TIP101 | Intro to Technical Interview Prep

Intro to Technical Interview Prep Fall 2025 (@ Section 3 | Tuesdays and Thursdays 5PM - 7PM PT)
Personal Member ID#: **134071**

Need help? Post on our **class slack channel** or email us at **support@codepath.org**

Session 1: Binary Trees

Overview

in this session, students will delve into evaluating mathematical expressions in full binary trees, finding corresponding nodes in cloned trees, checking for path sums, and determining if a binary tree is balanced. Additionally, they will practice replacing node values with the sum of their subtrees.

Through these exercises, students will deepen their understanding of tree evaluations, balancing techniques, and tree transformations. This session aims to build a solid foundation in handling complex tree structures and operations, preparing students for advanced problem-solving scenarios involving binary trees.

> You can find all resources from today including the session deck, session recording, and more on the resources tab

🚀 Part 1 : Instructor Lead Session

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.

👷 Part 2: Breakout Session

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as “solving the problem” - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
    - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff team will specify which tool to use for this class!
- Screen-share an implementation of your proposed solution
- Independently follow-along, or create an implementation, in your own IDE.

Your program leader/s will indicate which code sharing tool/s to use as a group, and will help break down or provide specific scaffolding with the main concepts above.

▶ **Note on Expectations**

🔎 Problem Solving Approach

We will approach problems using the six steps in the UMPIRE approach.

UMPIRE: Understand, Match, Plan, Implement, Review, Evaluate.

We'll apply these six steps to the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem
- **Match** identifies common approaches you've seen/used before
- **Plan** a solution step-by-step, and
- **Implement** the solution
- **Review** your solution
- **Evaluate** your solution's time and space complexity and think critically about the advantages and disadvantages of your chosen approach.

Breakout Problems Session 1

💡 Unit 9 Cheatsheet

To help your learning journey with binary trees, we've put together a guide to common concepts and syntax you will use throughout Unit 9 breakout problems. Use this cheatsheet as a quick reference guide as you work through the problems below.

▼ Problem Set Version 1

Problem 1: Is Symmetric Tree

Given the `root` of a binary tree, return `True` if the tree's left and right subtrees are mirrors of each other (i.e., tree is symmetric around its center). Return `False` otherwise.

Evaluate the time complexity of your function

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_symmetric(root):
        pass
```

Example Usage:

```
Example Tree #1:

       1
     /   \
    /     \
   2       2
  / \     / \
 3   4   4   3



Input: root = 1
Expected Output: True


Example Tree #2:

        1
      /   \
     /     \
    2       2
     \       \
      3       3


Input: root = 1
Expected Output: False
```

Problem 2: Root-to-Leaf Paths

Given the `root` of a binary tree, return a list of *all root-to-leaf paths in **any order***.

A **leaf** is a node with no children.

Evaluate the time complexity of your function.

```
        self.val = val
        self.left = left
        self.right = right

def binary_tree_paths(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

  1
 / \
2   3
 \
  5

Example Input: root = 1
Expected Output: ["1->2->5", "1->3"]
["1->3", "1->2->5"] is also valid

Example Input Tree #2:

  1

Example Input: root = 1
Expected Output: ["1"]
```

Problem 3: Minimum Difference in BST

Given the `root` of a binary search tree, return the minimum difference between the values of any two different nodes in the tree.

Evaluate the time complexity of your function.

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def min_diff_in_bst(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

    4
   / \
  2   6
 / \
1   3

Example Input: root = 4
Expected Output: 1
Explanation: The smallest difference between any two nodes is 1 (2 - 1 = 1, 3 - 2 = 1)

Example Input Tree  #2:

   1
  / \
 0  48
    / \
   12 49

Example Input: root = 1
Expected Output: 1
Explanation: The smallest difference between any two nodes is 1 (1 - 0 = 1)
```

▶ ✨ AI Hint: Representing Infinite Values

Problem 4: Increasing Order Search Tree

Given the `root` of a binary search tree, rearrange the tree in in-order so that the leftmost node of the tree is now the root of tree and every node has no left child and only one right child.

Return the root of the modified tree

Evaluate the time complexity of your function.

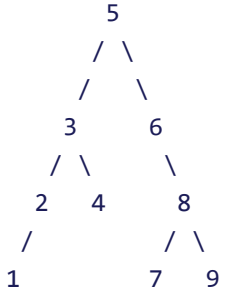
```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def increasing_bst(root):
        pass
```

Example Usage:

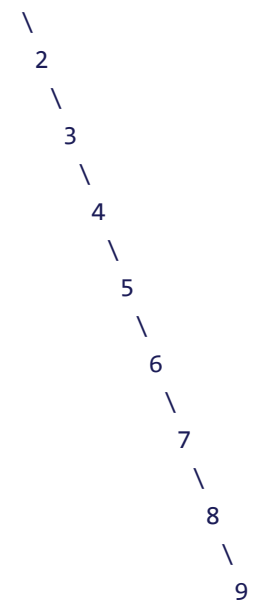
Example Input: root = 5
Expected Output: root = 1

Expected Output Tree #1:

Example Input Tree #2:



Input: root = 5
Expected Output: root = 1
Expected Output Tree #2:



Path Courses

Problem 5: Equal Tree Split

Given the `root` of a binary tree, return `True` if removing an edge between two nodes can split the tree into two trees with an equal number of nodes. Return `False` otherwise.

Evaluate the time complexity of the function.

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def can_split(root):
        pass
```

Example Usage:

```
Example Input Tree #1:

       1
      / \
     /   \
    2     3
   / \     \
  4   5     7

Example Input: root = 1
Expected Output: True
Explanation: Deleting the edge between node 1 and its left child, node 2 gives the following
two trees, each of size 3

  Tree 1    Tree 2
              1
               \
    2           3
   / \           \
  4   5           7



Example Input Tree #2:

       1
      /  \
     /    \
    2      3
   / \    / \
  4   5  6   7
```

```
Explanation: It is not possible to split the tree into two trees of equal size by deleting
an edge
```

Close Section

▸ Problem Set Version 2

▸ Problem Set Version 3