Assignment 2

Name: Pratham Chintawar
Roll No: 16
Batch: 2

Statement: Dijkstra Algorithm for Link State Routing Protocol in Python

Objective:
To find the shortest path using Dijkstra's Algorithm for Link State Routing Protocol (OSPF) in a given network topology using Python.

Theory

Link State Routing Protocols are dynamic routing protocols used in packet-switched networks to determine the best path for data packets. The most widely used link-state protocol is OSPF (Open Shortest Path First), which operates within an Autonomous System (AS). OSPF maintains a link-state database of the entire network topology and computes the shortest path using Dijkstra's Algorithm.

Dijkstra's algorithm is a graph search algorithm that finds the shortest path between a source node and all other nodes in a weighted graph. It is optimal and guarantees the least-cost path in a network.

Steps in Dijkstra's Algorithm:

1. Initialize distances from source to all nodes as infinity, except the source (distance 0).
2. Maintain a set of visited and unvisited nodes.
3. Select the unvisited node with the smallest known distance.
4. Update distances to adjacent nodes if a shorter path is found.
5. Repeat until all nodes are visited.

Program

```
import sys

INF = sys.maxsize

V = 7

vertex_names = ["A", "B", "C", "D", "E", "F", "G"]

def min_distance(dist, visited):

    min_val = INF

    min_index = -1

    for v in range(V):
```

```
        if not visited[v] and dist[v] <= min_val:
            min_val = dist[v]
            min_index = v
    return min_index

def dijkstra(graph, source):
    dist = [INF] * V
    visited = [False] * V
    path = [vertex_names[source]] * V

    print("Initial Values")
    print("Visited:", visited)
    print("Distances:", dist)
    print("Path:", path, "\n")

    dist[source] = 0

    for count in range(V - 1):
        u = min_distance(dist, visited)
        visited[u] = True

        print(f"Step {count + 1}:")
        print("Visited:", visited)
        print("Distances:", dist)

        for v in range(V):
            if (not visited[v] and graph[u][v] != 0 and dist[u] != INF and
                    dist[u] + graph[u][v] < dist[v]):
                dist[v] = dist[u] + graph[u][v]
```

```
            path[v] = path[u] + " -> " + vertex_names[v]

        print("Update distance and path for neighboring vertices")
        print("Distances:", dist)
        print("Path:", path, "\n")

    print("\nVertex\tDistance\tPath")
    for i in range(V):
        if i != source:
            print(f"{vertex_names[source]} -> {vertex_names[i]}\t{dist[i]}\t\t{path[i]}")

if __name__ == "__main__":
    graph = [
        [0, 2, 5, 0, 0, 0, 0],
        [2, 0, 1, 2, 0, 0, 0],
        [5, 1, 0, 3, 1, 4, 0],
        [0, 2, 3, 0, 2, 0, 3],
        [0, 0, 1, 2, 0, 0, 5],
        [0, 0, 4, 0, 0, 0, 2],
        [0, 0, 0, 3, 5, 2, 0]
    ]

    dijkstra(graph, 0)
```

Output:

```
Initial Values
Visited: [False, False, False, False, False, False, False]
Distances: [9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Path: ['A', 'A', 'A', 'A', 'A', 'A', 'A']

Step 1:
Visited: [True, False, False, False, False, False, False]
Distances: [0, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Update distance and path for neighboring vertices
Distances: [0, 2, 5, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Path: ['A', 'A -> B', 'A -> C', 'A', 'A', 'A', 'A']

Step 2:
Visited: [True, True, False, False, False, False, False]
Distances: [0, 2, 5, 9223372036854775807, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Update distance and path for neighboring vertices
Distances: [0, 2, 3, 4, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Path: ['A', 'A -> B', 'A -> B -> C', 'A -> B -> D', 'A', 'A', 'A']

Step 3:
Visited: [True, True, True, False, False, False, False]
Distances: [0, 2, 3, 4, 9223372036854775807, 9223372036854775807, 9223372036854775807]
Update distance and path for neighboring vertices
Distances: [0, 2, 3, 4, 4, 7, 9223372036854775807]
Path: ['A', 'A -> B', 'A -> B -> C', 'A -> B -> D', 'A -> B -> C -> E', 'A -> B -> C -> F', 'A']

Step 4:
Visited: [True, True, True, False, True, False, False]
Distances: [0, 2, 3, 4, 4, 7, 9223372036854775807]
Update distance and path for neighboring vertices
Distances: [0, 2, 3, 4, 4, 7, 9]
Path: ['A', 'A -> B', 'A -> B -> C', 'A -> B -> D', 'A -> B -> C -> E', 'A -> B -> C -> F', 'A -> B -> C -> E -> G']

Step 5:
Visited: [True, True, True, True, True, False, False]
Distances: [0, 2, 3, 4, 4, 7, 9]
Update distance and path for neighboring vertices
Distances: [0, 2, 3, 4, 4, 7, 7]
Path: ['A', 'A -> B', 'A -> B -> C', 'A -> B -> D', 'A -> B -> C -> E', 'A -> B -> C -> F', 'A -> B -> D -> G']

Step 6:
Visited: [True, True, True, True, True, False, True]
Distances: [0, 2, 3, 4, 4, 7, 7]
Update distance and path for neighboring vertices
Distances: [0, 2, 3, 4, 4, 7, 7]
Path: ['A', 'A -> B', 'A -> B -> C', 'A -> B -> D', 'A -> B -> C -> E', 'A -> B -> C -> F', 'A -> B -> D -> G']


Vertex  Distance        Path
A -> B  2               A -> B
A -> C  3               A -> B -> C
A -> D  4               A -> B -> D
A -> E  4               A -> B -> C -> E
A -> F  7               A -> B -> C -> F
A -> G  7               A -> B -> D -> G
```

Conclusion

The shortest path from the source node to all other nodes in the network was successfully determined using Dijkstra's Algorithm implemented in Python.