Experiment No : 06

Roll No : 24141005

Title: Shortest paths to other vertices using Dijkstra's algorithm from a given vertex in a Weighted connected graph.

Algorithm:

1. Initialize:

   For each vertex v in Graph:

   distance[v] ← ∞

   visited[v] ← false

   distance[source] ← 0

2. Repeat (V - 1) times:

   a. Select the vertex u with the smallest distance[u]

   that has not been visited yet

   b. Mark u as visited

   c. For each neighbor v of u:

   if (visited[v] == false) AND (edge u→v exists) AND

   (distance[u] + weight(u, v) < distance[v]) then

   distance[v] ← distance[u] + weight(u, v)

3. End Repeat

4. Print the shortest distance from source to all vertices

Program:

```
import java.util.*;
public class Dijkstra
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter number of vertices: ");
      int V = sc.nextInt();
      int[][] graph = new int[V][V];
```

```
        System.out.println("Enter the adjacency matrix (enter 0 if no edge):");
        for (int i = 0; i < V; i++)
            {
            for (int j = 0; j < V; j++)
                {
                graph[i][j] = sc.nextInt();
            }
        }
        System.out.print("Enter source vertex (0 to " + (V - 1) + "): ");
        int source = sc.nextInt();
        dijkstra(graph, source);
        sc.close();
    }
    public static void dijkstra(int[][] graph, int source)
    {
        int V = graph.length;
        int[] dist = new int[V];
        boolean[] visited = new boolean[V];
        Arrays.fill(dist, Integer.MAX_VALUE);
        dist[source] = 0;

        for (int count = 0; count < V - 1; count++)
            {
            int u = minDistance(dist, visited);
            visited[u] = true;
            for (int v = 0; v < V; v++)
                {
                if (!visited[v] && graph[u][v] != 0 && dist[u] != Integer.MAX_VALUE &&
dist[u] + graph[u][v] < dist[v])
                    {
                    dist[v] = dist[u] + graph[u][v];
                }
```

```
        }
      }
      System.out.println("\nVertex\tShortest Distance from Source " + source);
      for (int i = 0; i < V; i++)
        {
        System.out.println(i + "\t\t" + dist[i]);
      }
    }
    public static int minDistance(int[] dist, boolean[] visited)
   {
      int min = Integer.MAX_VALUE, minIndex = -1;
      for (int v = 0; v < dist.length; v++)
         {
        if (!visited[v] && dist[v] <= min)
           {
           min = dist[v];
           minIndex = v;
        }
      }
      return minIndex;
    }
}
```

OUTPUT:

```
C:\Users\diksh\OneDrive\Desktop\Roshani>java Dijkstra
Enter number of vertices: 4
Enter the adjacency matrix (enter 0 if no edge):
2 0 6 4
1 3 8 0
3 9 3 1
7 6 0 2
Enter source vertex (0 to 3): 3

Vertex  Shortest Distance from Source 3
0               7
1               6
2               13
3               0
```

Applications:

- Navigation Systems (e.g., Google Maps):

Finding the shortest or fastest route between two locations, considering factors like distance, traffic conditions, and road types (represented as edge weights).

- Network Routing:

Determining the optimal path for data packets to travel across a network, minimizing latency or hop count. Protocols like OSPF (Open Shortest Path First) utilize this principle.

- Telephone Networks:

Establishing connections and finding the most efficient path for calls or data transmission.

- Geographic Information Systems (GIS):

Analyzing spatial data, such as finding the shortest path between geographical points or optimizing delivery routes.

- Resource Allocation:

Optimizing resource distribution in various systems by finding the shortest path to deliver resources to different nodes.