```
import heapq
class LinkStateNetwork:
  def __init__(self):
    self.graph = {}

  def add_edge(self, src, dest, weight):
    if src not in self.graph:
      self.graph[src] = {}
    if dest not in self.graph:
      self.graph[dest] = {}
      self.graph[src][dest] = weight
      self.graph[dest][src] = weight  # Undirected graph

  def dijkstra(self, start):
    distances = {node: float('inf') for node in self.graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    shortest_paths = {node: [] for node in self.graph}

    while priority_queue:
      current_distance, current_node = heapq.heappop(priority_queue)
      if current_distance > distances[current_node]:
        continue
      for neighbor, weight in self.graph[current_node].items():
        distance = current_distance + weight
        if distance < distances[neighbor]:
          distances[neighbor] = distance
          shortest_paths[neighbor] = shortest_paths[current_node] + [current_node]
          heapq.heappush(priority_queue, (distance, neighbor))
    return distances, shortest_paths

  def print_routing_table(self, start):
    distances, paths = self.dijkstra(start)
    print(f"Link-State Routing Table from {start}:")
    for destination in distances:
      if distances[destination] < float('inf'):
        print(f"To {destination} via {' -> '.join(paths[destination] + [destination])}, cost: {distances[destination]}")
      else:
        print(f"To {destination}: Unreachable")

# Example usage of Link-State
if __name__ == "__main__":
  link_state_network = LinkStateNetwork()
  link_state_network.add_edge('A', 'B', 1)
  link_state_network.add_edge('A', 'C', 4)
  link_state_network.add_edge('B', 'C', 2)
  link_state_network.add_edge('B', 'D', 5)
  link_state_network.add_edge('C', 'D', 1)
  link_state_network.print_routing_table('A')
```

```
class DistanceVectorNetwork:
  def __init__(self):
    self.graph = {}
    self.distance_table = {}

  def add_edge(self, src, dest, weight):
    if src not in self.graph:
      self.graph[src] = {}
    if dest not in self.graph:
      self.graph[dest] = {}
    self.graph[src][dest] = weight
    self.graph[dest][src] = weight  # Undirected graph

  def initialize_distance_table(self, start):
    for node in self.graph:
      self.distance_table[node] = {n: float('inf') for n in self.graph}
    self.distance_table[start][start] = 0
    for neighbor in self.graph[start]:
      self.distance_table[start][neighbor] = self.graph[start][neighbor]

  def update_distance_table(self):
    for _ in range(len(self.graph) - 1):
      for src in self.graph:
        for dest in self.graph[src]:
          for neighbor in self.graph[src]:
            if self.distance_table[src][dest] > self.distance_table[src][neighbor] + self.distance_table[neighbor][dest]:
              self.distance_table[src][dest] = self.distance_table[src][neighbor] + self.distance_table[neighbor][dest]

  def print_routing_table(self, start):
    self.initialize_distance_table(start)
    self.update_distance_table()
    print(f"Distance Vector Routing Table from {start}:")
    for destination in self.distance_table[start]:
      if self.distance_table[start][destination] < float('inf'):
        print(f"To {destination}: Cost: {self.distance_table[start][destination]}")
      else:
        print(f"To {destination}: Unreachable")

# Example usage of Distance Vector
if __name__ == "__main__":
  distance_vector_network = DistanceVectorNetwork()
  distance_vector_network.add_edge('A', 'B', 1)
  distance_vector_network.add_edge('A', 'C', 4)
  distance_vector_network.add_edge('B', 'C', 2)
  distance_vector_network.add_edge('B', 'D', 5)
  distance_vector_network.add_edge('C', 'D', 1)
  distance_vector_network.print_routing_table('A')
```

Output:

```
Link-State Routing Table from A:
To A via A, cost: 0
To B via A -> B, cost: 1
To C via A -> C, cost: 4
To D via A -> B -> D, cost: 6
Distance Vector Routing Table from A:
To A: Cost: 0
To B: Cost: 1
To C: Cost: 4
To D: Unreachable
```