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.

Code: -

```
package com.wipro.assignment;
import java.util.*;

class Graph {
    private int numVertices;
    private Map<Integer, List<Edge>>
adjacencyMap;

    public Graph(int numVertices) {
        this.numVertices =
numVertices;
        adjacencyMap = new
HashMap<>();
        for (int i = 0; i <
numVertices; i++) {
            adjacencyMap.put(i, new
ArrayList<>());
        }
```

```
    }

    public void addEdge(int source,
int destination, int weight) {

adjacencyMap.get(source).add(new
Edge(destination, weight));
    }

    public int[] dijkstra(int source)
{
        PriorityQueue<Node>
priorityQueue = new
PriorityQueue<>(numVertices,
Comparator.comparingInt(n ->
n.weight));
        int[] distances = new
int[numVertices];
        Arrays.fill(distances,
Integer.MAX_VALUE);
        boolean[] visited = new
boolean[numVertices];

        priorityQueue.add(new
Node(source, 0));
        distances[source] = 0;
```

```
        while
(!priorityQueue.isEmpty()) {
            int currentNode =
priorityQueue.poll().vertex;
            visited[currentNode] =
true;
            List<Edge> neighbors =
adjacencyMap.get(currentNode);
            for (Edge neighbor :
neighbors) {
                int neighborVertex =
neighbor.destination;
                int newDistance =
distances[currentNode] +
neighbor.weight;
                if
(!visited[neighborVertex] &&
newDistance <
distances[neighborVertex]) {

distances[neighborVertex] =
newDistance;

priorityQueue.add(new
Node(neighborVertex, newDistance));
                }
            }
```

```
        }
        return distances;
    }

    private static class Node {
        int vertex;
        int weight;

        public Node(int vertex, int
weight) {
            this.vertex = vertex;
            this.weight = weight;
        }
    }

    private static class Edge {
        int destination;
        int weight;

        public Edge(int destination,
int weight) {
            this.destination =
destination;
            this.weight = weight;
        }
    }
}
```

```
public class Main {
    public static void main(String[]
args) {
        int numVertices = 9;
        Graph graph = new
Graph(numVertices);
        graph.addEdge(0, 1, 4);
        graph.addEdge(0, 7, 8);
        graph.addEdge(1, 2, 8);
        graph.addEdge(1, 7, 11);
        graph.addEdge(2, 3, 7);
        graph.addEdge(2, 8, 2);
        graph.addEdge(2, 5, 4);
        graph.addEdge(3, 4, 9);
        graph.addEdge(3, 5, 14);
        graph.addEdge(4, 5, 10);
        graph.addEdge(5, 6, 2);
        graph.addEdge(6, 7, 1);
        graph.addEdge(6, 8, 6);
        graph.addEdge(7, 8, 7);

        int source = 0;
        int[] distances =
graph.dijkstra(source);
```

```
        System.out.println("Shortest
distances from source vertex " +
source + ":");
        for (int i = 0; i <
numVertices; i++) {

System.out.println("Vertex " + i + ":
" + distances[i]);
        }
    }
}
```

Output: -