Data Structures

Node Deletion

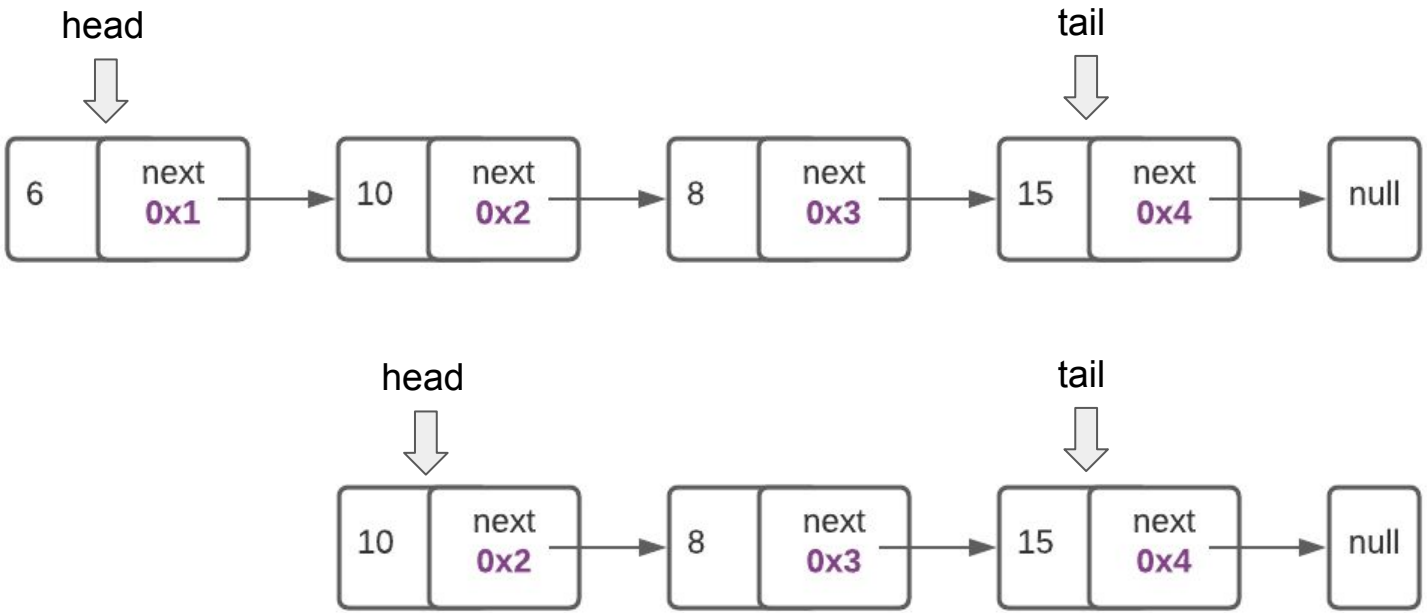
Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

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

Node deletion

- We typically might need 3 types of deletion for nodes
  - Delete the first node
  - Delete the last node
  - Delete the nth node or node with a value
- You know enough to code them by yourself
  - Think about the different cases for each
  - Draw the list before
  - Draw the list after each step

Delete first node

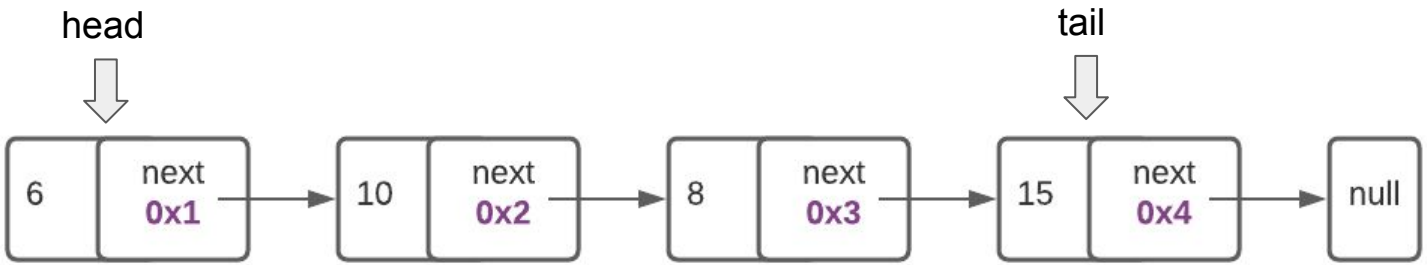


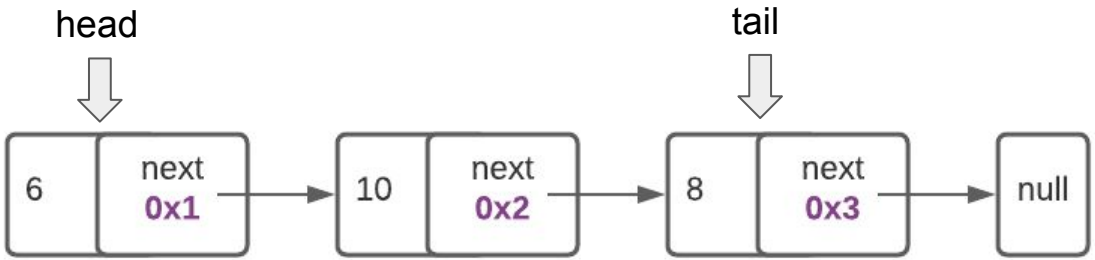
Delete first node

- We just need to make head.next the new head
  - Take copy first
- Properly handle the list when remains 0 or 1 elements


```
def delete_front(self):
    if not self.head:
        return

    next = self.head.next
    self._delete_node(self.head)
    self.head = next

    if self.length <= 1:
        self.tail = self.head

    self.debug_verify_data_integrity()
```

Delete last node



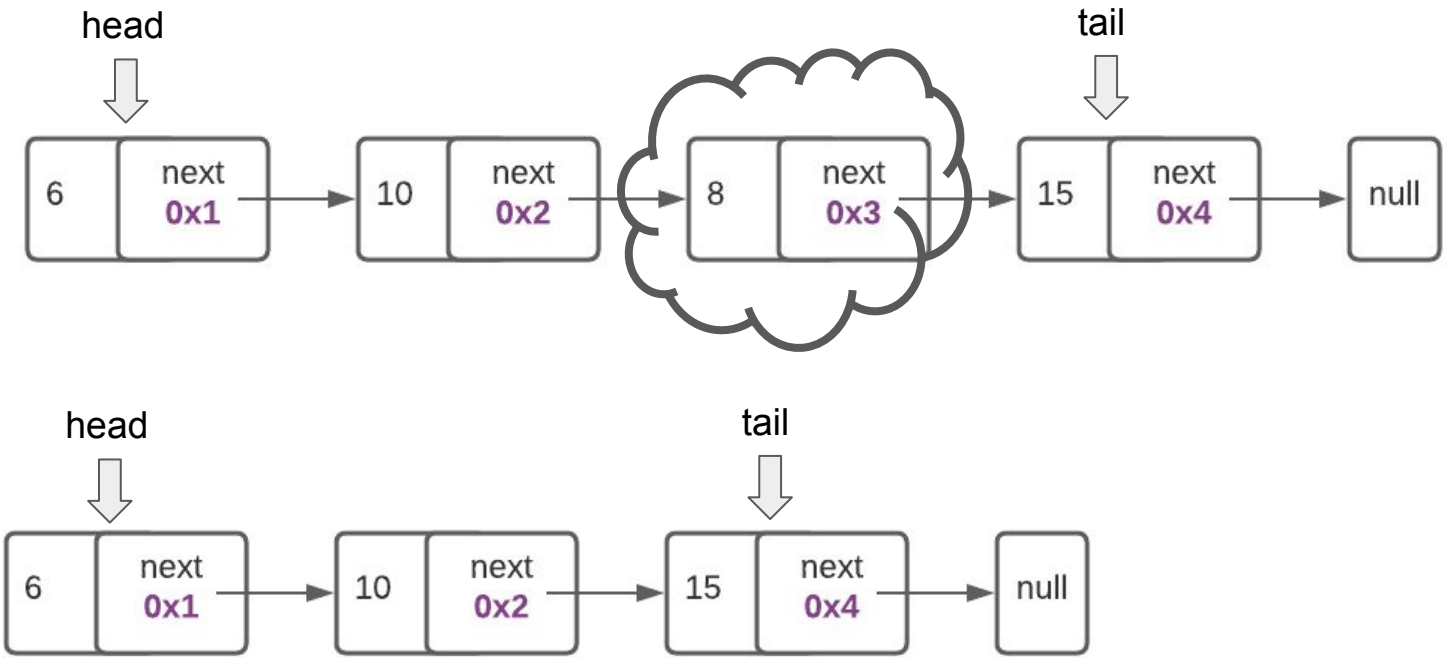


Delete last node

- We need to make the node immediately before the tail our new tail
- How to get it?
  - A simple loop can retrieve it
  - A better trick: use get_nth
  - get_nth(length-1) will retrieve the node immediately preceding the tail
- Also observe delete_last
  - We utilized 2 old functions!
- Delete the 'original' tail, make a new tail

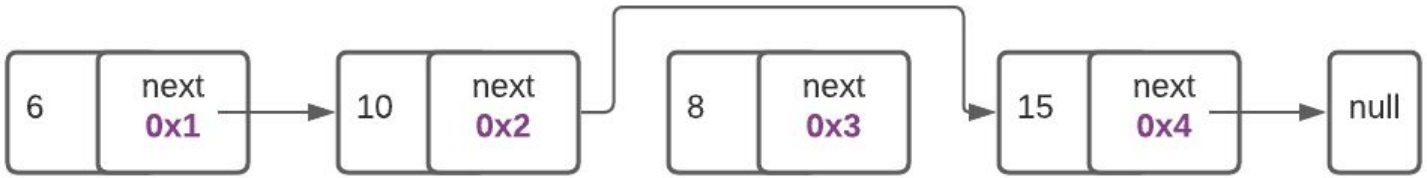
```
def delete_last(self):
    if self.length <= 1:
        self.delete_front()
        return

    # Tail is at length-1
    previous = self.get_nth(self.length - 1)

    self._delete_node(self.tail)
    self.tail = previous
    self.tail.next = None

    self.debug verify data integrity()
```

Delete nth node (e.g. 3)



Delete nth node

- The first node is a special case
- To handle this, we just need to link the (n-1) node with the (n+1) node
  - We need the node before nth and the node after nth
  - We need to connect them together
- Code trick
  - If we know the nth node, we can easily retrieve the n+1 node. But not the n-1 node!
  - Better: get the n-1 node. Then next is (n) and next.next is (n+1)



Delete nth node

```
def delete_nth_node(self, n):
    if n < 1 or n > self.length:
        print("Error. No such nth node")
    elif n == 1:
        self.delete_front()
    else:
        # Connect the node before nth with node after nth
        before_nth = self.get_nth(n - 1)
        nth = before_nth.next
        is_nth_tail = nth == self.tail
        # connect before node with after
        before_nth.next = nth.next

        if is_nth_tail:
            self.tail = before_nth

        self._delete_node(nth)
        self.debug_verify_data_integrity()
```

Tip

- In most of the medium/hard challenges in linked lists, we need to relink nodes
  - Delete nth node is an example for that
- You need to determine which links will be changed
  - From - To
- Order matters, especially if you are deleting
  - E.g. take its next first before deleting it
- Draw. Draw. Draw. Draw. Draw EVERY step

Linked List ADT

- The ADT of (Linked) List is a collection of data nodes accessed **sequentially**.
  - The main functionalities (interface): Add/Delete, Start, Next, Length
- We learned how to implement the list using a **linked list** data structure of (*head, tail and length*), and several variants to insert/delete items
- One clear disadvantage compared with arrays is the lack of **random access** which is O(1), but we gain flexibility in memory growth (no need for array reallocation)

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."