Dijkstra

Dijkstra's algorithm is a graph algorithm used to find the **shortest path** from a starting node to all other nodes in a weighted graph where all edge weights are non-negative. It starts at the source node and repeatedly selects the unvisited node with the smallest tentative distance, then updates the distances of its neighbors if a shorter path is found. This process continues until all nodes have been visited, ensuring the shortest distance from the source to every node is correctly calculated. It is widely used in routing, GPS navigation, and network optimization.

Advantages

- Finds the **shortest path** in a weighted graph with non-negative edges.
- Works for both **directed and undirected graphs**.
- Efficient with **priority queue optimization**.
- Widely used in **real-world applications** like GPS, networking, and traffic routing.

Limitations

- **Does not work with negative weight edges**.
- Can be **slow for very large graphs** without optimization.
- Needs a **priority queue or extra data structures** for efficiency.
- Only finds the shortest path from a **single source**, not all-pairs shortest paths.

Steps of Dijkstra

1. **Initialize distances**
   Set the distance of the source node to 0 and all others to infinity.
2. **Mark all nodes as unvisited**
3. **Select the unvisited node with the smallest distance**

4. **For each neighbor of the current node**
    - Calculate the distance from the source through the current node.
    - If this distance is smaller than the current known distance, update it.
5. **Mark the current node as visited**
6. **Repeat steps 3–5** until all nodes are visited
7. **Shortest paths determined**
   After all nodes are visited, the distance array contains the shortest distances from the source to every node.

Pseudocode

```
Dijkstra(Graph, source):
   create distance array dist[], set all values to ∞
   dist[source] = 0
   create a priority queue Q
   insert (source, 0) into Q

   while Q is not empty:
      u = extract node with minimum dist from Q

      for each neighbor v of u:
         if dist[u] + weight(u, v) < dist[v]:
            dist[v] = dist[u] + weight(u, v)
            insert (v, dist[v]) into Q

   return dist[]
```

Time Complexity

- **$O(V^2)$** for simple implementation (using array)
- **O((V + E) log V)** with **priority queue / min-heap optimization**

Where **V** = number of vertices, **E** = number of edges.

Dijkstra Code:

https://github.com/shanto470/algorithm_plm/blob/main/Dijkstra/basic/basic.cpp