Data Structures

Linked List Traversal

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

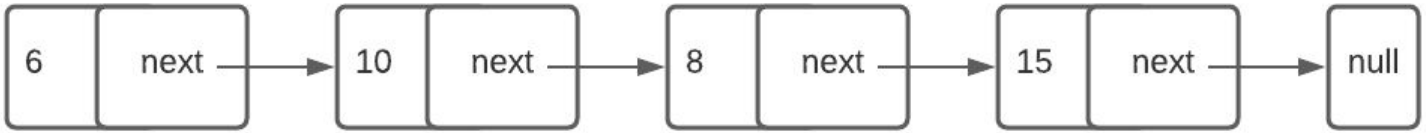
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Singly Linked List: traversal

- **Traversal** Terminology: Walk through the elements of a data structure.
- **Print function** is a traversal function, as it iterates over the elements
- Traversal is typically **O(n)** time for a complete iteration
  - All today traversal methods are O(n) time and O(1) memory
- We usually code the traversal **iteratively**, but we can do it **recursively**
- Many similar traversal problems exist
  - Examples: min, max, nodes sum, is sorted, search for an element, get_nth

Get Nth item

- Let’s implement: def get_nth(self, n)
  - This functions retrieves the nth node *where n is a 1-based integer*
    - If an nth node isn't found, it returns None
  - Below: get_nth(4) returns the Node with value 15
- Take 10 minutes to code it



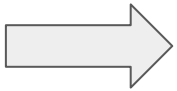
Get Nth item

- Just iterate and count up to N to find the nth node
- If the list ended without a matching count, then it doesn't exist

```
def get_nth(self, n):
    temp_head = self.head

    cnt = 1
    while temp_head is not None:
        if cnt == n:
            return temp_head
        temp_head = temp_head.next
        cnt += 1
    # still more steps needed - NOT found
    return None
```

Get Nth item

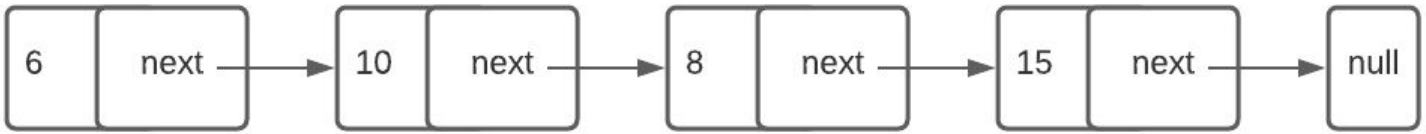
```
lst = LinkedList()
lst.insert_end(6)
lst.insert_end(10)
lst.insert_end(8)
lst.insert_end(15)

for n in range(1, 6):
    print(f'Find n={n} ==> {lst.get_nth(n)}')
```



```
Find n=1 ==> 6
Find n=2 ==> 10
Find n=3 ==> 8
Find n=4 ==> 15
Find n=5 ==> None
```

Search for an item

- Similarly, we can return a 0-based index of a node with a specific value
  - Or None if not found
- Here: list.index(15) ⇒ 3
  list.index(99) ⇒ None
- Implement it



Search for an item

```
def index(self, value):
    temp_head = self.head
    idx = 0

    while temp_head:    #  is not None
        if temp_head.data == value:
            return idx

        temp_head = temp_head.next
        idx += 1
    return None
```

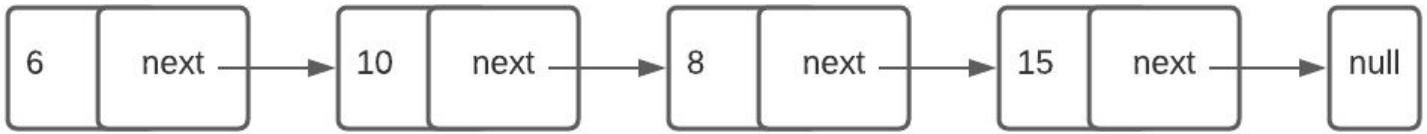
Search for an item

```
lst = LinkedList()
lst.insert_end(6)
lst.insert_end(10)
lst.insert_end(8)
lst.insert_end(15)

for value in [6, 10, 8, 15, 99]:
    print(f'Index of {value} ==> {lst.index(value)}')
```

```
Index of 6 ==> 0
Index of 10 ==> 1
Index of 8 ==> 2
Index of 15 ==> 3
Index of 99 ==> None
```

Improved Search

- Every time we find the element, we shift it one step **left**
- For example, searching for 15 will change this list to: 6 10 **15** 8
  - *Only values change, while the nodes objects are the same in the memory*
- Implement it



Improved Search

- A **common trick** is to maintain a copy of the previous node; this node is called 'previous' below:

```
def index_transposition(self, value):
    prev, cur = None, self.head
    idx = 0

    while cur:
        if cur.data == value:
            if not prev:
                return idx
            prev.data, cur.data = cur.data, prev.data
            return idx - 1
        prev, cur = cur, cur.next
        idx += 1
    return None
```

Improved Search

```
def index_transposition():
    lst = LinkedList()
    lst.insert_end(6)
    lst.insert_end(10)
    lst.insert_end(8)
    lst.insert_end(15)

    for value in [15, 15, 15, 15, 15]:
        print(f'Index of {value} ==> {lst.index_transposition(value)}')
        lst.print()

    for value in [8, 6, 99]:
        print(f'Index of {value} ==> {lst.index_transposition(value)}')
        lst.print()
```

```
Index of 15 ==> 2
6->10->15->8->
Index of 15 ==> 1
6->15->10->8->
Index of 15 ==> 0
15->6->10->8->
Index of 15 ==> 0
15->6->10->8->
Index of 15 ==> 0
15->6->10->8->
Index of 8 ==> 2
15->6->8->10->
Index of 6 ==> 0
6->15->8->10->
Index of 99 ==> None
6->15->8->10->
```

Code Enhancement

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

    def __repr__(self):...

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

        if initial_values:
            for value in initial_values:
                self.insert_end(value)
```

```
lst = LinkedList([6, 10, 8, 15])
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."