More linked lists
Announcements

• Lab 12 is optional, but doing it will be a good way to practice linked lists for the final exam
  - Labs will meet this week, but are also optional

• The final exam is on Saturday, May 6
  - 7-10pm in SCI 101
Today’s plan

• Review linked lists

• More linked list methods:
  - removeHead()
  - __str__
  - getAtIndex(index)
Review

• A **linked list** is a **data structure** that’s an alternative to a Python list.

• It has a similar **interface**, but a different **implementation**.

• The run times of the operations in this shared interface will vary based on the implementation.
Linked list structure

- Linked list of strings “A”, “B”, “C”
Review

• Each item in a linked list corresponds to a node. We represent this with a Node class that has two instance variables:

  • `self.data` (sometimes called `self.item`)
  • `self.next` (defaults to `None`)
Review

• A linked list is a chain of nodes that connect to each other through their `self.next` fields. We represent linked lists with the `LinkedList` class. It has three instance variables:

  • `self.head`
  
  • `self.tail`
  
  • `self.size`
Removing from the front

self.head

"X" → "A" → "B" → "C" None

self.size

4
Removing from the front

```
self.head

"X" --> "A" --> "B" --> "C" None

self.size → 4
```
Removing from the front

self.head

"A"  "B"  "C" None

self.size

3

self.tail
removeHead() method

- Watch out for the case where self.size == 1
Back to linked list motivation

• Now we can insert at the front or back and remove from the front, all in constant time.

• There are applications where we don’t need to access the middle of a list, and these constant-time operations are enough:
  
  - Orders to be filled by a restaurant, customer service requests (or any other queue)
  
  - The function stack!

• For such applications, linked lists will outperform Python lists.
Traversing a linked list

• Use a while loop that continues until the current Node is equal to None

• (Can also use a for loop since we know the number of nodes)

• Traversal will be used in a number of methods:
  - __str__, searching, indexing, etc.
Traversing a linked list

self.head

"A"

"B"

"C" None

current

self.size

3
Traversing a linked list

- `self.head` points to "A"
- `current` points to "B"
- `self.tail` points to "C" with `None`
- `self.size` is 3
Traversing a linked list

```
self.head

"A"   "B"   "C"   None
```

```
self.size

3
```

```python
self.tail

current
```
Traversing a linked list

- self.head
- self.size: 3
- self.tail
- current

Nodes: "A", "B", "C"