Task 1:

Dijkstra's Shortest Path Finder

Code Dijkstra's algorithm to find the shortest path from a start node to every other node in a weighted graph with positive weights.

ANS:

```
package com.Day15;
import java.util.*;
public class Dijkstra {
  // Class to represent a graph edge
  static class Edge {
     int target;
     int weight;
     Edge(int target, int weight) {
        this.target = target;
        this.weight = weight;
     }
  }
  // Class to represent a node in the priority queue
  static class Node implements Comparable<Node> {
     int vertex;
     int distance;
     Node(int vertex, int distance) {
        this.vertex = vertex;
        this.distance = distance;
     }
     @Override
     public int compareTo(Node other) {
        return Integer.compare(this.distance, other.distance);
     }
  }
  // Method to find the shortest path from a start node to all other
nodes
  public static int[] dijkstra(List<List<Edge>> graph, int start) {
     int numVertices = graph.size();
     int[] distances = new int[numVertices];
     boolean[] visited = new boolean[numVertices];
     PriorityQueue<Node> pq = new PriorityQueue<>();
     // Initialize distances array with infinity
```

```
        Arrays.fill(distances, Integer.MAX_VALUE);
        distances[start] = 0;
        // Add the start node to the priority queue
        pq.add(new Node(start, 0));
        while (!pq.isEmpty()) {
            // Extract the node with the smallest distance
            Node currentNode = pq.poll();
            int currentVertex = currentNode.vertex;
            // Skip if the node is already visited
            if (visited[currentVertex]) continue;
            visited[currentVertex] = true;
            // Process all adjacent nodes
            for (Edge edge : graph.get(currentVertex)) {
                int neighbor = edge.target;
                int newDist = distances[currentVertex] + edge.weight;
                // If a shorter path to the neighbor is found
                if (newDist < distances[neighbor]) {
                    distances[neighbor] = newDist;
                    pq.add(new Node(neighbor, newDist));
                }
            }
        }
        return distances;
    }
    public static void main(String[] args) {
        // Example graph represented as an adjacency list
        int numVertices = 5;
        List<List<Edge>> graph = new ArrayList<>();
        // Initialize graph with empty lists
        for (int i = 0; i < numVertices; i++) {
            graph.add(new ArrayList<>());
        }
        // Add edges to the graph (example graph)
        graph.get(0).add(new Edge(1, 10));
        graph.get(0).add(new Edge(4, 5));
        graph.get(1).add(new Edge(2, 1));
        graph.get(1).add(new Edge(4, 2));
```

```
        graph.get(2).add(new Edge(3, 4));
        graph.get(3).add(new Edge(0, 7));
        graph.get(3).add(new Edge(2, 6));
        graph.get(4).add(new Edge(1, 3));
        graph.get(4).add(new Edge(2, 9));
        graph.get(4).add(new Edge(3, 2));
        // Run Dijkstra's algorithm from start node 0
        int startNode = 0;
        int[] distances = dijkstra(graph, startNode);
        // Print the shortest distances from the start node to all other
nodes
        System.out.println("Shortest distances from node " + startNode
+ ":");
        for (int i = 0; i < distances.length; i++) {
            System.out.println("To node " + i + " : " + distances[i]);
        }
    }
}
```

OUTPUT:

```
Shortest distances from node 0:
To node 0 : 0
To node 1 : 8
To node 2 : 9
To node 3 : 7
To node 4 : 5
```