CSC111 Lecture 3: Mutating Linked Lists,

Hisbaan Noorani

January 18, 2021

Contents

1 Exercise 1: Index-based insertion

Our goal for this exercise is to extend our `LinkedList` class by implementing one of the standard mutating List ADT methods: inserting into a list by index. Here's the docstring of such a method:

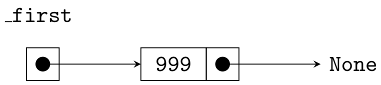
```python
class LinkedList:
    def insert(self, i: int, item: Any) -> None:
        """Insert the given item at index i in this list.

        Raise IndexError if i > len(self).
        Note that adding to the end of the list (i == len(self)) is okay.

        Preconditions:
            - i >= 0

        >>> lst = LinkedList([1, 2, 10, 200])
        >>> lst.insert(2, 300)
        >>> lst.to_list()
        [1, 2, 300, 10, 200]
        """
```

Before diving into any code at all, we'll gain some useful intuition by generating some test cases for this method based on two key input properties: the length of the list, and the relationship between `index` and the length of the list.

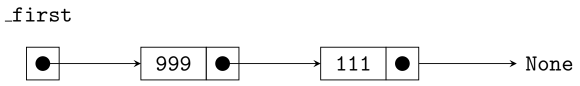
1. In the list below, modify each `self` diagram to show the state of the linked list after `999` is inserted.

   You should draw a new node containing `999`. In each case, you need to determine which arrows to modify to insert the new node into the correct location in the list.

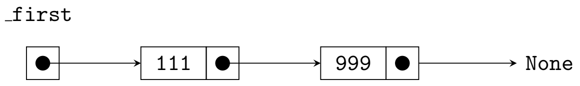
   (a) insert `999` at `0`:



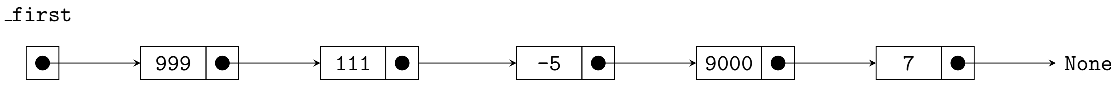
(b) insert 999 at 0:



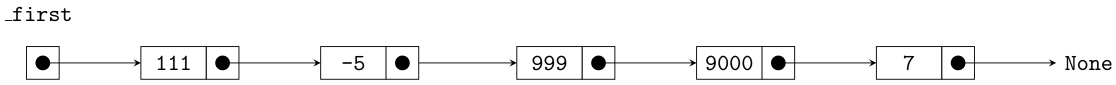
(c) insert 999 at 1:



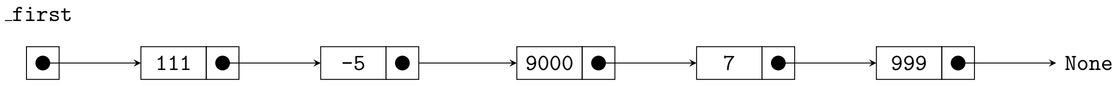
(d) insert 999 at 0:



(e) insert 999 at 2:



(f) insert 999 at 4:



2. Now let's start thinking about some code. Using your diagrams as a guide, answer the following questions:

   (a) For what values of len(self) and/or i would we need to reassign self._first to something new?

   We would only need to reassign self._first if we wanted to insert at index 0.

   (b) What is the relationship between len(self) and i that makes insert behave the same as LinkedList.append from this week's prep?

   For insert to behave like append, we want i = len(self). This will mutate the len(self) - 1 th node

   (c) In the len(self) == 4, i == 2 case, which *existing* node was actually mutated? Write down the index of this node in the list. (Hint: it's *not* the node at index 2!)

   The node at index 1 would be mutated (-5). The next attribute of the node would have to be modified to match the new inserted node.

3. Finally, using these ideas, implement the insert method. Note that you should have two cases: one for when you need to mutate self._first, and one where you don't.

   You'll need to make use of the *linked list traversal pattern*, as well as the extra "parallel loop variable" curr_index that we studied last week with LinkedList.__getitem__.

```python
class LinkedList:
    def insert(self, i: int, item: Any) -> None:
        """Insert the given item at index i in this list.

        Raise IndexError if i > len(self).
        Note that adding to the end of the list (i == len(self)) is okay.

        Preconditions:
            - i >= 0

        >>> lst = LinkedList([1, 2, 10, 200])
        >>> lst.insert(2, 300)
        >>> lst.to_list()
        [1, 2, 300, 10, 200]
        """
        new_node = _Node(item)

        if i == 0:
            new_node.next, self._first = self._first, new_node
        else:
            curr = self._first
            curr_index = 0

            while curr is not None:
                if curr_index == i - 1:
                    new_node.next, curr.next = curr.next, new_node
                    return

            raise IndexError
```