> *Dijkstra’s algorithm can efficiently find the shortest path from a source node to all other nodes in a graph.*

Here’s how you can use Dijkstra’s algorithm to solve the problem:

1. Create a set of unvisited nodes and initialize the distances of all nodes from the source node to infinity, except for the source node itself, which is initialized with a distance of 0.
2. Set the current node as the source node.
3. For the current node, calculate the tentative distance from the source node to its neighboring nodes. If the calculated distance is less than the previously recorded distance, update the distance.
4. Mark the current node as visited.
5. Select the unvisited node with the smallest tentative distance as the next current node and go to step 3.
6. Repeat steps 3 to 5 until all nodes have been visited or the target node is reached.
7. After visiting all nodes or reaching the target node, the shortest path from the source to the target is the sequence of nodes with the minimum distance recorded.

Example:

Consider the following weighted directed graph representing a network of cities connected by roads. Each node in the graph represents a city, and each directed edge represents a road connecting two cities with a certain distance. Find the shortest path from city A to city E.

Solution:

Graph:

```
A --2--> B --1--> C
|         ^       |
3         |       4
|         |       v
D --5----> E --7--> F
```

To solve this problem, we can apply Dijkstra’s algorithm.

Let’s go through the step-by-step solution:

Step 01: Create a set of unvisited nodes and initialize the distances of all nodes from the source node to infinity, except for the source node itself, which is initialized with a distance of 0.

- unvisited = {A, B, C, D, E, F}
- distances = {A: 0, B: inf, C: inf, D: inf, E: inf, F: inf}

Step 02: Set the current node as the source node (A).

Step 03: For the current node, calculate the tentative distance from the source node to its neighboring nodes. If the calculated distance is less than the previously recorded distance, update the distance.

- For A, neighbors are B (distance 2) and D (distance 3).

- Update the distances: distances[B] = 2, distances[D] = 3

Step 04: Mark the current node as visited (A).

Step 05: Select the unvisited node with the smallest tentative distance as the next current node and go to step 3.

- The next current node is B with a distance of 2.

Step 06: Repeat steps 3 to 5 until all nodes have been visited or the target node is reached.

- For B, neighbors are C (distance 1) and E (distance 4).
- Update the distances: distances[C] = 3, distances[E] = 6
- For C, neighbor is F (distance 4).
- Update the distance: distances[F] = 7
- For D, neighbor is E (distance 5).
- Update the distance: distances[E] = 5
- For E, neighbor is F (distance 7).
- Update the distance: distances[F] = 7
- For F, no outgoing edges.

Step 07: After visiting all nodes or reaching the target node (E), the shortest path from the source (A) to the target (E) is the sequence of nodes with the minimum distance recorded.

- The shortest distance from A to E is 5.
- The shortest path is A -> B -> E.

Therefore, the shortest path from city A to city E is A -> B -> E with a distance of 5.