TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)
Personal Member ID#: 117667

Session 2: Review

Session Overview

Congratulations on making it to the final session! 🎉 In this session, students will review everything that has been covered in the course so far, practicing matching the algorithmic strategies and data structures covered to different easy and medium level Leetcode problems.

> You can find all resources from today including session slide decks, session recordings, and more on the resources tab

📊 Part 1 : Instructor Led 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

To build a long-term organized approach to problem solving, we'll start with three main steps. We'll refer to them as **UPI: Understand, Plan, and Implement**.

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

We will learn to:

- **Understand** the problem,
- **Plan** a solution step-by-step, and
- **Implement** the solution

▶ Comment on UPI

▶ UPI Example

ℹ️ Note: Testing your Linked Lists (Printing)

The following function takes in the `head` of a linked list and prints out the values of each node int he list with an `->` between values of linked nodes. You may copy the function below to use it as needed while you complete the problem sets.

```
def print_linked_list(head):
    current = head
    if not head:
        print("Empty List")
    while current:
        print(current.value, end=" -> " if current.next else "\n")
        current = current.next
```

ℹ️ Note: Testing your Binary Tree (Printing)

To keep the amount of starter code manageable, we have chosen not to include a function to print a binary tree as part of each relevant problem statement. You may instead copy the function in the drop-down below `print_tree()` and use it as needed while you complete the problem sets.

▼ Print Binary Tree Function

Accepts the root of a binary tree and prints out the values of each node level by level from left to right. Values of `None` are used to indicate a null child node between non-null children on the same level. Prints `"Empty"` for an empty tree.

```
from collections import deque

# Tree Node class
class TreeNode:
    def __init__(self, value, left=None, right=None):
        self.val = value
        self.left = left
        self.right = right

def print_tree(root):
    if not root:
        return "Empty"
    result = []
    queue = deque([root])
    while queue:
        node = queue.popleft()
        if node:
            result.append(node.val)
            queue.append(node.left)
            queue.append(node.right)
        else:
            result.append(None)
    while result and result[-1] is None:
        result.pop()
    print(result)
```

Example Usage:

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

root = Node(1, Node(2, Node(4)), Node(3, Node(5), Node(6)))

print_tree(root)
print_tree(None)
```

Example Output:

```
[1, 2, 3, 4, None, 5, 6]
'Empty'
```

ℹ️ Note: Testing your Binary Tree (Generating a Tree)

Now that you have practice manually building trees for testing in previous sessions, we are providing a function that builds binary trees based off of a list of values to speed up the testing process. We have chosen not to include this function in the starter code for each problem to keep the length of problems manageable. You may instead copy the function in the drop-down below `build_tree()` and use it as needed while you complete the problem sets.

▼ Build Binary Tree Function

Takes in a list `values` where each element in the list corresponds to a node in the binary tree you would like to build. The values should be in level order (from top to bottom, left to right). Use `None` to indicate a null child between non-null children on the same level.

Some problems may ask you to build a tree where nodes have both keys and values. This function may be used to build trees with just values *and* trees with both keys and values:

- If building a tree with only values, `values` should be given in the form: `[value1, value2, value3, ...]`.
- If building a tree with both keys and values `values` should be given in the form `[(key1, value1), (key2, value2), (key3, value3), ...]`.

Returns the `root` of the binary tree made from `values`.

```
from collections import deque

# Tree Node class
class TreeNode:
  def __init__(self, value, key=None, left=None, right=None):
      self.key = key
      self.val = value
      self.left = left
      self.right = right

def build_tree(values):
  if not values:
      return None

  def get_key_value(item):
      if isinstance(item, tuple):
          return item[0], item[1]
      else:
          return None, item

  key, value = get_key_value(values[0])
  root = TreeNode(value, key)
  queue = deque([root])
  index = 1

  while queue:
      node = queue.popleft()
      if index < len(values) and values[index] is not None:
          left_key, left_value = get_key_value(values[index])
          node.left = TreeNode(left_value, left_key)
          queue.append(node.left)
      index += 1
      if index < len(values) and values[index] is not None:
          right_key, right_value = get_key_value(values[index])
          node.right = TreeNode(right_value, right_key)
          queue.append(node.right)
      index += 1

  return root
```

Example Usage:

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

tree_with_just_values = [1, 2, 3, 4, None, 5, 6]
val_tree = build_tree(tree_with_just_values)

"""
          (1, 'A')
          /       \
       (2, 'B')   (3, 'C')
      /           /      \
  (4, 'D')    (5, 'E')   (6, 'F')
"""

tree_with_keys_and_values = [(1, 'A'), (2, 'B'), (3, 'C'), (4, 'D'), None, (5, 'E
key_val_tree = build_tree(tree_with_keys_and_values)

# Using print_tree() function included above
print_tree(val_tree)
print_tree(key_val_tree) # Only values will be printed
```

Example Output:

```
[1, 2, 3, 4, None, 5, 6]
['A', 'B', 'C', 'D', None, 'E', 'F']
```

Breakout Problems Session 2

▾ Standard Problem Set Version 1

Problem 1: Valid Anagram

Given two strings `s` and `t`, write a function `is_anagram()` that returns `True` if `t` is an anagram of `s` and `False` otherwise.

```
def is_anagram(s, t):
    pass
```

Example Usage:

```
print(is_anagram("anagram", "nagaram"))
print(is_anagram("rat", "car"))
```

Example Output:

```
True
False
```

Problem 2: Count Binary Substrings

Given a binary string `s`, return the number of non-empty substrings that have the same number of `0`'s and `1`'s, and all the `0`'s and all the `1`'s in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

```
def count_binary_substrings(s):
    pass
```

Example Usage:

```
print(count_binary_substrings("00110011"))
print(count_binary_substrings("10101"))
```

Example Output:

```
6
Example 1 Explanation: There are 6 substrings that have equal number of consecutive
"01", "1100", "10", "0011", and "01".
Notice that some of these substrings repeat and are counted the number of times they
is not a valid substring because all the 0's (and 1's) are not grouped together.

4
Example 2 Explanation: There are 4 substrings: "10", "01", "10", "01" that have equa
1's and 0's.
```

Problem 3: Diameter of a Binary Tree

Given the `root` of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

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

def tree_diameter(root):
    pass
```

Example Usage:

```
"""
     1
    /  \
   2    3
  / \
 4   5
"""
# Using build_tree() function included at top of page
tree_1 = build_tree([1, 2, 3, 4, 5])
print(tree_diameter(tree_1))

"""
   1
  /
 2
"""
tree_2 = build_tree([1, 2])
print(tree_diameter(tree_2))
```

Example Output:

```
3
Example 1 Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

1
```

Problem 4: Meeting Rooms

Given an array of meeting time intervals where `intervals[i] = [start`$_i$`, end`$_i$`]`, return `True` if a person could attend all meetings and `False` otherwise.

```
def can_attend_meetings(intervals):
    pass
```

Example Usage:

```
intervals_1 = [[0,30],[5,10],[15,20]]
intervals_2 = [[7,10],[2,4]]

print(can_attend_meetings(intervals_1))
print(can_attend_meetings(intervals_2))
```

Example Output:

```
False
True
```

Problem 5: Best Time to Buy and Sell Stock

You are given an array `prices` where `prices[i]` is the price of a given stock on the `ith` day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return `0`.

```
def max_profit(prices):
    pass
```

Example Usage:

```
prices_1 = [7,1,5,3,6,4]
prices_2 = [7,6,4,3,1]

print(max_profit(prices_1))
print(max_profit(prices_2))
```

Example Output:

```
5
Example 1 Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profi
Note that buying on day 2 and selling on day 1 is not allowed because you must buy b

0
Example 2 Explanation: In this case, no transactions are done and the max profit = 0
```

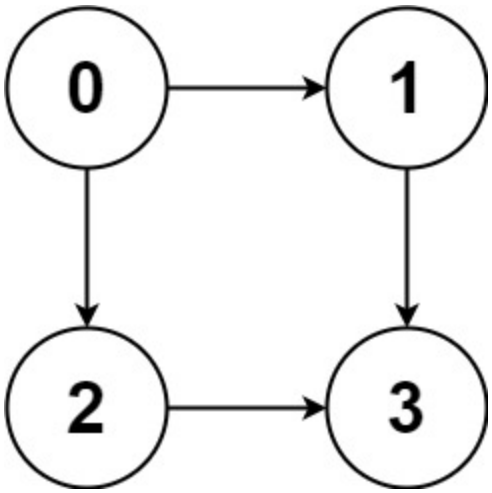
Problem 6: Find All Paths From Source to Target

Given a directed acyclic graph (DAG) of `n` nodes labeled from `0` to `n - 1`, find all possible paths from node `0` to node `n - 1` and return them in any order.

The graph is given as follows: `graph[i]` is a list of all nodes you can visit from node `i` (i.e., there is a directed edge from node `i` to node `graph[i][j]`).

```
def all_paths(graph):
    pass
```

Example Usage 1:

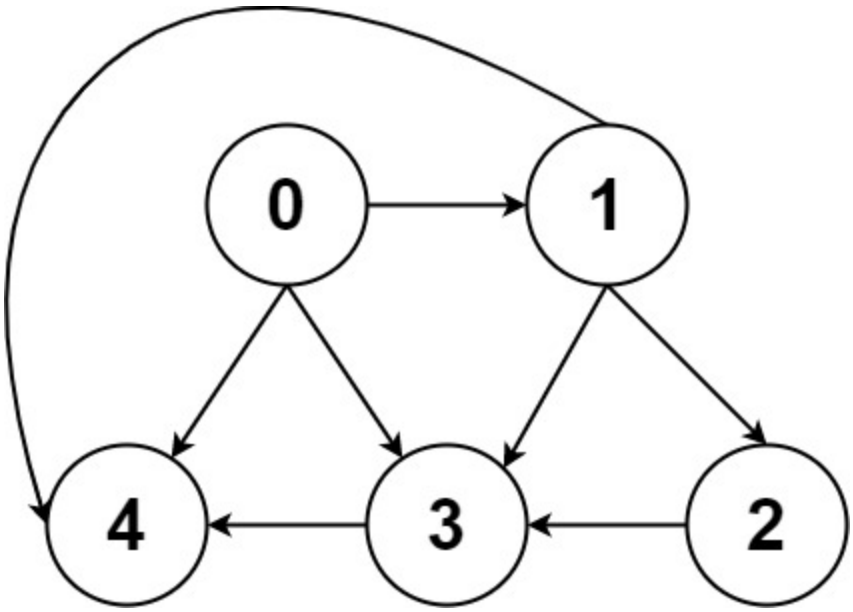



```
graph_1 = [[1,2],[3],[3],[]]
print(all_paths(graph_1))
```

Example Output 1:

```
[[0,1,3],[0,2,3]]
Example 1 Explanation: There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.
```

Example Usage 2:




```
graph_2 = [[4,3,1],[3,2,4],[3],[4],[]]
print(all_paths(graph_2))
```

Example Output 2:

```
[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]
```

▾ Standard Problem Set Version 2

Problem 1: Sort Array by Increasing Frequency

Given an array of integers `nums`, sort the array in increasing order based on the frequency of the values. If multiple values have the same frequency, sort them in decreasing order.

Return the sorted array.

```
def freq_sort(nums):
    pass
```

Example Usage:

```
print(freq_sort([1,1,2,2,2,3]))
print(freq_sort([2,3,1,3,2]))
print(freq_sort([-1,1,-6,4,5,-6,1,4,1]))
```

Example Output:

```
[3,1,1,2,2,2]
Example 1 Explanation: '3' has a frequency of 1, '1' has a frequency of 2, and
'2' has a frequency of 3.

[1,3,3,2,2]
Example 2 Explanation: '2' and '3' both have a frequency of 2, so they are sorted
in decreasing order.

Output: [5,-1,4,4,-6,-6,1,1,1]
```

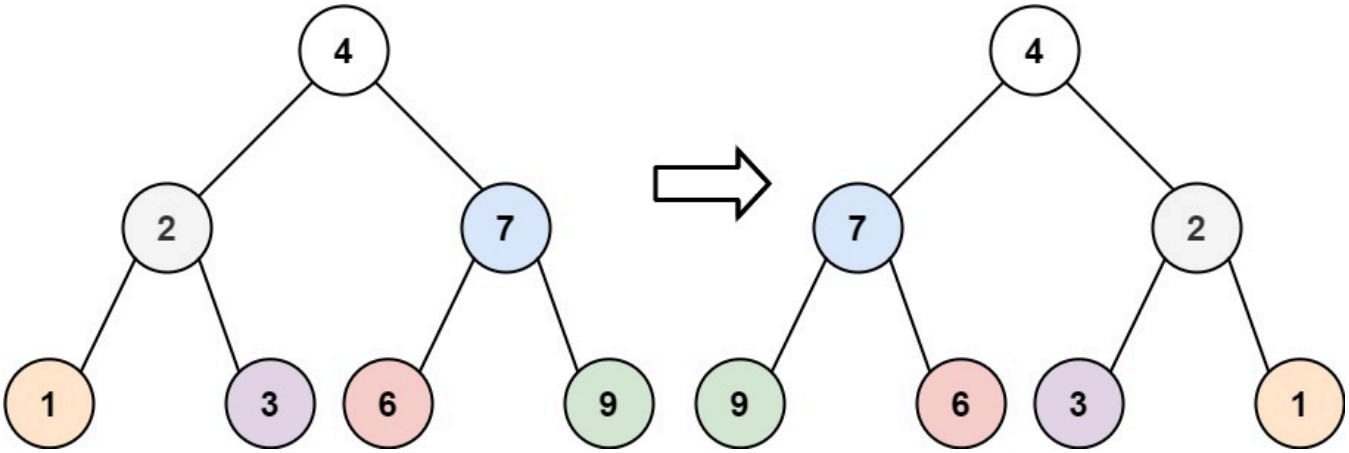
Problem 2: Invert Binary Tree

Given the `root` of a binary tree, invert the tree, and return the root of the modified tree.

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

def invert_tree(root):
    pass
```

Example Usage 1:



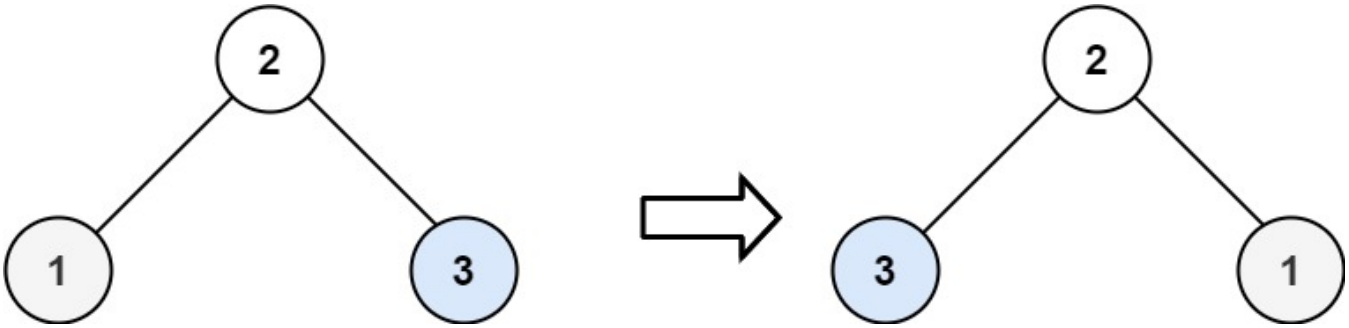

```
# Using build_tree() function included at top of page
tree_1 = build_tree([4,2,7,1,3,6,9])

# Using print_tree() function included at top of page
print_tree(tree_1)
```

Example Output 1:

```
[4,7,2,9,6,3,1]
```

Example Usage 2:




```
# Using build_tree() function included at top of page
tree_2 = build_tree([2,1,3])

# Using print_tree() function included at top of page
print_tree(tree_2)
```

Example Output:

```
[2,3,1]
```

Problem 3: Valid Parentheses

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, return `True` if the input string is valid and `False` otherwise.

An input string is valid if:

- Open brackets are closed by the same type of brackets.

- Open brackets are closed in the correct order.
- Every close bracket has a corresponding open bracket of the same type.

```
def valid_parentheses(s):
    pass
```

Example Usage:

```
print(valid_parentheses("()"))
print(valid_parentheses("()[]{}"))
print(valid_parentheses("(]"))
print(valid_parentheses("([])"))
print(valid_parentheses("([)]"))
```

Example Output:

```
True
True
False
True
False
```

Problem 4: Is Subsequence

Given two strings `s` and `t`, return `True` if `s` is a subsequence of `t`, or `False` otherwise.

A subsequence of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

```
def is_subsequence(s, t):
    pass
```

Example Usage:

```
print(is_subsequence("abc", "ahbgdc"))
print(is_subsequence("axc", "ahbgdc"))
```

Example Output:

```
True
False
```

Problem 5: Number of Provinces

There are `n` cities. Some of them are connected, while some are not. If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

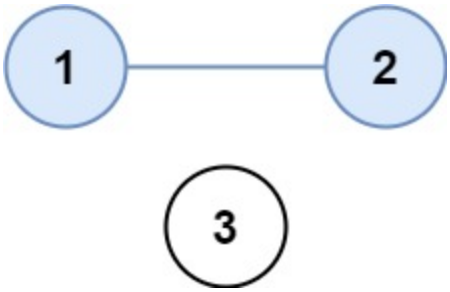
A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `is_connected` where `is_connected[i][j] = 1` if the `ith` city and the `jth` city are directly connected, and `is_connected[i][j] = 0` otherwise.

Return the total number of provinces.

```
def num_provinces(is_connected):
    pass
```

Example Usage 1:

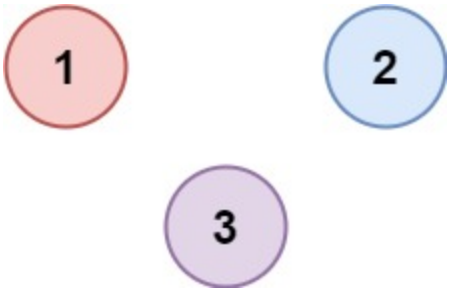



```
is_connected_1 = [[1,1,0],[1,1,0],[0,0,1]]
print(num_provinces(is_connected_1))
```

Example Output 1:

```
2
```

Example Usage 2:




```
is_connected_2 = [[1,0,0],[0,1,0],[0,0,1]]
print(num_provinces(is_connected_2))
```

Example Output 2:

```
3
```

Problem 6: Split Linked List in Parts

Given the `head` of a singly linked list and an integer `k`, split the linked list into `k` consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being `None`.

The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

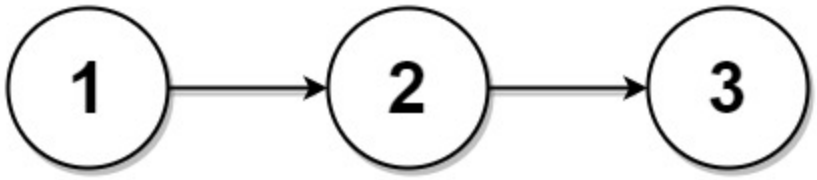
Return an array of the `k` parts.

```
class Node:
    def __init__(self, value, next=None):
        self.value = value
        self.next = next

def split_list(head, k):
    pass
```

Example Usage:

Example 1:




```
list_1 = Node(1, Node(2, Node(3)))
list_1_output = split_list(list_1, 5)

for part in list_1:
    print_linked_list(part)

list_2 = Node(1, Node(2, Node(3, Node(4, Node(5,
                Node(6, Node(7, Node(8, Node(9, Node(10))))))))))
list_2_output = split_list(list_2, 3)

for part in list_2:
    print_linked_list(part)
```

Example Output:

```
1
2
3
Empty List
Empty List
Example 1 Explanation:
The first element output[0] has output[0].val = 1, output[0].next = null.
The last element output[4] is null, but its string representation as a ListNode is [

1 -> 2 -> 3 -> 4
5 -> 6 -> 7
8 -> 9 -> 10
Example 2 Explanation:
The input has been split into consecutive parts with size difference at most 1, and
parts are a larger size than the later parts.
```

Close Section

▸ Advanced Problem Set Version 1

▸ Advanced Problem Set Version 2