CSC148-Section:L0301/L0401 Week#4-Monday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Linked Lists
  - walking a linked list
  - Implement methods in LinkedListNode
  - Implement methods in LinkedList

a node class

```
class LinkedListNode:
    """
    Node to be used in linked list

    === Attributes ===
    value - data this LinkedListNode represents
    next_ - successor to this LinkedListNode
    """
    value: object
    next_: 'LinkedListNode'

    def __init__(self, value: object, next_:'LinkedListNode'=None)->None:
        """
        Create LinkedListNode self with data value and successor next_.
        """
        self.value, self.next_ = value, next_
```

Computer Science
UNIVERSITY OF TORONTO

a wrapper class for list

The list class keeps track of information about the entire list - such as its front, back, and size.

```
class LinkedList:
    """
    Collection of LinkedListNodes
    === Attributes ==
    front - first node of this LinkedList
    back - last node of this LinkedList
    size - number of nodes in this LinkedList a non-negative integer
    """
    front: LinkedListNode
    back: LinkedListNode
    size: int
    def __init__(self) -> None:
        """
        Create an empty linked list.
        """
        self.front, self.back, self.size = None, None, 0
```

Computer Science
UNIVERSITY OF TORONTO

walking a list

- Make a reference to (at least one) node, and move it along the list:

```
cur_node = self.front
while <some condition here...>:
    # do something here...
    cur_node = cur_node.nxt
```

Implement Methods in LinkedListNode

- __str__
- __eq__

__str__ in LinkedListNode

```
def __str__(self) -> str:
    """
    Return a user-friendly representation of this LinkedListNode.

    >>> n = LinkedListNode(5, LinkedListNode(7))
    >>> print(n)
    5 ->7 ->|
    """
```

__str__ in LinkedListNode

```
def __str__(self) -> str:
    """
    Return a user-friendly representation of this LinkedListNode

    >>> n = LinkedListNode(5, LinkedListNode(7))
    >>> print(n)
    5 ->7 ->|
    """
    cur_node = self
    result=''
    while cur_node is not None:
        result += '{} ->'.format(cur_node.value)
        cur_node = cur_node.next_
    return result+'|'
```

__eq__ in LinkedListNode

```
def __eq__(self, other: Any) -> bool:
    """
    Return whether LinkedListNode self is equivalent to other.

    >>> LinkedListNode(5).__eq__(5)
    False
    >>> n1 = LinkedListNode(5, LinkedListNode(7))
    >>> n2 = LinkedListNode(5, LinkedListNode(7, None))
    >>> n1.__eq__(n2)
    True
    """
```

__eq__ in LinkedListNode

```
def __eq__(self, other: Any) -> bool:
    """
     . . . . .
    """
    left_node=self
    right_node=other
    while (left_node is not None
           and right_node is not None
           and type(left_node) == type(right_node)
           and left_node.value == right_node.value):
        left_node = left_node.next_
        right_node = right_node.next_
    return right_node is None and left_node is None
```

Implement Methods in LinkedList

- __getitem__
- __contains__

__getitem__

```
def __getitem__(self, index: int) -> object:
    """
    Return the value at LinkedList self's position
    index,which must be a valid position in
    LinkedList self.

    >>> lnk = LinkedList()
    >>> lnk.prepend(1)
    >>> lnk.prepend(0)
    >>> lnk.__getitem__(1)
    1
    >>> lnk[-1]
    1
    """
```

- Python call when:

>>>lnk[0]

>>>lnk[2]

>>>lnk[-1]

__getitem__

- what is the output?

```
>>> lnk = LinkedList()
>>> lnk.prepend(2)
>>> lnk.prepend(1)
>>> lnk.prepend(0)


>>> lnk[1]
>>> lnk[3]
>>> lnk[-1]
>>> lnk[-2]
>>> lnk[-3]
>>> lnk[-4]
```

__getitem__

- We need to handle negative index?

```
if index < 0:
    index += self.size
```

```
>>> lnk = LinkedList()
>>> lnk.prepend(2)
>>> lnk.prepend(1)
>>> lnk.prepend(0)
```

```
>>> lnk[1]
>>> lnk[3]
>>> lnk[-1]
>>> lnk[-2]
>>> lnk[-3]
>>> lnk[-4]
```

Computer Science
UNIVERSITY OF TORONTO

__getitem__

- We need to handle out of range index?

```
>>> lnk = LinkedList()
>>> lnk.prepend(2)
>>> lnk.prepend(1)
>>> lnk.prepend(0)
>>> lnk[1]
>>> lnk[3]
>>> lnk[-1]
>>> lnk[-2]
>>> lnk[-3]
>>> lnk[-4]
```

```
if index >= self.size or self.size == 0 or index < 0:
    raise IndexError('index is out of bounds')
```

__getitem__

- Go through the list nodes till arrive to index?

```
current = self.front
for steps in range(index):
    current = current.next_
return current.value
```

```
>>> lnk = LinkedList()
>>> lnk.prepend(2)
>>> lnk.prepend(1)
>>> lnk.prepend(0)
>>> lnk[1]
>>> lnk[3]
>>> lnk[-1]
>>> lnk[-2]
>>> lnk[-3]
>>> lnk[-4]
```

__getitem__

```
def __getitem__(self, index: int) -> object:
    """ see prev slide for docstrings
    """
    if index < 0:
        index += self.size
    if index >= self.size or self.size == 0 or index < 0:
        raise IndexError('index is out of bounds')
    current = self.front
    for steps in range(index):
        current = current.next_
    return current.value
```

__contains__

```
def __contains__(self, value: object) -> bool:
    """
    Return whether LinkedList self
    contains value.

    >>> lnk = LinkedList()
    >>> lnk.prepend(0)
    >>> lnk.prepend(1)
    >>> lnk.prepend(2)
    >>> lnk.__contains__(1)
    True
    >>> lnk.__contains__(3)
    False
    """
```

```
>>> lnk = LinkedList()
>>> lnk.prepend(0)
>>> lnk.prepend(1)
>>> lnk.prepend(2)
>>> lnk.__contains__(1)
True
>>> lnk.__contains__(3)
False
>>> 2 in lnk
True
>>> 3 in lnk
False
.....
```

__contains__

```
def __contains__(self, value: object) -> bool:
    """
    Return whether LinkedList self contains value.
    """
    current = self.front
    while current is not None:
        if current.value == value:
            return True
        current = current.next_
    return False
```

Where Can I find the code presented in class

- You can find the full code in the course website under section **MWF2 (L0301) and MWF3 (L0401)**
- with the following file names:
  - Linked_list_Monday.py
  - (Note: the code has **no docstrings** and **might not be efficient** and it can be written in much better way. However, it is made this way with repetition of some lines to **keep you focused** on the concepts of **linked lists**)
- Download them Try different things with them and practice
  - Do not be afraid of doing mistakes