SHORTEST PATH

Shortest Path Algorithms

Algorithm	Use Case	Graph Type	Time Complexity	Notes
BFS	When all edges have **equal weight**	Unweighted / same-weighted edges	O(V + E)	Simplest and fastest when all weights are equal.
Dijkstra	When edge weights are **non-negative**	Weighted, no negative weights	O((V + E) log V) with heap	Greedy, efficient for SSSP (Single Source Shortest Path).
Bellman-Ford	When edge weights can be **negative**	Weighted, allows negative weights	O(V * E)	Slower, but handles negative weights and detects negative cycles.
Floyd-Warshall	For **all-pairs shortest paths**	Dense graphs, small number of nodes	$O(V^3)$	Easy to implement; handles negative weights (but not negative cycles).
A* Search	For shortest path with a **goal node** and **heuristic**	Weighted, heuristic needed	Depends on heuristic quality	Often used in pathfinding (e.g. maps, games); faster than Dijkstra if heuristic is good.
Johnson's Algorithm	**All-pairs shortest paths** in sparse graphs with **negative weights**	Weighted, allows negative weights	$O(V^2 \log V + V * E)$	Reweights graph with Bellman-Ford, then runs Dijkstra from each node.
SPFA	Practical variant of Bellman-Ford, often faster	Weighted, allows negative weights	Avg: O(E), Worst: O(VE)	Queue-based; faster in practice, not guaranteed. Handles negative weights.
Bidirectional Search	When you know **start and target**, speeds up search in undirected graphs	Unweighted or uniformly weighted	O(b^(d/2)) in best case	Runs two simultaneous searches (from start and end); fast when goal is known.

Problem – 743. Network Delay Time

Medium

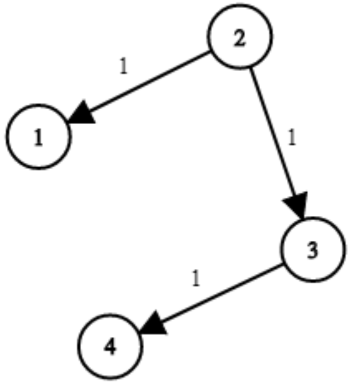
LeetCode leetcode.com/problems/network-delay-time

Problem

- You are given a network of nodes n with destination and time to reach that node
- You are given a starting node **k** and the number of nodes in the network **n**
- A signal is sent from node **k** to all nodes in the network
- Find the minimum time required for all nodes to receive the signal from **k**
- **Example:**

  k = 2   n =4

  Output: 3



Solution – 743. Network Delay Time

Medium

LeetCode leetcode.com/problems/network-delay-time

Solution

- This is solved using Djikstra algorithm
- Build the graph by storing in the destination node and the time from a source node:

  `graph[node] = [[node, distance]]`

  `graph[2] = [[1,1], [2,3]]`
- Set up a min-heap for Djikstra (priority_queue) with distance and node
- Perform Djikstra algorithm and store the shortest paths
- Check if all nodes were reached
- Return the longest distance among shortest paths

Code – 743. Network Delay Time

Medium

LeetCode leetcode.com/problems/network-delay-time

Code

Time: **O((n + e) log n)** Space: **O(n + e)** where n is the number of nodes and e the number of edges

```cpp
int networkDelayTime(vector<vector<int>>& times, int n, int k) {
    // node => (destination, distance)
    unordered_map<int, vector<pair<int, int>>> graph;
    for (const auto& time : times) {
        // time[0] = source node, time[1] = dest node, time[2] = time
        graph[time[0]].emplace_back(time[1], time[2]);
    }

    // min heap: distance from the origin 'k' to 'node'
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> minHeap;
    minHeap.emplace(0, k); // distance, starting node

    // shortest path from each node to the origin ‘k’ (node, distance)
    unordered_map<int, int> dist;

    // we start exploring the nodes from the minimum
    // distance to the origin 'k'
    while (!minHeap.empty()) {
        auto [distance, node] = minHeap.top();
        minHeap.pop();
        // already visited, skip
        if (dist.count(node)) continue;
        // set the distance
        dist[node] = distance;
        // look at the connections
        for (const auto& [n, d] : graph[node]) {
            // n[node, distance]
            // quick optimization, not necessary
            if (dist.count(n)) continue;
            // add the distance since we want
            // the distance from the origin
            minHeap.emplace(distance + d, n);
        }
    }

    // check if all nodes were visited
    if (dist.size() != n) return -1;

    // all the minimum distances are calculated
    // find the max one since we want to reach
    // all nodes
    int minTime = 0;
    for (const auto& [n, d] : dist) {
        minTime = max(minTime, d);
    }

    return minTime;
}
```

Problem – 1306. Jump Game III

Medium

LeetCode leetcode.com/problems/jump-game-iii

Problem

- Variation of Jump Game
- You are given an array of integers and a start index (integer)
- You are initially positioned at start index of the array
- You can jump either to **pos + array[pos]** or **pos - array[pos]**
- Check if you can reach any index with value 0

Solution – 1306. Jump Game III

Medium

LeetCode leetcode.com/problems/jump-game-iii

Solution

- Once you are in any position, you have two choices: either go **i + array[i]** or **i - array[i]**
- You should explore both directions recursively
- Once you **find 0,** return **true**
- Keep track of visited positions, **return false** once visited

Solution – 1306. Jump Game III

Medium

LeetCode leetcode.com/problems/jump-game-iii

Solution: I like to use the following thought process:

- We know we have to explore both situations: pos + arr[pos] and pos – array[pos]:

```
bool dfs(vector<int>& arr, int pos) {
        return dfs(arr, pos + arr[pos]) || dfs(arr, pos - arr[pos]);
}
```

- Add the most obvious base cases: **found zero**

```
bool dfs(vector<int>& arr, int pos) {
        if (arr[pos] == 0) return true;
...
```

- And then add the other obvious base case: **out of bounds, return false**

```
bool dfs(vector<int>& arr, int pos) {
        if (pos >= arr.size()) return false;
        if (arr[pos] == 0) return true;
```

- Then check visited:

```
bool dfs(vector<int>& arr, int pos, vector<bool>& visited) {
        if (pos >= arr.size()) return false;
        if (visited[pos]) return false;
        if (arr[pos] == 0) return true;
        visited[pos] = true;
        return dfs(arr, pos + arr[pos]) || dfs(arr, pos - arr[pos]);
}
```

Code – 1306. Jump Game III

Medium

LeetCode leetcode.com/problems/jump-game-iii

Code

Time: **O(n)** Space: **O(1)**

```
bool dfs(vector<int>& arr, int pos, vector<bool>& visited) {
    if (pos >= arr.size()) return false;
    if (visited[pos]) return false;
    if (arr[pos] == 0) return true;
    visited[pos] = true;
    return dfs(arr, pos + arr[pos], visited) || dfs(arr, pos - arr[pos], visited);
}

bool canReach(vector<int>& arr, int start) {
    vector<bool> visited(arr.size(), false);
    return dfs(arr, start, visited);
}
```