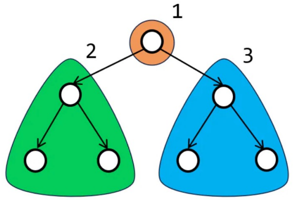
TREE

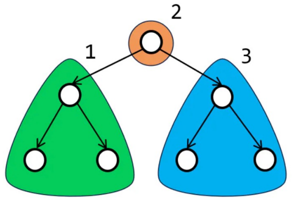
Tree Traversals

Depth-First Traversals

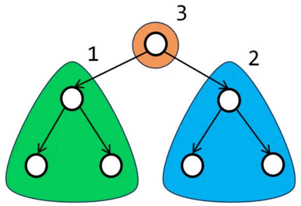
- **Pre-order**: Root – Left – Right



- **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);
}
```

Tree Traversals

Example of post-order and level-order

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

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

// Level-order traversal using a queue
void levelOrderTraversal(TreeNode* root) {
    if (root == nullptr) return;
    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        TreeNode* current = q.front();
        q.pop();
        cout << current->val << " ";
        if (current->left != nullptr) q.push(current->left);
        if (current->right != nullptr) q.push(current->right);
    }
}
```

BFS Using Stack

BFS with std::stack

- This might be useful for problems when you want to return and resume (for example, 872. Leaf-Similar Trees)

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

// Pre-order traversal
void bfs(std::stack<TreeNode*>& tree) {
    while(!tree.empty()) {
         TreeNode* root = tree.top();
         tree.pop();
         // do something ...
         if (root->right) tree.push(root->right);
         if (root->left) tree.push(root->left);
    }
}
```

Problem – Maximum Depth of Binary Tree

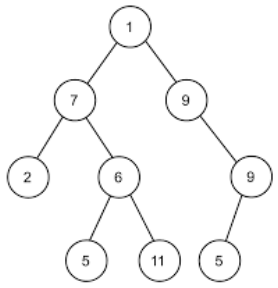
Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

Problem Statement

- Given the root of a binary tree, find the *maximum depth*
- **Example:**

  root = [1,7,9,2,6,null,9,null,null,5,11,5,null]
- **Output**: 4



Solution – Maximum Depth of Binary Tree

Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

Solution

- Perform **post-order** traversal: left - right - root
- Recursively go left and right to find each value
- Return the max of each one

Code – Maximum Depth of Binary Tree

Easy

LeetCode https://leetcode.com/problems/maximum-depth-of-binary-tree

```
int maxDepth(TreeNode* root) {
    if (!root) return 0;
    // find max left
    int maxLeft = maxDepth(root->left);
    // find max right
    int maxRight = maxDepth(root->right);
    // return max +1 (account for root)
    return max(maxLeft, maxRight) + 1;
}
```

Problem – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Problem

- You are given the root of two trees
- Write a function to check if they are the same
- **Example:**

  p = [1,2,3], q = [1,2,3]

  Output: true

Solution – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Solution

- Traverse both trees (**p** and **q**) recursively and check if the nodes are the same
- Start by the base case:

  are **p** and **q** null? return true
- One of them are null? return false, because they should be the same
- Finally, check if **p->val** is equal to **q->val** and also for both and left, recursively

Code – 100. Same Tree

Easy

LeetCode leetcode.com/problems/same-tree

Code

Time: **O(n)** where n is the number of nodes Space: **O(h)** where h is the height of the tree. Best case is usually O(log n) for balanced trees, but skewed trees is usually O(n)

```
bool isSameTree(TreeNode* p, TreeNode* q) {
    // base case: leaf is null. If both are null, then return true
    if (!p && !q) return true;
    // if both are not NULL, then they must have value.
    // If one of them doesn't have value, then they're different, return false
    if (!p || !q) return false;
    // they must have the same value
    // as any other nodes in the tree
    return p->val == q->val &&
           isSameTree(p->left, q->left) &&
           isSameTree(p->right, q->right);
}
```

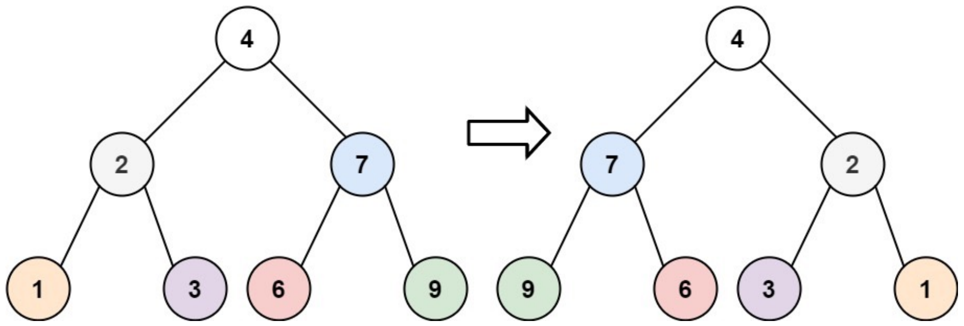
Problem – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Problem

- You are given the root of a binary tree
- Invert the tree and return the root
- **Example:**



Solution – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Solution

- Recursively traverse the tree
- Create a new pointer **temp** that points to **left** node
- Set **left** node to **right**
- Set **right** node to **temp**
- Call the function recursively for **left** and **right**
- Return root

Code – 226. Invert Binary Tree

Easy

LeetCode leetcode.com/problems/invert-binary-tree

Code

Time: **O(n)** Space: **O(h)** where h is the height of the tree

```
TreeNode* invertTree(TreeNode* root) {
    // base case
    if (!root) return nullptr;

    // create a new pointer to left
    TreeNode* temp = root->left;
    // invert
    root->left = root->right;
    root->right = temp;

    // recursively invert left and right
    invertTree(root->left);
    invertTree(root->right);

    return root;
}
```

Problem – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

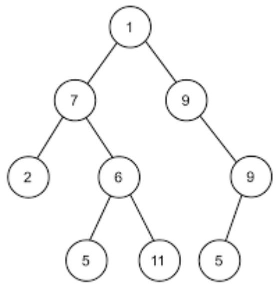
Problem Statement

- It is given the **root** of a binary tree and an integer **target sum**
- **Example:**

  root = [1,7,9,2,6,null,9,null,null,5,11,5,null]

  target sum = 10
- Return true if there is a path from root to leaf that adds up to 10
- **Output**: true

  Node 1 + Node 7 + Node 2 = 10



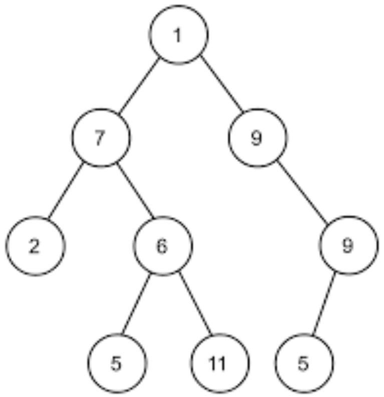
Solution – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

Solution

- Start from root node (1)
- Subtract from target number (example 10 – 1 = 9)
- Continue going down the tree, until the target is 0, return true
- After visiting all nodes, if the target is not zero, return false



Code – Path Sum

Easy

LeetCode https://leetcode.com/problems/path-sum

```
bool hasPathSum(TreeNode* root, int targetSum) {
    if (!root) {
        return false;
    }
    // we want targetSum to be zero
    targetSum -= root->val;
    // if there is no left, no right, we've reached the end of the path
    // so if the targetSum is zero, then the nodes summed up to the targetSum
    if (!root->left && !root->right && targetSum == 0) {
        return true;
    }
    // propagate to left and right
    return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);
}
```

Also, a small performance tweak can be made by avoiding writing *targetSum:* `targetSum -= root->val`
This will avoid a memory write access, making the calculation directly in the CPU, but also at a cost of readability

```
    if (!root->left && !root->right && targetSum - root->val == 0) {
     …
    return hasPathSum(root->left, targetSum - root->val) || hasPathSum(root->right, targetSum - root->val);
```

Problem – 102. Binary Tree Level Order Traversal

Medium

LeetCode leetcode.com/problems/binary-tree-level-order-traversal

Problem Statement / Solution / Code

Time: **O(-)** Space: **O(-)**

- ...

Problem – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Problem

- Design an algorithm to serialize and deserialize a binary tree
- You have to build two interfaces: serialize that returns a string, and deserialize that returns the whole tree as TreeNode pointer
- The string can be represented at any format (comma-separated, space separated etc)

Solution – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Solution

- **Serialize**: traverse the tree pre-order, and append its value to a string

  Null value should also be represented

  Example: [1,2,null,null,3 ...]

  Call “traverse” to do it recursively

- **Deserialize**: split the string into tokens

  read each token and re-build the tree by adding a new node

  Call “buildTree” to do it recursively

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Code Time: O() Space: O()

```
string serialize(TreeNode* root) {
    // traverse the tree in pre-order: root, left, right
    // generate a string with comma separator,
    // example: 1,2,N,N,3 ...
    string result;
    traverse(root, result);
    return result;
}

TreeNode* deserialize(string data) {
    // split the input data
    vector<string> tokens = split(data);
    // index to be used to access the elements from tokens recursively.
    // Hence, we need to create it here to pass by reference.
    // Note that index is bounded by the number of tokens, so it won't overflow
    int index = 0;
    TreeNode* root = buildTree(tokens, index);
    return root;
}
```

continue...

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

```
TreeNode* buildTree(vector<string>& tokens, int& index) {
        // read the current token based on the index
        const string& token = tokens[index];
        // increment index before checking for null
        ++index;
        // base case: null node
        if (token == "N") {
            return nullptr;
        }
        // build root
        TreeNode* node = new TreeNode(stoi(token));
        // build left
        node->left = buildTree(tokens, index);
        // build right
        node->right = buildTree(tokens, index);
        return node;
    }
```

```
// traverse in pre-order (root, left, right)
// and append the values to the string 's'
// append 'N' if it is NULL
void traverse(TreeNode* root, string& s) {
    if (!s.empty()) s += ",";
    // base case, we need to append null
    if (!root) {
        s += "N";
        return;
    }
    // visit root
    s += to_string(root->val);
    // visit left
    traverse(root->left, s);
    // visit right
    traverse(root->right, s);
}

// helper function in C++ to split string
vector<string> split(const string& s) {
    vector<string> result;
    stringstream ss(s);
    string token;
    while(getline(ss, token, ',')) {
        result.push_back(token);
    }
    return result;
}
```

Code – 297. Serialize and Deserialize Binary Tree

Hard

LeetCode leetcode.com/problems/serialize-and-deserialize-binary-tree

Some interesting alternative to split

- C++ 23 have an interesting way to split using std::views::split

```
vector<string> split(string s) {
    auto result = s |
        views::split(',') |
        views::transform([](auto&& subRange) {
                return string(subRange.start(), subRange.end());
                });
}
```

- To understand, this follow a structure similar to unix pipes:

  echo "123,N,556" | split | transform

- std::views::split returns ranges, something like:

  [range("123"), range("N"), range("556")]

- std::views::transform converts each subrange into an actual string

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Problem

- You are given the root of a binary tree root and the root of another binary tree subRoot
- Determine whether subRoot is a **subtree** of root.
- A subtree of a binary tree is a node in the tree along with all of its descendants
- The tree itself is also considered a subtree.

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Solution

- Define a helper function isSameTree(a, b) that checks if two trees rooted at a and b are identical in both structure and node values.
- Traverse the root tree, and for each node:
- Use isSameTree(node, subRoot) to check if a matching subtree starts at that node.
- Return true if any such match is found; otherwise, return false.

Problem – 572. Subtree of Another Tree

Easy

LeetCode leetcode.com/problems/subtree-of-another-tree

Code

Time: **O(m * n)** Space: **O(h)** where n is the number of nodes of the tree and m the number of nodes of the subtree, and h the height of the tree.

```cpp
bool isSame(TreeNode* q, TreeNode* r) {
    // both are null, so they're the same
    if (!q && !r) return true;
    // if they're not null, both must be not null
    if (!q || !r) return false;
    // now check the values
    if (q->val != r->val) return false;
    // check left and right
    return isSame(q->left, r->left) && isSame(q->right, r->right);
}

bool isSubtree(TreeNode* root, TreeNode* subRoot) {
    if (!root) return false;
    // Check starting from the root first
    if (isSame(root, subRoot)) {
        return true;
    }
    // they are not the same starting from the root,
    // but still subRoot may be in the middle of root. So check it recursively
    return isSubtree(root->left, subRoot) || isSubtree(root->right, subRoot);
}
```

Problem – 105. Construct Binary Tree from Preorder and Inorder Traversal Medium

LeetCode leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal

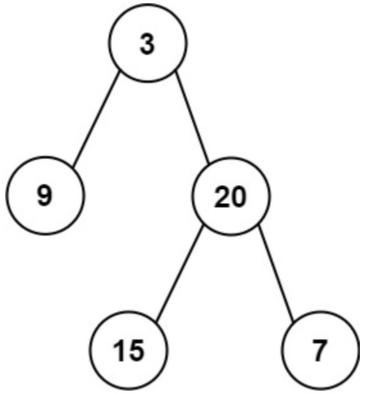
Problem

- You are given two **array of integers: preorder and inorder**
- **preoder** is the pre-order traversal of a binary tree
- **inorder** is the in-order traversal of a binary tree
- Based on those two arrays, construct the binary tree
- **Example**

  Input: preorder = [3,9,20,15,7]

  inorder = [9,3,15,20,7]

  Output: the head pointer of the binary tree



Solution – 105. Construct Binary Tree from Preorder and Inorder Traversal

Medium

LeetCode leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal

Solution

- 2

Code – 105. Construct Binary Tree from Preorder and Inorder Traversal

Medium

LeetCode leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal

Code

Time: **O(-)** Space: **O(-)**

- 1
- 2

Problem – 98. Validate Binary Search Tree

Medium

LeetCode leetcode.com/problems/validate-binary-search-tree

Problem

- You are given the root of a **binary tree**
- Determine if the tree is a valid **binary search tree** according to the following:
- The left subtree contains all elements less than the node's key
- The right subtree contains all elements greater than the node's key
- Both left and right must also be binary search trees

Solution – 98. Validate Binary Search Tree

Medium

LeetCode leetcode.com/problems/validate-binary-search-tree

Solution

- For each node, there is a **maximum** and a **minimum** value where node lies
- The range of the root is (-∞, ∞)
- Traverse from **top to bottom**
- If you are checking the right subtree, all the elements must be higher than a minimum value
- When traversing right, make sure to update the minimum value
- The same is true for the left subtree – all the elements must be less than the maximum value

Solution – 98. Validate Binary Search Tree

Medium

LeetCode leetcode.com/problems/validate-binary-search-tree

Solution (example)

```
    5
   / \
  1   6
     / \
    3   7
```

- The minimum of all elements at the right of “5” should be greater than 5
- So this tree is invalid
- For each node, the values must range between a min and a max:

node = (max, min)

node 5 = (-∞, +∞) can be any value

node 1 = (-∞, 5) must be any value less than 5

node 6 = (5, +∞) this node can be any value greater than 5

node 3 = (5, 6) must be greater than the subtree the node belongs and less than the immediate parent

Code – 98. Validate Binary Search Tree

Medium

LeetCode leetcode.com/problems/validate-binary-search-tree

Code

Time: **O(n)** Space: **O(n)** where n is the number of the nodes in the tree

```
// Use long long for min/max due the constraints
bool isValid(TreeNode* root, long long min, long long max) {
    // base case: if we reach the bottom, that means we've
    // checked all nodes along the way and everything is fine
    if (!root) return true;

    // compare the min/max values
    if (root->val <= min || root->val >= max) return false;
    // go right and left. When going right, the minimum should be updated and
    // when going left, the maximum should be updated by its immediate parent
    return isValid(root->right, root->val, max) && isValid(root->left, min, root->val);
}

bool isValidBST(TreeNode* root) {
    return isValid(root, LONG_MIN, LONG_MAX);
}
```

Problem – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

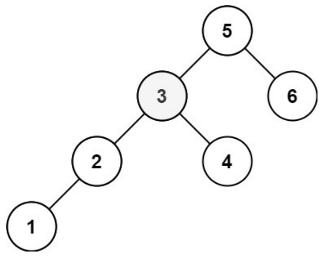
Problem Statement / Solution

- You are given the **root** of a binary search tree and an **integer k**
- Find the k^{th} smallest value
- **Example**

  From all values in the tree: 1,2,3,4,5,6

  k = 3 so find the 3^{th} smallest value

  Output is 3: 1,2,**3**,4,5,6 (3th)



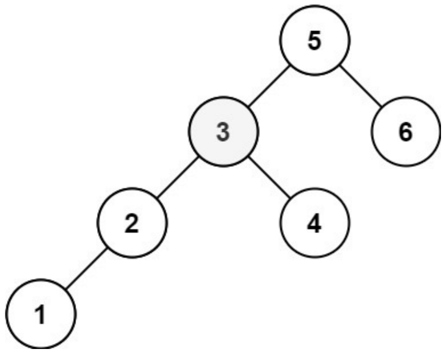
Solution – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

Solution

- Note that the smallest element is in the left leaf
- Therefore, there is an order from small → big values from left → root → right
- Perform in-order traversal **k** times and stop in the desired node



Code – Kth Smallest Element in a BST

Medium

LeetCode leetcode.com/problems/kth-smallest-element-in-a-bst

Code

Time: **O(k)** Space: **O(h)** where **h** is the height of the tree

```
// in-order traversal: left, node: right
void traverse(TreeNode* node, int& k, int& result) {
    // base case
    if (!node) return;
    // visit left first
    traverse(node->left, k, result);
    // visit node
    k--;
    if (k == 0) {
        result = node->val;
        return;
    }
    // visit right
    traverse(node->right, k, result);
}

int kthSmallest(TreeNode* root, int k) {
    // perform pre-order traversal
    int result;
    traverse(root, k, result);
    return result;
}
```

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Problem

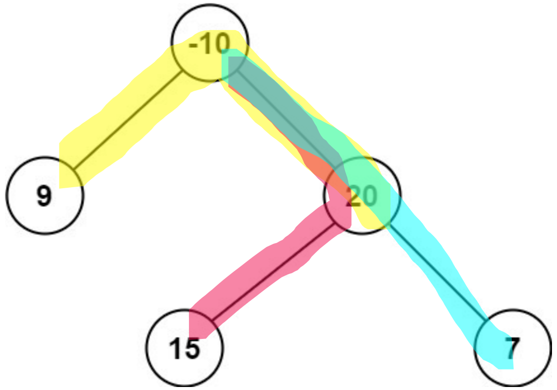
- You are given the root node of a binary tree
- Return the **max path sum** of any path
- A path can be linear (from the root all the way down to the leaf) or the three node: root, left and right)
- A path can start at any node
- **Example:**

9 → -10 → 20 is a valid path

9 → -10 → 20 → 7 is **NOT** a valid path

-10 → 20 → 15 is a valid path

20 → 7 is a valid path



Solution – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Solution

- Use **post-order** traversal (bottom-up recursion)
- At each node:

  - Recursively compute left and right max path gains

  - Consider all 3 possible paths:

    1. Turn path: left + root + right

    2. Linear path: root + left

    3. Linear path: root + right

  - Also consider just the root (if the children is negative)
- Track the maximum path seen so far
- Only return linear path (root + one child) upward to maintain the valid structure
- Also, prune negative gain before returning

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Code

Time: **O(n)** Space: **O(h)** where **n** is the number of nodes and **h** is the height of the tree.

```
int findMaxSum(TreeNode* root, int& maxSum) {
    if (!root) return 0;
    int left = findMaxSum(root->left, maxSum);
    int right = findMaxSum(root->right, maxSum);
    // 1st possible path: exactly the only 3 nodes: root, right and left
    int threeNodes = left + right + root->val;
    // 2nd possible path, linear recursive path: root + left
    int secondPath = root->val + left;
    // 3rd possible path, linear recurrsive path: root + right
    int thirdPath = root->val + right;

    // check if we should consider left, right or only root itself
    int bestPath = max({root->val, secondPath, thirdPath});

    // maxSum can be the accumulated 2nd and 3rd (linear path)
    // or the threeNodes path
    maxSum = max({maxSum, bestPath, threeNodes});

    // Prune subtree: we start from the bottom, so we can set 0
    // to ignore left or right path
    return max(0, bestPath);
}

int maxPathSum(TreeNode* root) {
    int maxSum = INT_MIN;
    findMaxSum(root, maxSum);
    return maxSum;
}
```

Problem – 124. Binary Tree Maximum Path Sum

Hard

LeetCode leetcode.com/problems/binary-tree-maximum-path-sum

Code (compact)

Time: **O(n)** Space: **O(h)** where **n** is the number of nodes and **h** is the height of the tree.

```
int find(TreeNode *node, int& totalMax) {
    if (!node) return 0;
    int leftGain = max(0, find(node->left, totalMax));
    int rightGain = max(0, find(node->right, totalMax));
    int currentMax = node->val + leftGain + rightGain;
    totalMax = max(totalMax, currentMax);
    return node->val + max(leftGain, rightGain);
}

int maxPathSum(TreeNode* root) {
    int totalMax = INT_MIN;
    find(root, totalMax);
    return totalMax;
}
```

Problem – 872. Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Problem Statement

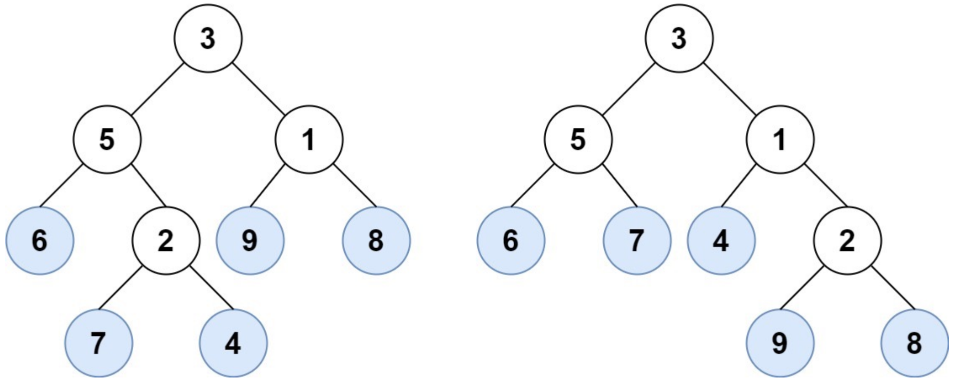
- You are given two trees
- The goal is to compare if they have the same leaves
- The leaves should be in the same order
- Example:

  First tree:

  leaves = 6,7,4,9,8 (blue nodes)

  Second tree:

  leaves = 6,7,4,9,8
- Return true if the leaves are the same



Solution – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Solution

- Get the first leaf value from tree 1
- Get the first leaf value from tree 2
- Compare, if they are different, return false immediately
- Otherwise, continue finding the next leaf value for tree 1 and 2

Implementation

- Create two stacks **stack<TreeNode*> left** and **stack<TreeNode*> right**
- Add the

Code – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Code

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2   Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```cpp
// returns the value of the leaf, or -1 if empty
int getLeaf(stack<TreeNode*>& tree) {
    // tree is a reference, we will always pop an element from it
    while(!tree.empty()) {
        // get the top element from the stack
        TreeNode* node = tree.top();
        // already visited, so remove from stack
        tree.pop();
        // is this a leaf?
        if (!node->left && !node->right) {
            // yes, return the value
            return node->val;
        }
        // push the right FIRST to the stack
        if (node->right) tree.push(node->right);
        // left should be on top of the stack
        if (node->left) tree.push(node->left);
    }
    return -1;
}
```

```cpp
bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    // initialize the stacks, add root1 and root2
    std::stack<TreeNode*> leftTree, rightTree;
    leftTree.push(root1);
    rightTree.push(root2);

    while(true) {
        // get the leaves to compare
        int leaf1 = getLeaf(leftTree);
        int leaf2 = getLeaf(rightTree);
        // exit immediately if one leaf is different
        if (leaf1 != leaf2) return false;
        // stop when there are no leaves left
        if (leaf1 == -1 || leaf2 == -1) break;
    }
    return true;
}
```

Code – Leaf-Similar Trees

Easy

LeetCode leetcode.com/problems/leaf-similar-trees

Code (another approach)

Time: **O(n + m)** where n and m are the numbers of nodes for trees 1 and 2 Space: **O(h1 + h2)** where h1 and h2 represents the height of the tree

```
void extractLeafs(TreeNode* node, vector<int>& leafValues) {
    // base case, return
    if (!node) return;
    // if it looks like a leaf, no left child
    // like a leaf, no right child like a leaf,
    // then it's probably  a leaf
    // add to the vector
    if (!node->left && !node->right) {
        leafValues.push_back(node->val);
    }
    // continue looking at left and right
    extractLeafs(node->left, leafValues);
    extractLeafs(node->right, leafValues);
}

bool leafSimilar(TreeNode* root1, TreeNode* root2) {
    vector<int> tree1Values;
    vector<int> tree2Values;
    // extract all leafs from tree 1
    extractLeafs(root1, tree1Values);
    // extract all leafs from tree 2
    extractLeafs(root2, tree2Values);
    // compare
    return tree1Values == tree2Values;
}
```

Problem – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Problem Statement

- You are given a binary tree and have to find "**good**" **nodes**
- A **good node** is a **node** where the values in the path are always than the **node**
- The **root node** is always a **good node**
- Example:

  - root 3 is a good node

  left side:

  - left **leaf 1** is not a good node because **1 < 3**

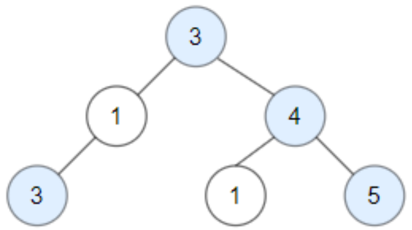
  - **leaf 3** is a good node because **3 > 1** and **3 == 3**

  right side:

  - **leaf 4** is a good node because **4 > 3**

  - **leaf 1** is not a good node because **1 < 4**

  - **leaf 5** is a good node because **5 > 4 > 3**



Solution – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Solution

- Use DFS traversal to explore the tree from the root to all leaf nodes
- As you traverse, **keep track of the maximum value** along the path from root to node
- Update max value once you find a node value greater than the max value
- **Recursive logic**

  Base case: if the node is *`nullptr`*, return 0

  At each node:

  - Compare its value to max so far
  - If it is a good node, increase a local count
  - Recursively repeat this process for the left and right children, passing along the updated max value

Code – 1448. Count Good Nodes in Binary Tree

Medium

LeetCode leetcode.com/problems/count-good-nodes-in-binary-tree

Code

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

```
int traverse(TreeNode* root, int maxValue) {
    if (!root) return 0;
    // is this a good node?
    int count = 0;
    if (root->val >= maxValue) {
        maxValue = root->val;
        count = 1;
    }
    count += traverse(root->left, maxValue);
    count += traverse(root->right, maxValue);
    return count;
}

int goodNodes(TreeNode* root) {
    if (!root) return 0;
    return traverse(root, root->val);
}
```