Dijkstra's Algorithm: A Greedy Approach

Sumanth

June 18, 2025

Abstract

Dijkstra's Algorithm is a classic **greedy algorithm** designed to compute the shortest paths from a single source vertex to all other vertices in a weighted graph with non-negative edge weights. This document details the algorithm's formulation, pseudocode, complexity analysis, and practical applications. It includes enhancements such as path reconstruction, input validation, and considerations for adjacency matrix and priority queue implementations, along with limitations and extensions.

1 Introduction

Dijkstra's Algorithm solves the **single-source shortest path** problem in a weighted graph with non-negative edge weights. By greedily selecting the unvisited vertex with the smallest tentative distance, it iteratively updates distances to neighboring vertices. The algorithm is widely used in navigation systems, network routing, and AI pathfinding due to its efficiency and simplicity.

2 Problem Formulation

Given a weighted graph $G = (V, E)$ with $|V| = n$ vertices, $|E| = m$ edges, non-negative edge weights $w(u, v)$, and a source vertex s, compute the shortest path distances from s to all other vertices. Optionally, reconstruct the shortest paths.

The output includes:

- An array $\text{dist}[v]$, where $\text{dist}[v]$ is the shortest path distance from s to vertex v.
- A predecessor array $\text{pred}[v]$ for path reconstruction.

2.1 Key Idea

The algorithm maintains a set of unvisited vertices and iteratively selects the one with the smallest tentative distance. For each selected vertex, it **relaxes** the distances to its neighbors, updating them if a shorter path is found.

3 Dijkstra's Algorithm

The algorithm can be implemented using an adjacency matrix or an adjacency list with a priority queue. The following pseudocode uses an adjacency matrix for simplicity:

Algorithm 1 Dijkstra's Algorithm (Adjacency Matrix)

Input: Graph $G = (V, E)$, source vertex s, cost matrix w
Output: Distance array dist, predecessor array pred
Initialize $\text{dist}[v] \leftarrow \infty$ for all $v \in V$, $\text{dist}[s] \leftarrow 0$
Initialize $\text{pred}[v] \leftarrow -1$ for all $v \in V$
Initialize $\text{visited}[v] \leftarrow$ false for all $v \in V$
$\text{visited}[s] \leftarrow$ true
for each vertex $v \neq s$ **do**
    $\text{dist}[v] \leftarrow w[s][v]$
    if $w[s][v] \neq \infty$ **then**
        $\text{pred}[v] \leftarrow s$
    end if
end for
for $i = 1$ to $n - 1$ **do**
    Let u be the unvisited vertex with minimum $\text{dist}[u]$
    if u does not exist **then**
        break
    end if
    $\text{visited}[u] \leftarrow$ true
    for each vertex v where $\text{visited}[v] =$ false **do**
        if $\text{dist}[u] + w[u][v] < \text{dist}[v]$ **then**
            $\text{dist}[v] \leftarrow \text{dist}[u] + w[u][v]$
            $\text{pred}[v] \leftarrow u$
        end if
    end for
end for
Return dist, pred

3.1 Path Reconstruction

The shortest path from source s to vertex v is reconstructed by backtracking through the predecessor array pred. Starting from v, follow pred$[v]$ until reaching s, collecting vertices in reverse order.

4 Complexity Analysis

- **Time Complexity**:
  - Adjacency matrix: $O(n^2)$, suitable for dense graphs.
  - Adjacency list with min-heap: $O((m+n)\log n)$, efficient for sparse graphs.
  - Fibonacci heap (theoretical): $O(m+n\log n)$.
- **Space Complexity**:
  - $O(n)$ for distance, visited, and predecessor arrays.
  - $O(n^2)$ for the adjacency matrix.
  - $O(n)$ additional space for a priority queue in heap-based implementations.

5 Practical Applications

Dijkstra's Algorithm is used in:

- **GPS Navigation**: Computing shortest routes between locations.
- **Network Routing**: Protocols like OSPF for optimal packet routing.
- **AI Pathfinding**: Navigating characters in grid-based or graph-based game environments.
- **Transportation Systems**: Optimizing logistics and delivery routes.

6 Example

Consider a graph with 5 vertices and the following adjacency matrix (∞ denotes no edge):

	1	2	3	4	5
1	0	4	8	∞	∞
2	∞	0	2	5	∞
3	∞	∞	0	1	∞
4	∞	∞	∞	0	3
5	∞	∞	∞	∞	0

Starting from vertex 1, Dijkstra's Algorithm yields:

Vertex	1	2	3	4	5
Distance	0	4	6	7	10
Path	1	$1 \to 2$	$1 \to 2 \to 3$	$1 \to 2 \to 3 \to 4$	$1 \to 2 \to 3 \to 4 \to 5$

7 Limitations and Extensions

- **Limitations**:
  - Fails for graphs with negative edge weights (use Bellman-Ford instead).
  - Adjacency matrix implementation is less efficient for sparse graphs.
- **Extensions**:
  - **A* Algorithm**: Enhances Dijkstra's with heuristics for faster pathfinding.
  - **Bidirectional Dijkstra**: Runs from both source and target to reduce search space.
  - **Parallel Implementation**: Distributes computation for large graphs.

8 Implementation Considerations

- **Priority Queue**: For sparse graphs, use a min-heap (e.g., `std::priority_queue`) to achieve $O((m+n)\log n)$.
- **Adjacency Matrix**: The provided implementation uses an adjacency matrix for simplicity ($O(n^2)$), suitable for dense graphs.
- **Numeric Stability**: Use INT_MAX for infinity and check for overflow during relaxation. **Path** Maintain a predecessor array to reconstruct paths, as shown in the example.
- **Input Validation**: Ensure valid vertex count and source, and handle unreachable vertices explicitly.

9 References

- GeeksForGeeks: Dijkstra's Algorithm
- Wikipedia: Dijkstra's Algorithm
- Cormen, T. H., et al., *Introduction to Algorithms*, Chapter on Graph Algorithms.