Tips

- When to build an **adjacency list** from the input data?

  - When the graph is **sparse**

  - When working with **non-grid graphs**

  - When you want to perform multiple or complex traversals and need fast neighbour lookups.

DFS Boilerplate

Recursive DFS

Time Complexity: **O(N + E)** Space Complexity: **O(N)**

```
void dfs(int node, const vector<vector<int>>& graph,
         vector<bool>& visited) {
    if (visited[node]) return;
    visited[node] = true;

    for (int neighbor : graph[node]) {
        dfs(neighbor, graph, visited);
    }
}

void runDFS(int n, const vector<vector<int>>& graph) {
    vector<bool> visited(n, false);
    dfs(0, graph, visited);  // start from node 0
}
```

N = Number of nodes
E = Number of edges

Iterative DFS

Time Complexity: **O(N + E)** Space Complexity: **O(N)**

```
void runDFS(int n, const vector<vector<int>>& graph) {
    vector<bool> visited(n, false);
    stack<int> stk;
    stk.push(0);  // start from node 0

    while (!stk.empty()) {
        int node = stk.top();
        stk.pop();

        if (visited[node]) continue;
        visited[node] = true;

        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                stk.push(neighbor);
            }
        }
    }
}
```

BFS Boilerplate

Iterative BFS

Time Complexity: **O(N + E)** Space Complexity: **O(N)**

```
void runBFS(int n, const vector<vector<int>>& graph) {
    vector<bool> visited(n, false);
    queue<int> q;
    q.push(0);  // start from node 0
    visited[0] = true;

    while (!q.empty()) {
        int node = q.front();
        q.pop();

        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}
```

N = Number of nodes
E = Number of edges

⏱️ Time Complexity notes

- Every node is enqueued/dequeued once
- Every edge is checked once

Problem – 841. Keys and Rooms

Medium

LeetCode leetcode.com/problems/keys-and-rooms

Problem

- You are given an **array of array rooms**, example:

  `[[1,2], [2],[0]]`
- **Each element** in the **outer array** represents a **room**
- **Each array** inside the array represents a **set of keys** that open the rooms
- **Rooms** are the **index of the array**
- In the example**, room 0** have the keys for **room 1 and 2**
- **Room 1** have the keys for **room 2**
- **Room 2** have the key for **room 0**
- **Room 0** is the only room unlocked. You start by visiting room 0, grab the key and unlock other rooms
- **Return true** if you can unlock all rooms

Solution – 841. Keys and Rooms

Medium

LeetCode leetcode.com/problems/keys-and-rooms

Solution

- This is a graph problem that can be solved using DFS
- Treat each room as a node
- Treat each set of keys as edges the goes from room A to B
- Visit room 0, and then start visiting the neighbours
- Once you visited all rooms/nodes, check if the visited size is the same as the number of exiting rooms

Code – 841. Keys and Rooms

Medium

LeetCode leetcode.com/problems/keys-and-rooms

Code

Time: **O(N + E)** Space: **O(N)** where N is the number of rooms and E the number of keys. Space complexity is N due to the visited set and call stack

```
void dfs(int room, vector<vector<int>>& rooms, unordered_set<int>& visited) {
    if (visited.count(room)) return;
    visited.insert(room);
    for (const auto& roomNumber : rooms[room]) {
        dfs(roomNumber, rooms, visited);
    }
}

bool canVisitAllRooms(vector<vector<int>>& rooms) {
    unordered_set<int> visited;
    dfs(0, rooms, visited);
    return rooms.size() == visited.size();
}
```

Problem – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

Problem Statement

- Given a node reference, create a deep copy of the graph
- The class node has two variables: val and neighbours

```
class Node {
  public int val;
  public List<Node> neighbors;
}
```

- **Output** is the node reference of the copy

Solution – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

Solution

- First check the edge cases (is the node null?)
- Create a hash map to store the nodes that is already created

  `unordered<int, Node*> graph;`
- Check if the current node already exists in the graph
- If not, create a new Node object and store in the hashmap
- Visit all the neighbors and add the neighbors to this current node

Code – Clone Graph

Medium

LeetCode https://leetcode.com/problems/clone-graph

```
std::unordered_map<int, Node*> graph;

Node* cloneGraph(Node* node) {
    if (node == NULL) {
        return NULL;
    }
    // does this node object exists?
    if (graph.find(node->val) == graph.end()) {
        // node wasn't visited yet, store in the hashmap
        graph[node->val] = new Node(node->val);
        // visit all neighnours
        for (const auto& n : node->neighbors) {
            graph[node->val]->neighbors.push_back(cloneGraph(n));
        }
    }
    return graph[node->val];
}
```

Problem – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Problem

- You are given the number of courses and a course pre-requisite array
- Course pre–requisite indicates the dependency between courses
- **Example**:

  `numCourses = 2, prerequisites = [[1,0],[0,1]]`

  To take course 1, you must take course 0 first

  to take course 0, you must take course 1 first
- In this example, this schedule is not possible since one course depends on the other
- Return if the schedule is valid or not

Solution – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Solution

- Model as a graph problem
- Create a dependency graph between courses: **course A** depends on **course B**
- If there is a cycle, **A** to **B** and **B** to **A**, the schedule is invalid
- For the implementation: first convert the schedule to adjacency list
- Use DFS and track two status: **VISITING** and **VISITED**
- Go over each node in the adjacency list, and perform a DFS
- Once you find a node which status is **VISITING**, you've detected a cycle

Code – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Code

Time: **O(n + p)** Space: **O(n + p)** where n is the number of courses and p the number of edges in the graph

```
enum class VisitState {
    NOT_VISITED,
    VISITING,
    VISITED
};

bool hasCycle(int node, const unordered_map<int, vector<int>>& adjList,
              unordered_map<int, VisitState>& visited) {
    // if we are revisiting a node in the current path, there's a cycle
    if (visited[node] == VisitState::VISITING) return true;

    // if we've already completed visiting this node, no need to check again
    if (visited[node] == VisitState::VISITED) return false;

    // mark the node as being visited
    visited[node] = VisitState::VISITING;

    for (int neighbor : adjList.at(node)) {
        if (hasCycle(neighbor, adjList, visited)) return true;
    }

    // mark the node as fully visited
    visited[node] = VisitState::VISITED;
    return false;
}
```

```
bool canFinish(int numCourses, const vector<vector<int>>& prerequisites) {
    // build the adjacency list: course -> list of its prerequisites
    unordered_map<int, vector<int>> adjList;
    for (const auto& dependencyPair : prerequisites) {
        int course = dependencyPair[0];
        int prerequisite = dependencyPair[1];
        adjList[course].push_back(prerequisite);
    }

    unordered_map<int, VisitState> visited;

    // check each course for cycles
    for (int course = 0; course < numCourses; ++course) {
        if (adjList.count(course)) {
            if (hasCycle(course, adjList, visited)) return false;
        }
    }

    return true; // no cycles detected
}
```

Code – 207. Course Schedule

Medium

LeetCode leetcode.com/problems/course-schedule

Code (simplified)

Time: **O(n + p)** Space: **O(n + p)** where n is the number of courses and p the number of edges in the graph

```
bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
    // construct the graph
    vector<vector<int>> adjList(numCourses);
    // states:
    // unvisited = 0, visiting = -1, visited = 1
    vector<int> visited(numCourses, 0);
    for (const auto& pre : prerequisites) {
        adjList[pre[1]].push_back(pre[0]);
    }
    for (int n = 0; n < adjList.size(); ++n) {
        if (hasCycle(adjList, visited, n /* starting node */)) {
            return false;
        }
    }
    return true;
}

bool hasCycle(vector<vector<int>>& adjList, vector<int>& visited, int node) {
    if (visited[node] == -1) return true;
    if (visited[node] == 1) return false;

    // visiting
    visited[node] = -1;
    for (const auto& n: adjList[node]) {
        if (hasCycle(adjList, visited, n)) {
            return true;
        }
    }
    // already visited
    visited[node] = 1;
}
```

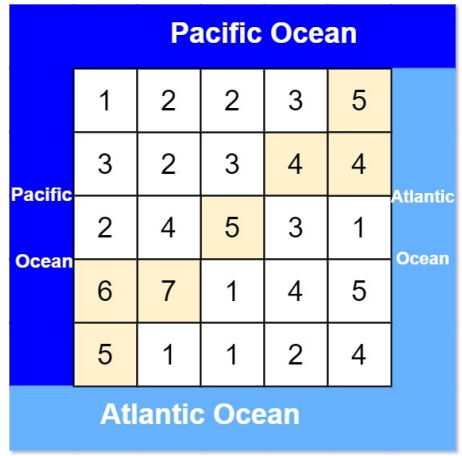
Problem – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Problem

- You are given a **matrix heights[m][n]** representing elevations on an island
- **Pacific Ocean** borders the **top and left**, **Atlantic Ocean** borders the **bottom and right**
- **Rainwater can flow** from a cell to its **north, south, east, or west** neighbor **if the neighbor's height is ≤ current**
- Find all coordinates **(r, c)** from which **water can flow to both oceans**
- Return a list of such coordinates: [[r1, c1], [r2, c2], ...]



Solution – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Solution 1 (ineficient)

- Iterate the whole matrix and perform a DFS
- Visit the neighbour if the value is less or equal the current value
- If it reaches some ocean, mark either atlantic or pacific as true
- Return true once both are true or false after finishing DFS

Code – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Code

Time: **O((m × n) ²)** Space: **O(m × n)**

```
private:
vector<vector<int>> directions = {{0,1},{0,-1},{1,0},{-1,0}};

public:
vector<vector<int>> pacificAtlantic(vector<vector<int>>& heights) {
    int m = heights.size();
    int n = heights[0].size();

    vector<vector<int>> result;

    for (int row = 0; row < m; ++row) {
        for (int col = 0; col < n; ++col) {
            bool pacific = false;
            bool atlantic = false;
            vector<vector<bool>> visited(m, vector<bool>(n, false));

            if (dfs(make_pair(row, col), heights, visited, pacific,
atlantic)) {
                result.push_back({row, col});
            }
        }
    }
    return result;
}
```

```
bool dfs(pair<int, int> coordinates, vector<vector<int>>& heights,
        vector<vector<bool>>& visited, bool& pacific, bool& atlantic) {
    if (pacific && atlantic) return true;
    int row = coordinates.first;
    int col = coordinates.second;
    if (visited[row][col]) return false;

    if (col == 0 || row == 0) pacific = true;
    if (col == heights[0].size() - 1 || row == heights.size() - 1)
         atlantic = true;

    visited[row][col] = true;

    for (const auto& d : directions) {
        int nextRow = row + d[0];
        int nextCol = col + d[1];
        if (nextRow < 0 || nextCol < 0 || nextRow >= heights.size() ||
            nextCol >= heights[0].size()) continue;
        if (heights[nextRow][nextCol] <= heights[row][col]) {
            if (dfs(make_pair(nextRow, nextCol), heights,
                    visited, pacific, atlantic)) {
                    return true;
            }
        }
    }
    return pacific && atlantic;
}
```

Solution – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Solution 2 (better)

- **Start** from the **border of pacific ocean** and find all reachable squares using DFS
- Then, **start** from **the border of atlantic ocean** and find all reachable squares using DFS
- Check the squares where both are accessible

Code – 417. Pacific Atlantic Water Flow

Medium

LeetCode leetcode.com/problems/pacific-atlantic-water-flow

Code

Time: **O(m × n)** Space: **O(m × n)**

```
private:
vector<vector<int>> directions = {{0,1},{0,-1},{1,0},{-1,0}};
public:
vector<vector<int>> pacificAtlantic(vector<vector<int>>& heights) {
    int m = heights.size();
    int n = heights[0].size();

    // squares where pacific is reachable
    vector<vector<bool>> pacific(m, vector<bool>(n, false));
    // squares where atlantic is reachable
    vector<vector<bool>> atlantic(m, vector<bool>(n, false));

    // check pacific and atlantic
    for (int row = 0; row < m; ++row) {
        dfs(row, 0, heights, pacific);
        dfs(row, n - 1, heights, atlantic);
    }
    for (int col = 0; col < n; ++col) {
        dfs(0, col, heights, pacific);
        dfs(m - 1, col, heights, atlantic);
    }
    // check where both are reachable
    vector<vector<int>> result;
    for (int row = 0; row < m; ++row) {
        for (int col = 0; col < n; ++col) {
            if (pacific[row][col] && atlantic[row][col]) {
                result.push_back({row,col});
            }
        }
    }
    return result;
}
```

```
void dfs(int row, int col, vector<vector<int>>& heights,
vector<vector<bool>>& visited) {
    // check bounds
    if (visited[row][col]) return;

    visited[row][col] = true;

    for (const auto& d : directions) {
        int nextRow = d[0] + row;
        int nextCol = d[1] + col;
        if (nextRow < 0 || nextCol < 0 ||
                nextRow >= heights.size() || nextCol >= heights[0].size())
continue;
        if (heights[nextRow][nextCol] >= heights[row][col]) {
            dfs(nextRow, nextCol, heights, visited);
        }
    }
}
```

Problem – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Problem

- You are given a **matrix m x n**
- The **matrix** represents a map where “1” is land and “0” is water
- **Return** the total number of island: “connected” ones form one island
- Example:

Input:

```
grid =  {{"1","1","1","1","0”},
         {"1","1","0","1","0”},
         {"1","1","0","0",”1"],
         {"0","0","0",”1",”1” }}
```

Output: 2

Solution – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Solution

- **Loop** through each element in the matrix
- Once you find “1”, count the land and;
- Perform a **DFS (or BFS)** around each ‘1’ cell
- **When traversing**, mark “0” after visited
- **Continue** until you finish processing all elements in the matrix

Code – 200. Number of Islands

Medium

LeetCode leetcode.com/problems/number-of-islands

Code

Time: **O(m × n)** Space: **O(m × n)** As we visit each cell only once, time complexity is the size of the matrix. The space complexity comes from the DFS recursion stack.

```
int numIslands(vector<vector<char>>& grid) {
    int m = grid.size(), n = grid[0].size();
    int count = 0;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (grid[i][j] == '1') {
                count++;
                dfs(grid, i, j);
            }
        }
    }
    return count;
}

void dfs(vector<vector<char>>& grid, int i, int j) {
    int m = grid.size(), n = grid[0].size();

    if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == '0')
        return;

    grid[i][j] = '0';

    dfs(grid, i + 1, j);
    dfs(grid, i - 1, j);
    dfs(grid, i, j + 1);
    dfs(grid, i, j - 1);
}
```

Problem – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Problem

- You are given an unsorted **array of integers** nums
- **Return the length** of the **longest consecutive** elements sequence
- Algorithm **must run** on **O(n)** time

Solution – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Solution

- **Create** a set with the same elements of **nums**
- Loop through this set and check if it is the beginning of a sequence
- **Example**:

  `[4,100,3,2,101]`

  4: Look up 3 (found) → 4 is **NOT** the beginning (skip it)

  100: Look up 99 (not found) → 100 **IS** the beginning → count 100, 101 → length = 2

  3: Look up 2 (found) → 3 is **NOT** the beginning (skip it)

  2: Look up 1 (not found) → 2 **IS** the beginning → count 2, 3, 4 → length = 3

  101: Look up 100 (found) → 101 is **NOT** the beginning (skip it)
- Track the **maximum length**

Code – 128. Longest Consecutive Sequence

Medium

LeetCode leetcode.com/problems/longest-consecutive-sequence

Code

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

```
int longestConsecutive(vector<int>& nums) {
    unordered_set<int> seq(nums.begin(), nums.end());
    int maxStreak = 0;
    for (const auto& n : seq) {
        // is beginning of sequence?
        // previous exist?
        int streak = 0;
        if (seq.find(n - 1) == seq.end()) {
            // check next
            int currNum = n;
            while (seq.find(currNum) != seq.end()) {
                streak++;
                currNum++;
            }
        }
        maxStreak = max(maxStreak, streak);
    }

    return maxStreak;
}
```

Problem – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Problem

- You are given a graph of **n** nodes, and an array of **edges (source, destination)**
- Edges indicates the edge between node **source** and **destination**
- Find the total number of isolated components (subgraphs)
- **Example:**

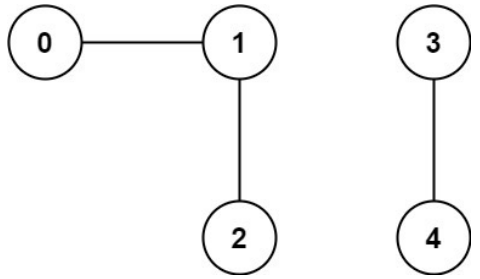
  Input:

  `n = 5, edges = [[0,1],[1,2],[3,4]]`

  Output: 2

  0 is connected to 1, 1 is connected to 2

  3 is a new subgraph connected to 4



Solution – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Solution

- Build an adjacency list from edges
- From each node, check if its visited
- If it is not visited, mark as a new “component” or subgraph
- Perform a DFS from that node

Code – 323. Number of Connected Components

Medium

LeetCode leetcode.com/problems/number-of-connected-components-in-an-undirected-graph

Code

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

```
int countComponents(int n, vector<vector<int>>& edges) {
    vector<bool> visited(n, false);
    vector<vector<int>> adjList(n);

    // build adjancency list
    for (const auto& edge: edges) {
        adjList[edge[0]].push_back(edge[1]);
        adjList[edge[1]].push_back(edge[0]);
    }
    int totalComponents = 0;
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(adjList, visited, i);
            totalComponents++;
        }
    }

    return totalComponents;
}

void dfs(vector<vector<int>>& adjList, vector<bool>& visited, int node) {
    if (visited[node]) return;
    visited[node] = true;
    for (const auto& neighbour : adjList[node]) {
        dfs(adjList, visited, neighbour);
    }
}
```

Problem – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

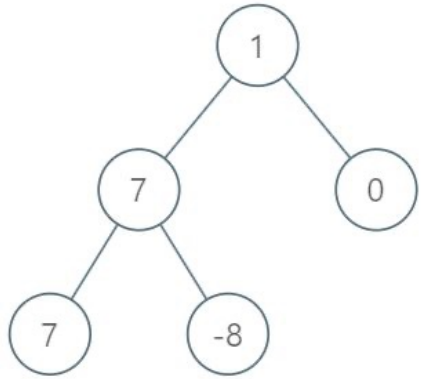
Problem Statement

- Given the root of a binary tree, find the smallest level with the maximum sum
- For example, the tree below has the follow sums for each level:

  level 1 (root) = 1

  level 2 = 7 + 0 = 7

  level 3 = 7 – 8 = -1
- Therefore, **level 2** has the maximum sum



Solution – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

Solution

- Have a queue with the nodes for the current level
- Sum the values from that level by taking the nodes from the queue
- Example, we know that level 1 has one node. Hence, pop the first node from the queue

  If level 2 has 2 nodes, pop two nodes, sum the values
- In addition, add left and right to the end of the queue to process the next level

Code – Maximum Level Sum of a Binary Tree

Medium

LeetCode https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree

```
int maxLevelSum(TreeNode* root) {
    std::queue<TreeNode*> nodes;
    int currentLevel = 0;
    int maxLevel = 1;
    int maxSum = INT_MIN;

    nodes.push(root);

    // traverse the graph
    while(!nodes.empty()) {
        int levelSum = 0;
        int levelSize = nodes.size();
        currentLevel++;

        // sum the values in current level
        for (int i = 0; i < levelSize; ++i) {
            TreeNode* node = nodes.front();
            levelSum += node->val;
            nodes.pop();

            if (node->left) nodes.push(node->left);
            if (node->right) nodes.push(node->right);
        }

        if (levelSum > maxSum) {
            maxLevel = currentLevel;
            maxSum = levelSum;
        }
    }

    return maxLevel;
}
```

Problem – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Problem

- You are given a starting URL startURL and an interface HtmlParser with a method getUrls(url)
- getUrls(url) returns a vector of strings with the URLs found on the given page
- Start crawling from startUrl and recursively visit all reachable URLs
- Only visit URLs that share the same hostname as startUrl
- Return a list of all visited URLs (in any order)

Solution – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Solution

- This is a graph problem framed as an object
- Both BFS and DFS are valid options
- Each url represent a node, and `getUrls` retrieve the neighbours
- Visit each node and add to the result if they have the same hostname

Code – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Code (BFS)

Time: **O(n + m)** Space: **O(n + w)** where **n** is the number of unique URLs, **m** the number of links (edges), **w** is the explicit queue

```
vector<string> crawl(string startUrl, HtmlParser htmlParser) {
    string hostname = getHostname(startUrl);
    queue<string> urls;
    unordered_set<string> visited;
    vector<string> result;

    urls.push(startUrl);
    visited.insert(startUrl);
    result.push_back(startUrl);

    while(!urls.empty()) {
        string url = urls.front();
        urls.pop();
        for (const auto& u : htmlParser.getUrls(url)) {
            // is it the same hostname?
            // have I already visited this one?
            if (visited.count(u)) continue;
            if (getHostname(u) != hostname) continue;
            urls.push(u);
            result.push_back(u);
            visited.insert(u);
        }
    }
    return result;
}
```

```
string getHostname(const string& url) {
    int start = url.find("://") + 3;
    int end = url.find("/", start);
    return url.substr(start, end - start);
}
```

Problem – 1236. Web Crawler

Medium

LeetCode https://leetcode.com/problems/web-crawler

Code (DFS)

Time: **O(n + m)** Space: **O(n + h)** where **n** is the number of unique URLs, **m** the number of links (edges), **h** is the recursive stack

```
string getHostname(const string& url) {
    int start = url.find("://") + 3;
    int end = url.find("/", start);
    return url.substr(start, end - start);
}

void dfs(const string& hostname, const string& url, HtmlParser& htmlParser,
unordered_set<string>& visited, vector<string>& result) {
    if (visited.count(url)) return;
    result.push_back(url);
    visited.insert(url);

    for (const auto& u: htmlParser.getUrls(url)) {
        if (getHostname(u) == hostname) {
            dfs(hostname, u, htmlParser, visited, result);
        }
    }
}

vector<string> crawl(string startUrl, HtmlParser htmlParser) {
    string hostname = getHostname(startUrl);
    unordered_set<string> visited;
    vector<string> result;
    dfs(hostname, startUrl, htmlParser, visited, result);
    return result;
}
```

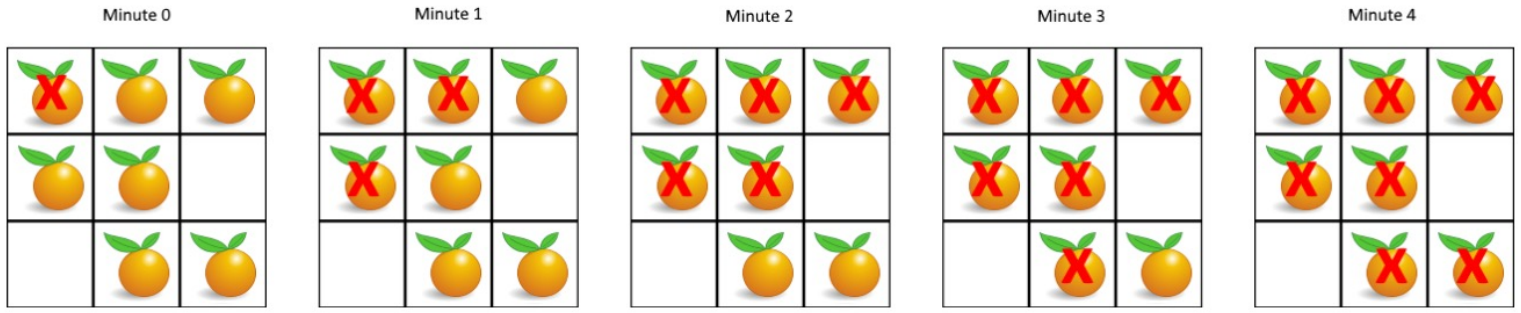
Problem – 994. Rotting Oranges

Medium

LeetCode leetcode.com/problems/rotting-oranges

Problem

- You are given a **m x n** grid
- Each cell represents the following:
- **0** is an empty cell
- **1** is a fresh orange
- **2** is a rotten orange
- Every minute (snapshot), any fresh orange is contaminated by adjacent oranges
- Return the **minimum number of minutes** required for all fresh oranges to become rotten
- If it is **impossible** to rot all fresh oranges, return -1



Solution – 994. Rotting Oranges

Medium

LeetCode leetcode.com/problems/rotting-oranges

Solution

- This is another BFS problem: for each rotten orange, visit adjacent fresh oranges
- Start by finding all rotten oranges in the grid. No need to convert grid to adjacent list
- Initialize a `queue<pair<int, int>>` to perform the BFS. Add the rotten oranges to this queue
- Start the traversal

  while (!q.empty()) { ... }
- **This part is important!** you want to calculate the "minutes". So you have to first go over the current size of the queue and "process" all the elements, meaning, rot the adjacent fresh oranges
- Use directions vector to calculate the adjacent positions:

  const vector<pair<int, int>> directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
- Keep track of the number of fresh oranges
- By the end, check if the number of fresh oranges is zero. If so, return minutes, or -1 otherwise.

Code – 994. Rotting Oranges

Medium

LeetCode leetcode.com/problems/rotting-oranges

Code

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

```
int orangesRotting(vector<vector<int>>& grid) {
    // go over the grid, find the rotten ones
    // count the number of fresh oranges
    // add to a queue
    // queue should contain the positions x,y
    int fresh = 0;
    // we'll increase the minutes before visiting
    int minutesElapsed = -1;
    queue<pair<int, int>> q;

    int m = grid.size();
    int n = grid[0].size();

    for (int row = 0; row < m; ++row) {
        for (int col = 0; col < n; ++col) {
            if (grid[row][col] == 1) ++fresh;
            if (grid[row][col] == 2) q.push({row, col});
        }
    }

    // no fresh oranges
    if (fresh == 0) return 0;

    // at each minute: pop all the queue, visit the neighbours
    // set a fresh one to rotten
    // decrease the number of fresh
    vector<pair<int, int>> directions = {{0,1},{0,-1},{1,0},{-1,0}};
    while(!q.empty()) {
        int qSize = q.size();
        // at each minute, it rottens all oranges
        // therefore, fully consumes the queue
        minutesElapsed++;

        for (int i = 0; i < qSize; ++i) {
            auto [row, col] = q.front();
            q.pop();

            // visit neighbours, check boundaries
            // and if its not visited yet
            for (const auto& [dRow, dCol] : directions) {
                int nRow = row + dRow;
                int nCol = col + dCol;
                if (nRow >= 0 && nCol >= 0 && nRow < m && nCol < n && grid[nRow][nCol] == 1) {
                    grid[nRow][nCol] = 2;
                    fresh--;
                    q.push({nRow, nCol});
                }
            }
        }
    }
    // once you reach the end, count if rotten == fresh
    return (fresh == 0) ? minutesElapsed : -1;
}
```

Backtracking

Common pattern in backtracking

- Useful for problems like: generating all permutations / combinations
- N-Queens
- Sudoku
- Letter combinations

```
void backtrack(/* problem-specific args */) {
    if (/* base case */) {
        // store result
        return;
    }

    for (/* each choice */) {
        // make choice
        state.push_back(choice);

        // explore further
        backtrack(/* updated args */);

        // undo choice (backtrack)
        state.pop_back();
    }
}
```

Problem – 46. Permutations

Medium

LeetCode leetcode.com/problems/permutations

Problem

- You are given an array of numbers, **Example**:

  `nums = [1,2]`

- Return all possible the permutations:

  `output = [[1,2], [2,1]]`

Solution – 46. Permutations

Medium

LeetCode leetcode.com/problems/permutations

Solution

- Permutations problem can be solved using backtracking
- Create a function following the template:

```
backtrack(path, result, nums) {
    if path.size == nums.size()
        add to the result and return
     for i in nums
        continue if already used this i
        mark i as used
        add to the current path
        backtrack(...)
        mark i as unused
        remove from the current path
```

Code – 46. Permutations

Medium

LeetCode leetcode.com/problems/permutations

Code

Time: **O(n! * n)** Space: **O(n)** **Time:** there are n! permutations. Each permutation takes O(n) time to construct. **Space**: the path and used vector grow as n grows. n represents the size of nums

```
void backtrack(vector<int>& path, vector<int>& nums, vector<bool>& used, vector<vector<int>>& result) {
    if (path.size() == nums.size()) {
        result.push_back(path);
        return;
    }
    for (int i = 0; i < nums.size(); ++i) {
        if (used[i]) continue;
        used[i] = true;
        path.push_back(nums[i]);
        backtrack(path, nums, used, result);
        path.pop_back();
        used[i] = false;
    }
}
vector<vector<int>> permute(vector<int>& nums) {
    vector<int> path;
    vector<vector<int>> result;
    vector<bool> used(nums.size(), false);
    backtrack(path, nums, used, result);
    return result;
}
```

Problem – 17. Letter Combinations of a Phone Number

Medium

LeetCode leetcode.com/problems/letter-combinations-of-a-phone-number

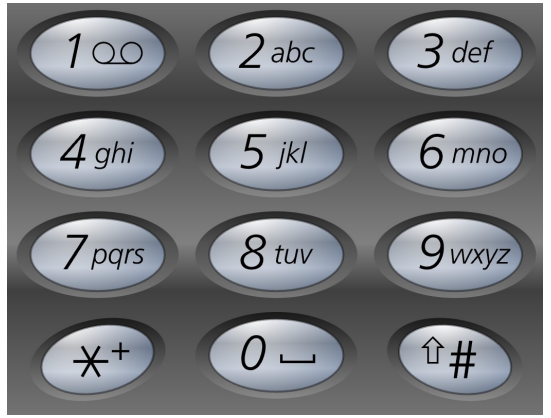
Problem

- You are given a string **digits** containing numbers such as “2” or “234” etc
- Each digit correspond to a digit of a phone number
- The digits map to a group of characters from the phone. For example, 2 → “abc”, 3 → “def” ...
- Return all possible letter combinations from the digits
- **Example**

  Input: 23

  Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

  2 maps to “abc” and 3 maps to “def”, so generate all combinations



Solution – 17. Letter Combinations of a Phone Number

Medium

LeetCode leetcode.com/problems/letter-combinations-of-a-phone-number

Solution

- Map the keyboard to a vector of strings:

  `std::vector<string> = { “”, “”, “abc”, “def”, “ghi”, "jkl", "mno", "pqrs", "tuv", "wxyz"}`

  First 2 characters are empty to map exactly the phone digit position
- Use backtracking to generate all combinations
- **Example**: digits “2” and “3” maps to “abc” and “def”:

  visit “a”

  visit “d”

  reached the end of the digits, add “ad”

  backtrack to “a”

  visit “e”

  reached the end of the digits, add “ae”

  …

Problem – 17. Letter Combinations of a Phone Number

Medium

LeetCode leetcode.com/problems/letter-combinations-of-a-phone-number

Code

Time: **O(4^n)** Space: **O($n * 4^n$)** where n is the number of digits.

For each digit, you have to generate a combination of max 4 characters (the maximum phone digits, for example, 7 represents “pqrs”)

```
void backtrack(vector<string>& result, string& current,
               const vector<string>& phone, string& digits, int index) {
    if (index == digits.size()) {
        result.push_back(current);
        return;
    }
    // retrieve current digit
    char currentDigit = digits[index];
    // retrieve chars from that digit
    string chars = phone[currentDigit - '0'];

    // go over each char to backtrack
    for (const char& c : chars) {
        current.push_back(c);
        backtrack(result, current, phone, digits, index + 1);
        current.pop_back();
    }
}

vector<string> letterCombinations(string digits) {
    if (digits.empty()) return {};

    const vector<string> phone = {
        "", "", "abc", "def", "ghi",
        "jkl", "mno", "pqrs", "tuv", "wxyz"
    };
    vector<string> result;
    string current;
    backtrack(result, current, phone, digits, 0);
    return result;
}
```

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

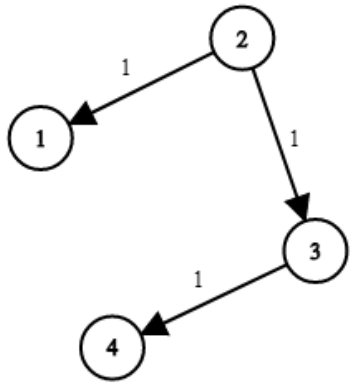
LeetCode leetcode.com/problems/network-delay-time

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