BansilalRamnathAgarwalCharitableTrust's

VishwakarmaInstituteofTechnology, Pune-37

(AnautonomousInstituteofSavitribaiPhulePuneUniversity)

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Division	**AI-B**
Batch	**1**
GR-no	**12320050**
Rollno	**75**
Name	**Tanpure Rushikesh**

Lab Assignment: 10

Problem Statement : Write a program to implement:

a. Network Routing: Shortest path routing, AODV

CODE:

```
import java.util.*;

// Represents a node in the network
class Node {
    private int id;
    private Map<Node, Integer> neighbors;

    public Node(int id) {
        this.id = id;
        this.neighbors = new HashMap<>();
    }

    public int getId() {
        return id;
    }

    public void addNeighbor(Node neighbor, int distance) {
        neighbors.put(neighbor, distance);
    }

    public Map<Node, Integer> getNeighbors() {
        return neighbors;
    }
}

// Represents a network with nodes and connections between them
class Network {
    private Map<Integer, Node> nodes;

    public Network() {
        this.nodes = new HashMap<>();
    }

    public void addNode(Node node) {
        nodes.put(node.getId(), node);
    }

    public Node getNode(int id) {
```

```
        return nodes.get(id);
    }

    // AODV routing algorithm to find the shortest path
    public List<Node> shortestPath(int sourceId, int destinationId) {
        Node source = getNode(sourceId);
        Node destination = getNode(destinationId);

        // Initialize distances and previous node
        Map<Node, Integer> distance = new HashMap<>();
        Map<Node, Node> previous = new HashMap<>();
        for (Node node : nodes.values()) {
            distance.put(node, Integer.MAX_VALUE);
            previous.put(node, null);
        }
        distance.put(source, 0);

        // Priority queue for nodes to be processed
        PriorityQueue<Node> queue = new
PriorityQueue<>(Comparator.comparingInt(distance::get));
        queue.add(source);

        // Dijkstra's algorithm
        while (!queue.isEmpty()) {
            Node current = queue.poll();
            if (current == destination) break;

            for (Map.Entry<Node, Integer> entry :
current.getNeighbors().entrySet()) {
                Node neighbor = entry.getKey();
                int altDistance = distance.get(current) + entry.getValue();
                if (altDistance < distance.get(neighbor)) {
                    distance.put(neighbor, altDistance);
                    previous.put(neighbor, current);
                    queue.add(neighbor);
                }
            }
        }

        // Reconstruct the shortest path
        List<Node> path = new ArrayList<>();
        Node current = destination;
        while (previous.get(current) != null) {
            path.add(0, current);
            current = previous.get(current);
        }
        if (path.isEmpty() || path.get(0) != source) return null; // No path found
        path.add(0, source);
        return path;
    }
}

public class AODV {
```

```
    public static void main(String[] args) {
        // Create a network
        Network network = new Network();
        Node node1 = new Node(1);
        Node node2 = new Node(2);
        Node node3 = new Node(3);
        Node node4 = new Node(4);

        node1.addNeighbor(node2, 1);
        node1.addNeighbor(node3, 2);
        node2.addNeighbor(node4, 3);
        node3.addNeighbor(node4, 1);

        network.addNode(node1);
        network.addNode(node2);
        network.addNode(node3);
        network.addNode(node4);

        // Find the shortest path between nodes
        int sourceId = 1;
        int destinationId = 2;
        List<Node> shortestPath = network.shortestPath(sourceId, destinationId);

        // Print the shortest path
        if (shortestPath != null) {
            System.out.println("Shortest path from node " + sourceId + " to node "
+ destinationId + ":");
            for (Node node : shortestPath) {
                System.out.print(node.getId() + " ");
            }
        } else {
            System.out.println("No path found from node " + sourceId + " to node "
+ destinationId);
        }
    }
}
```

OutPut:

```
PS C:\Users\rames\Documents\CN\practical10> c:; cd 'c:\Users\rames\Documents\(
-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Use
8f06a60a97b161d0e\redhat.java\jdt_ws\practical10_1968cec9\bin' 'AODV'
Shortest path found from node 1 to node 4
1 3 4
```