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

```
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)**

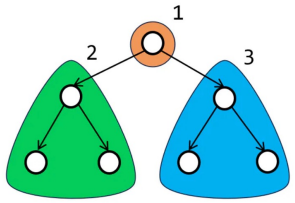
```
bool dfs(vector<int>& arr, int pos, vector<bool>& visited) {
    if (pos >= arr.size()) return false;
    if (visited[pos]) return false;
    if (arr[pos] == 0) return true;
    visited[pos] = true;
    return dfs(arr, pos + arr[pos], visited) || dfs(arr, pos - arr[pos], visited);
}

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

TREE

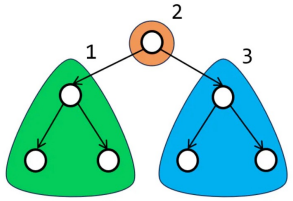
Tree Traversals

Depth-First Traversals

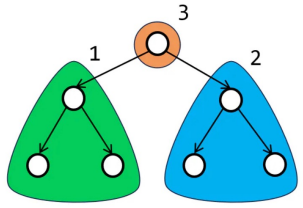
- **Pre-order**: Root – Left – Right



- **In-order**: Left – Root – Right



- **Post-order**: Left – Right – Root



Breadth-First Traversal (Level Order Traversal)

Visit every node on a level before moving to a lower level.

Image credits:
https://deriveit.org/coding/tree-dfs-113

Tree Traversals

Depth-First Traversals

Use a recursive algorithm to traverse according to the order

- **Pre-order**: Root – Left – Right

```
if (!root) return;
doSomething();
visit(node->left);
visit(node->right);
```

- **In-order**: Left – Root – Right

```
if (!root) return;
visit(node->left);
doSomething();
visit(node->right);
```

- **Post-order**: Left – Right – Root

```
if (!root) return;
visit(node->left);
visit(node->right);
doSomething();
```

Tree Traversals

Example of pre-order and in-order

```
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

// Pre-order traversal
void preorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    cout << root->val << " ";
    preorderTraversal(root->left);
    preorderTraversal(root->right);
}

// In-order traversal
void inorderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    inorderTraversal(root->left);
    cout << root->val << " ";
    inorderTraversal(root->right);
}
```