جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Lap: (6)

Linked List II

Subject: Data Strectuer

Class: second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

Update the Node of a Linked List

This code defines a method called updateNode in a linked list class. It is used to update the value of a node at a given position in the linked list.

 # 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 != index):
            position = position+1
            current_node = current_node.next
        if current_node != None:
            current_node.data != val
            else:
                print("Index not present")
```

Delete Node in a Linked List

Remove First Node from Linked List

This method removes the first node of the linked list simply by making the second node head of the linked list.

```
def remove_first_node(self):
  if(self.head == None):
    return
  self.head = self.head.next
```

Remove Last Node from Linked List

In this method, we will delete the last node. First, we traverse to the second last node using the while loop, and then we make the next of that node None and last node will be removed.

```
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
```

Delete a Linked List Node at a given

Position In this method, we will remove the node at the given index, this method is similar to the insert_at_inded() method. In this method, if the head is None we simply return else we initialize a current_node with self.head and position with 0. If the position is equal to the index we called the remove_first_node() method else we traverse to the one node before that we want to remove using the while loop. After that when we out of the while loop we check that current_node is equal to None if not then we make the next of current_node equal to the next of node that we want to remove else we print the message “Index not present” because current_node is equal to None.

```
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+1 != index):
            position = position+1
            current_node = current_node.next
        if current_node != None:
            current_node.next = current_node.next.next
        else:
            print("Index not present")
```

Delete a Linked List Node of a given Data

This method removes the node with the given data from the linked list. In this method, firstly we made a current_node equal to the head and run a while loop to traverse the linked list. This while loop breaks when current_node becomes None or the data next to the current node is equal to the data given in the argument. Now, After coming out of the loop if the current_node is equal to None it means that the node is not present in the data and we just return, and if the data next to the current_node is equal to the data given then we remove that node by making next of that removed_node to the next of current_node. And this is implemented using the if else condition.

```
def remove_node(self, data):
```

```
    current_node = self.head
    while(current_node != None and current_node.next.data != data):
        current_node = current_node.next
    if current_node == None:
        return
    else:
        current_node.next = current_node.next.next
```

Linked List Traversal in Python

This method traverses the linked list and prints the data of each node. In this method, we made a current_node equal to the head and iterate through the linked list using a while loop until the current_node become None and print the data of current_node in each iteration and make the current_node next to it.

```
def printLL(self):
    current_node = self.head
    while(current_node):
        print(current_node.data)
        current_node = current_node.next
```

Get Length of a Linked List

in Python This method returns the size of the linked list. In this method, we have initialized a counter 'size' with 0, and then if the head is not equal to None we traverse the linked list using a while loop and increment the size with 1 in each iteration and return the size when current_node becomes None else we return 0.

```
def sizeOfLL(self):
    size = 0
    if(self.head):
```

```
        current_node = self.head
        while(current_node):
            size = size+1
    current_node = current_node.next
        return size
    else:
        return 0
```

Example of the Linked list in Python

In this example, After defining the Node and LinkedList class we have created a linked list named "llist" using the linked list class and then insert four nodes with character data 'a', 'b', 'c', 'd' and 'g' in the linked list then we print the linked list using printLL() method linked list class after that we have removed some nodes using remove methods and then print the linked list again and we can see in the output that node is deleted successfully. After that, we also print the size of the linked list.

```
class Node:
  def __init__(self, data):
    self.data = data
    self.next = None
# Create a LinkedList class
class LinkedList:
  def __init__(self):
    self.head = None
  # Method to add a node at the beginning 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
  # 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 + 1 != index):
              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")
    # 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
    # Method to 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 != index):
                position = position + 1
                current_node = current_node.next
            if current_node != None:
                current_node.data = val
            else:
```

```
            print("Index not present")
    # 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 and current_node.next.next):
            current_node = current_node.next
        current_node.next = None


    # Method to remove a node 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 + 1 != index):
            position = position + 1
            current_node = current_node.next
        if current_node != None:
            current_node.next = current_node.next.next
        else:
            print("Index not present")
    # Method to remove a node from linked list
    def remove_node(self, data):
        current_node = self.head
        while(current_node != None and current_node.next and current_node.next.data != data):
            current_node = current_node.next
        if current_node == None or current_node.next == None:
            return
        else:
            current_node.next = current_node.next.next
    # Print the size of linked list
    def sizeOfLL(self):
        size = 0
        if self.head:
            current_node = self.head
            while(current_node):
                size = size + 1
                current_node = current_node.next
```

```
        return size
    else:
        return 0
  # Print method for the linked list
  def printLL(self):
    current_node = self.head
    while(current_node):
        print(current_node.data)
        current_node = current_node.next
# create a new linked list
llist = LinkedList()
# add nodes to the linked list
llist.insertAtEnd('a')
llist.insertAtEnd('b')
llist.insertAtBegin('c')
llist.insertAtEnd('d')
llist.insertAtIndex('g', 2)
# print the linked list
print("Node Data")
llist.printLL()
# remove a node from the linked list
print("\nRemove First Node")
llist.remove_first_node()
print("Remove Last Node")
```

```
llist.remove_last_node()

print("Remove Node at Index 1")

llist.remove_at_index(1)

# print the linked list again

print("\nLinked list after removing a node:")

llist.printLL()

print("\nUpdate node Value")

llist.updateNode('z', 0)

llist.printLL()

print("\nSize of linked list:", end=" ")

print(llist.sizeOfLL())
```

Output:

```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:/Users/ALFA/Desktop/python_3/123.PY
Node Data
c
a
g
b
d

Remove First Node
Remove Last Node
Remove Node at Index 1

Linked list after removing a node:
a
b

Update node Value
z
b

Size of linked list: 2
```