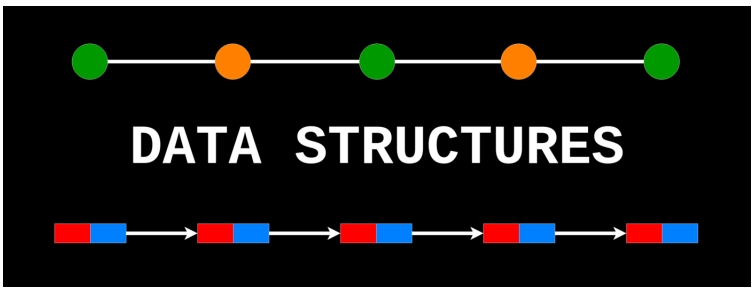
Computer Science 1 — CSCI-141
Linked Lists — Homework 11

10/25/2020



1 Overview

This week in lecture we learned how to take the singly linked node we developed for stacks and queues and saw how it can also be used to represent a *linked list*. A linked list has the same properties of a *stack* and *queue*, but allows insertion and removal from anywhere in the collection. In this homework you will see how we can do even more operations with linked list, including *concatenating* two lists together, and *reversing* a list.

Recall in lecture we developed two different versions of a linked list that supported the same functionality but were implemented in very different ways.

- An *immutable* list implemented *recursively* using `FrozenNode`'s.
- A *mutable* list implemented *iteratively* using `MutableNode`'s.

In this homework you will be implementing *four* additional linked list functions:

1. Get the size of an immutable list.
2. Concatenate two immutable lists together.
3. Reverse a mutable list.
4. Remove an element by index in a mutable list.

2 Starter Code

1. Download and unzip the starter code for this assignment.

   `https://www.cs.rit.edu/~csci141/Homeworks/11/code.zip`

2. Create a new PyCharm project and copy/move the starter code into it. There are three source modules **that you should not modify**.
   - `node_types.py`: Definitions for `FrozenNode` and `MutableNode`.
   - `linked_list_type.py`: Definition for the `LinkedList` structure that encapsulates a `MutableNode` as the head of the linked list.
   - `test_linked_lists.py`: A complete unit test module for the functions you will implement.

3. Create the source file, `linked_lists.py`, in the same location as the starter code. This is where you will implement the four linked list functions for this assignment.

3 Getting Started: Stubbing

To begin with, you should *stub* out the four functions so that the unit test program compiles and runs, but does not pass any of the tests. Here is a brief description of each function, listed in the order you will eventually fully implement them in.

1. `size_of`: A function that returns the size of an immutable list. It takes as an argument the `head`, a `FrozenNode`, and returns the number of nodes in the linked sequence.

2. `extend`: A function that concatenates two immutable lists into a new one. It takes as arguments two `head`'s, each a `FrozenNode`, and returns the head of a new immutable list, another `FrozenNode`, that represents the concatenation of the first onto the second list.

3. `reverse`: A function that takes a mutable list and reverses the elements. It takes as an argument the original mutable list, a `LinkedList`. The function returns `None` because it modifies the list passed in.

4. `remove_index`: A function that removes an element from a mutable list at a specific index. It takes two arguments, the mutable list, a `LinkedList`, and an integer `index`. The function returns `None` because the intent is the list that is passed in is modified.

To begin with:

- Define each of these functions in `linked_lists.py`.
- Each function should simply return `None`.
- Once fully stubbed, run the unit test module, `test_linked_lists.py`. The program should run to completion without error, and all of the tests should fail.
- If there are any runtime errors, make sure you named the functions correctly and with the correct number of arguments.

Once everything compiles and runs, add the following import statements to the top of your program:

```
from node_types import FrozenNode, MutableNode
from linked_list_type import LinkedList
```

This will allow PyCharm to assist you when working with the various user defined types in this assignment.

4 Implementation

For all tasks, the general requirements are:

- You may not use any built in Python data structures, e.g. a `list` or `dict`.
- All functions that work with immutable lists are guaranteed to have nodes that are of type `FrozenNode`.
- All functions that work with mutable lists are guaranteed to have types `LinkedList` and `MutableNode`.
- The functions that work with immutable lists, `size_of` and `extend`, must be implemented recursively. No loops are allowed.
- The functions that work with mutable lists, `reverse` and `remove_index`, must be implemented using a loop. Recursion is not allowed.
- The use of global variables is prohibited.

4.1 Task 1: Size of an immutable list

Implement the `size_of` function for determining the number of nodes in a linked sequence of `FrozenNode`'s starting with the head.

4.1.1 Hints

- While you cannot alter the main function signature, you are allowed to implement a separate recursive helper function with an accumulator.

4.1.2 Output

Run the unit test module, `test_linked_lists.py`, and make sure all the tests for `size_of` pass.

```
testing size_of...
size_of([]) passed
size_of([A]) passed
size_of([A,B]) passed
size_of([A,B,C]) passed
size_of([A,B,C,D,E]) passed
```

4.2 Task 2: Extend two immutable lists

Implement the `extend` function for concatenating together two immutable lists into a new immutable list that is the result.

4.2.1 Hints

- The concept here is that you build an entirely new immutable list of `FrozenNode`'s by recursing over the nodes of the first list.
- When you reach the end of the first list, you connect the last element of the first list to the head of the second list with a new `FrozenNode`.
- Look at the lecture `append` function for immutable lists. It is very similar.

4.2.2 Output

Run the unit test module, `test_linked_lists.py`, and make sure all the tests for `extend` pass.

```
testing extend...
extend([], []) passed
extend([], [A]) passed
extend([A], []) passed
extend([A], [B,C]) passed
extend([A,B,C], [D,E,F,G]) passed
```

4.3 Task 3: Reverse a mutable list

Implement the `reverse` function for reversing the elements in a mutable list.

4.3.1 Hints

- Your loop should work in a way such that the first node of the original list is the last node of the reversed list, and so on.
- Consider creating new `MutableNode`'s as you traverse the original list to create a reversed sequence. That sequence will eventually become the new head of the original list.
- If you get stuck, consider how you would reverse a string, `"ABC"` to become `"CBA"`, starting with a resulting string that is initially empty and using string concatenation.

4.3.2 Output

Run the unit test module, `test_linked_lists.py`, and make sure all the tests for `reverse` pass.

```
testing reverse...
reverse([]) passed
reverse([A]) passed
reverse([A,B]) passed
reverse([A,B,C]) passed
reverse([A,B,C,D,E]) passed
```

4.4 Task 4: Remove by index from a mutable list

Implement the `remove_index` function for removing the element at a zero based index from a mutable list.

4.4.1 Hints

- To begin with, assume the index is in the *range* of the list so that it represents a valid element to remove. Your function should pass all of the first set of tests below.
- Next, handle invalid indices, e.g. negative numbers, or numbers that are out of the range of the size of the list. In order to do this, you should *raise* an `IndexError` when the problem is detected. To do this in code:

  ```
  raise IndexError('Index out of bounds!')
  ```

  Doing this correctly should cause your function to pass the second set of tests below.

4.4.2 Output

Run the unit test module, `test_linked_lists.py`, and make sure all the tests for `remove_index` pass.

```
testing remove_index...
remove_index([A,B,C,D,E], 0) passed
remove_index([B,C,D,E], 1) passed
remove_index([B,D,E], 2) passed
remove_index([B,D], 0) passed
remove_index([D], 0) passed

testing remove_index_bad...
remove_index([A,B,C,D,E], -1) passed
remove_index([A,B,C,D,E], 5) passed
```

5 Grading

- Task 1 - `size_of`: 25%
- Task 2 - `extend`: 25%
- Task 3 - `reverse`: 25%
- Task 4 - `remove_index`: 25%
- Failure to provide function and module docstrings, or to follow the submission instructions: up to a 10% penalty

6 Submission

Follow these instructions exactly to submit your work.

1. Navigate in an explorer to your PyCharm project and make a zip file containing only your `linked_lists.py` module.
2. Make sure your zip file is named `hw11.zip`.
3. Upload your zip file to the Assignment dropbox by the due date.