Data Structures

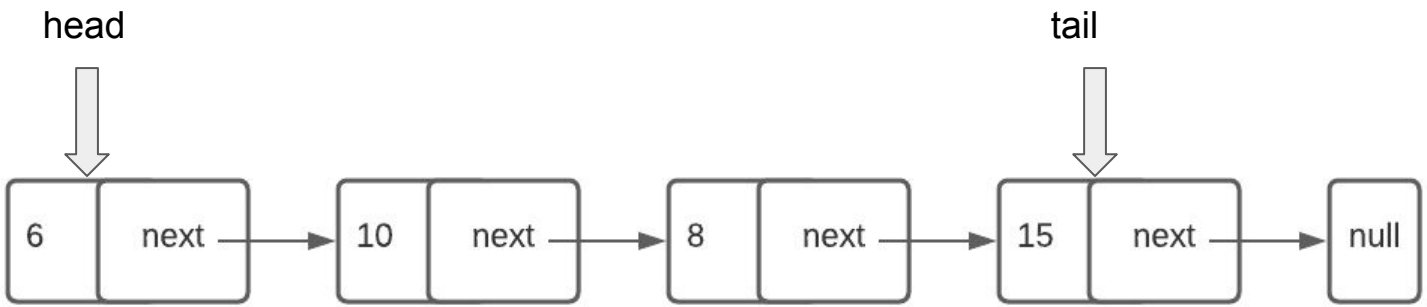
Linked-list-based Queue

Mostafa S. Ibrahim
Teaching, Training and Coaching since 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)

Similar to a Linked List-based Stack

- Clearly, we can use a linked list to solve the fixed memory issue
  - Downside: clearing a queue is O(n) using a linked list
- Head is actually our front and tail is our rear
  - All that we need to do is to add to the tail to enqueue, and remove the head to dequeue!



Aggregation

- We can re-use the code from the Singly Linked List section of the course, and rename parts of it
- In OOP/SWE, aggregation/composition allows your class to make use of objects constructed from other classes
- Many data structure ADTs can be implemented in terms of other data structures
    - We can implement a stack using a queue
    - We can implement a queue using a stack
    - And so on, regardless of complexity

Queue

- Use an object from the linked list
- Queue just **delegates** the calls to the linked list
- Note: now delete front returns the deleted value
- Observe: No test for full()
  - Linked list is not constrained

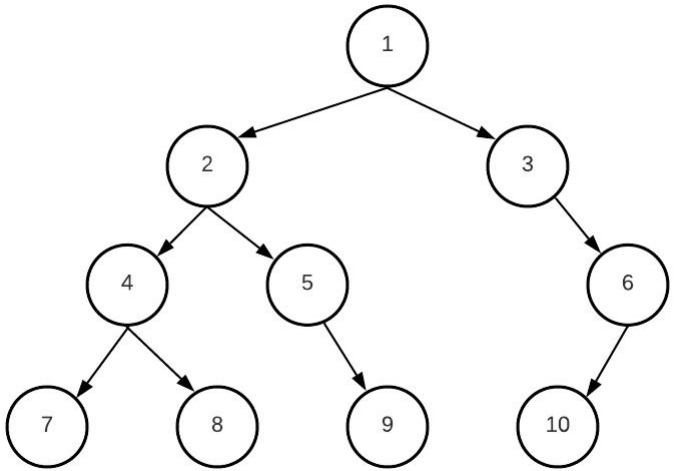
```
class Queue:
    def __init__(self):
        self.lst = LinkedList()

    def enqueue(self, value):
        self.lst.insert_end(value)

    def dequeue(self):
        assert not self.empty()
        return self.lst.delete_front()

    def empty(self):
        return self.lst.length == 0

    def display(self):
        if self.empty():
            print("EMPTY\n")
        else:
            print(self.lst)
```

Tips

- Stack: Follows FILO
  - It helps us easily reverse data - It involves some recursion: (()(((())))
- Queue: Follows FIFO
  - It can be used to simulate queues (restaurant, hospital, customer calls)
- **Lessons from homework (to keep in mind)**
  - We can use an available data structure to implement another (aggregation)
  - If you move data from the Queue to a Stack, and then back to the Queue: the Queue is now **reversed**
  - If a Queue has N elements, to add a new element at the front, we first enqueue the new element as normal. Then we enqueue the dequeue value, n times - enqueuing and dequeuing almost simultaneously
    - [1, 2, 3, 4] ⇒ insert front (5) ⇒ [1, 2, 3, 4, 5] ⇒ [5, 1, 2, 3, 4]
  - It can be used to act as a **sliding window** over data: [1, 2, 3, 4, 5, 6, 7, 8]
    - window=3: [1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6], … etc

Linear and Non-linear Data Structures

- **Linear** data structure: elements arranged & connected in a **sequential** manner
  - Array, Vector, Linked List Stack and Queue are linear data structures
- A **non-linear** data structure: elements can have **multiple paths** to connect to other elements.
  - Example include Tree / Trie (upcoming topics)
  - On the right, nodes are linked to form a binary tree



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."