

Dijkstra's Algorithm

● MEDIUM

■ Introduction

Dijkstra's algorithm finds the shortest path from a source node to all other nodes in a WEIGHTED graph (non-negative weights). It is the foundation of GPS navigation, network routing, and all weighted shortest-path problems.

■ Core Intuition

Greedy + Priority Queue

- Maintain a `dist[]` array: shortest known distance from source to each node.
- Use a Min-Heap (Priority Queue): always process the node with minimum current distance.
- When processing node `u`: for each neighbor `v`, check if $\text{dist}[u] + \text{weight}(u,v) < \text{dist}[v]$.
- If yes, update `dist[v]` and add $(\text{dist}[v], v)$ to the heap.
- Key insight: once a node is popped from heap, its shortest distance is FINALIZED.

■ Visual

Dijkstra's Trace

Graph (weighted):

0 --3-- 1 --1-- 2

| |

4 2

| |

3 --5-- 4

Dijkstra from 0:

`dist[] = [0, INF, INF, INF, INF]`

Heap: $[(0,0)]$

Pop (0,0): update $1 \rightarrow 3$, $3 \rightarrow 4$. Heap: $[(3,1), (4,3)]$

Pop (3,1): update $2 \rightarrow 4$, $4 \rightarrow 5$. Heap: $[(4,2), (4,3), (5,4)]$

Pop (4,2): no better path. Heap: $[(4,3), (5,4)]$

Pop (4,3): update nothing. Heap: $[(5,4)]$

`dist[] = [0, 3, 4, 4, 5]`

■ Java Implementation

Java

```
import java.util.*;

class Solution {

    public int[] dijkstra(int V, List<List<int[]>> adj, int src) {

        int[] dist = new int[V];

        Arrays.fill(dist, Integer.MAX_VALUE);

        dist[src] = 0;

        // Min-heap: [distance, node]

        PriorityQueue<int[]> pq = new PriorityQueue<>((a,b) -> a[0]-b[0]);

        pq.offer(new int[]{0, src});

        while (!pq.isEmpty()) {

            int[] curr = pq.poll();

            int d = curr[0], node = curr[1];

            if (d > dist[node]) continue; // Stale entry, skip

            for (int[] edge : adj.get(node)) {

                int neighbor = edge[0], weight = edge[1];

                int newDist = dist[node] + weight;

                if (newDist < dist[neighbor]) {

                    dist[neighbor] = newDist;

                    pq.offer(new int[]{newDist, neighbor});

                }

            }

        }

        return dist;

    }

}
```

Time & Space Complexity

Case	Time	Space	Why
Time	$O((V+E) \log V)$	$O(V)$	Each edge processed once; heap operations $O(\log V)$
Space	$O(V + E)$	$O(V)$	Adjacency list $O(V+E)$; dist array + heap $O(V)$

Common Mistakes

Critical Pitfalls

- Using Dijkstra with **NEGATIVE** weights — undefined behavior! Use Bellman-Ford instead.
- Not skipping stale heap entries ($d > \text{dist}[\text{node}]$) — causes incorrect results.
- Using max-heap instead of min-heap — always use min-heap (compare by distance).
- Forgetting to initialize `dist[]` with `MAX_VALUE` except source.

Practice Problems

Case	Time	Space	Why
Medium	LC 743	Network Delay Time	Classic Dijkstra's
Medium	LC 787	Cheapest Flights K Stops	Modified Dijkstra / Bellman-Ford
Hard	LC 1514	Path with Max Probability	Dijkstra with probabilities
Hard	LC 778	Swim in Rising Water	Dijkstra on grid

Mental Model

Remember Forever

- "Dijkstra is a greedy algorithm: always expand the nearest unfinalized node. Like expanding a growing circle of confirmed-shortest paths from the source."