Data Structures and Algorithms

(DSA)

Lab Report 5

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Submitted To:

Engineer Sheharyar Khan

Obtained Marks: Total Marks: 8

Marks Distribution:

Total Lab Activity Marks: 4

Total Lab Report Marks: 4

Lab 5

Guided Tasks

Task 1: Implementing a Doubly Linked List (DLL)

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None  # New previous pointer

class DoublyLinkedList:
    def __init__(self):
        self.head = None  # Points to the first node
        self.tail = None  # Points to the last node

    def append(self, data):
        """Appends a node at the end of the DLL."""
        node = Node(data)
        if self.head is None:
            self.head = node
            self.tail = node
        else:
            self.tail.next = node
            node.prev = self.tail  # Linking the previous node
            self.tail = node  # Update tail to the new last node

    def display_forward(self):
        """Traverses the list from head to tail."""
        current = self.head
        while current:
            print(current.data, "<->", end=" ")
            current = current.next
        print("None")

    def display_backward(self):
        """Traverses the list from tail to head."""
        current = self.tail
        while current:
            print(current.data, "<->", end=" ")
            current = current.prev
        print("None")
```

```
# Example Usage
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.display_forward()  # Output: 10 <-> 20 <-> 30 <-> None
dll.display_backward()  # Output: 30 <-> 20 <-> 10 <-> None
```

Output:

```
10 <-> 20 <-> 30 <-> None
30 <-> 20 <-> 10 <-> None
```

Task 2: Insertion Operations in DLL

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

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

    def insert_at_beginning(self, data):
        """Inserts a node at the beginning of the DLL."""
        node = Node(data)
        if self.head is None:
            self.head = node
        else:
            node.next = self.head
            self.head.prev = node
            self.head = node

    def insert_at_position(self, data, pos):
        """Inserts a node at a specific position in the DLL."""
        node = Node(data)
        if pos == 0:
            self.insert_at_beginning(data)
            return
```

```
        current = self.head
        for _ in range(pos - 1):
            if current is None:
                print("Position out of bounds")
                return
            current = current.next

        if current is None:
            print("Position out of bounds")
            return
```

```
        node.next = current.next
        if current.next:
            current.next.prev = node
        current.next = node
        node.prev = current

    def display_forward(self):
        """Displays the DLL from head to tail."""
        current = self.head
        while current:
            print(current.data, "<->", end=" ")
            current = current.next
        print("None")

# Example Usage
dll = DoublyLinkedList()
dll.insert_at_beginning(50)
dll.insert_at_position(25, 1)
dll.display_forward()  # Output: 50 <-> 25 <-> None
```

Output:

```
50 <-> 25 <-> None
```

Task 3: 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 4: Music Playlist System using a Doubly Linked List

```
class Song:
    def __init__(self, title):
        self.title = title
        self.next = None
        self.prev = None

class MusicPlaylist:
    def __init__(self):
        self.head = None
        self.tail = None
        self.current_song = None
    def add_song(self, title):
        """Adds a song to the playlist."""
        song = Song(title)
        if self.head is None:
            self.head = song
            self.tail = song
            self.current_song = song
        else:
            self.tail.next = song
            song.prev = self.tail
            self.tail = song
```

```
    def play_next(self):
        """Moves to the next song in the playlist."""
        if self.current_song and self.current_song.next:
            self.current_song = self.current_song.next
            print(f"Now playing: {self.current_song.title}")
        else:
            print("End of playlist reached!")

    def play_previous(self):
        """Moves to the previous song in the playlist."""
        if self.current_song and self.current_song.prev:
            self.current_song = self.current_song.prev
            print(f"Now playing: {self.current_song.title}")
        else:
            print("Already at the first song!")

    def display_playlist(self):
        """Displays the full playlist in order."""
        current = self.head
        while current:
            print(current.title, "<->", end=" ")
            current = current.next
        print("None")

# Example Usage
playlist = MusicPlaylist()
playlist.add_song("Song 1")
playlist.add_song("Song 2")
playlist.add_song("Song 3")

print("\nMusic Playlist:")
playlist.display_playlist()
```

```
print("\nNavigating the Playlist:")
playlist.play_next()  # Now playing: Song 2
playlist.play_next()  # Now playing: Song 3
playlist.play_previous()  # Now playing: Song 2
playlist.play_previous()  # Now playing: Song 1
playlist.play_previous()  # Already at the first song!
```

Output:

```
Music Playlist:
Song 1 <-> Song 2 <-> Song 3 <-> None

Navigating the Playlist:
Now playing: Song 2
Now playing: Song 3
Now playing: Song 2
Now playing: Song 1
Already at the first song!
```

Task 5: Instagram Story Viewer using a Circular Linked List

```python
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-DLL Basic Operations

Implement a class for Doubly Linked List that supports append, display, and delete from start.

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

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

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

    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" <-> ")
            temp = temp.next
        print("None")

    def delete_from_start(self):
```

```
        if not self.head:
            print("List is empty")
            return
        self.head = self.head.next
        if self.head:
            self.head.prev = None

# Usage
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.display()
dll.delete_from_start()
dll.display()
```

Output:

```
10 <-> 20 <-> 30 <-> None
20 <-> 30 <-> None
```

2. 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)
```

3. DLL Reverse Traversal

Implement a method to print a DLL in reverse order.

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

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

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

    def reverse_traverse(self):
        temp = self.head
        if not temp:
            print("List is empty")
            return
        while temp.next:
            temp = temp.next
        while temp:
            print(temp.data, end=" <-> ")
            temp = temp.prev
        print("None")

# Usage
```

```
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.reverse_traverse()
```

Output:

```
30 <-> 20 <-> 10 <-> None
```

4. 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)
```

5. DLL Length Calculation

Implement a function that returns the length of a DLL.

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

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

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

    def get_length(self):
```

```
        count = 0
        temp = self.head
        while temp:
            count += 1
            temp = temp.next
        return count

# Usage
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
print("Length:", dll.get_length())
```

Output:

```
Length: 3
```

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. Game Leaderboard (DLL)

Implement a leaderboard where scores are stored in a Doubly Linked List, sorted by highest score.

```
class Node:
    def __init__(self, name, score):
        self.name = name
        self.score = score
        self.next = None
        self.prev = None

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

    def add_score(self, name, score):
        new_node = Node(name, score)
        if not self.head or self.head.score < score:
            new_node.next = self.head
            if self.head:
                self.head.prev = new_node
            self.head = new_node
            return
        temp = self.head
        while temp.next and temp.next.score >= score:
            temp = temp.next
        new_node.next = temp.next
        if temp.next:
            temp.next.prev = new_node
        temp.next = new_node
        new_node.prev = temp

    def display_leaderboard(self):
        temp = self.head
        while temp:
            print(f"{temp.name}: {temp.score}")
            temp = temp.next

# Usage
```

```
board = Leaderboard()
board.add_score("Alice", 85)
board.add_score("Bob", 92)
board.add_score("Charlie", 78)
board.display_leaderboard()
```

Output:

```
Bob: 92
Alice: 85
Charlie: 78
```

3. 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.
```

Advanced Problems

1. Facebook Messenger Chat History (DLL)

Implement a chat history feature using a Doubly Linked List to navigate through messages.

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

class ChatHistory:
    def __init__(self):
        self.head = None
        self.tail = None

    def add_message(self, message):
```

```
        new_node = Node(message)
        if not self.head:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.next = new_node
            new_node.prev = self.tail
            self.tail = new_node

    def show_history(self):
        temp = self.tail
        while temp:
            print(temp.message)
            temp = temp.prev

# Usage
chat = ChatHistory()
chat.add_message("Hello")
chat.add_message("How are you?")
chat.add_message("I'm good, thanks!")
chat.show_history()
```

Output:

```
I'm good, thanks!
How are you?
Hello
```

2. Undo/Redo System (DLL)

Implement an Undo/Redo system for a text editor using Doubly Linked Lists.

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

class TextEditor:
    def __init__(self):
        self.head = None
        self.current = None

    def write(self, text):
        new_node = Node(text)
        if not self.head:
            self.head = new_node
            self.current = new_node
        else:
            new_node.prev = self.current
            self.current.next = new_node
            self.current = new_node

    def undo(self):
```

```
        if self.current and self.current.prev:
            self.current = self.current.prev
        print("Current Text:", self.current.text if self.current
else "Empty")

    def redo(self):
        if self.current and self.current.next:
            self.current = self.current.next
        print("Current Text:", self.current.text if self.current
else "Empty")

# Usage
editor = TextEditor()
editor.write("Hello")
editor.write("World")
editor.undo()
editor.redo()
```

Output:

```
Current Text: Hello
Current Text: World
```

3. Browser History Navigation (DLL)

Implement forward and backward navigation in a web browser using a Doubly Linked List.

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

class BrowserHistory:
    def __init__(self):
        self.current = None

    def visit(self, url):
        new_node = Node(url)
        if not self.current:
            self.current = new_node
        else:
            new_node.prev = self.current
            self.current.next = new_node
            self.current = new_node

    def back(self):
        if self.current and self.current.prev:
            self.current = self.current.prev
        print("Current Page:", self.current.url if self.current else
"No history")

    def forward(self):
```

```
        if self.current and self.current.next:
            self.current = self.current.next
        print("Current Page:", self.current.url if self.current else
"No forward history")

# Usage
browser = BrowserHistory()
browser.visit("google.com")
browser.visit("facebook.com")
browser.back()
browser.forward()
```

Output:

```
Current Page: google.com
Current Page: facebook.com
```