Python Chapter 6: Linked Lists

Ezequiel Torres

June 23, 2024

Table of contents

Class Example

A linked list is a data structure which holds a collection of data in a chain. It's pros are being able to insert and delete much faster than a normal array. But, it is much slower to index.

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

Linked List Insertion at Beginning

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

    # Method to add a node at begin of LL
    def insertAtBegin(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        else:
            new_node.next = self.head
            self.head = new_node
```

Linked List Insertion at Index

```
# Method to add a node at any index
# Indexing starts from 0.
def insertAtIndex(self, data, index):
    new_node = Node(data)
    current_node = self.head
    position = 0
    if position == index:
        self.insertAtBegin(data)
    else:
        while(current_node != None and position
            position = position+1
            current_node = current_node.next

        if current_node != None:
            new_node.next = current_node.next
            current_node.next = new_node
        else:
            print("Index not present")
```

Linked List Insertion at End

```
# Method to add a node at the end of LL
def insertAtEnd(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        return

    current_node = self.head
    while(current_node.next):
        current_node = current_node.next

    current_node.next = new_node
```

Linked List Update Node

```
# Update node of a linked list
# at given position
def updateNode(self, val, index):
    current_node = self.head
    position = 0
    if position == index:
        current_node.data = val
    else:
        while(current_node != None and position
            position = position+1
            current_node = current_node.next

        if current_node != None:
            current_node.data = val
        else:
            print("Index not present")
```

Linked List Remove First and Last Node

```
# Method to remove first node of linked list
    def remove_first_node(self):
        if(self.head == None):
            return
        self.head = self.head.next
    # Method to remove last node of linked list
    def remove_last_node(self):
        if self.head is None:
            return
        current_node = self.head
        while(current_node.next.next):
            current_node = current_node.next
        current_node.next = None
```

Linked List Remove at index

```
# Method to remove at given index
def remove_at_index(self, index):
    if self.head == None:
        return
    current_node = self.head
    position = 0
    if position == index:
        self.remove_first_node()
    else:
        while(current_node != None and position
            position = position+1
            current_node = current_node.next
        if current_node != None:
            current_node.next = current_node.ne
        else:
            print("Index not present")
```

Linked List Remove Node

```
# Method to remove a node from linked list
    def remove_node(self, data):
        current_node = self.head
        if current_node.data == data:
            self.remove_first_node()
            return
        while(current_node != None and current_node
            current_node = current_node.next
        if current_node == None:
            return
        else:
            current_node.next = current_node.next.n
```