Data Structures and Algorithms

(DSA)

Lab Report 6

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Submitted To:

Engineer Sheharyar Khan

Obtained Marks: Not Evaluated Total Marks: 8

Marks Distribution:

Total Lab Activity Marks:4

Total Lab Report Marks: 4

Lab 5

Guided Tasks (Circular Linked List)

Task 1: Implementing a Circular Linked List (CLL)

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

class CircularLinkedList:
    def __init__(self):
        self.head = None

    def delete(self, data):
        """Deletes a node by value in a Circular Linked List."""
        if self.head is None:
            print("List is empty!")
            return

        current = self.head
        prev = None
        while True:
            if current.data == data:
                if prev is not None:
                    prev.next = current.next
                else:
                    # If only one node is in the list
                    if current.next == self.head:
                        self.head = None
                    else:
```

```
                        self.head = current.next
                        temp = self.head
                        while temp.next != current:
                            temp = temp.next
                        temp.next = self.head
                return
```

```
            prev = current
            current = current.next
            if current == self.head:
                break
        print("Element not found!")

    def display(self):
        """Displays the Circular Linked List."""
        if self.head is None:
            print("List is empty!")
            return
        current = self.head
        while True:
            print(current.data, "->", end=" ")
            current = current.next
            if current == self.head:
                break
        print("(Back to head)")

# Example Usage
cll = CircularLinkedList()
# Assume we have a method to add elements before deleting
dummy_node = Node(10)
dummy_node.next = dummy_node  # Pointing to itself to create a circular link
cll.head = dummy_node

cll.delete(10)
cll.display()  # Output: List is empty!
```

Output:

```
List is empty!
```

Task 2: Instagram Story Viewer using a Circular Linked List

```
class Story:
    def __init__(self, content):
        self.content = content
        self.next = None

class StoryViewer:
    def __init__(self):
        self.head = None
        self.current_story = None

    def add_story(self, content):
        """Adds a new story to the circular linked list."""
        story = Story(content)
        if self.head is None:
            self.head = story
            story.next = self.head  # Circular link
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = story
            story.next = self.head  # Pointing back to head
    def view_next_story(self):
        """Moves to the next story."""
        if self.current_story is None:
            self.current_story = self.head  # Start from the first story
        else:
```

```
            self.current_story = self.current_story.next  # Move to next
        print(f"Viewing Story: {self.current_story.content}")

    def display_stories(self):
        """Displays all stories."""
        if self.head is None:
            print("No stories available!")
            return
        temp = self.head
        while True:
            print(f"Story: {temp.content}")
            temp = temp.next
            if temp == self.head:
                break

# Example Usage
stories = StoryViewer()
stories.add_story("User1's Story")
stories.add_story("User2's Story")
stories.add_story("User3's Story")

print("\nAll Stories in Viewer:")
stories.display_stories()

print("\nSimulating Story Viewing:")
stories.view_next_story()  # Viewing: User1's Story
stories.view_next_story()  # Viewing: User2's Story
stories.view_next_story()  # Viewing: User3's Story
stories.view_next_story()  # Viewing: User1's Story (Cycle Restarts)
```

Output:

```
All Stories in Viewer:
Story: User1's Story
Story: User2's Story
Story: User3's Story

Simulating Story Viewing:
Viewing Story: User1's Story
Viewing Story: User2's Story
Viewing Story: User3's Story
Viewing Story: User1's Story
```

EXERCISE

Easy Problems

1. CLL Traversal

Implement a Circular Linked List and traverse it in a loop.

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

```
class CircularLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def traverse(self):
        if not self.head:
            print("List is empty")
            return
        temp = self.head
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")

# Usage
cll = CircularLinkedList()
cll.append(1)
cll.append(2)
cll.append(3)
cll.traverse()
```

Output:

```
1 -> 2 -> 3 -> (back to head)
```

2. CLL Deletion

Implement a method to delete a node in Circular Linked List.

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

class CircularLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
```

```
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def delete_node(self, key):
        if not self.head:
            print("List is empty")
            return
        temp = self.head
        prev = None
        while True:
            if temp.data == key:
                if prev:
                    prev.next = temp.next
                else:
                    last = self.head
                    while last.next != self.head:
                        last = last.next
                    self.head = temp.next
                    last.next = self.head
                return
            prev, temp = temp, temp.next
            if temp == self.head:
                break
        print("Node not found")

    def traverse(self):
        if not self.head:
            print("List is empty")
            return
        temp = self.head
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(back to head)")

# Usage
cll = CircularLinkedList()
cll.append(1)
cll.append(2)
cll.append(3)
cll.traverse()
cll.delete_node(2)
cll.traverse()
```

Output:

```
1 -> 2 -> 3 -> (back to head)
1 -> 3 -> (back to head)
```

Intermediate Problems

1. Circular Scheduling System (CLL)

Implement a task scheduling system where tasks repeat cyclically using a Circular Linked List.

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

class TaskScheduler:
    def __init__(self):
        self.head = None

    def add_task(self, task):
        new_node = Node(task)
        if not self.head:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def execute_tasks(self, cycles=2):
        if not self.head:
            print("No tasks to execute.")
            return
        temp = self.head
        for _ in range(cycles):
            print(f"Executing: {temp.task}")
            temp = temp.next
        print("(Tasks repeated)")

# Usage
scheduler = TaskScheduler()
scheduler.add_task("Task 1")
scheduler.add_task("Task 2")
scheduler.add_task("Task 3")
scheduler.execute_tasks()
```

Output:

```
Executing: Task 1
Executing: Task 2
(Tasks repeated)
```

2. Round Robin CPU Scheduling (CLL)

Simulate Round Robin CPU scheduling using a Circular Linked List.

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

class CPU_Scheduler:
    def __init__(self):
        self.head = None

    def add_process(self, process, time):
        new_node = Node(process, time)
        if not self.head:
            self.head = new_node
            new_node.next = self.head
        else:
            temp = self.head
            while temp.next != self.head:
                temp = temp.next
            temp.next = new_node
            new_node.next = self.head

    def execute(self, quantum=3):
        if not self.head:
            print("No processes to execute.")
            return
        temp = self.head
        while True:
            if temp.time > 0:
                execute_time = min(temp.time, quantum)
                temp.time -= execute_time
                print(f"Executing {temp.process} for {execute_time}
units")
                if temp.time == 0:
                    print(f"{temp.process} completed.")
            temp = temp.next
            if temp == self.head and all(node.time == 0 for node in
self._iter_nodes()):
                break

    def _iter_nodes(self):
        temp = self.head
        if not temp:
            return
        while True:
            yield temp
            temp = temp.next
            if temp == self.head:
```

```
                break

# Usage
scheduler = CPU_Scheduler()
scheduler.add_process("P1", 5)
scheduler.add_process("P2", 7)
scheduler.add_process("P3", 4)
scheduler.execute()
```

Output:

```
Executing P1 for 3 units
Executing P2 for 3 units
Executing P3 for 3 units
Executing P1 for 2 units
P1 completed.
Executing P2 for 3 units
Executing P3 for 1 units
P3 completed.
Executing P2 for 1 units
P2 completed.
```