February 5, 2025

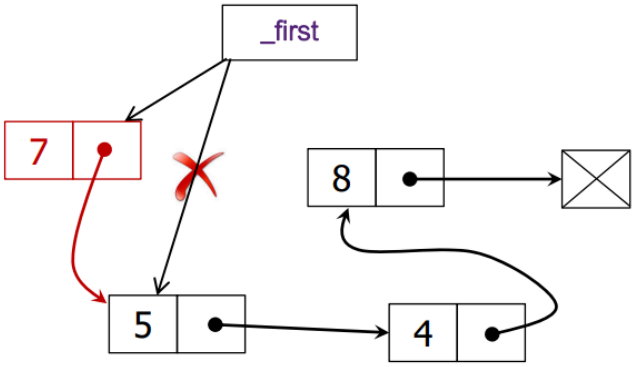
LEC 12 - Linked List Insertion and Deletion

Insertion

- We might want to implement all sorts of insert variations depending on what operations we want the linked list to support
  - Prepend → Add to start
  - Append → Add to end
  - Insert → Add at given index
- It is helpful to use diagrams to visualize such operations

Prepend

- Simply adjust the `_first` reference

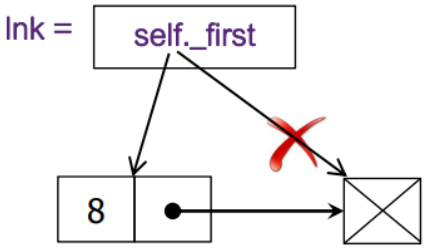


Edge Cases

- `len(self) == 0`
  - Instead of adding a new `_Node` object, we must mutate `_first`

Append

- We need to adjust some node inside the list
  - Either the first or last node
- We need to consider some cases
  - List is empty → Modify `_first`
  - List has 1 element → Modify `next` for `_first`



February 5, 2025

LEC 12 - Linked List Insertion and Deletion

Edge Cases

- `len(self) == 0`
    - We need to mutate `_first`

Key Ideas

- Figure out when to modify `self._first` vs a `_Node` object in the list
- When index > 0, iterate to the (index - 1)th node to update links

---

Delete / Pop

- Pop from an index allows us to also pop form the front and back of the list

Delete From Front

- Make `_first` reference the second node
- The former node will automatically be handled by garbage collection (memory management)

Tracking Previous Nodes

Strategy 1:

- Iterate to the node before the desired position

```
i = 0
curr = self._first
while not (curr is None or i == index - 1):
      curr = curr.next
      i += 1
```

February 5, 2025

LEC 12 - Linked List Insertion and Deletion

Strategy 2:

- Track the previous node explicitly

```
i = 0
prev = None
curr = self._first
while not (curr is None or i == index):
      prev, curr = curr, curr.next
      i += 1
```