University of Engineering and Technology Taxila
Computer Engineering Department

Data Structure and Algorithm

Lab Manual # 04

Lab Instructor: Engr. Shahryar Khan

University of Engineering and Technology Taxila
Computer Engineering Department

Lab Title: Practicing Linked Lists for Efficient Data Management

Lab Overview

This lab focuses on Singly Linked Lists and their real-world applications. Students will implement insertion, deletion, searching, traversing, and updating nodes while exploring Linked Lists' practical use cases. The tasks will be implemented using classes and objects in Python. By the end of this lab, students will develop an in-depth understanding of Linked Lists and how they are utilized in various computer science applications such as memory management, navigation systems, task scheduling, and version control systems.

Lab Objectives

- Implement Singly Linked List using classes and objects in Python.
- Perform basic and advanced operations (insertion, deletion, traversal, searching, and updating).
- Understand memory efficiency and dynamic allocation using Linked Lists.
- Solve real-world problems, such as task management, version control systems, and navigation history tracking.
- Compare Linked Lists with other data structures like Arrays.

Lab Requirements

- **Python Environment:** Ensure Python 3.10+ is installed. Use the official Python website for downloads (https://python.org).
- **VSCode Installation:** Download and install Visual Studio Code (https://code.visualstudio.com).
- **Documentation**: Use the official Python documentation for reference (https://docs.python.org/3).Reference Official Documentation: Use the Python official documentation (https://docs.python.org/3/) and sample programs to enhance learning.

Note: For the guided tasks type the code yourself.

Guided Tasks

Task 1: Basic Singly Linked List Implementation

Objective: Implement a simple Singly Linked List class to understand node creation and traversal.

Implementation:

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

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

    def append(self, data):
        """Append a new node to the end of the list."""
        node = Node(data)
        if not self.head:
            self.head = node
        else:
            current = self.head
            while current.next:
                current = current.next
```

```
        current.next = node


    def display(self):
        """Print all elements of the list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")


# Example Usage
sll = SinglyLinkedList()
sll.append(10)
sll.append(20)
sll.append(30)
sll.display()
```

Task 2: Insertion at the Beginning, Middle, and End

Objective: Implement three types of insertions in a linked list.

Implementation:

```
class SinglyLinkedList:
    def insert_at_beginning(self, data):
        """Insert a node at the beginning of the linked list."""
        node = Node(data)
        node.next = self.head
        self.head = node
```

```
  def insert_at_position(self, data, pos):
    """Insert a node at a specific position."""
    node = Node(data)
    current = self.head
    for _ in range(pos - 1):
      if current.next is None:
        break
      current = current.next
    node.next = current.next
    current.next = node


# Example Usage
sll.insert_at_beginning(5)
sll.insert_at_position(15, 2)
sll.display()
```

Task 3: Deletion of Nodes

Objective: Implement deletion of head node, middle node, and last node.

Implementation:

```
class SinglyLinkedList:
  def delete(self, data):
    """Delete a node by value."""
    current = self.head
    prev = None

    while current:
      if current.data == data:
```

```
            if prev:
                prev.next = current.next
            else:
                self.head = current.next
            return
        prev = current
        current = current.next


# Example Usage
sll.delete(20)  # Deletes 20 from the list
sll.display()
```

Task 4: Searching in a Linked List

Objective: Implement search operation to check if a value exists in the Linked List.

Implementation

```
class SinglyLinkedList:
    def search(self, data):
        """Search for an element in the linked list."""
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False


# Example Usage
print(sll.search(15))  # True
```

```
print(sll.search(100))  # False
```

Task 5: Reverse a Linked List

Objective: Implement a function to reverse the linked list.

Implementation:

```
class SinglyLinkedList:
    def reverse(self):
        """Reverse the linked list."""
        prev = None
        current = self.head
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        self.head = prev

# Example Usage
sll.reverse()
sll.display()
```

---------- xxxxx ------- xxxx ----------

Exercise Questions

Easy Problems

1. **Student Name List**: Store and manage a **list of student names** in a linked list.
2. **Task Scheduler**: Implement a simple **task manager** where users can add/remove tasks.
3. **Contact List**: Create a **contact list** using linked lists where users can **search by name**.

4. **Undo Feature in Editor**: Implement a **basic undo feature** where previous actions are stored.
5. **Simple Playlist Manager**: Store a list of **songs** and provide a method to display them.

Intermediate Problems

1. **Version Control System**: Simulate a **Git commit history** where commits are stored in a linked list.
2. **Hospital Patient Queue**: Implement a **queue system** where patients are treated in order.
3. **Web Browser Navigation**: Implement a **forward/backward navigation** in a web browser.
4. **File Management System**: Simulate a **hierarchical file system** using linked lists.
5. **Movie Recommendation System**: Store user ratings and suggest similar movies.

Advanced Problems

1. **Facebook Messenger Chat History**
   - Implement a **chat system** where messages are stored in a **linked list** and retrieved in order.
   - **Hint**: Store messages as **nodes** with timestamps.
2. **LinkedIn Profile Connections**
   - Implement a **user profile system** where each user is a node connected to other users.
   - **Hint**: Each node contains a list of **connections**.
3. **Google Docs Edit History**
   - Simulate **edit history tracking** in Google Docs.
   - **Hint**: Each node stores a **version of the document**.
4. **Pathfinding Algorithm in Maps**
   - Store a **series of locations** in a **linked list** and allow traversal.
   - **Hint**: Each node represents a **location**.
5. **Blockchain Implementation**
   - Simulate a **simple blockchain** where each block stores transactions.
   - **Hint**: Use linked list nodes to represent **blocks**.

All the best