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 2: Working with Linked Lists

Overview

In this coding assignment, students will focus on mastering various operations involving linked lists through hands-on problem-solving, enhancing their understanding of this crucial data structure in computer science.

> 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**

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 2

💡 **Unit 6 Cheatsheet**

To help your learning journey with linked lists, we've put together a guide to common concepts and syntax you will use throughout Unit 6 breakout problems. Use this cheatsheet as a quick reference guide as you work through the problems below.

▼ Problem Set Version 1

Problem 1: Detect Circular Linked List

A circular linked list is a linked list where the tail node points at the head node. Given the head of a linked list, write a function `is_circular()` that returns `True` if the linked list is circular and `False` otherwise.

Note: a circular list is more than just a cycle - specifically connecting the first and last nodes.

Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

```
        pass
```

Example Usage:

```
# num1 -> num2 -> num3 -> num1
print(is_circular(num1))

# var1 -> var2 -> var3
print(is_circular(var1))
```

Example Output:

```
True
False
```

Problem 2: Find Last Node in a Linked List Cycle

Given the head of a singly linked list, write a function that returns the last node in the cycle. If there is no cycle in the linked list, return None.

```
def find_last_node_in_cycle(head):
        pass
```

Example Input: `num1 -> num2 -> num3 -> num4 -> num2`

Example Output: `num4`

▶ ✨ AI Hint: Multiple Pass Technique

Problem 3: Partition List Around Value

Given the `head` of a linked list and a value `val`, partition a linked list around `val` such that all nodes with values less than `val` come before nodes with values greater than or equal to `val`.

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

def partition(head, val):
        pass
```

Example Input:

Result Linked List: `1 -> 2 -> 2 -> 4 -> 3 -> 5` or `2 -> 2 -> 1 -> 5 -> 4 -> 3`

▶ 💡 **Hint: Temporary Head Technique**

Problem 4: Convert Binary Number in a Linked List to Integer

You are given the head of a linked list. Each value in the linked list is either 0 or 1, and the entire linked list represents a binary number. Return an integer that is the decimal value of the number represented by the linked list. The most significant bit is at the head of the linked list.

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

def binary_to_int(head):
        pass
```

Example Usage:

```
# 1 -> 0 -> 1
num3 = Node(1)
num2 = Node(0, num3)
num1 = Node(1, num2)   # head of the list

int_num = binary_to_int(num1)
# 101 in binary is 5
print(int_num)  # Output: 5
```

Example Output: `5`

▶ ✨ **AI Hint: Binary to Decimal**

Problem 5: Add Two Numbers Represented by Linked Lists

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

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

```
    pass
```

Example Usage:

```
# list 1: 2 -> 4 -> 3 (342)
# list 2: 5 -> 6 -> 4 (465)
# head_a = 2, head_b = 5

sum = add_two_numbers(head_a, head_b)
print(sum)
```

Example Output: `7 -> 0 -> 8`

Explanation: `342 + 465 = 807, so the list is 7 -> 0 -> 8.`

Problem 6: Reverse Sublist of a Linked List

Given the head of a linked list and indices `m` and `n`, reverse the linked list between positions `m` and `n`. Assume the linked list uses **1-based indexing** and the `0 <= m <= n <= length of list`. Return the head of the list.

```
def reverse_between(head, m, n):
        pass
```

Example Usage:

```
# input list: 1 -> 2 -> 3 -> 4 -> 5
reverse_between(head, 2, 5)
```

Result Linked List: `1 -> 5 -> 4 -> 3 -> 2`

Close Section

- ▸ **Problem Set Version 2**
- ▸ **Problem Set Version 3**