Dijkstra

➢ What is Dijkstra?

Dijkstra's Algorithm is a shortest path algorithm used to find the minimum distance from a starting node (source) to all other nodes in a graph **with non-negative edge weights**.

It is widely used in maps, GPS navigation, routing, networking, and many real-world applications.

The main idea is: always expand the **closest unvisited node**, update the cost of its neighbors, and repeat.

➢ Why learn Dijkstra?

- It is the foundation of many shortest-path problems.
- Used in real-life applications like GPS route planning.
- Helps understand priority queues and greedy algorithms.
- Essential for solving many competitive programming and computer science problems.
- Works efficiently for large graphs.
- Demonstrates the greedy strategy in practice.

➢ What you should know first

- Basics of graphs (nodes, edges, weights).
- Adjacency list or adjacency matrix representation.
- Priority queue / min-heap concept.
- Arrays, loops, functions, and basic algorithmic thinking.

➢ Key Idea

- You start at the source with distance **0**.
- All other nodes begin with distance **infinity**.
- Always pick the **closest unvisited node**.
- Update the distances of all its neighbors.
- If a new path is shorter, replace the old distance.

- Once a node is chosen as the closest, its distance becomes **final**.
- Repeat until all nodes are processed or the queue becomes empty.

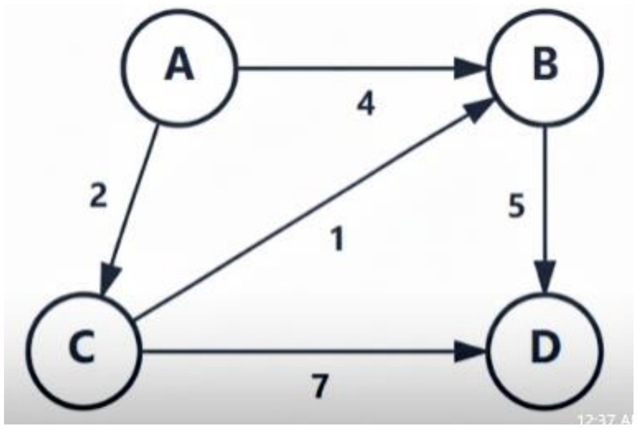
➢ How Dijkstra Works

- Start with all distances = infinity, except the source (0).
- Pick the unvisited node with the smallest known distance.
- For each neighbor, update the `distance = current distance + edge weight` if smaller.
- Mark the current node as visited.
- Repeat until all nodes are processed.

➢ Pseudocode

```
Dijkstra(G, source) {
for each vertex u in G {
  dist[u] = infinity
  parent[u] = null
}
dist[source] = 0
PQ = priority_queue()
PQ.push( (0, source) )
while (PQ is not empty) {
  (du, u) = PQ.pop()
  for each v in Adj[u] {
    w = weight(u, v)
    if (dist[u] + w < dist[v]) {
      dist[v] = dist[u] + w
      parent[v] = u
      PQ.push( (dist[v], v) )
    }
  }
}
for each vertex u in G {
  print dist[u]
}  }
```

➢ **Diagram**



Dijkstra's Algorithm from node A:

Initial:

Distance: A=0, B=∞, C=∞, D=∞

Unvisited: {A, B, C, D}

Step 1: Visit A (distance=0)

Update: B=4, C=2

Unvisited: {B, C, D}

Step 2: Visit C (distance=2)

Update: B=min(4, 2+1)=3, D=9

Unvisited: {B, D}

Step 3: Visit B (distance=3)

Update: D=min(9, 3+5)=8

Unvisited: {D}

Step 4: Visit D (distance=8)

Done!

Shortest distances from A:

A→A: 0

A→C: 2

A→B: 3 (via C)

A→D: 8 (via C→B)

Shortest paths:

A→B: A→C→B (total: 3)

A→C: A→C (total: 2)

A→D: A→C→B→D (total: 8)

- **Time & Space Complexity**

  Using Priority Queue (recommended)

  - **Time:** O((V + E) log V)
  - **Space:** O(V)

  Using simple array (slower)

  - **Time:** O(V^2)

- **After learning Dijkstra**

  Dijkstra's algorithm helps you find the shortest path from one point to all other points in a graph. It works by always picking the closest unvisited point, updating the distances to its neighbors, and then moving on. You keep doing this until all points are checked. It's useful for things like GPS directions or finding the fastest route in a network.