**Hello everyone,**

I hope you're enjoying your vacation… because I'm here to stop it! Just kidding, but now, with all seriousness, I want to take a brief moment to share a few words with you.

For some of you, I won't be able to attend the final test, and there's a chance this might be our last interaction in the formal framework of university, at least in the teacher-student posture for the LDS discipline. Maybe I'll meet some of you again in future labs, but we never know.

In any case, I'm truly glad I had the opportunity to meet you and be part of your learning experience, especially during your first semester of college, a big and probably scary step until you get used to it. Now, I won't say college is easy as it depends on your passion and even more on your determination. But one thing is certain: it's not impossible.

My most important advice? **Take things as they come, do them one by one, but with all your determination and discipline, do them like you love it.** I wish you good luck in everything you're about to do, and be brave.

As I've said before on various occasions, if you ever need help and I can help you with something, even if it's just a simple talk, advice, or something bigger (maybe even help with another discipline), don't hesitate to contact me. If I can help, I'll do my best.
Just keep in mind (with good regard to the law): **NO, I'm not going to help you hack someone's phone because they made you mad!**

But jokes aside, if that's the case, you can get my contact from your group representatives, they have it and will share this document through them.

Finally, I want to wish you all a **Merry Christmas and a Happy New Year!** 🎄

Sincerely,

**Denis** 🦖

## 🟢 Problem 1: Count Non-Leaf Nodes in a Generic Tree

**Problem Statement:**

You are given a generic tree where each node contains an integer value and can have zero or more children. Write a recursive function that returns the number of nodes that are **not leaves** (i.e., nodes that have at least one child).

**Constraints:**

- The function must be recursive.
- If the tree is empty, return 0.

**Function Signature:**

```python
def count_internal_nodes(root: GNode) -> int
```

**Example:**

```
Tree:
10
├── 5
│   ├── 2
│   └── 3
└── 7
    └── 6

Output: 3  (nodes 10, 5, 7)
```

**Problem 2: Preorder Traversal of a Binary Tree**

**Problem Statement:**

Write a function that returns the preorder traversal of a binary tree as a list of integers. Preorder means visiting nodes in the order: **root → left → right**.

**Constraints:**

- The function must be recursive.
- If the tree is empty, return an empty list.

**Function Signature:**

```python
def preorder(root: BNode) -> list[int]
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Output: [5, 3, 2, 4, 8, 10]
```

## 🟠 Problem 3: Count Non-Leaf Nodes in a Binary Tree (Postorder Traversal)

**Problem Statement:**

Write a recursive function that counts the number of nodes in a binary tree that are **not leaves** (nodes with at least one child). The traversal must be done in **postorder** (left → right → root).

**Constraints:**

- Use recursion.
- If the tree is empty, return 0.

**Function Signature:**

```
def count_internal_postorder(root: BNode) -> int
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Internal nodes: 5, 3, 8 → Output: 3
```

**Problem 4: Level-Order Traversal of a Binary Tree**

**Problem Statement:**

Write a function that returns the level-order traversal of a binary tree as a list of lists, where each inner list contains the values of nodes at that level. Level-order means visiting nodes level by level from top to bottom.

**Constraints:**

- You may use a queue (e.g., collections.deque).
- If the tree is empty, return an empty list.

**Function Signature:**

```python
def level_order(root: BNode) -> list[list[int]]
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Output: [[5], [3, 8], [2, 4, 10]]
```

**Problem 5: Height of a Binary Tree**

**Problem Statement:**

Write a recursive function that returns the **height** of a binary tree. The height is the number of edges on the longest path from the root to a leaf. Define the height of an empty tree as -1.

**Constraints:**

- Use recursion.
- If the tree has only one node, height = 0.

**Function Signature:**

```python
def height(root: BNode) -> int
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Height: 2
```

**Problem 6: Count Leaf Nodes in a Generic Tree**

**Problem Statement:**

Write a recursive function that returns the number of **leaf nodes** in a generic tree. A leaf node is a node with no children.

**Constraints:**

- Use recursion.
- If the tree is empty, return 0.

**Function Signature:**

```
def count_leaves(root: GNode) -> int
```

**Example:**

```
Tree:
10
├── 5
│   ├── 2
│   └── 3
└── 7
    └── 6

Leaves: 2, 3, 6 → Output: 3
```

**Problem 7: Sum of Values at a Given Depth in a Generic Tree**

**Problem Statement:**

Write a function that returns the sum of all node values at a given depth k in a generic tree. The root is at depth 0.

**Constraints:**

- Use recursion.
- If k is greater than the tree height, return 0.

**Function Signature:**

```python
def sum_at_depth(root: GNode, k: int) -> int
```

**Example:**

```
Tree:
10
├── 5
|   ├── 2
|   └── 3
└── 7
    └── 6

Depth 1: nodes 5, 7 → Output: 12
```

**Problem 8: Maximum Width of a Binary Tree**

**Problem Statement:**

Write a function that returns the maximum width of a binary tree. The width of a level is the number of nodes present at that level.

**Constraints:**

- Use level-order traversal.
- If the tree is empty, return 0.

**Function Signature:**

```
def max_width(root: BNode) -> int
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Widths: level 0 → 1, level 1 → 2, level 2 → 3 → Output: 3
```

**Problem 9: Validate a Binary Search Tree (BST)**

**Problem Statement:**

Write a function that checks if a binary tree is a valid Binary Search Tree (BST). A BST satisfies:

- Left subtree values < root value
- Right subtree values > root value
- Applies recursively to all nodes

**Constraints:**

- Use recursion.
- Return True if valid, False otherwise.

**Function Signature:**

```python
def is_valid_bst(root: BNode) -> bool
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Output: True
```

**Problem 10: Find the Kth Smallest Element in a BST**

**Problem Statement:**

Write a function that returns the k-th smallest element in a BST (1-indexed). If k is invalid, return None.

**Constraints:**

- Use inorder traversal (sorted order).
- Assume all values are unique.

**Function Signature:**

```python
def kth_smallest(root: BNode, k: int) -> int | None
```

**Example:**

```
Tree:
      5
     / \
    3   8
   / \    \
  2   4    10

Inorder: [2, 3, 4, 5, 8, 10]
k = 3 → Output: 4
```

## 🔴 Problem 11: Lowest Common Ancestor (LCA) in a Binary Tree

**Problem Statement:**

Write a function that returns the **lowest common ancestor (LCA)** of two nodes in a binary tree, given their values. The LCA of two nodes is the deepest node that has both nodes as descendants (a node can be a descendant of itself).

**Constraints:**

- Use recursion.
- If one or both values are not present, return None.

**Function Signature:**

```python
def lca(root: BNode, a: int, b: int) -> BNode | None
```

**Example:**

```
Tree:
        8
      /   \
     3     10
    / \      \
   1   6      14
      / \     /
     4   7   13

LCA of 4 and 7 → Output: 6
LCA of 4 and 14 → Output: 8
```

## 🔴 Problem 12: Diameter of a Binary Tree and Path

**Problem Statement:**

Write a function that returns the **diameter** of a binary tree and the actual path of nodes forming this diameter. The diameter is the length (number of edges) of the longest path between any two nodes in the tree.

**Constraints:**

- Use recursion.
- Return both the diameter and the path as a list of node values.

**Function Signature:**

```
def diameter_with_path(root: BNode) -> tuple[int, list[int]]
```

**Example:**

```
Tree:
        8
      /   \
     3     10
    / \      \
   1   6      14
      / \     /
     4   7   13

Diameter: 5
Path: [1, 3, 6, 7, 8, 10, 14] (or any valid longest path)
```

## 🔴 Problem 13: ASCII Visual Representation of a Binary Tree

**Problem Statement:**

Write a function that prints a binary tree in a **visual ASCII format** showing branches and spacing. The output should clearly represent the tree structure.

**Constraints:**

- Use recursion.
- The output should be printed to the console.

**Function Signature:**

```python
def print_tree_ascii(root: BNode) -> None
```

**Example:**

```
     8
   /   \
  3     10
 / \      \
1   6      14
   / \     /
  4   7   13
```

## 🦖 A Final Challenge: Build Your Own Tree App 🦖

As we've often discussed, real-world and industry applications of the topics and data structures we study are everywhere, or at least we've tried to show you glimpses of that. Most of you haven't yet had the chance to experience what it feels like to integrate these concepts into something tangible.

Based on this, I propose a little exercise in imagination and creativity, **play the role of a developer and an integrator**. Using the functions you've developed for the problems above, imagine each one as a small component. Now, combine these components thoughtfully to create a **small application** that brings them all together. Think of them as puzzle pieces, when arranged properly, they form a complete picture: in our case, a working app.

**What should this app do?**

- Allow the user to **build a tree** (generic, binary, or BST).
- Provide a **simple GUI** (or a clean console menu) to interact with the tree.
- Enable the user to **visualize the tree** in a readable format.
- Offer buttons or options to run the algorithms you implemented: traversals, counts, height, width, validation, and more.
- Most importantly: **extend it as you wish**. Add new features, improve the interface, or invent something creative that makes sense for trees. There are no limits, this is your playground.

**Why do this?**

Because this is how real-world development works, combining smaller, well-tested pieces into a bigger system. It's also a great way to practice thinking like an engineer, not just a coder.

**Extra Tips**

- If you decide to take on this challenge, we recommend creating a **Git repository** for version control and uploading your work online (GitHub, GitLab, etc.).
- If you want feedback or help, just ask, **contact me or Andrei** and we'll gladly review your work, give an opinion or just lend a hand if you need assistance and guide.
- Collaboration is welcome, you can work solo or in pairs.

**Final Note**

The intent behind this challenge is simple, to prepare you for entering an industry that is becoming more competitive and full of challenges, from AI disruption to economic uncertainty. But with efforts like this, you'll build a stronger foundation of knowledge and skills that will serve you well.

Remember, **every big step starts with small projects like this**. So, if you're eager, take the leap. Experiment, innovate, and have fun. You're building not just an app, you're building confidence and capability for your future.

**Good luck, and keep pushing forward, you've got this!**