198. Given a graph represented by an adjacency matrix, implement Dijkstra's Algorithm to find the shortest path from a given source vertex to all other vertices in the graph. The graph is represented as an adjacency matrix where graph[i][j] denote the weight of the edge from vertex i to vertex j. If there is no edge between vertices i and j, the value is Infinity (or a very large number).

Program:

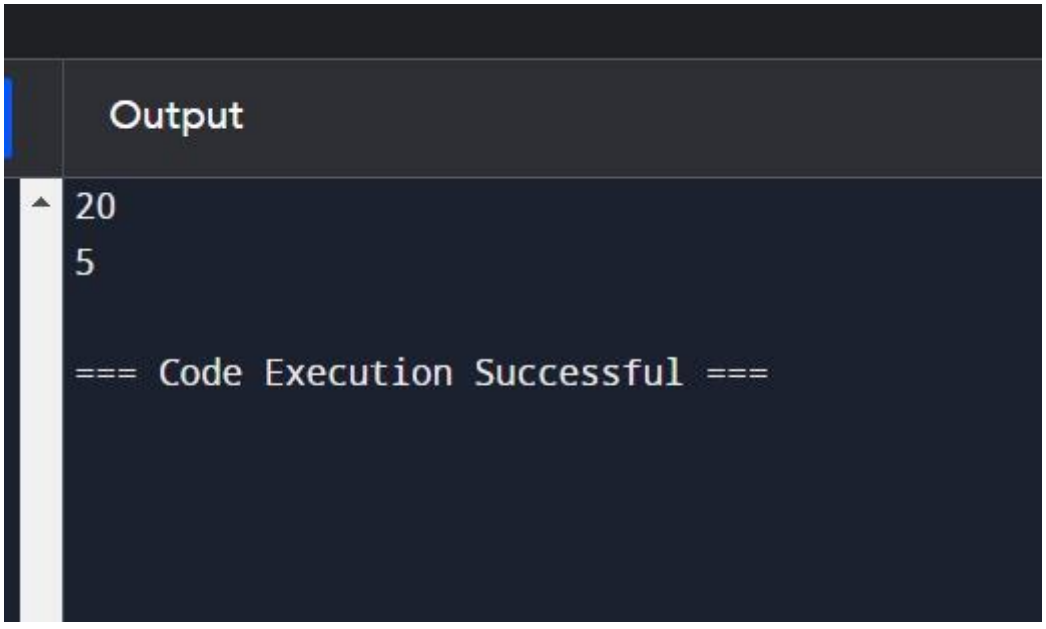
```
import heapq



def dijkstra(graph, source):
    n = len(graph)
    dist = [float('inf')] * n
    dist[source] = 0
    heap = [(0, source)]  # (distance, vertex)
    while heap:
        d, u = heapq.heappop(heap)
        # If current distance is greater than known shortest distance, skip it
        if d > dist[u]:
            continue
        for v in range(n):
            if graph[u][v] != float('inf'):  # There is an edge from u to v
                if dist[u] + graph[u][v] < dist[v]:
                    dist[v] = dist[u] + graph[u][v]
                    heapq.heappush(heap, (dist[v], v))
    return dist
# Example 1
graph1 = [
    [0, 10, 3, float('inf'), float('inf')],
    [float('inf'), 0, 1, 2, float('inf')],
    [float('inf'), 4, 0, 8, 2],
```

```
    [float('inf'), float('inf'), float('inf'), 0, 7],
    [float('inf'), float('inf'), float('inf'), 9, 0]
]
source1 = 0
print(dijkstra(graph1, source1))  # Output: [0, 7, 3, 9, 5]
# Example 2
graph2 = [
    [0, 5, float('inf'), 10],
    [float('inf'), 0, 3, float('inf')],
    [float('inf'), float('inf'), 0, 1],
    [float('inf'), float('inf'), float('inf'), 0]
]
source2 = 0
print(dijkstra(graph2, source2))
```

Output:




Time complexity: O(n^2)