Carleton University
Department of Systems and Computer Engineering
SYSC 2100 — Algorithms and Data Structures — Summer 2023

Lab 7 - Implementing ADT Deque Using Doubly-Linked List

Getting Started

1. Review *Important Considerations When Submitting Files to Brightspace*, which can be found in the course outline.

2. Launch Wing 101 and configure Wing's code reformatting feature. Instructions can be found in Appendix A of document, *SYSC 2100 - Style Guide for Python Code*.

   All code you submit for grading must be formatted. If you decide to disable automatic reformatting, make sure you manually reformat your code (Appendix A.2). At a minimum, we recommend that you reformat your file as you finish each exercise.

3. Download lab7_linkeddeque.py from the *Lab Materials* module in Brightspace.

4. Open lab7_linkeddeque.py in Wing 101. Locate these assignment statements at the top of the file:

   ```
   __author__ = ''
   __student_number__ = ''
   ```

   Replace the empty strings with strings containing your name and student number. (Don't modify the variable names.)

5. Your solutions to the exercises must conform to the coding conventions described in *SYSC 2100 - Style Guide for Python Code*.

6. **Important**: if you decide to write a script in lab7_linkeddeque.py, it **must** be placed in an `if __name__ == '__main__':` block, like this:

   ```
   class LinkedDeque:
       # methods not shown

   if __name__ == '__main__':
       empty_deque = LinkedDeque()
       dq = LinkedDeque([1, 2, 3, 2, 1])
           ...
   ```

   Doing this ensures that the script will only be executed when you run lab7_linkeddeque.py as the "main" module, and won't be executed when the file is imported into another module; for example, the testing/grading program provided to the TAs.

Overview of Class `LinkedDeque`

Recall that ADT Deque (double-ended queue) is a generalization of a FIFO queue and a LIFO

stack. An implementation of ADT Deque should provide efficient append and pop operations at either side of the deque; that is, appends and pops should have complexity O(1).

Python provides an implementation of ADT Deque. For information about Python's `deque` class, see *The Python Standard Library*, Section *`deque` objects*: https://docs.python.org/3/library/collections.html#collections.deque.

Section 5.1.2 of the *The Python Tutorial*, *Using Lists as Queue*s, shows how to use a `deque` as a FIFO queue. The URL is https://docs.python.org/3/tutorial/index.html.

Class `LinkedDeque` in lab7_linkeddeque.py contains an incomplete implementation of ADT Deque that uses a doubly-linked list as its data structure. The design of this data structure is presented in document *SYSC_2100_Lab_7_Doubly_Linked_Lists.pdf.* You should refer to this document while working on this lab.

Class `LinkedDeque` provides methods `__init__`, `__str__`, `__repr__`, `__len__`, and `__iter__`. **Do not modify these methods.** The class also has "stub" implementations of methods `_insert_before`, `_remove`, `append`, `appendleft`, `pop` and `popleft`. If you call any of these methods on a `LinkedDeque` object, Python will raise a `NotImplementedError` exception. You'll complete the implementation of these methods during this lab.

Unlike Python's `deque` class, `LinkedDeque` doesn't support any list operations.

Exercise 1: A link to a Python Tutor script that will help you visualize `LinkedDeque` objects is provided in the Lab 7 module in Brightspace. Open the script by clicking the link. Scroll down until you reach the code after the definition of `LinkedDeque`.

Trace the execution of `__init__` when `dq = LinkedDeque()` is executed. After the `LinkedDeque` object is initialized, what does instance variable `_dummy` refer to? An empty `LinkedDeque` has one node (the dummy node), so why is instance variable `_num_items` bound to 0 instead of 1?

Exercise 2: *SYSC_2100_Lab_7_Doubly_Linked_Lists.pdf* presents the pseudocode algorithm for the `insert_before` operation. Open lab7_linkeddeque.py in Wing 101. Read the docstring for "private" method `_insert_before`, delete the `raise` statement, then implement the method's body by translating the pseudocode into Python.

Exercise 3: Read the docstring for `append`, delete the `raise` statement, then implement the method. The computational complexity of this method must be *O*(1). Hint: the method body requires exactly one line of code, which will be a call to `self._insert_before`. Method `append` adds a new node at the tail of the deque's linked list, which means it inserts the node immediately before which node?

Use the Python shell to run a few tests on `append`. When your method works, you'll be able to create a new deque by passing an iterable to the constructor; for example,

```
dq = LinkedDeque([1, 4, 3, 6])
```

will create a `LinkedDeque` in which 1 will be in the deque's left side; that is, stored in the head node. 6 will be in the deque's right side; that is, stored in the tail node.

Copy `_insert_before` and `append` into the Python Tutor script you used in Exercise 1. Replace the `pass` statements with the method bodies. Trace the calls to `append`. Make sure you understand why your `append` method works with (1) an empty deque (a linked list that has only a dummy node) and (2) a deque that has one or more items.

Exercise 4: Read the docstring for `appendleft`, delete the `raise` statement, then implement the method. The computational complexity of this method must be *O*(1). Hint: the method body requires exactly one line of code, which will be a call to `self._insert_before`. Method `appendleft` adds a new node at the head of the deque's linked list, which means it inserts the node immediately before which node?

Use the Python shell to run a few tests on `appendleft`.

Copy `appendleft` into the Python Tutor script. Trace the calls to `appendleft`. Make sure you understand why your `appendleft` method works with (1) an empty deque (a linked list that has only a dummy node) and (2) a deque that has one or more items.

Exercise 5: *SYSC_2100_Lab_7_Doubly_Linked_Lists.pdf* presents the pseudocode algorithm for the `remove` operation. Read the docstring for "private" method `_remove`, delete the `raise` statement, then implement the method's body by translating the pseudocode into Python.

Exercise 6: Read the docstring for `pop`, delete the `raise` statement, then implement the method. Your `pop` method must call `_remove`. The computational complexity of this method must be *O*(1).

When the deque is empty, the method should raise an `IndexError` exception containing the message, `"LinkedDeque: pop from an empty deque"`.

Hint: the method requires no more than 5 lines of code.

Use the Python shell to run a few tests on `pop`.

Copy `pop` into the Python Tutor script. Trace the calls to `pop`. Make sure you understand why your `pop` method works with (1) a deque that has exactly one item and (2) a deque that has two or more items.

Exercise 7: Read the docstring for `popleft`, delete the `raise` statement, then implement the method. Your `popleft` method must call `_remove`. The computational complexity of this method must be *O*(1).

When the deque is empty, the method should raise an `IndexError` exception containing the message, `"LinkedDeque: pop from an empty deque"`.

Hint: the method requires no more than 5 lines of code.

Use the Python shell to run a few tests on `popleft`.

Copy `popleft` into the Python Tutor script. Trace the calls to `popleft`. Make sure you understand why your `popleft` method works with (1) a deque that has exactly one item and (2) a deque that has two or more items.

Wrap-up

- Before submitting lab7_linkeddeque.py, review your code. Does it conform to the coding conventions, as specified in the *Getting Started* section? Has it been formatted? Did you edit the `__author__` and `__student_number__` variables?
- Submit lab7_linkeddeque.py to Brightspace. Make sure you submit the file that contains your solutions, not the unmodified file you downloaded from Brightspace! You are permitted to make changes to your solutions and resubmit the files as many times as you want, up to the submission due date. Only the most recent submission will be saved by Brightspace.
- Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the submission due date.