Ex.No.8: From a given vertex in a weighted connected graph, develop a program to find the shortest paths to other vertices using Dijkstra's algorithm.

```
# Dijkstra's Algorithm in Python

import sys

# Providing the graph

vertices = [[0, 0, 1, 1, 0, 0, 0],

          [0, 0, 1, 0, 0, 1, 0],

          [1, 1, 0, 1, 1, 0, 0],

          [1, 0, 1, 0, 0, 0, 1],

          [0, 0, 1, 0, 0, 1, 0],

          [0, 1, 0, 0, 1, 0, 1],

          [0, 0, 0, 1, 0, 1, 0]]


edges = [[0, 0, 1, 2, 0, 0, 0],

        [0, 0, 2, 0, 0, 3, 0],

        [1, 2, 0, 1, 3, 0, 0],

        [2, 0, 1, 0, 0, 0, 1],

        [0, 0, 3, 0, 0, 2, 0],

        [0, 3, 0, 0, 2, 0, 1],

        [0, 0, 0, 1, 0, 1, 0]]


# Find which vertex is to be visited next

def to_be_visited():

    global visited_and_distance

    v = -10

    for index in range(num_of_vertices):

        if visited_and_distance[index][0] == 0 \

            and (v < 0 or visited_and_distance[index][1] <=

                visited_and_distance[v][1]):
```

```
            v = index
    return v



num_of_vertices = len(vertices[0])
visited_and_distance = [[0, 0]]
for i in range(num_of_vertices-1):
    visited_and_distance.append([0, sys.maxsize])
for vertex in range(num_of_vertices):
    # Find next vertex to be visited
    to_visit = to_be_visited()
    for neighbor_index in range(num_of_vertices):
        # Updating new distances
        if vertices[to_visit][neighbor_index] == 1 and \
                visited_and_distance[neighbor_index][0] == 0:
            new_distance = visited_and_distance[to_visit][1] \
                + edges[to_visit][neighbor_index]
            if visited_and_distance[neighbor_index][1] > new_distance:
                visited_and_distance[neighbor_index][1] = new_distance


        visited_and_distance[to_visit][0] = 1
i = 0
# Printing the distance
for distance in visited_and_distance:
    print("Distance of ", chr(ord('a') + i),
          " from source vertex: ", distance[1])
    i = i + 1
```