Linked Lists

Mayank Pratap Singh

Week 4

1 Introduction

A linked list is a linear collection of data elements called nodes, where the linear order is not given by their physical placement in memory. Instead, each node points to the next node by means of a pointer. Linked lists provide a way to store a collection of items and support efficient insertion and deletion.

2 Representation in Memory

Each node in a linked list is composed of:

- **Data field**: Contains the data.
- **Link field**: Contains the address of the next node.

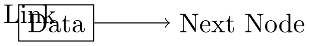

Figure 1: Node Structure

3 Traversing a Linked List

Traversing a linked list involves processing each node exactly once. The algorithm for traversing is:

- Set $P :=$ START.
- Repeat while $P \neq$ NULL:
  - Process $DATA[P]$.
  - Set $P :=$ LINK[P].

4 Searching a Linked List

Searching involves finding the location of an ITEM in the list. There are two scenarios:

4.1 Unsorted List

- Set $P :=$ START.
- Repeat while $P \neq$ NULL:
  - If $ITEM =$ DATA[P], then set $LOC := P$ and exit.
  - Else set $P :=$ LINK[P].
- If search is unsuccessful, set $LOC :=$ NULL.

4.2 Sorted List

- Set $P :=$ START.
- Repeat while $P \neq$ NULL:
  - If $ITEM <$ DATA[P], then set $P :=$ LINK[P].
  - Else if $ITEM =$ DATA[P], then set $LOC := P$ and exit.
  - Else set $LOC :=$ NULL and exit.
- If search is unsuccessful, set $LOC :=$ NULL.

5 Memory Allocation and Garbage Collection

Memory allocation for linked lists involves maintaining a list of available space (free storage list or free pool). Garbage collection periodically collects all deleted space onto the free storage list.

6 Insertion into a Linked List

Insertion is the process of adding a new node. It requires two pointer maneuvers:

- Remove the first node from the AVAIL list.
- Set $DATA[N] := ITEM$.
- Set $LINK[N] :=$ START and $START := N$.

7 Deletion from a Linked List

Deletion involves removing a node by adjusting pointers:

- If $LOC1 = \text{NULL}$, set $START := \text{LINK}[START]$.
- Else set $LINK[LOC1] := \text{LINK}[LOC]$.
- Return the deleted node to the AVAIL list by setting $LINK[LOC] := AVAIL$ and $AVAIL := LOC$.

8 Types of Linked Lists

8.1 Singly Linked List

A singly linked list allows traversal in one direction only.

8.2 Doubly Linked List

A doubly linked list allows traversal in both directions by maintaining two pointers in each node:

- **Next link field**: Points to the next node.
- **Previous link field**: Points to the previous node.

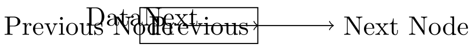

Figure 2: Doubly Linked List Node Structure

8.3 Circular Linked List

In a circular linked list, the null pointer in the last node is replaced with the address of the first node, forming a circle.

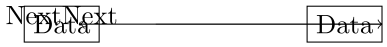

Figure 3: Circular Linked List

9 Comparison of Array and Linked List

10 Python Implementation of Linked List

	Array	**Linked List**
Memory Allocation	Contiguous	Non-contiguous
Insertion/Deletion	O(n)	O(1)
Random Access	Available	Not available
Memory Overhead	Less	More (due to pointers)

Table 1: Comparison of Array and Linked List

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

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

    def insert_at_beginning(self, new_data):
        new_node = Node(new_data)
        new_node.next = self.head
        self.head = new_node

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

# Example usage
llist = LinkedList()
llist.insert_at_beginning(10)
llist.insert_at_beginning(20)
llist.insert_at_beginning(30)
llist.print_list()
```