PROBLEM ANALYSIS

Dijkstra Algorithms:

Dijkstra's shortest path algorithm is similar to Prim's because both use local shortest choices to build a global solution. But unlike Prim's, Dijkstra's does **not** create a minimum spanning tree; instead, it finds the shortest paths from one source vertex to all other vertices in a graph. It works on both directed and undirected graphs.

Because it computes distances from a **single source**, it is known as the single-source shortest path algorithm, and its result is a shortest-path tree. The algorithm takes a graph **G(V, E)** and a source vertex **S**, and outputs the shortest paths from S to every reachable vertex.

How it works:

1. Set initial distances for all vertices: 0 for the source vertex, and infinity for all the other.
2. Choose the unvisited vertex with the shortest distance from the start to be the current vertex. So the algorithm will always start with the source as the current vertex.
3. For each of the current vertex's unvisited neighbor vertices, calculate the distance from the source and update the distance if the new, calculated, distance is lower.
4. We are now done with the current vertex, so we mark it as visited. A visited vertex is not checked again.
5. Go back to step 2 to choose a new current vertex, and keep repeating these steps until all vertices are visited.
6. In the end we are left with the shortest path from the source vertex to every other vertex in the graph.

Core Idea

Dijkstra repeatedly selects the closest unvisited node, then tries to improve ("relax") the distances of its neighbors.

We keep a distance table:

- dist[v] = current shortest known distance from source → v
  Initially:
- dist[source] = 0
- For all other nodes: dist[v] = ∞ (infinity)

We also maintain:

- A priority queue (min-heap) to always pick the next closest node efficiently.
- A visited set to avoid reprocessing nodes.

Step-by-Step Process (Simplified)

1. **Start**

- Give all nodes a distance of infinity (∞) because we don't know the shortest path yet.
- Set the source node's distance to 0.

2. **Use a priority queue**
   - Put the source node in a queue to pick the closest node first.
3. **Repeat until the queue is empty**
   - Take the node with the **smallest distance** (call it u).
   - If u is already visited, skip it.
   - Mark u as visited.
4. **Update neighbors (Relaxation)**
   - Look at each neighbor v of u.
   - Check if going through u gives a **shorter distance**:

     if dist[u] + weight(u,v) < dist[v]:

     dist[v] = dist[u] + weight(u,v)
   - If it is shorter, update dist[v] and add v to the queue.
5. **Finish**
   - When the queue is empty, the shortest distances from the source to all nodes are found.

Pseudocode:

```
1  function Dijkstra(Graph, source):
2
3      for each vertex v in Graph.Vertices:
4          dist[v] ← INFINITY
5          prev[v] ← UNDEFINED
6          add v to Q
7      dist[source] ← 0
8
9      while Q is not empty:
10         u ← vertex in Q with minimum dist[u]
11         Q.remove(u)
12
13         for each arc (u, v) in Q:
14             alt ← dist[u] + Graph.Edges(u, v)
15             if alt < dist[v]:
16                 dist[v] ← alt
17                 prev[v] ← u
18
19     return dist[], prev[]
```