# Dijkstra’s Algorithm in DSA: Beginner to Advanced Guide

**Subtitle:** Learn how to find shortest paths in weighted graphs efficiently using Dijkstra’s algorithm.

**Website Name:** haas.dev
**Website Link:** https://dev-roast-app.vercel.app

## Introduction

Dijkstra’s Algorithm is used to find the **shortest path from a source node to all other nodes in a weighted graph with non-negative edges**. It’s a fundamental algorithm in **routing, navigation, and graph-based problem solving**.

## Step 1: Core Concepts

- Works on **weighted graphs** with **non-negative edge weights**
- Maintains a **distance array** initialized to infinity, except the source (0)
- Uses a **priority queue** to always expand the node with the smallest tentative distance
- Updates distances for neighboring nodes if a **shorter path is found**

## Step 2: Graph Representation

- **Adjacency List** is preferred for efficiency

```
let graph = {
  0: [[1,4],[2,1]],
  1: [[3,1]],
  2: [[1,2],[3,5]],
  3: []
};
```

- Each entry [node, weight] represents an edge and its weight

## Step 3: Implementation in JavaScript

```
class MinHeap {
  constructor() { this.heap = []; }
  push(val){ this.heap.push(val); this.heapifyUp(); }
```

```
  pop(){
    if(this.heap.length === 1) return this.heap.pop();
    let top = this.heap[0];
    this.heap[0] = this.heap.pop();
    this.heapifyDown();
    return top;
  }
  heapifyUp(){ let i=this.heap.length-1; while(i>0){ let p=Math.floor((i-1)/2); if(this.heap[p][1]<=this.heap[i][1]) break; [this.heap[p],this.heap[i]]=[this.heap[i],this.heap[p]]; i=p; }}
  heapifyDown(){ let i=0,l=this.heap.length; while(true){ let left=2*i+1,right=2*i+2,smallest=i; if(left<l && this.heap[left][1]<this.heap[smallest][1]) smallest=left; if(right<l && this.heap[right][1]<this.heap[smallest][1]) smallest=right; if(smallest===i) break; [this.heap[i],this.heap[smallest]]=[this.heap[smallest],this.heap[i]]; i=smallest; }}
  isEmpty(){ return this.heap.length===0; }
}

function dijkstra(graph, src, n){
  let dist = Array(n).fill(Infinity);
  dist[src] = 0;
  let pq = new MinHeap();
  pq.push([src,0]);

  while(!pq.isEmpty()){
    let [u, d] = pq.pop();
    if(d>dist[u]) continue;
    for(let [v,w] of graph[u]){
      if(dist[v] > dist[u]+w){
        dist[v] = dist[u]+w;
        pq.push([v, dist[v]]);
```

```
    }
  }
 }
 return dist;
}


// Example usage
let distances = dijkstra(graph, 0, 4);
console.log(distances); // [0,3,1,4]
```

# Step 4: Applications

1. **Shortest path from a single source**
2. **Routing and navigation problems**
3. **Network optimization**
4. **Weighted graph traversal in competitive programming**

# Step 5: Mini Exercises

1. Find shortest path from node 0 in a given weighted graph
2. Implement Dijkstra using a simple array (no priority queue)
3. Track the **actual path** along with distances
4. Solve a competitive programming problem using Dijkstra on sparse/dense graphs

# Step 6: Common Mistakes

- Using Dijkstra on **graphs with negative edges** → gives incorrect results
- Forgetting to update distances or skip outdated heap entries
- Using adjacency matrix for very large graphs → inefficient
- Not tracking the parent for path reconstruction

# Step 7: Practice Plan for Dijkstra

Day 1:

- Implement basic Dijkstra using array → 3 problems

Day 2:

- Implement priority queue-based Dijkstra → 5 problems

Day 3:

- Path reconstruction + multiple queries → 3–5 problems

Day 4:

- Competitive programming style problems → 5–7 problems

# Key Takeaways

- Dijkstra finds **shortest paths efficiently in non-negative weighted graphs**
- Priority queue is crucial for **O(E log V) performance**
- Widely used in **routing, network, and problem-solving**
- Practice builds **graph traversal intuition and optimization skills**

Visit haas.dev for more Dijkstra algorithm guides, problem sets, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app