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: Linked Lists II

Session Overview

In this session, students will deepen their ability to work with linked lists, solving problems involving traversal, node manipulation, and cycle detection. These exercises will cover techniques common to many linked lists problems and enhance their understanding of this crucial data structure.

> 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**

Breakout Problems Session 2

▶ Standard Problem Set Version 1

▶ Standard Problem Set Version 2

▼ Advanced Problem Set Version 1

Problem 1: Next in Queue

Each user on a music app should have a queue of songs to play next. Implement the **class** `Queue` using a singly linked list. Recall that a queue is a First-In-First-Out (FIfO) data structure where elements are added to the end (the tail) and removed from the front (the head).

Your queue must have the following methods:

- `__init()__` : Initializes an empty queue (provided)
- `enqueue()` : Accepts a tuple of two strings `(song, artist)` and adds the element with the specified tuple to the end of the queue.
- `dequeue()` : Removes and returns the element at the front of the queue. If the queue is empty, returns `None` .

- `peek()` : Returns the value of the element at the front of the queue without removing it. If the queue is empty, returns `None` .
- `is_empty()` : Returns `True` if the queue is empty, and `False` otherwise.

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

# For testing
def print_queue(head):
    current = head.front
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

class Queue:
    def __init__(self):
        self.front = None
        self.rear = None

    def is_empty(self):
        pass

    def enqueue(self):
        pass

    def dequeue(self):
        pass

    def peek(self):
        pass
```

Example Usage:

```
# Create a new Queue
q = Queue()

# Add elements to the queue
q.enqueue(('Love Song', 'Sara Bareilles'))
q.enqueue(('Ballad of Big Nothing', 'Elliot Smith'))
q.enqueue(('Hug from a Dinosaur', 'Torres'))
print_queue(q)

# View the front element
print("Peek: ", q.peek())

# Remove elements from the queue
print("Dequeue: ", q.dequeue())
print("Dequeue: ", q.dequeue())

# Check if the queue is empty
print("Is Empty: ", q.is_empty())

# Remove the last element
print("Dequeue: ", q.dequeue())

# Check if the queue is empty
print("Is Empty:", q.is_empty())
```

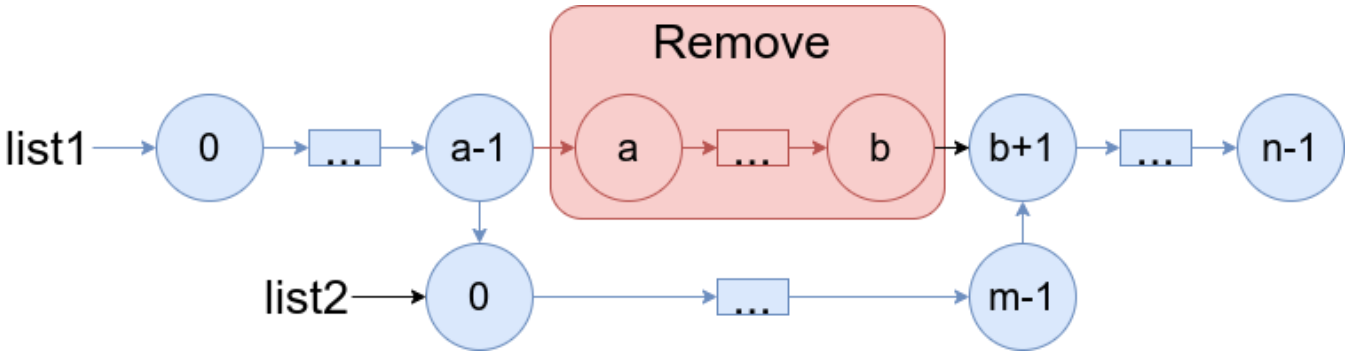
Example Output:

```
('Love Song', 'Sara Bareilles') -> ('Ballad of Big Nothing', 'Elliot Smith')
-> ('Hug from a Dinosaur', 'Torres')
Peek:  ('Love Song', 'Sara Bareilles')
Dequeue:  ('Love Song', 'Sara Bareilles')
Dequeue:  ('Ballad of Big Nothing', 'Elliot Smith')
Is Empty:  False
Dequeue:  ('Hug from a Dinosaur', 'Torres')
Is Empty: True
```

Problem 2: Merge Playlists

You are given the head of two linked lists, `playlist1` and `playlist2` with lengths `n` and `m` respectively. Remove `playlist1`'s nodes from the `ath` to the `bth` node and put `playlist2` in its place. Assume the lists are 0-indexed.

The blue edges and nodes in the figure below indicate the result:



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
                self.next = next

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def merge_playlists(playlist1, playlist2, a, b):
    pass
```

Example Usage:

```
playlist1 = Node(('Flea', 'St. Vincent'),
                Node(('Juice', 'Lizzo'),
                    Node(('Tenderness', 'Jay Som'),
                        Node(('Ego Death', 'The Internet'),
                            Node(('Empty', 'Kevin Abstract'))))))

playlist2 = Node(('Dreams', 'Solange'), Node(('First', 'Gallant')))

print_linked_list(merge_playlists(playlist1, playlist2, 2, 3))
```

Example Output:

```
('Flea', 'St.Vincent') -> ('Juice', 'Lizzo') -> ('Dreams', 'Solange') -> ('First', '(
-> ('Empty', 'Kevin Abstract')
```

Problem 3: Shuffle Playlist

You are given the head of a singly linked list `playlist`. The list can be represented as:

`L0 → L1 → … → Ln - 1 → Ln`

Shuffle the playlist to have the following form:

`L0 → Ln → L1 → Ln − 1 → L2 → Ln − 2 → …`

You may not modify the values in the list's nodes. Only the order of the nodes themselves may be changed. Return the head of the shuffled list.

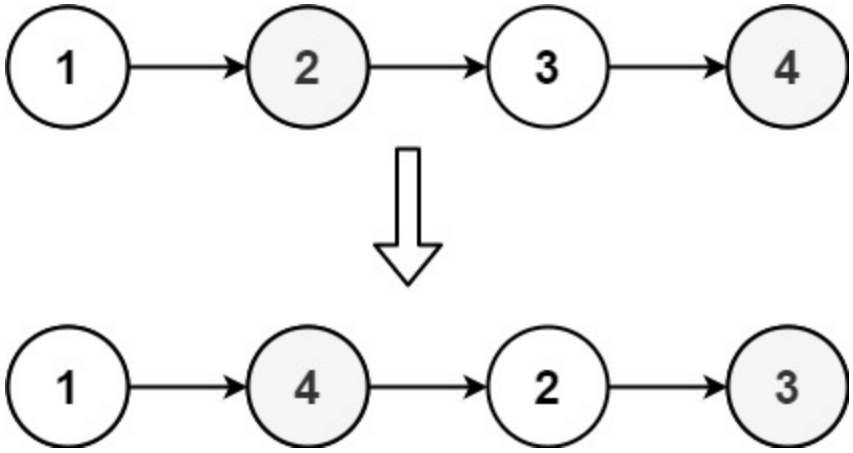
Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def shuffle_playlist(playlist):
    pass
```

Example Usage:




```
playlist1 = Node(1, Node(2, Node(3, Node(4))))

playlist2 = Node(('Respect', 'Aretha Franklin'),
                Node(('Superstition', 'Stevie Wonder'),
                    Node(('Wonderwall', 'Oasis'),
                        Node(('Like a Prayer', 'Madonna'),
                            Node(('Bohemian Rhapsody', 'Queen'))))))

print_linked_list(shuffle_playlist(playlist1))
print_linked_list(shuffle_playlist(playlist2))
```

Example Output:

```
1 -> 4 -> 2 -> 3
('Respect', 'Aretha Franklin') -> ('Bohemian Rhapsody', 'Queen') -> ('Superstition',
('Like a Prayer', 'Madonna') -> ('Wonderwall', 'Oasis')
```

Problem 4: Shared Music Taste

Given the heads of two singly linked lists `playlist_a` and `playlist_b`, return the node at which the two lists intersect. If the two lists have no intersection at all, return `None`.

There are no cycles anywhere in either linked list. The linked lists must retain their original structure after the function returns.

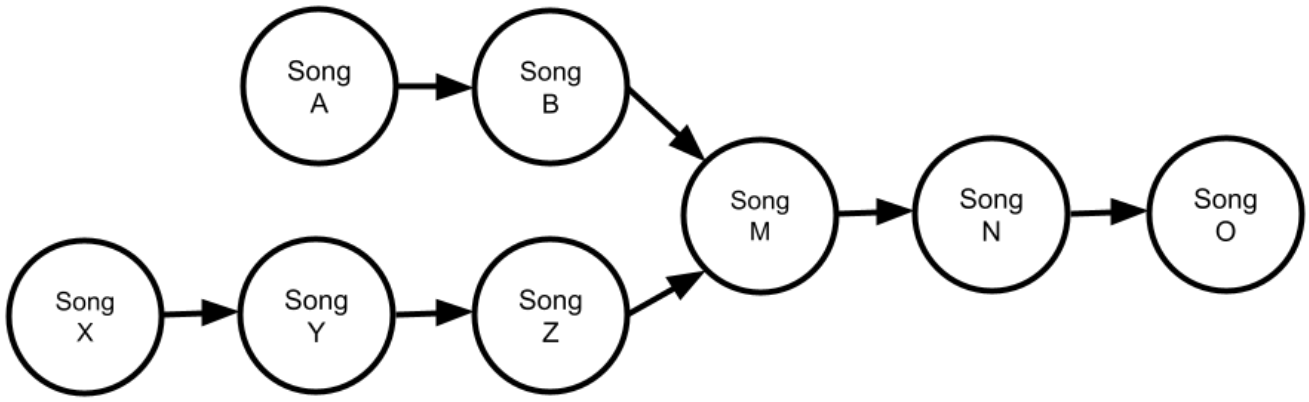
Evaluate the time and space complexity of your solution. Define your variables and provide a rationale for why you believe your solution has the stated time and space complexity.

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

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def playlist_overlap(playlist_a, playlist_b):
    pass
```

Example Usage:



```
playlist_a = Node('Song A', Node('Song B'))
playlist_b = Node('Song X', Node('Song Y', Node('Song Z')))
shared_segment = Node('Song M', Node('Song N', Node('Song O')))

playlist_a.next.next = shared_segment
playlist_b.next.next.next = shared_segment

print((playlist_overlap(playlist_a, playlist_b)).value)
```

Example Output:

```
Song M
```

Problem 5: Double Listening Count

A new artist is blowing up and the number of people listening to their music has doubled in the last month. Given the head of a non-empty linked list `monthly_listeners` representing a non-negative integer without leading zeroes, return the `head` of the linked list after doubling its integer value.

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
                self.next = next

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def double_listeners(monthly_listeners):
    pass
```

Example Usage:

```
monthly_listeners1 = Node(1, Node(8, Node(9))) # 189
monthly_listeners2 = Node(9, Node(9, Node(9))) # 999

print_linked_list(double_listeners(monthly_listeners1))
print_linked_list(double_listeners(monthly_listeners2))
```

Example Output:

```
3 -> 7 -> 8
Example 1 Explanation: 189 * 2 = 378

1 -> 9 -> 9 -> 8
Example 2 Explanation: 999 * 2 = 1998
```

Close Section

▼ Advanced Problem Set Version 2

Problem 1: Stack 'Em Up!

The library has a stack of returned books waiting to be shelved. Help the library to manage the stack by implementing the **class** `Stack` using a singly linked list. Recall that a stack is a Last-In-First-Out (LIFO) data structure where elements are added to the front (the head) and removed from the front (the head).

Your stack must have the following methods:

- `__init()__` : Initializes an empty stack (provided)
- `push()` : Accepts a tuple of two strings `(title, author)` and adds the element with the specified tuple to the front/top of the stack.
- `pop()` : Removes and returns the element at the front/top of the stack. If the stack is empty, returns `None` .
- `peek()` : Returns the value of the element at the front/top of the stack without removing it. If the stack is empty, returns `None` .
- `is_empty()` : Returns `True` if the stack is empty, and `False` otherwise.

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

# For testing
def print_stack(head):
    current = head.front
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

class Stack:
    def __init__(self):
        self.front = None

    def is_empty(self):
        pass

    def push(self):
        pass

    def pop(self):
        pass

    def peek(self):
        pass
```

Example Usage:

```
# Create a new Stack
stack = Stack()

# Add elements to the stack
stack.push(('Educated', 'Tara Westover'))
stack.push(('Gone Girl', 'Gillian Flynn'))
stack.push(('Dune', 'Frank Herbert'))
print_stack(stack)

# View the front element
print("Peek: ", stack.peek())

# Remove elements from the stack
print("Pop: ", stack.pop())
print("Pop: ", stack.pop())

# Check if the stack is empty
print("Is Empty: ", stack.is_empty())

# Remove the last element
print("Pop: ", stack.pop())

# Check if the queue is empty
print("Is Empty:", stack.is_empty())
```

Example Output:

```
('Dune', 'Frank Herbert') -> ('Gone Girl', 'Gillian Flynn') -> ('Educated', 'Tara We
Peek:  ('Dune', 'Frank Herbert')
Pop:  ('Dune', 'Frank Herbert')
Pop:  ('Gone Girl', 'Gillian Flynn')
Is Empty:  False
Pop:  ('Educated', 'Tara Westover')
Is Empty: True
```

Problem 2: Surprise Me

Given the head of a singly linked list of books in a library `catalogue`, suggest a random book to a customer by returning a random node's value from the linked list. Each node must have the same probability of being chosen.

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
        self.next = next

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def get_random(playlist):
    pass
```

Example Usage:

```
catalogue = Node(('Homegoing', 'Yaa Gyasi'),
                 Node(('Pachinko', 'Min Jin Lee'),
                          Node(('The Night Watchman', 'Louise Erdrich'))))

print(get_random(catalogue))
```

Example Output:

```
('Homegoing', 'Yaa Gyasi')
Explanation: It should be equally likely that ('Pachinko', 'Min Jin Lee') or
('The Night Watchman', 'Louise Erdrich') is returned
```

▶ 💡 **Hint: Random Library**

Problem 3: Properly Reshelve

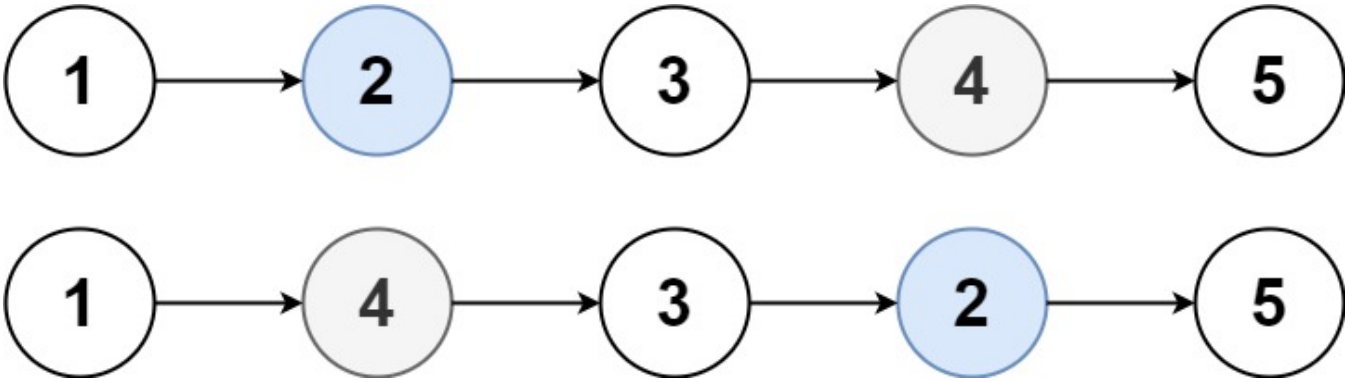
A well-intentioned reader has improperly put back a book on the shelf. Given the head of a linked list `shelf` where each node represents a book on the shelf, and a value `k` return the head of the linked list after swapping the values of the `kth` node from the beginning and the `kth` node from the end. Assume the list is 1-indexed. Assume `1 <= k < n` where `n` is the length of `shelf`.

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

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def swap_books(shelf, k):
        pass
```

Example Usage:




```
shelf = Node('Book 1', Node('Book 2', Node('Book 3', Node('Book 4', Node('Book 5'))))

print_linked_list(swap_books(shelf, 2))
```

Example Output:

```
Book 1 -> Book 4 -> Book 3 -> Book 2 -> Book 5
```

Problem 4: Book Display

You want to display popular new books the library has just received in a fun way to visitors.

Given two integers `m` and `n` which represent dimensions of a matrix and the head of a linked list `new_reads` where each node represents a book, generate a `m x n` matrix that contains the values of each book in `new_reads` presented in sprial order (clockwise), starting from the top-left of the matrix. If there are remaining empty spaces, fill them with `None`.

Return the generated matrix.

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

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def spiralize_books(m, n, new_reads):
        pass
```

Example Usage:

Example 1

Book 1	Book 2	Book 3	Book 4	Book 5
Book 12	Book 13	None	None	Book 6
Book 11	Book 10	Book 9	Book 8	Book 7

Example 2

Book 1	Book 2	Book 3	None

```
new_reads1 = Node('Book 1', Node('Book 2', Node('Book 3', Node('Book 4', Node('Book 
Node('Book 7', Node('Book 8', Node('Book 9', Node('Book 10', Node('Book 11', Node('B
new_reads2 = Node('Book 1', Node('Book 2', Node('Book 3')))

print(spiralize_books(3, 5, new_reads1))
print(spiralize_books(1, 4, new_reads2))
```

Example Output:

```
[
    ['Book 1', 'Book 2', 'Book 3', 'Book 4', 'Book 5'],
    ['Book 12', 'Book 13', None, None, 'Book 6'],
    ['Book 11', 'Book 10', 'Book 9', 'Book 8', 'Book 7']
]

[['Book 1', 'Book 2', 'Book 3', None]]
```

Problem 5: Book Similarity

The library sequences books by topic so that it's easy to find related books. Given the head of a linked list `all_books` where each node contains a unique integer values representing a different book in the library, and an integer array `subset` that contains a subset of the values in `all_books`, return the number of *similar* book components in `subset`. Two books are similar if they appear consecutively in the linked list.

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

# For testing
def print_linked_list(head):
    current = head
    while current:
        print(current.value, end=" -> " if current.next else "")
        current = current.next
    print()

def similar_book_count(all_books, subset):
        pass
```

Example Usage:

```
all_books1 = Node(0, Node(1, Node(2, Node(3))))
subset1 = [0, 1, 3]

all_books2 = Node(0, Node(1, Node(2, Node(3, Node(4)))))
subset2 = [0, 3, 1, 4]

print(similar_book_count(all_books1, subset1))
print(similar_book_count(all_books2, subset2))
```

Example Output:

```
2
Example 1 Explanation: 0 and 1 are similar, so [0, 1] and [3] are the two similar co

2
Example 2 Explanation: 0 and 1 are similar, 3 and 4 are similar,
so [0, 1] and and [3, 4] are the similar components.
```

Close Section