Data Structures

Doubly Linked List

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)

The Node & Linked List data structures

```
class Node:
    def __init__(self, data, next):
        self.data = data
        self.next = next
```

```
class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
        self.length = 0
```

- Different **design choices** will impact your code in data structures
  - Time & Memory Speed
  - Data assumption (e.g. data reversed)
  - Code simplicity!

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

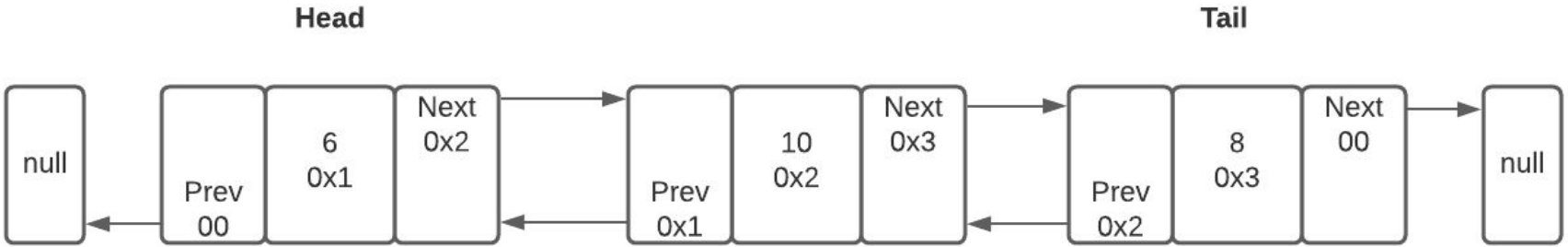
Doubly Linked List

- Our node has an linknode; this time to the **previous node**!
- This allows us to move **backward**, as well as forward, easily!

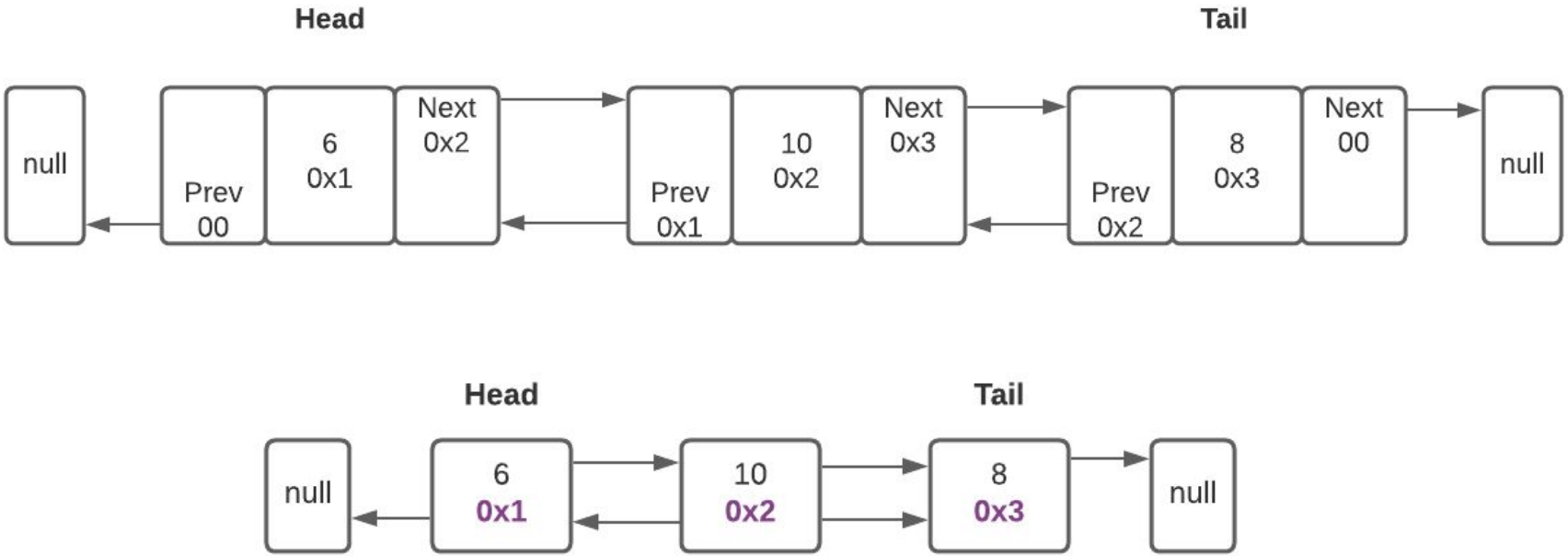
```
class Node:
    def __init__(self, data, next=None, prev=None):
        self.data = data
        self.next = next
        self.prev = prev

    def __repr__(self):
        return f'{self.data}'
```

Let’s visualize

- In **SLL**, node (6) is connected (with next) to node (10)
  - node1.next is node2
- In **DLL**, in addition to the above connection, node(10) is connected to node(6) with the **prev link**!
  - node2.prev is node 1
- Head has no previous node (None/Null)
- Tail has no next node (null)



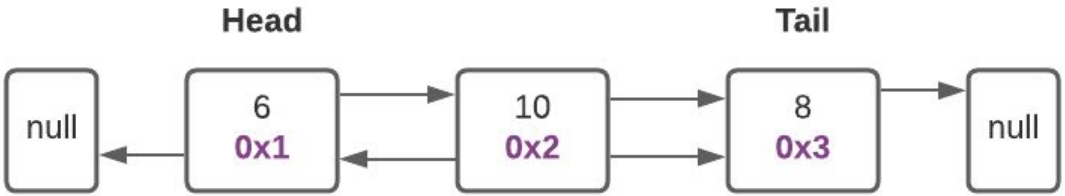
Simplifying the drawing



Print reversed

- Now we can go both forward and backward

```
def print_reversed(self):
    cur = self.tail

    while cur is not None:
        print(cur.data, end='->')
        cur = cur.prev
    print('None')
```



Visualizing the list

- Attached my updated code for the debugging template
  - Kindly review for the updates. Ignore insert_end function
- Here is how we can visualize the doubly list

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

```
None    6     -> 10     head
6       10    -> 8
10      8     -> 15
8       15    -> None   tail
```

Why?!

- In many scenarios, we need to get the previous node!
- We can easily get this node in O(n)!
- By adding a **'previous'** node, we can have it in O(1).
- **In return**, you have to maintain data integrity for this added pointer!
    - Minor concern: It takes up more space (another object)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."