CS218 - Data Structures
FAST NUCES Peshawar Campus
Dr. Nauman (recluze.net)

September 2, 2019

1 Circular Linked List in Python

Raster images of the notebook 06-circular-linked-list

Circular Linked List (or Ring Data Structure)

This is also very similar to a regular linked list -- except the last node is "linked" to the head thus creating a ring.

Due to this, we can't "loop until None is reached" because we will never have a None in the next now -- even for a single node! So, we need to keep track of the head and always loop "until we reach back to head".

In [ ]:
```
class Node:
    def __init__(self, data=None):
        self.val = data
        self.next = None

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

Let's explain this reasoning through trying to output the ring.

In [40]:
```
def __str__(self):
    ret_str = '['
    temp = self.head
    while temp is not None:
        ret_str += str(temp.val) + ', '
        temp = temp.next

        if temp == self.head:  # different for ring  (change)
            break

    ret_str = ret_str.rstrip(', ')
    ret_str += ']'

    return ret_str

Ring.__str__ = __str__
```

In [41]:
```
def _get_last(self):   # helper function (change)
    # no node, no last
    if self.head is None:
        return None

    # at least two nodes: advance once
    temp = self.head.next
    while temp.next != self.head:
        temp = temp.next

    return temp

Ring._get_last = _get_last
```

In [55]:
```
def insert(self, index, val):
    new_node = Node(val)

    last = self._get_last()  # need last for ring (change)

    # insertion at index 0 is different
    if index == 0:

        new_node.next = self.head
        self.head = new_node


        # also need to set the last pointer to this new head (change)
        if last is None:
            self.head.next = self.head # first node ever being inserted
        else:
            last.next = new_node


        return
```

```
    if self.head is None: #  and index > 0:
        raise IndexError("CAnnot insert at" + str(index) + " because list is empty")

    # for other indices
    temp = self.head

    counter = 0
    while temp is not None and counter < index:     # temp will never be None!
        prev = temp
        temp = temp.next
        counter += 1

    prev.next = new_node
    new_node.next = temp


Ring.insert = insert
```

In [62]:
```
r = Ring()
# r.insert(1, 1)
r.insert(0, 1)
r.insert(0, 2)
r.insert(1, 3)
r.insert(7, 5)  # different behavrior since it's a ring!
print(r)
```

```
[2, 5, 3, 1]
```

In [58]:
```
r._get_last().next == r.head  # just check if it's actually a ring
```

Out[58]:
```
True
```

In [ ]:
```
# r = Ring()
# r.insert(1, 2)
# print(r)
```

In [63]:
```
def remove(self, val):
    # empty ring case
    if self.head is None:
        return


    temp = self.head
    last = self._get_last() # (change)

    # first node matches case  # (change)
    if temp.val == val:
        if last == self.head:
            self.head = None # just one node. Now gone

        else:
            self.head = temp.next
            last.next = self.head
        return
```

```
    # let's move to next nodes
    # temp holds the value of the node that will be deleted
    prev = temp       # (change)
    temp = temp.next  # (change)

    while temp != self.head:  # (change)
        if temp.val == val:
            break

        prev = temp
        temp = temp.next

    if temp == self.head:  # not found
        return

    prev.next = temp.next  # just lose the reference to delete node

Ring.remove = remove
```

In [70]:
```
r = Ring()
r.insert(0, 1)
# r.insert(1, 2)
# r.insert(1, 3)
# r.insert(7, 5)

print(r)
r.remove(1)
print(r)
```

```
[1]
[1]
```