EXPERIMENT 7

PRAKASH P. BISWAS
9947
COMPS-B

Complexity: The Complexity of Dijkstra's Algorithm is O(n^2)

Input:

```
V = 100
INFI = 999

def dijkstra(graph, numVertices, source):
    dist = [INFI] * numVertices
    visited = [False] * numVertices
    dist[source] = 0

    for _ in range(numVertices):
        minDist = INFI
        minVertex = -1

        # Find the vertex with the minimum distance that is not yet
visited
        for v in range(numVertices):
            if not visited[v] and dist[v] < minDist:
                minDist = dist[v]
                minVertex = v

        visited[minVertex] = True

        # Update the distances to the neighboring vertices
        for v in range(numVertices):
            if not visited[v] and graph[minVertex][v] != INFI:
                newDist = dist[minVertex] + graph[minVertex][v]
                if newDist < dist[v]:
                    dist[v] = newDist

    print("Vertex\tDistance from Source")
    for i in range(numVertices):
        print(f"{i}\t{dist[i]}")

def main():
    adjMat = [[INFI] * V for _ in range(V)]
    vertices = int(input("Enter the number of vertices: "))
    edges = int(input("Enter the number of edges: "))

    # Input the edges and weights into the adjacency matrix
    for _ in range(edges):
        source = int(input("Enter the source for edge: "))
        destination = int(input("Enter the destination for edge: "))
        weight = int(input("Enter the weight for edge: "))
        adjMat[source][destination] = weight
        adjMat[destination][source] = weight  # Assuming the graph is
undirected

    # Print the adjacency matrix
```

```
    print("Adjacency Matrix:")
    for row in adjMat:
        for w in row:
            if w == INFI:
                print(".", end=" ")
            else:
                print(w, end=" ")
        print()

    source_vertex = 0
    dijkstra(adjMat, vertices, source_vertex)

if __name__ == "__main__":
    main()
```

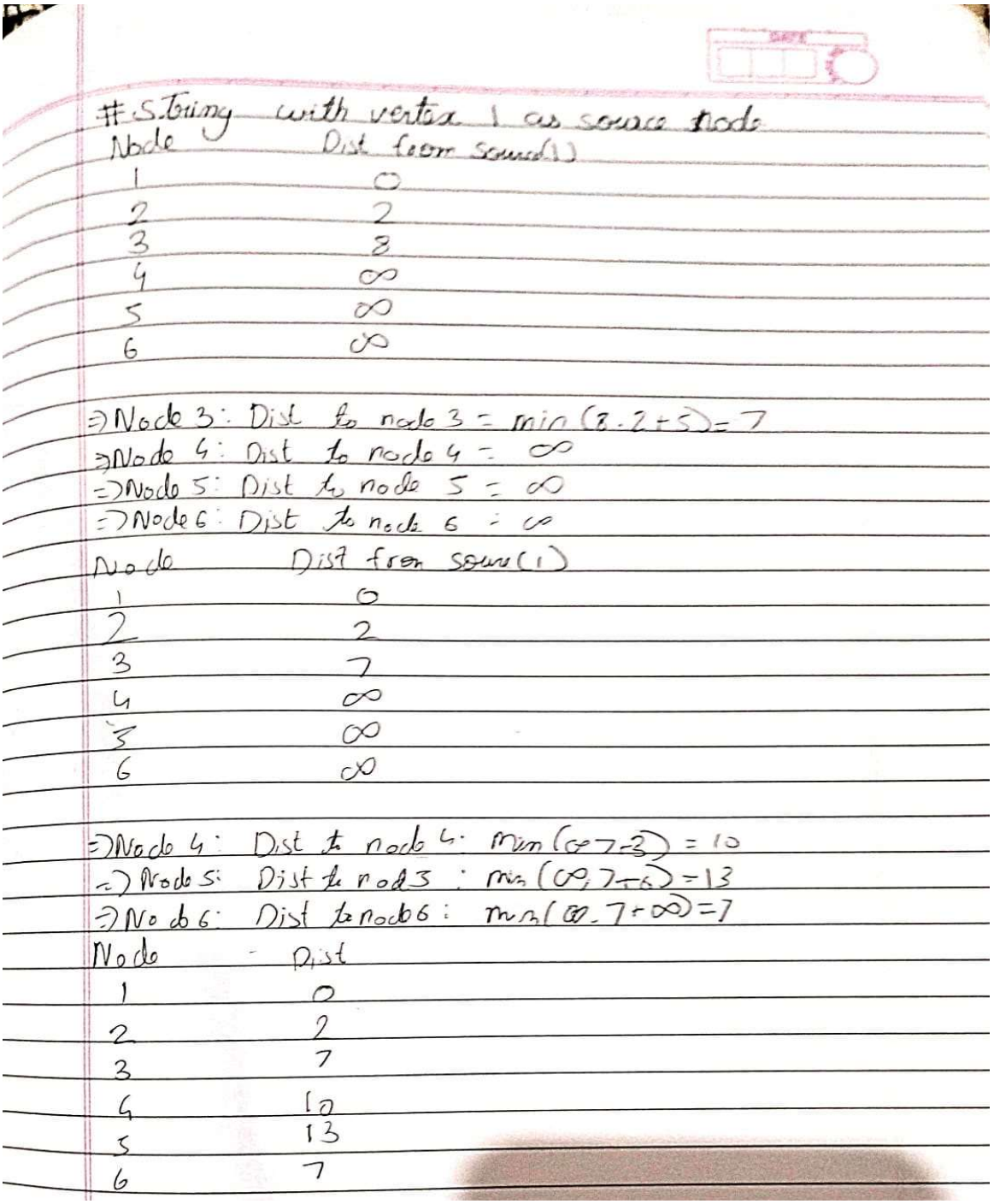
Output:

```
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the source for edge: 0
Enter the destination for edge: 1
Enter the weight for edge: 2
Enter the source for edge: 0
Enter the destination for edge: 2
Enter the weight for edge: 4
Enter the source for edge: 1
Enter the destination for edge: 2
Enter the weight for edge: 1
Enter the source for edge: 1
Enter the destination for edge: 3
Enter the weight for edge: 7
Enter the source for edge: 2
Enter the destination for edge: 3
Enter the weight for edge: 3
Enter the source for edge: 2
Enter the destination for edge: 4
Enter the weight for edge: 5
Enter the source for edge: 3
Enter the destination for edge: 4
Enter the weight for edge: 2

Adjacency Matrix:
0 2 4 . .
2 . 1 7 .
4 1 . 3 5
. 7 3 . 2
. . 5 2 .

Vertex  Distance from Source
0   0
1   2
2   3
3   6
4   8
```

Postlab:



#S.Trimy with vertex 1 as source node

Node	Dist from source(1)
1	0
2	2
3	8
4	∞
5	∞
6	∞

⇒Node 3: Dist to node 3 = min(8, 2+5) = 7
⇒Node 4: Dist to node 4 = ∞
⇒Node 5: Dist to node 5 = ∞
⇒Node 6: Dist to node 6 = ∞

Node	Dist from source(1)
1	0
2	2
3	7
4	∞
5	∞
6	∞

⇒Node 4: Dist to node 4: min(∞, 7+3) = 10
⇒Node 5: Dist to node 5: min(∞, 7+6) = 13
⇒Node 6: Dist to node 6: min(∞, 7+∞) = 7

Node	Dist
1	0
2	2
3	7
4	10
5	13
6	7

: Shortest distance	Shortest path
Node 1 = 0	1 → 1
Node 2 = 2	1 → 2
Node 3 = 7	1 → 2 → 3
Node 4 = 10	1 → 2 → 3 → 4
Node 5 = 13	1 → 2 → 3 → 4
Node 6 = 7	1 → 2 → 3 → 4 → 6
	1 → 2 → 3 → 4 → 6 → 5