Lab 5: Linked Lists Solution

4) Additional exercises

Generalizing *__getitem__*

The implementation we've provided for *__getitem__* has many shortcomings compared to Python's built-in lists.

Two features that it doesn't currently support are negative indexes and slices (e.g., *my_list[2:5]*).

Your first task here is to investigate the different ways in which Python supports these operations for built-in Python lists; you can do this by experimenting yourself in the Python console, or by doing some reading online.

Then, modify the linked list implementation of *__getitem__* so that it handles both negative indexes and slices.

Note that a slice in Python is actually a class: the expression *my_list[2:5]* is equivalent to *my_list.__getitem__(slice(2, 5))*.

Use *isinstance* to determine whether the input to *__getitem__* is an integer or a slice.

The fully general method signature of *__getitem__* should become:

```
    def __getitem__(self, index: Union[int, slice]) -> Union[Any,
   LinkedList]
```

Note: slicing should always return a new *LinkedList* object.

This means that for a given slice, you'll need to create a *LinkedList* and new *_Nodes* as well, in a similar manner to how you implemented the more powerful initializer at the end of Task 1.

Negative Index:

```
    class LinkedList:
    ...
        def __getitem__(self, index: Union[int , slice]) -> Union[Any ,
    LinkedList]:
            """Return the item at position <index> in this list.

```

```
        Raise IndexError if <index> is >= the length of this list.

        >>> lst = LinkedList([1, 2, 10, 200])
        >>> lst[-1]
        200
        >>> lst[2]
        10
        >>> lst[-10]
        Traceback (most recent call last):
            ...
        IndexError
        >>> str(lst[:2])
        '[1 -> 2]'
        >>> str(lst[10:1:-1])
        '[200 -> 10]'
        """

        if isinstance(index, slice):
            curr = self._first
            curr_index = 0
            start, stop, step = index.indices(len(self))
            # 1. initialize list
            i_list = range(start, stop, step)
            result = []

            for i in i_list:
                # 2. fetch value from linked list
                try:
                    item = self[i]

                    # 3. if it exists, insert to result
                    result.append(item)
                except IndexError:
                    # 4. if it doesn't exist, then continue
                    continue

            return LinkedList(result)

        else:
            curr = self._first
            curr_index = 0
            index = index if index >= 0 else self._length + index

            if index < 0:
                raise IndexError

            while curr is not None and curr_index < index:
                curr = curr.next
                curr_index += 1

            assert curr is None or curr_index == index

            if curr is None:
```

```
                raise IndexError
            else:
                return curr.item
```

Matplotlib Practice

Use *matplotlib* to plot the results of your timing experiments, using the same approach as last week (See matplotlib section in lab 4).