LECTURE 13 – SHORTEST PATH ALGORITHMS

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 13

In this lecture, we'll implement the algorithms for finding the shortest path from given vertex to other vertices like Dijkstra, Floyd-Warshall and Bellman-Ford.

TOPICS WE'LL COVER:

- Types of Shortest Path Problems
- Dijkstra's algorithm
- Bellman-Ford
- Floyd-Warshall
- Practice Problems

GOALS FOR THIS LECTURE:

- Understand the concept of shortest paths in weighted graphs;
- Learn when to use Dijkstra, Floyd–Warshall, and Bellman–Ford;
- Implement each algorithm in code;
- Apply these algorithms to graph problems.

KAZAKH BRITISH TECHNICAL UNIVERSITY

TYPES OF SHORTEST PATH PROBLEMS

- **Types of Shortest Path Problems**
- Dijkstra's algorithm
- Bellman-Ford
- Floyd-Warshall
- Practice Problems

There are two types of shortest path problems: single-source shortest path (SSSR) and all-pair shortest path (APSP).

Single-source shortest path:

Find shortest distance from one start node s to all others.

- Dijkstra — works with non-negative weights;
- Bellman–Ford — works with negative weights, detects cycles.

All-pairs shortest path:

Find distances between every pair of nodes.

- Floyd–Warshall — works with positive & negative weights, can detect the negative cycles, but does not work correctly when graph contains negative cycles, and computes all-pairs shortest paths.

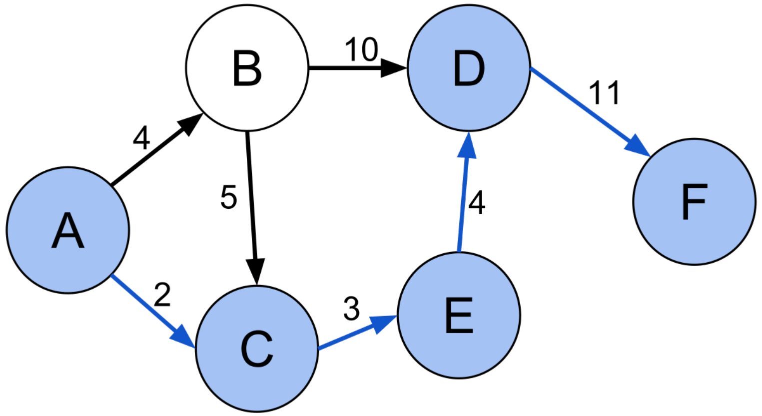

Figure 1 – "Shortest path (A, C, E, D, F), blue, between vertices A and F in the weighted directed graph" [1]

[1] https://en.wikipedia.org/wiki/Shortest_path_problem

DIJKSTRA’S ALGORITHM

Types of Shortest Path Problems

Dijkstra’s algorithm

Bellman-Ford

Floyd-Warshall

Practice Problems

We use Dijkstra’s algorithm when we have weighted graph with **only positive weights**, and have to *find the shortest path* from **one source** vertex **to others**.

Idea:

- We choose neighbor nodes greedily using a priority queue to always expand the “closest” node;
- Once a node is finalized, its shortest distance is never updated.

Applications:

- GPS navigation;
- Routing in networks;
- Path-planning.

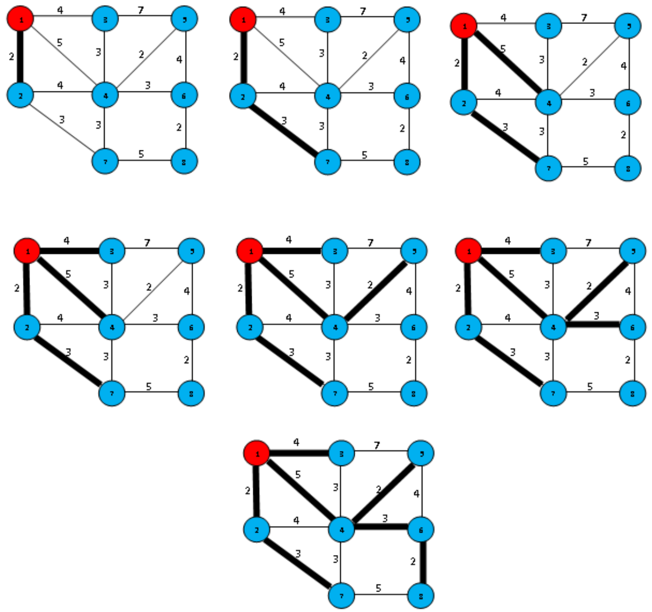

Figure 2 – Procedure of the algorithm [2]

[2] https://pages.cs.wisc.edu/~yzhao/

DIJKSTRA'S ALGORITHM (CONT.)

Types of Shortest Path Problems

Dijkstra's algorithm

Bellman-Ford

Floyd-Warshall

Practice Problems

To implement the current algorithm, we have **6 steps** to encounter:

1. Initialize all distances to ∞, except source = 0;
2. Put source in priority queue;
3. Extract node with smallest distance;
4. Relax all outgoing edges;
5. If new distance < old → update;
6. Continue until queue is empty.

Time Complexity:

- Using priority queue: **O((V + E) log V)**;
- Using adjacency matrix: $O(V^2)$.

```
vector<int> dijkstra(vector<vector<pii>>& adj, int start) {
    int V = adj.size();
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    vector<int> dist(V, INT32_MAX);

    dist[start] = 0; pq.emplace(0, start);

    while(!pq.empty()) {
        pii top = pq.top();
        pq.pop();

        int d = top.first;
        int u = top.second;

        if (d > dist[u]) continue;

        for (pii &p: adj[u]) {
            int v = p.first, w = p.second;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.emplace(dist[v], v);
            }
        }
    }
    return dist;
}
```

Figure 3 – Dijkstra's algorithm

BELLMAN-FORD

- Types of Shortest Path Problems
- Dijkstra's algorithm
- **Bellman-Ford**
- Floyd-Warshall
- Practice Problems

This algorithm effectively works with the negative weights, and is able to detect negative cycles as well. The working principle is **'relaxation of the edges'**.

Note: The negative cycle is a cycle, whose sum of edge weights is negative.

For an edge (u , v) with weight w, if reaching v through u gives a smaller distance than the current one, we update it:

$$\begin{gathered} if\ dist[v] > dist[u] + w, \\ then\ set\ dist[v] = dist[u] + w. \end{gathered}$$

This steps is applied to every edge exactly $\boldsymbol{V}$ **– 1** times.

Time complexity: O(V × E).

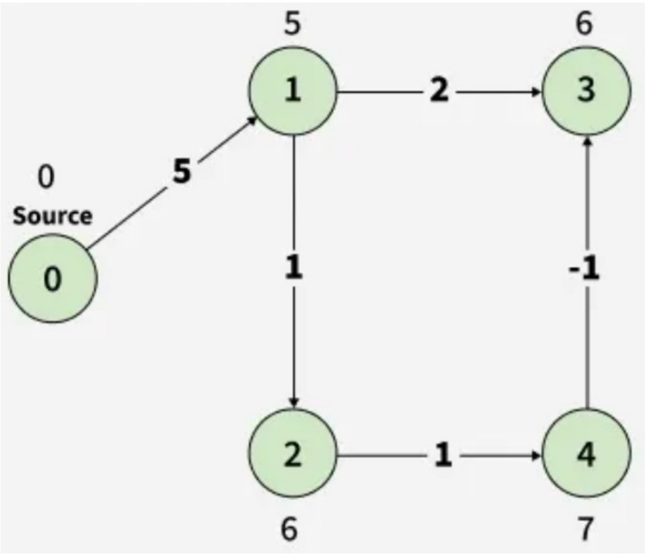

Figure 4 – Example graph for Bellman-Ford [3]

[3] https://www.geeksforgeeks.org/dsa/bellman-ford-algorithm-dp-23/

BELLMAN-FORD (CONT.)

The algorithm works in 3 steps:

1. Initialize distances;
2. Repeat V – 1 times:
   - Go through every edge
   - Relax it
3. 1 more iteration:
   - If any distance improves → negative cycle

Applications

- Currency arbitrage detection;
- Time-travel constraint (transport scheduling);
- Longest path (by negating weights, DAG only);

```cpp
vector<int> bellmanFord(int src, vector<vector<int>>& edges, int V) {
    vector<int> dist(V, INF);
    dist[src] = 0;

    for (int i=0; i < V; i++) {
        // one additional relaxation needs to detect negative cycle
        for (vector<int> edge: edges) {
            int u = edge[0];
            int v = edge[1];
            int w = edge[2];

            if (dist[u] != INF && dist[u] + w < dist[v]) {
                if (i == V - 1) return {-1};
                dist[v] = dist[u] + w;
            }
        }
    }
    return dist;
}
```

Figure 5 – Bellman-Ford algorithm

FLOYD-WARSHALL

Types of Shortest Path Problems

Dijkstra's algorithm

Bellman-Ford

Floyd-Warshall

Practice Problems

The algorithm works with 2D-array, that stores the distance between i-th & j-th vertices.

Example:

$$\text{dist[][]} = [[0, 4, 10^8, 5, 10^8], [10^8, 0, 1, 10^8, 6], [2, 10^8, 0, 3, 10^8], [10^8, 10^8, 1, 0, 2], [1, 10^8, 10^8, 4, 0]]$$

The graph may contain ***negative edge weights****, but it does not contain any negative weight cycles.*

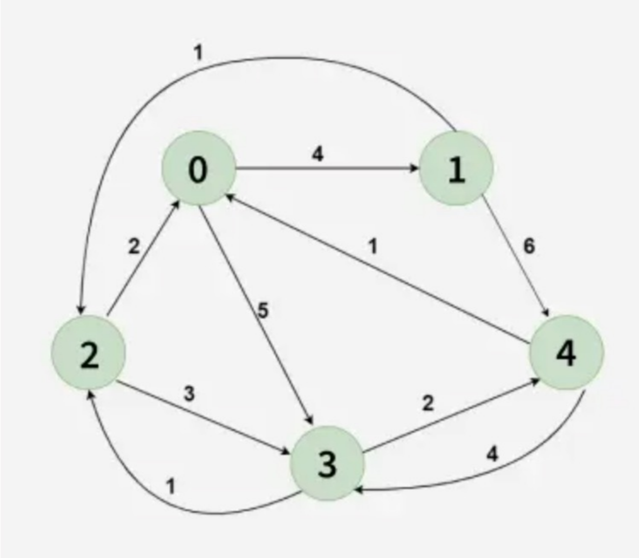

Figure 6 – Example undirected graph [4]

Note: the algorithm works both directed & undirected graphs.

[4] https://www.geeksforgeeks.org/dsa/floyd-warshall-algorithm-dp-16/

FLOYD-WARSHALL (CONT.)

- Types of Shortest Path Problems
- Dijkstra's algorithm
- Bellman-Ford
- **Floyd-Warshall**
- Practice Problems

Idea of the algorithm:

It improves a distance matrix by gradually allowing each vertex to act as an intermediate point.

For each vertex k, we check if any path from i to j becomes shorter by passing through k.

Since vertices 0 to k–1 were already considered, we use those previously computed shortest paths to update better ones that include k as a middle node.

Time complexity: $O(V^3)$

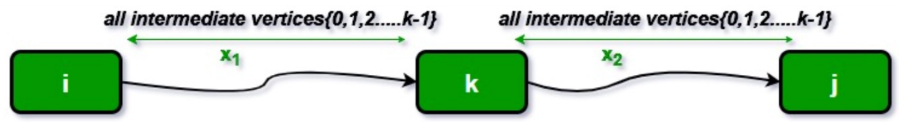

Figure 7 – Representation of intermediate vertices [4]

```cpp
#define INF 1e9

void floydWarshall(vector<vector<int>> &dist, int n) {
    for (int k=0; k < n; ++k) {
        for (int i=0; i < n; ++i) {
            for (int j=0; j < n; ++j) {
                if (dist[i][k] != INF && dist[k][j] != INF) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

Figure 8 – Example undirected graph

[4] https://www.geeksforgeeks.org/dsa/floyd-warshall-algorithm-dp-16/

SUMMARY

Types of Shortest Path Problems

Dijkstra’s algorithm

Bellman-Ford

Floyd-Warshall

Practice Problems

Table 1 – Summary for learned algorithms.

Algorithm	Handles Neg. Weights	Negative Cycle?	Complexity	Best Use
Dijkstra	No	No	O(E log V)	Large, positive weights
Bellman–Ford	Yes	Detects	O(VE)	Negative edges
Floyd–Warshall	Yes	No	$O(V^3)$	APSP, small graph

PRACTICE PROBLEMS

KAZAKH BRITISH TECHNICAL UNIVERSITY

We have one task to solve:

- 1334. Find the City With the Smallest Number of Neighbors at a Threshold Distance.

Q & A