199. Given a graph represented by an edge list, implement Dijkstra's Algorithm to find the shortest path from a given source vertex to a target vertex. The graph is represented as a list of edges where each edge is a tuple (u, v, w) representing an edge from vertex u to vertex v with weight w.

Program:

```
import heapq

def dijkstra(edges, n, source, target):
    graph = {i: [] for i in range(n)}
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))  # if undirected graph
    dist = [float('inf')] * n
    dist[source] = 0
    heap = [(0, source)]  # (distance, vertex)
    while heap:
        d, u = heapq.heappop(heap)
        if u == target:
            return dist[target]
        if d > dist[u]:
            continuimport heapq
def dijkstra(edges, n, source, target):
    graph = {i: [] for i in range(n)}
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))  # if undirected graph
    dist = [float('inf')] * n
    dist[source] = 0
    heap = [(0, source)]  # (distance, vertex)
    while heap:
```

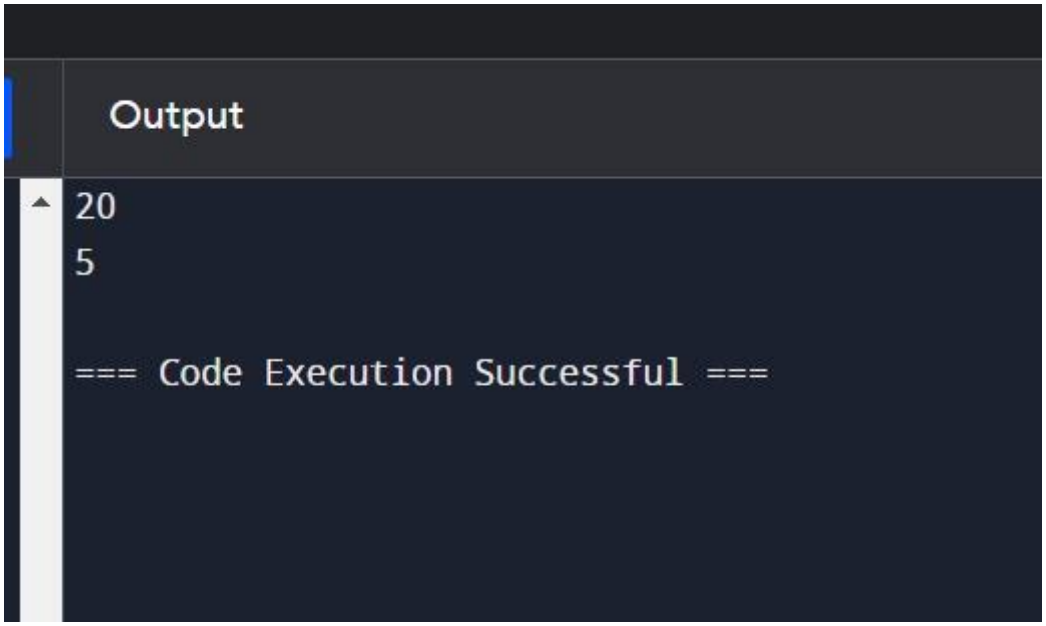
```
        d, u = heapq.heappop(heap)
        if u == target:
            return dist[target]
        if d > dist[u]:
            continue

        for v, weight in graph[u]:
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
                heapq.heappush(heap, (dist[v], v))
    return dist[targetimport heapq
def dijkstra(edges, n, source, target):
    graph = {i: [] for i in range(n)}
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))  # if undirected graph
    dist = [float('inf')] * n
    dist[source] = 0
    heap = [(0, source)]  # (distance, vertex)
    while heap:
        d, u = heapq.heappop(heap)
        if u == target:
            return dist[target]
        if d > dist[u]:
            continue
        for v, weight in graph[u]:
            if dist[u] + weight < dist[v]:
```

```
            dist[v] = dist[u] + weight
            heapq.heappush(heap, (dist[v], v))
    return dist[target]
# Example 1
edges1 = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
        (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
n1 = 6
source1 = 0
target1 = 4
print(dijkstra(edges1, n1, source1, target1))  # Output: 20
# Example 2
edges2 = [(0, 1, 10), (0, 4, 3), (1, 2, 2), (1, 4, 4), (2, 3, 9),
        (3, 2, 7), (4, 1, 1), (4, 2, 8), (4, 3, 2)]
n2 = 5
source2 = 0
target2 = 3
print(dijkstra(edges2, n2, source2, target2))  # Output:
# Example 1
edges1 = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
        (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
n1 = 6
source1 = 0
target1 = 4
print(dijkstra(edges1, n1, source1, target1))  # Output: 20
# Example 2
edges2 = [(0, 1, 10), (0, 4, 3), (1, 2, 2), (1, 4, 4), (2, 3, 9),
        (3, 2, 7), (4, 1, 1), (4, 2, 8), (4, 3, 2)]
```

```
n2 = 5
source2 = 0
target2 = 3
print(dijkstra(edges2, n2, source2, target2))  # Output:
        for v, weight in graph[u]:
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
                heapq.heappush(heap, (dist[v], v))
    return dist[target]
# Example 1
edges1 = [(0, 1, 7), (0, 2, 9), (0, 5, 14), (1, 2, 10), (1, 3, 15),
          (2, 3, 11), (2, 5, 2), (3, 4, 6), (4, 5, 9)]
n1 = 6
source1 = 0
target1 = 4
print(dijkstra(edges1, n1, source1, target1))  # Output: 20
# Example 2
edges2 = [(0, 1, 10), (0, 4, 3), (1, 2, 2), (1, 4, 4), (2, 3, 9),
          (3, 2, 7), (4, 1, 1), (4, 2, 8), (4, 3, 2)]
n2 = 5
source2 = 0
target2 = 3
print(dijkstra(edges2, n2, source2, target2))
```

Output:



Time complexity: O((V + E) log V)