CSC148H Week 5

Ilir Dema, Michael Miljanovic

Summer 2021

List Implementations

There are two major list implementations

- An Array-based list stores references to elements in a contiguous block of memory.
  - This is how Python lists work
- A linked list can store elements anywhere, but each element must store a reference to the next element in the list.

Implementing Queues

- You can implement a queue using a Python list (this week's lab)
- That kind of implementation requires you to enqueue at one end and dequeue from the other
- Only one of these can be fast on a Python list (left side is slow; right side is fast)
- Let's look at some timing ...

Linked Lists

- We'll implement our own list ADT using a **linked list**
- We will use two classes for our implementation
  - `_Node`: allows us to string together elements of a list
  - `LinkedList`: stores a reference to the first `Node` in the list... and we can even store other attributes to make some list operations faster!

Nodes

Our `_Node` class allows us to make chains of nodes.

```
>>> a = _Node(4)
>>> a.item
4
>>> b = _Node(5)
>>> c = _Node(6)
>>> a.next = b
>>> b.next = c
>>> a.next.item
5
```

Our goals this week

This week we want to

1. Work with linked lists by implementing some of the operations of Python's built-in list.
2. Analyze the running time of our linked list methods and compare them to the built-in list.

Worksheet 1

Worksheet 1, linked list traversals

Takeaways

- Code templates are useful.
- Code templates aren't everything.
- Writing a stopping condition is often easier to understand than writing a loop condition.

Linked-List Insertion

- We might want to implement all sorts of insert variations depending on what operations we want the linked list to support, e.g.
  - Prepend
  - Append
  - Insert at a given index

Prepend

Inserting at the front of a linked list.

- Change what `_first` refers to
- No need to iterate through the linked list

Append

Inserting at the back of a linked list.

- You worked on this in this week's prep
- Iterate through the linked list to find the last node

Worksheet 2

Worksheet 2, inserting into linked list

The Problem of Previous

Strategy 1: iterate to the node before the desired position.

```
i = 0
curr = self._first
while not (curr is None or i == index - 1):
    curr = curr.next
    i += 1
```

The Problem of Previous...

Strategy 2: explicitly track the previous node.

```
i = 0
prev = None
curr = self._first
while not (curr is None or i == index):
    prev, curr = curr, curr.next
    i += 1
```

Worksheet 3

Worksheet 3, running time of linked list operations

Another Linked List Design

Let's store more info in our linked list.

- `_first`: refers to first node (like we've been doing)
- `_last`: refers to the last node
- `_size`: refers to the length of the linked list

Questions:

- Which method implementations change?
- What are the performance implications?

Another Linked List Design...

Suppose that this new linked list implementation only supports insert and delete operations

- How does it benefit us to store `_last` and `_size`?
- At which end of a linked list would it be best to insert an element? Which index would it be easiest to remove?
- How does this implementation fare against the previous one?