5.6 DIJKSTRA'S ALGORITHM USING EDGE LIST

Question:

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.

AIM

To compute the shortest path from a source vertex to a target vertex in a weighted graph using Dijkstra's algorithm and an edge list representation.

ALGORITHM

1. Convert the edge list into an adjacency list graph[u] = [(v, w), ...].
2. Use a min-heap (priority queue) to always expand the node with the smallest known distance.
3. Initialize a dictionary dist to store the shortest distance to each node.
4. Start from the source node with distance 0.
5. While the heap is not empty:
    - Pop the node with the smallest distance.
    - If it's the target, return the distance.
    - For each neighbor, calculate the new distance.
    - If the new distance is better, update and push to the heap.
6. If the target is unreachable, return -1.

PROGRAM

```
import heapq
from collections import defaultdict

def dijkstra(n, edges, source, target):
    graph = defaultdict(list)
    for u, v, w in edges:
        graph[u].append((v, w))
        graph[v].append((u, w))
    dist = [float('inf')] * n
    dist[source] = 0
    heap = [(0, source)]
    while heap:
        d, u = heapq.heappop(heap)
        if u == target:
            return d
        for v, w in graph[u]:
            if d + w < dist[v]:
                dist[v] = d + w
                heapq.heappush(heap, (dist[v], v))
    return -1

n = int(input("Enter number of vertices: "))
m = int(input("Enter number of edges: "))
edges = []
for _ in range(m):
    u, v, w = map(int, input("Edge: ").split())
    edges.append((u, v, w))
source = int(input("Source: "))
target = int(input("Target: "))
print("Shortest distance from source to target:", dijkstra(n, edges, source, target))
```

Input:

Enter number of vertices: 6
Enter number of edges: 9
Edges: 0 1 7
Edges: 0 2 9
Edges: 0 5 14
Edges: 1 2 10
Edges: 1 3 15
Edges: 2 3 11
Edges: 2 5 2
Edges: 3 4 6
Edges: 4 5 9
Source: 0
Target: 4

Output:

```
Enter number of vertices: 6
Enter number of edges: 9
Edge: 0 1 7
Edge: 0 2 9
Edge: 0 5 14
Edge: 1 2 10
Edge: 1 3 15
Edge: 2 3 11
Edge: 2 5 2
Edge: 3 4 6
Edge: 4 5 9
Source: 0
Target: 4
Shortest distance from source to target: 20
>>>
```

RESULT:

Thus the program is successfully executed, and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity: O(E log N), where *E* is the number of edges and *N* is the number of nodes
- Space Complexity: O(N + E), for graph and heap