Dijkstra's Algorithm(Basic)

1. Introduction

Dijkstra's Algorithm is one of the most famous algorithms in computer science used to find the shortest path from a starting node (source) to all other nodes in a weighted graph.
It was invented by Edsger Dijkstra in 1956.

Where it works:

- Graph can be directed or undirected
- All edge weights must be non-negative

What it finds:

- The *minimum cost* (shortest distance) to reach each node from the source
- Optionally, the actual shortest path (if we store parents)

2. Why Use Dijkstra's Algorithm?

It is used when:

- You want the shortest path in a map (GPS)
- You want the cheapest/fastest route
- You have weighted graphs with no negative edges
- You need optimized paths for networking, games, and robotics

3. Key Idea

The algorithm works by repeatedly selecting the next closest node that has not been processed yet.

Concept summary:

1. Start at the source with distance 0.
2. All other nodes start with distance ∞ (infinity).
3. Pick the nearest unvisited node.
4. Update (relax) the distances of its neighbors.
5. Mark the node as visited.
6. Continue until all nodes are visited.

This process ensures the distance you assign to a node is always the **shortest possible**.

4. Terms You Should Know

Distance Table

A list/map storing the shortest known distance to every node.

Visited Set

Nodes that have been permanently processed.

Priority Queue

Used to always get the node with the minimum distance efficiently.

5. Step-by-Step Working (Detailed)

Let's walk through how the algorithm runs:

Step 1: Initialization

- Distance[source] = 0
- Distance[all other nodes] = ∞
- Priority queue starts with (0, source)

Step 2: Pick closest unvisited node

Pop the smallest distance from the priority queue.

Step 3: Relax edges

For each neighbor:

if distance[current] + weight < distance[neighbor]:

distance[neighbor] = distance[current] + weight

This step tries to improve the known distance.

Step 4: Mark node as visited

Once we process the node, we don't update it again.

Step 5: Repeat

Continue until the queue becomes empty.

6. Time Complexity

Depends on how the priority queue is implemented:

Implementation	**Time Complexity**
No heap (simple array)	$O(V^2)$
Binary Heap (min-heap)	O((V + E) log V)
Fibonacci Heap	O(E + V log V)

Most practical usage: **O((V + E) log V)**.

PSEUDOCODE

Dijkstra's Algorithm Pseudocode in C++

```
function dijkstraalgorithm(G, S)
    for each node N in G
        dist[N] <- infinite
        prev[N] <- NULL
        If N != S, add N to Priority Queue Q
    dist[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unmarked neighbour N of U
            temporaryDist <- dist[U] + edgeWeight(U, N)
            if temporaryDist < dist[N]
                dist[N] <- temporaryDist
                prev[N] <- U
    return dist[], prev[]
```