Assignment 10 (Linked List)

> **Name: Mohammed Varaliya**

> **Roll No: 54**

Questions

1. **A method display() to print all elements in the list.**
2. **A method calculate_total() to calculate sum of all the elements.**
3. **A method node_count() for counting nodes in a linked list.**
4. **A method max() for finding the node that holds maximum value in a linked list.**
5. **A method search() to find a node with a specified value.**

---

1. **A method display() to print all elements in the list.**
    a. **Code:**

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

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

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)
```

```
        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()
```

```
# A method display() to print all elements in the list.

def display_list_iterative(self):
    cur_node = self.head
    while cur_node:
        print(cur_node.data, end=" -> ")
        cur_node = cur_node.next
```

1. **Explanation:**
    a. The `display_list_iterative` method prints all the elements of the singly linked list in a readable format. Here's how it works:
    b. **Initialization**: It starts by setting `cur_node` to the `head` of the list.
    c. **Traversal**: It enters a loop where it prints the `data` of the current node followed by an arrow (`->`), then moves to the next node by updating `cur_node` to `cur_node.next`.
    d. **End of List**: When it reaches the last node (i.e., when `cur_node` becomes `None`), the loop stops. After that, it prints `None` to signify the end of the list.

b. **Output:**

```
2 -> 5 -> 8 -> 1 -> None
```

---

2. **A method calculate_total() to calculate sum of all the elements.**
    a. **Code:**

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

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

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")

    def cal_total_node(self):
        sum = 0
        cur_node = self.head
        while cur_node:
          sum += cur_node.data
          cur_node = cur_node.next

        return su

if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method calculate_total() to calculate sum of all the elements.
    print("\nSum of all nodes in singly linked list are: ", llist.cal_total_node())
```

```
# A method calculate_total() to calculate sum of all the elements.

def cal_total_node(self):
    sum = 0
    cur_node = self.head
    while cur_node:
      sum += cur_node.data
      cur_node = cur_node.next

    return sum
```

1. **Explanation:**
    a. The `cal_total_node` method calculates the sum of all elements in the singly linked list. Here's how it works:
    b. **Initialization**: It starts by initializing a variable `sum` to 0, which will store the total sum of node values.
    c. **Traversal**: It then iterates through the list starting from the `head`. For each node, it adds the `data` of the node to `sum` and moves to the next node (`cur_node = cur_node.next`).
    d. **End of List**: The loop continues until it reaches the end of the list (i.e., when `cur_node` is `None`).
    e. **Return Total**: Once all nodes are processed, it returns the total sum of all node values.

b. **Output:**

```
2 -> 5 -> 8 -> 1 -> None

Sum of all nodes in singly linked list are:  16
```

---

3. **A method node_count() for counting nodes in a linked list.**
    a. **Code:**

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

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

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)
```

```
        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")

    def node_count(self):
        total = 0
        cur_node = self.head
        while cur_node:
          total += 1
          cur_node = cur_node.next

        return total


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method node_count() for counting nodes in a linked list.
    print("\nTotal num of Nodes present in the singly linked list are: ", llist.node
_count())
```

```
# A method node_count() for counting nodes in a linked list.

def node_count(self):
    total = 0
    cur_node = self.head
    while cur_node:
      total += 1
      cur_node = cur_node.next

    return total
```

1. **Explanation:**
    a. The `node_count` method counts the number of nodes in the singly linked list. Here's how it works:
    b. **Initialization**: It initializes a variable `total` to 0, which will keep track of the number of nodes.
    c. **Traversal**: It starts from the `head` of the list and iterates through each node. For every node, it increments the `total` by 1 and moves to the next node (`cur_node = cur_node.next`).

d. **End of List**: The loop continues until it reaches the end of the list (i.e., when `cur_node` becomes `None`).

e. **Return Count**: Once all nodes have been processed, it returns the total count of nodes.

b. **Output:**

```
2 -> 5 -> 8 -> 1 -> None

Total num of Nodes present in the singly linked list are:  4
```

---

4. **A method max() for finding the node that holds maximum value in a linked list.**

a. **Code:**

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

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

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")

    def max_node(self):
        max = 0
        cur_node = self.head
        while cur_node:
          if cur_node.data > max:
```

```
            max = cur_node.data
            cur_node = cur_node.next
          else:
            cur_node = cur_node.next

        return max


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method max() for finding the node that holds maximum value in a linked list.
    print("\nNode with maximum data in the singly linked list is: ", llist.max_node
())
```

```
# A method max() for finding the node that holds maximum value in a linked list.

def max_node(self):
    max = 0
    cur_node = self.head
    while cur_node:
      if cur_node.data > max:
        max = cur_node.data
        cur_node = cur_node.next
      else:
        cur_node = cur_node.next

    return max
```

1. **Explanation:**
    a. The `max_node` method finds the node that holds the maximum value in the singly linked list. Here's how it works:
    b. **Initialization**: It starts by setting the `max` variable to 0 (or any value smaller than the smallest expected node value).
    c. **Traversal**: It then iterates through the list starting from the `head`. For each node, it compares the node's `data` with the current `max`. If the node's value is greater than `max`, it updates `max` with the node's value.
    d. **End of List**: The loop continues until it has checked all nodes (i.e., when `cur_node` becomes `None`).
    e. **Return Maximum**: Finally, it returns the highest value found in the list.

b. **Output:**

```
2 -> 5 -> 8 -> 1 -> None

Node with maximum data in the singly linked list is:  8
```

---

5. **A method search() to find a node with a specified value.**

a. **Code:**

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

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

    def append(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def prepend(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            return

        cur_node = self.head
        self.head = new_node
        new_node.next = cur_node

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")

    def search_node(self, node):
        index = 1
        cur_node = self.head
        while cur_node:
            if cur_node.data == node:
                print(f"\nNode {node} found at index {index}")
                return
            index += 1
            cur_node = cur_node.next

        print(f"\nNode {node} is not present in the singly linked list")


if __name__ == "__main__":

    llist = SinglyLinkedList()
    llist.append(2)
```

```
    llist.append(5)
    llist.append(8)
    llist.append(1)

    # A method display() to print all elements in the list.
    llist.display_list_iterative()

    # A method search() to find a node with a specified value.
    llist.search_node(1)
```

```
# A method search() to find a node with a specified value.

def search_node(self, node):
    index = 1
    cur_node = self.head
    while cur_node:
        if cur_node.data == node:
            print(f"\nNode {node} found at index {index}")
            return
        index += 1
        cur_node = cur_node.next

    print(f"\nNode {node} is not present in the singly linked list")
```

1. **Explanation:**
    a. The `search_node` method searches for a node with a specified value in the singly linked list. Here's how it works:
    b. **Initialization**: It initializes an `index` variable to 1 (since the list is 1-indexed) and starts at the `head` of the list.
    c. **Traversal**: It iterates through the list, checking each node's `data`. If the node's `data` matches the specified value (`node`), it prints the index where the node is found and returns.
    d. **End of List**: If the loop reaches the end of the list without finding the specified value, it prints a message saying the node is not present in the list.
    e. **Return Result**: The method either prints the index of the found node or a message indicating the node is not in the list.

b. **Output:**

```
2 -> 5 -> 8 -> 1 -> None

Node 1 found at index 4
```

---

1. **GitHub Code Explanation with example walkthrough.**

MCA-Coursework/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-10 at main · Mohammedvaraliya/MCA-Coursework
Contribute to Mohammedvaraliya/MCA-Coursework development by creating an account on GitHub.

https://github.com/Mohammedvaraliya/MCA-Coursework/tree/main/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-10
