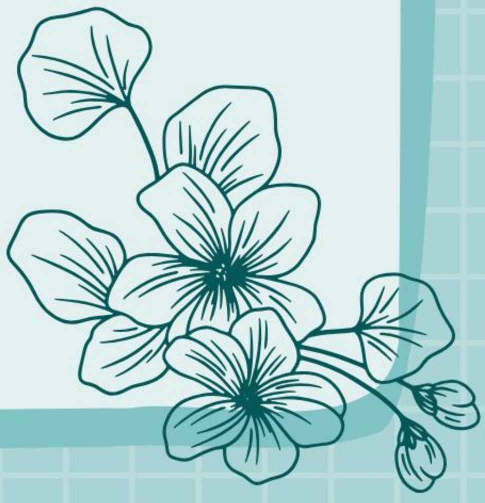
DSA In Python Linked list

Handwritten notes

-by Shanmuga Priya

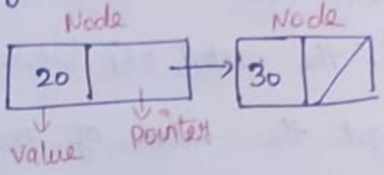
www.linkedin.com/in/shanmuga-priya-e-tech2



Linked List

1) what is Linked List?

→ Linked list is a collection of Nodes where each node contain value & pointer to next node.



→ Linked list are more flexible (i.e adding & removing node is easier)

→ we cannot access Linked list with index as they are not continous.

→ Here each node have their own memory address.

→ It consits of Head and Tail node.

Head: first node

Tail: last node where next is null.

2) Time Complexity analysis of Linked list?

append: O(1), Prepend: O(1), adding middle: O(n)

remove (pop): O(n), remove first node: O(1), removing middle: O(n)

lookup: O(n),

3) Comparison of Time complexity with list?

operations	Linked List	List
Append	O(1)	O(1)
Pop	O(n)	O(1)
prepend	O(1)	O(n)
pop first	O(1)	O(n)
Insert	O(n)	O(n)
remove	O(n)	O(n)
lookup by Index	O(n)	O(1)
lookup by value	O(n)	O(n)

4) How to create a node?

-> Node is nothing but a dict/obj with value and next which points to next node.

-> so L.L is the nested obj where each obj is a node.

-> we can get the value of node using ".value" and move to another node using ".next"

eg:
```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = none. -> as it is the 1st node
```

5) How to create a Linked List?

-> there are 3 steps involved in creating a L.L.

Step 1: creating a new node

Step 2: Create a head pointer and point to new node

Step 3: Create a tail pointer and point to new node.

eg:
```
class Linkedlist:
    def __init__(self, value):
        # creating a new node based on Node class
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1
my_linked_list = LinkedList(4)
```

-> It will create a new node with value 4 & assign a head & tail pointer to it

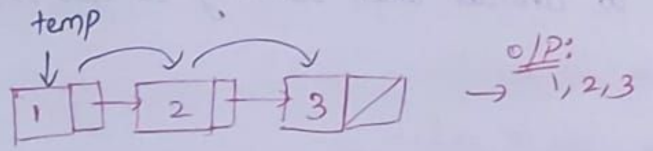
```
print(my_linked_list.head.value) -> o/p: 4.
```

6) How to print all the values in the L.L?

-> Step 1: we create a pointer "temp" which initially points on head.

-> Step 2: It print the value of that node & move to next node & start printing the value of that node.

-> Step 3: This printing & moves to next node continous until the next is not NULL.



eg:
```
def print_L.L (self):
    while temp:
        print ( temp. value) -> Printing of value
        temp = temp.next -> moving to next node
```

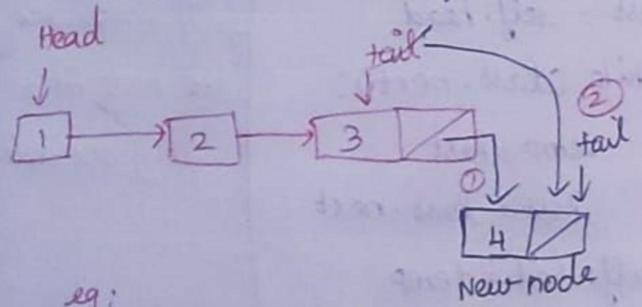
7) How to append a new node to the L.L?

step 1: Create a new node.

Step 2: make the tail next point to new node

Step 3: move tail to the new node.

-> If there is no item in L.L that means the new node is the 1st node so both head and tail both points to that new node.



eg:
```
def append ( self, value):
    new_node = Node (value) -> creating new node
    if self. head is None:
        self. head = new_node   |
                                 |-> if its a 1st node
        self. tail = new_node   |
else:
    self.tail.next = new_node
    self.tail = new_node.
self.length += 1.
```
→ Changing the tail node points to new node and tail to new node.

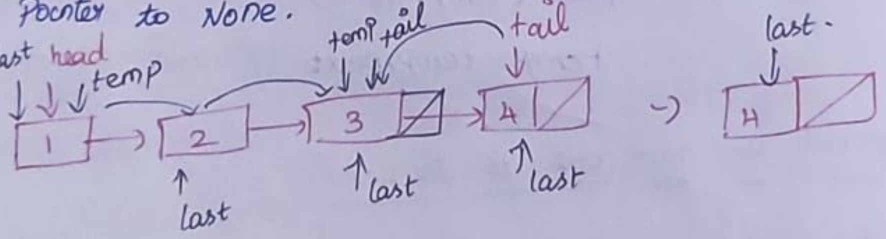
8) How to pop a node from L.L?

→ there are 2 cases to consider when removing an last node from a L.L.

*) No node in a L.L

*) Only one node in a L.L

→ loop through the L.L ~~and~~ until it reaches the Previous node of the tail node and make tail point to that node and make that node's Pointer to None.



eg:
```
def pop(self):
    if self.length == 0:
        return None
    temp = self.head
    last = self.head
    while (last.next):
        temp = last
        last = last.next
    self.tail = temp
    self.tail.next = None
    self.length -= 1.
    if self.length == 0:
        self.head = None
        self.tail = None
    return last.
```
→ No node in L.L.

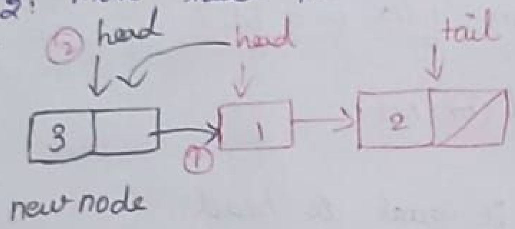
→ if there are more than one node in a L.L. (or) Only one node in a L.L

→ if there is only one node after removing that we need to make both head tail to None.

9) How to Prepend an node in L.L?

step 1: create a new node and set next point to the head

step 2: move head pointer to the new node.

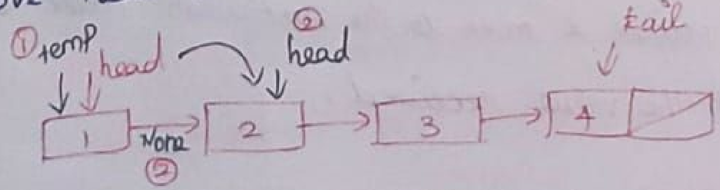


eg:
```
def prepend (self, value):
    new_node = Node(value)        -> creation of new node
    if self.length == 0:
        self.head = new_node      -> check it is a 1st node
        self.tail = new_node
    else:
        new_node.next = self.head -> Step 1
        self.head = new_node.     -> Step 2
    self.length += 1.
```

10) How to pop 1st node in a L.L?

Step 1: take a "temp" pointer and place it in a head.

Step 2: move head to temp.next and make temp.next to none.



eg:
```
def pop_first( self ):
    if self.length == 0:       -> if there is no node in L.L
        return None
    temp = self.head
    self.head = temp.next      -> step ① & step ②
    temp.next = None
    self.length -= 1
```

```
if self.length == 0:        ] -> if there is only 1 node after
    self.tail = None         ]    removing nodes from L.L.
return temp.
```

11) How to get an element at a Particular index in a L.L?

step 1: checking the index value is valid (or) not.

step 2: take a "temp" pointer & set it equal to head

step 3: Move this "temp" pointer to the Particular index using for loop & get the value.

eg:
```
def get (self, index):
    if index < 0 (or) index >= self.length:   ] -> checking index valid.
        return None

    temp = self.head               ]
    for _ in range (index):        ] -> step 2 & step 3
        temp = temp.next           ]
    return temp.value.             ]
```

12) How to set an value at a particular index in a L.L?

step 1: checking for index validity

step 2: create a "temp" pointer & move to the particular index using loop.

step 3: set value with the value received.

eg:
```
def set_value (self, index, value):
    temp = self.get (index)  -> logic to move to that particular
                                index is already defined in get

    if temp:
        temp.value = value   ] -> if temp is not None
        return True          ]
    return False -> if temp is None.
```

(3) How to insert a new node at a Particular index in a L.L?

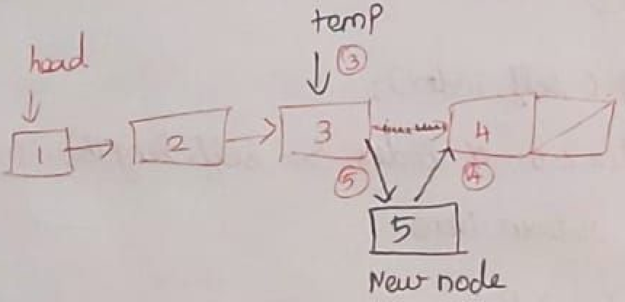
step1: create a new node and check for index validity

Step2: If the index is '0' we use 'Prepend' method, if the index is 'self.length' we use 'append' method.

Step3: take a "temp" pointer and move to one position before the index.

Step4: make temp.next equal to new_node.next to connect new node with the given index node.

Step5: make temp.next to new node. to connect it with the previous node.



eg:

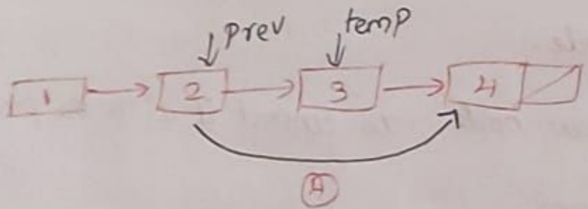
```
def insert(self, index, value):
    if index<0 or index > self.length:      ] -> index validati
        return False
    if index == 0:                           ] -> index is '0' we
        return self.prepend(value)               prepend it
    if index == self.length :                ] -> if index is equa
        return self.append(value)                to length we app
                                                 it
    new_node = Node(value)                   ] -> step 3, 4, 5
    temp = self.get(index-1)
    new_node.next = temp.~~g~~next
    temp.next = new-node
    self.length += 1
    return True.
```

(4) How to remove an item at a Particular index in a L.L?

step1: index validation & index equals to '0' use 'pop-first' method
if it equal to last item we use 'pop' method.

step2: take 2 pointer "prev" and "temp". we move "temp" to the position where the node has to be removed and "prev" to the previous node before it.

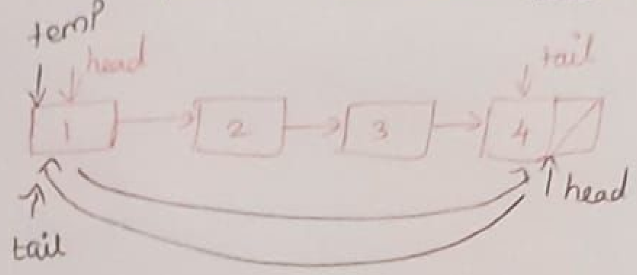
step3: we make prev.next to temp.next & temp.next to None.



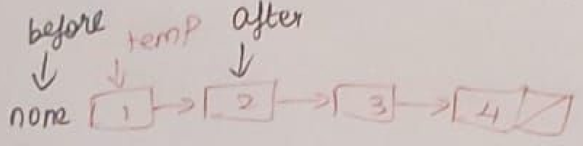
eg:
```
def remove (self, index):
    if index < 0 or index >= self.length:   ] -> index validation
        return None
    if index == 0:                           ] -> if 1st node
        return self.pop_first()
    if index == self.length - 1:             ] -> if last node
        return self.pop()
    prev = self.get(index - 1)
    temp = prev.next
    prev.next = temp.next                    ] -> step 2 & 3
    temp.next = None
    self.length -= 1
    return temp.
```

15) How to Reverse a L·L?

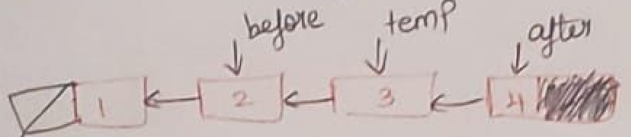
step 1: take "temp" pointer to act as intermediate for switching "head" & "tail" pointer. we move head to tail & tail to temp.



step 2: we are going to take another 2 pointers "before" & "after" which will be placed before & after "temp" pointer.



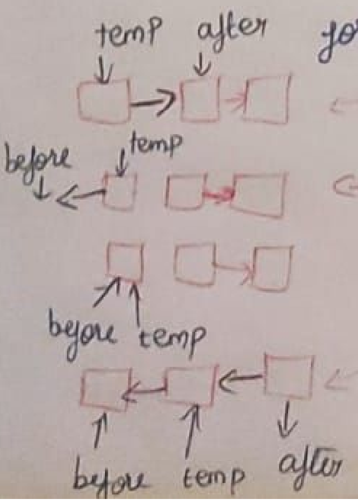
step 3: we are going to loop through the L·L & change the next values points to previous nodes using this 3 pointers



eg:
```
def reverse(self):
    temp = self.head
    self.head = self.tail        -> step ①
    self.tail = temp

    after = temp.next            -> ②
    before = None .

    for _ in range(self.length):
        after = temp.next
        temp.next = before       -> ③
        before = temp
        temp = after
```



temp
after
before
temp
before
temp
before
temp
after