

CS 231B Fall '24 Assignment 4: LinkedLists with a *Tail*

This assignment will primarily focus on the `LinkedList` data structure.

Last week, we had you implement the following four methods on an `ArrayList`:

- `void addFirst(T item)`: makes `item` the first entry in the list
- `void addLast(T item)`: makes `item` the last entry in the list
- `T removeFirst()`: removes and returns the first entry in the list
- `T removeLast()`: removes and returns the final entry in the list

Your first task this week is to do the same on a `LinkedList`. Specifically, [download this starter file](#). The `Node` class should look very familiar from lecture notes. The `head` and `size` fields are also the same. However, we've added one more field for you to maintain: the `tail`. The `tail` is meant to track the last `Node` in the list (analogous to `head` and the first node in the list). You'll notice that the signatures of the four methods above have been included in the file: your task for this Assignment is to fill in the code for each to appropriately implement the methods *without using any loops* (similar to Assignment 3) while *maintaining all three fields correctly*.

It is reasonably natural to implement `addFirst`, `addLast`, and `removeFirst` without loops, in a way that the number of operations they do does not change as the list gets longer. Or, to use some of the vocabulary introduced in class on Friday Sept. 20, they are all *constant time* methods.

However, `removeLast` would need a loop (do you understand why? Feel free to ask your Prof. about it, if you'd like!), and the number of operations it would do would vary proportionally with the size of the list—it would do more operations on a list with more elements in it. Let's fix that!

To make `removeLast` so it does not use a loop, and its number of operations does not depend on the number of elements in the list, you will add one more instance field to the `Node` class: a `Node<T> prev` field (`prev` is short for *previous*). This is analogous to the `next` field: just as `next` refers to the `Node` immediately following in a linked list (or `null` if none exists), `prev` refers to the `Node` immediately preceding in the list (or `null` if none exists). With the `prev` field for each node, a “walker” on the list could go either forward *or backward* by one node on the list in one step.

To make this work, first, add getter/setter methods for the `prev` field in the `Node` class. Then, make sure your code in `addFirst`, `addLast`, `removeFirst` and `removeLast` is written so that `prev` is correctly set for all the nodes in the list, and make sure `removeLast` works

correctly without any loops, so it is also a constant time method (the same number of operations no matter how many elements are in the list).

Restrictions:

- Do not add any new fields to either the `Node` or `LinkedList` class that are not described here.
- The uploaded file should have these four above methods with no loops, as constant time methods.

To submit: Download the attached file above, complete the tasks as described above, and upload the **.java** file to your HomeworkAssignments 4 folder in your Google Drive.