Ex No: 9

Date:

WRITE A PROGRAM TO FIND THE SHORTEST PATH BETWEEN VERTICES USING OPEN SHORTEST PATH FIRST ROUTING ALGORITHM.

Aim:

To simulate the process of finding the shortest path between vertices in a network using the Open Shortest Path First (OSPF) algorithm.

Theory:

After the completion of this experiment, students will be able to:

- Understand the OSPF algorithm and its role in routing within networks.
- Simulate the process of finding the shortest path using Dijkstra's algorithm, the core of OSPF.
- Analyze how the shortest path is determined in a network of interconnected nodes.
- Interpret the relationship between link costs and routing decisions in maintaining optimal network performance.
- Understand the importance of link-state routing protocols in managing dynamic network topologies.

The OSPF algorithm is a link-state routing protocol used in Internet Protocol (IP) networks. It uses Dijkstra's algorithm to compute the shortest path tree for each route. The OSPF algorithm works by finding the shortest path between a given source node and all other nodes in the network based on link costs.

Algorithm:

1. **Initialization:**
    - Start with the source node and set the distance to itself as 0.
    - Set the distance to all other nodes as infinity.
    - Mark all nodes as unvisited.
2. **Selection:**
    - Choose the unvisited node with the smallest tentative distance and mark it as the current node.
3. **Distance Calculation:**
    - For the current node, consider all its unvisited neighbors.
    - Calculate the tentative distance from the source node to each neighbor.
    - If the calculated distance is less than the known distance, update the shortest distance to that neighbor.

4. **Mark as Visited:**
    - Once all neighbors of the current node have been considered, mark the current node as visited.
5. **Repeat:**
    - Repeat steps 2 to 4 until all nodes have been visited or the smallest tentative distance among the unvisited nodes is infinity.
6. **Termination:**
    - The algorithm terminates when all nodes have been visited, and the shortest path from the source node to all other nodes is determined.

Example:

Consider a network of 6 nodes with the following edges and costs:

- (0, 1) with cost 4
- (0, 2) with cost 3
- (1, 2) with cost 1
- (1, 3) with cost 2
- (2, 3) with cost 4
- (3, 4) with cost 2
- (4, 5) with cost 6

Steps:

- Initialize distances from the source node (e.g., Node 0) to all other nodes.
- For each node, find the shortest path to its neighbors and update the distances.
- Repeat until all nodes have been visited.

Example Calculation:

For the source node 0:

- Distance to 1: 4
- Distance to 2: 3
- Distance to 3: 6
- Distance to 4: 8
- Distance to 5: 14

These distances represent the shortest path from node 0 to all other nodes.

Program:

```
import java.util.*;

class OSPF {
    private int vertices;
    private LinkedList<Edge>[] adjList;

    // Edge class to represent a weighted edge in the graph
    static class Edge {
        int target, weight;

        Edge(int target, int weight) {
            this.target = target;
            this.weight = weight;
        }
    }

    // Constructor to initialize the graph with a given number of vertices
    OSPF(int vertices) { // Changed from Graph to OSPF
        this.vertices = vertices;
        adjList = new LinkedList[vertices];
        for (int i = 0; i < vertices; i++) {
            adjList[i] = new LinkedList<>();
        }
    }

    // Method to add an edge to the graph
    void addEdge(int source, int target, int weight) {
        adjList[source].add(new Edge(target, weight));
        adjList[target].add(new Edge(source, weight)); // for undirected graph
    }

    // Method to find the shortest path using Dijkstra's algorithm
    void dijkstra(int source, int target) {
        PriorityQueue<Edge> priorityQueue = new PriorityQueue<>(vertices,
Comparator.comparingInt(e -> e.weight));
        int[] distances = new int[vertices];
        Arrays.fill(distances, Integer.MAX_VALUE);
        distances[source] = 0;
        priorityQueue.add(new Edge(source, 0));

        while (!priorityQueue.isEmpty()) {
            int u = priorityQueue.poll().target;

            for (Edge edge : adjList[u]) {
```

```
                int v = edge.target;
                int weight = edge.weight;

                if (distances[u] + weight < distances[v]) {
                    distances[v] = distances[u] + weight;
                    priorityQueue.add(new Edge(v, distances[v]));
                }
            }
        }

        // Print the shortest path to the specified target node
        System.out.println("Shortest path from vertex " + source + " to vertex " + target
+ " is " + distances[target]);
    }

    // Main method to demonstrate the program
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        OSPF graph = new OSPF(6); // Changed from Graph to OSPF
        graph.addEdge(0, 1, 4);
        graph.addEdge(0, 2, 3);
        graph.addEdge(1, 2, 1);
        graph.addEdge(1, 3, 2);
        graph.addEdge(2, 3, 4);
        graph.addEdge(3, 4, 2);
        graph.addEdge(4, 5, 6);

        System.out.print("Enter the source vertex: ");
        int source = scanner.nextInt();

        System.out.print("Enter the target vertex: ");
        int target = scanner.nextInt();

        graph.dijkstra(source, target);
    }
}
```

Sample Output:

Enter the source vertex: 0

Enter the target vertex: 4

Shortest path from vertex 0 to vertex 4 is 8

Screenshot of output:

Result:

The OSPF algorithm effectively calculates the shortest path between nodes, ensuring optimal routing decisions in IP networks. This simulation demonstrates how the algorithm dynamically adjusts to find the most efficient routes, which is crucial in maintaining network performance.