Introduction to Linked list

Assignment Solutions

PW SKILLS

1. In a singly linked list, deletion of data requires modification of how many pointers?

1
2
3
Depends upon the node being deleted.

Ans: It depends upon the node being deleted. Specifically, for the most common cases:

- Deleting the head node requires modifying 1 pointer (the head pointer).
- Deleting any other node requires modifying 1 pointer (the next pointer of the previous node to point to the next node of the node being deleted).

2. Predict the output for linked list = 1->2->3->4->5:

```
def traverse(head):
while head is not None and head.next is not None:
 print(head.data, end=" ")
     head = head.next.next
```

1 2 3 4 5
1 3 5
2 4
1 3

Ans: 1 3 5

Explanation: The `traverse` function iterates over the linked list, printing the data of the current node, and then moving two steps forward (`head = head.next.next`). This skips every second node.

3. Implement a Linked List class.
The user-defined LL should have insert (head,tail,idx) , delete(head,tail,idx) , get(idx) and display functions.

Ans:

```
class ListNode:
    def __init__(self, data=0, next=None):
        self.data = data
        self.next = next

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

    def insert(self, idx, data):
        new_node = ListNode(data)
        if idx == 0:  # Insert at head
            new_node.next = self.head
            self.head = new_node
            if self.tail is None:
                self.tail = new_node
            return
```

```
current = self.head
        count = 0
        while current and count < idx - 1:
            current = current.next
            count += 1

        if current is None:
            if self.tail is None:
                self.head = new_node
            else:
                self.tail.next = new_node
            self.tail = new_node
        else:
            new_node.next = current.next
            current.next = new_node
            if new_node.next is None:
                self.tail = new_node

    def delete(self, idx):
        if self.head is None:
            return

        if idx == 0:  # Delete head
            self.head = self.head.next
            if self.head is None:
                self.tail = None
            return

        current = self.head
        count = 0
        while current.next and count < idx - 1:
            current = current.next
            count += 1

        if current.next:
            current.next = current.next.next
            if current.next is None:
                self.tail = current

    def get(self, idx):
        current = self.head
        count = 0
        while current and count < idx:
            current = current.next
            count += 1

        if current is None:
            return None
        return current.data

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" → ")
            current = current.next
        print("None")

# Usage example
ll = LinkedList()
ll.insert(0, 1)  # Insert 1 at head
ll.insert(1, 2)  # Insert 2 at tail
ll.insert(2, 3)  # Insert 3 at tail
ll.insert(1, 4)  # Insert 4 at index 1
ll.display()  # Expected Output: 1 → 4 → 2 → 3 → None

print("Element at index 2:", ll.get(2))  # Expected Output: 2

ll.delete(1)  # Delete element at index 1
ll.display()  # Expected Output: 1 → 2 → 3 → None

ll.delete(0)  # Delete head
ll.display()  # Expected Output: 2 → 3 → None
```