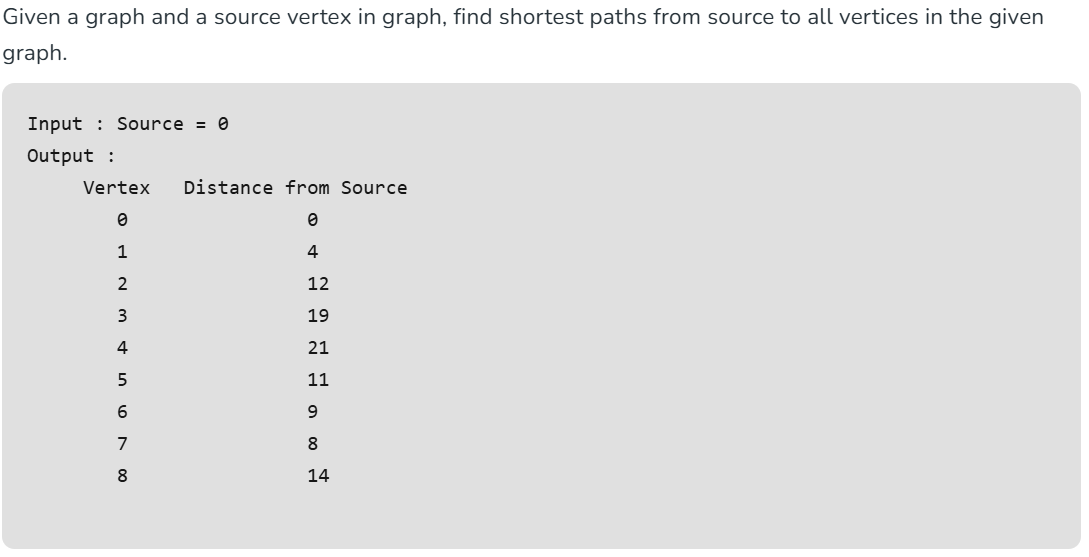
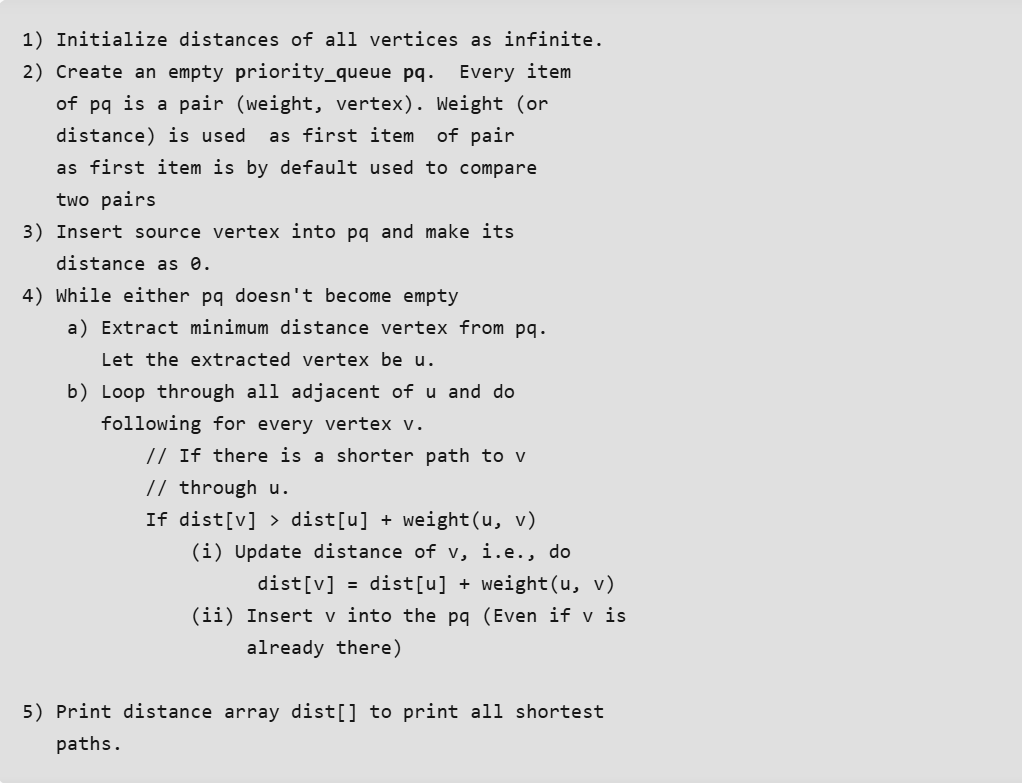
**Dijkstra’s Shortest Path Algorithm using priority\_queue**





import java.util.\*;

// Class to represent a graph and implement Dijkstra's

// shortest path algorithm

class Graph {

private int V; // Number of vertices

private List<int[]>[] adj; // Adjacency list to store

// graph edges

// Inner class to represent a pair of vertex and its

// weight

class iPair implements Comparable<iPair> {

int vertex, weight;

iPair(int v, int w)

{

vertex = v;

weight = w;

}

// Comparison method for priority queue

public int compareTo(iPair other)

{

return Integer.compare(this.weight,

other.weight);

}

}

// Constructor to initialize the graph

Graph(int V)

{

this.V = V;

adj = new ArrayList[V];

for (int i = 0; i < V; ++i)

adj[i] = new ArrayList<>();

}

// Method to add an edge to the graph

void addEdge(int u, int v, int w)

{

adj[u].add(new int[] { v, w });

adj[v].add(new int[] { u, w });

}

// Method to find the shortest paths from source vertex

// to all other vertices

void shortestPath(int src)

{

PriorityQueue<iPair> pq = new PriorityQueue<>();

int[] dist = new int[V];

Arrays.fill(dist, Integer.MAX\_VALUE);

pq.add(new iPair(src, 0));

dist[src] = 0;

// Dijkstra's algorithm

while (!pq.isEmpty()) {

int u = pq.poll().vertex;

for (int[] neighbor : adj[u]) {

int v = neighbor[0];

int weight = neighbor[1];

// Relaxation step

if (dist[v] > dist[u] + weight) {

dist[v] = dist[u] + weight;

pq.add(new iPair(v, dist[v]));

}

}

}

// Print shortest distances from source

System.out.println("Vertex Distance from Source");

for (int i = 0; i < V; ++i)

System.out.println(i + "\t\t" + dist[i]);

}

}

// Main class containing the main method to test the graph

// and Dijkstra's algorithm

public class GFG {

public static void main(String[] args)

{

int V = 9;

Graph g = new Graph(V);

// Adding edges to create the graph

g.addEdge(0, 1, 4);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 8);

g.addEdge(1, 7, 11);

g.addEdge(2, 3, 7);

g.addEdge(2, 8, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 4, 9);

g.addEdge(3, 5, 14);

g.addEdge(4, 5, 10);

g.addEdge(5, 6, 2);

g.addEdge(6, 7, 1);

g.addEdge(6, 8, 6);

g.addEdge(7, 8, 7);

// Finding and printing the shortest paths from

// source vertex 0

g.shortestPath(0);

}

}