified in the command-line arguments.
2. Creating the network graph using the `create_network` function.
3. Starting each router's thread and waiting for all routers to converge.

2.3 Distance Vector Algorithm Details

The core of the simulation lies in the Distance Vector Algorithm:

- Each router periodically sends its routing table to all its neighbors.
- Upon receiving a routing table from a neighbor, the router calculates the potential cost to each destination via the neighbor.
- If a shorter path is found, the routing table is updated, and the updates are propagated to neighbors.
- Convergence is achieved when no further updates occur, and all routers have consistent routing tables.

2.4 Convergence Detection

Convergence is detected using the `convergence_flag`, a shared data structure marking whether each router has stopped making updates. The simulation ends when all routers are marked as converged.

2.5 Logging Mechanism

Each router logs its activities, including:

- Routing table updates.
- Receipt of distance vectors from neighbors.
- Calculation details for new routing costs.

The logs are stored in separate files, one for each router, for detailed debugging and analysis.

2.6 Conclusion

This implementation demonstrates a threaded simulation of the Distance Vector Routing protocol, showcasing the principles of distributed routing and convergence in network protocols.

3 Test Case 1 - Network Topology

3.1 Graph Description

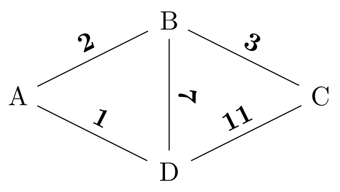
This graph consists of 4 nodes: **A**, **B**, **C**, and **D**. The edges and their respective weights are as follows:

- A-B: 2
- A-D: 1
- B-C: 3
- B-D: 7
- C-D: 11

The graph is undirected, meaning that the edges are bidirectional.

3.2 Graph Visualization

The graph can be represented as shown below:



3.3 Routing Table Initialization

Each node begins with an initial routing table with all costs set to infinity except for the node's own cost, which is set to 0. The cost to directly connected neighbors is initialized with the given edge weights. For instance:

- Router A: { A: 0, B: 2, D: 1 }
- Router B: { A: 2, B: 0, C: 3, D: 7 }
- Router C: { B: 3, C: 0, D: 11 }
- Router D: { A: 1, B: 7, C: 11, D: 0 }

3.4 Routing Updates

Each router broadcasts its routing table to its neighbors every iteration. The routing table is updated based on the Bellman-Ford algorithm, which updates the distances between routers to the shortest known paths.

4 Test Case 2 - Network Topology

4.1 Graph Description

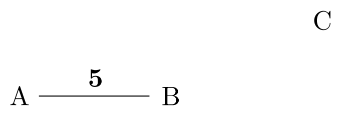
This graph consists of 3 nodes: **A**, **B**, and **C**. The edge and its respective weight are as follows:

- A-B: 5

This graph is undirected with no edges connecting node C to the other nodes.

4.2 Graph Visualization

The graph can be represented as shown below:



4.3 Routing Table Initialization

Each router begins with an initial routing table where all costs are set to infinity except for the node's own cost. For instance:

- Router A: { A: 0, B: 5 }
- Router B: { A: 5, B: 0 }
- Router C: { C: 0 } (No connections)

4.4 Routing Updates

Router A and B will send their routing tables to each other, but router C will not receive any updates due to lack of connections.

5 Test Case 4 - Network Topology

5.1 Graph Description

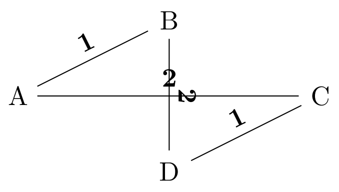
This graph consists of 4 nodes: **A**, **B**, **C**, and **D**. The edges and their respective weights are as follows:

- A-B: 1
- A-C: 2
- B-D: 2
- C-D: 1

The graph is undirected.

5.2 Graph Visualization

The graph can be represented as shown below:



5.3 Routing Table Initialization

Each router starts with a routing table where all costs are set to infinity except for the node's own cost. The initial routing tables are:

- Router A: { A: 0, B: 1, C: 2 }
- Router B: { A: 1, B: 0, D: 2 }
- Router C: { A: 2, C: 0, D: 1 }
- Router D: { B: 2, C: 1, D: 0 }

5.4 Routing Updates

Routers broadcast their routing tables to their neighbors. The routing tables will be updated using the Bellman-Ford algorithm. The update steps include adjusting the cost for each destination node based on the shortest path through neighboring routers.