LAB#18

Example#1: Write a program to compute the shortest path from a specific node to all the other nodes by using the Dijkstra's Algorithm:

Solution:

```
from collections import deque

class MyGraph:
    def __init__(self, num_vertices):
        self.num_vertices = num_vertices
        self.adj_list = {v: [] for v in range(num_vertices)}

    def add_edge(self, u, v, weight):
        if 0 <= u < self.num_vertices and 0 <= v < self.num_vertices:
            self.adj_list[u].append((v, weight))
        else:
            print('Invalid vertex')

    def print_adj_list(self):
        for v, neighbors in self.adj_list.items():
            print(v, ':', neighbors)

```

```
    def dijkstra_shortest_path(self, source):
        distances = [float('inf')] * self.num_vertices
        distances[source] = 0

        queue = deque([source])

        while queue:
            current_vertex = queue.popleft()

            for neighbor, weight in self.adj_list[current_vertex]:
                current_distance = distances[current_vertex] + weight

                if current_distance < distances[neighbor]:
                    distances[neighbor] = current_distance
                    queue.append(neighbor)

        return distances
```

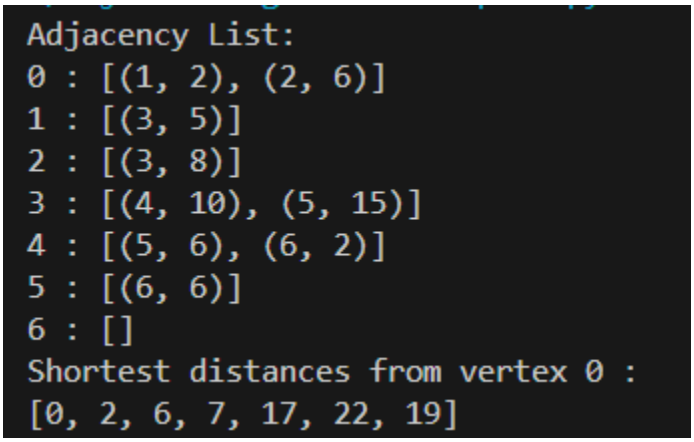
```
# Create a weighted graph instance
g = MyGraph(7)

# Add weighted edges
g.add_edge(0, 1, 2)
g.add_edge(0, 2, 6)
g.add_edge(1, 3, 5)
g.add_edge(2,3, 8)
g.add_edge(3,4,10)
g.add_edge(3,5,15)
g.add_edge(4,5,6)
g.add_edge(5,6, 6)
g.add_edge(4, 6, 2)


print("Adjacency List:")
g.print_adj_list()


source_vertex = 0
shortest_distances = g.dijkstra_shortest_path(source_vertex)
print("Shortest distances from vertex", source_vertex, ":")
print(shortest_distances)
```

Output:



Class Assignment

Q: Repeat the program above, considering the source as 1, and then provide an explanation of the code.