Linked List Practice

May 3, 2020

1 Linked List Practice

Implement a linked list class. You have to define a few functions that perform the desirbale action. Your `LinkedList` class should be able to:

- Append data to the tail of the list and prepend to the head
- Search the linked list for a value and return the node
- Remove a node
- Pop, which means to return the first node's value and delete the node from the list
- Insert data at some position in the list
- Return the size (length) of the linked list

```
In [3]: class Node:
            def __init__(self, value):
                self.value = value
                self.next = None
```

```
In [4]: class LinkedList:
            def __init__(self):
                self.head = None

            def to_list(self):
                out = []
                node = self.head
                while node:
                    out.append(node.value)
                    node = node.next
                return out
```

Task 1. Write definition of `prepend()` function and test its functionality

```
In [5]: # Define a function outside of the class
        def prepend(self, value):
                """ Prepend a value to the beginning of the list. """
                if self.head is None:
                    self.head = Node(value)
                    return
        # Move to the head (the first node)
        newHeadNode = Node(value)
        newHeadNode.next = self.head
        self.head = newHeadNode


# This is the way to add a function to a class after it has been defined
LinkedList.prepend = prepend
```

Here is an example of a Makefile you could create for this exercise:

```
cmd1:
ăăăăăăăă@echo "$@"

cmd2:
ăăăăăăăă@echo "$@"

all: cmd1 cmd2
```

Note that after `cmd1` and `cmd2`, and before `@echo`, should be a tab. The `@` at the start of these lines prevents `make` from automatically printing the lines, while `"$@"` is the variable for a string containing the target name, in this case either `cmd1` or `cmd2`. To double-check that `make` is actually showing the command name from within the command itself, try to `echo` something else from within one of them, such as `Hello World!`, and check the results.

Show Solution

"'{toggle} Click the button o your right to reveal the solution!

def prepend(self, value): """ Prepend a node to the beginning of the list """

```
if self.head is None:
    self.head = Node(value)
    return

new_head = Node(value)
new_head.next = self.head
self.head = new_head
```

"'

```
In [6]: # Test prepend
        linked_list = LinkedList()
        linked_list.prepend(1)
        assert linked_list.to_list() == [1], f"list contents: {linked_list.to_list()}"
```

Task 2. Write definition of `append()` function and test its functionality

```
In [12]:  def append(self, value):
                 if self.head is None:
                     self.head = Node(value)
                     return
```

```
                # Move to the tail (the last node)
                node = self.head
                counter = 0
                while node.next:
                    counter += 1
                    node = node.next
                node.next = Node(value)
                return


        LinkedList.append = append
```

```
In [13]: # Test append - 1
         linked_list.append(3)
         linked_list.prepend(2)
         assert linked_list.to_list() == [2, 1, 3], f"list contents: {linked_list.to_list()}"
```

```
        ---------------------------------------------------------------------------

        AssertionError                            Traceback (most recent call last)

        <ipython-input-13-6b5e5c170f4e> in <module>()
          2 linked_list.append(3)
          3 linked_list.prepend(2)
    ----> 4 assert linked_list.to_list() == [2, 1, 3], f"list contents: {linked_list.to_list()}"


        AssertionError: list contents: [2, 2, 1, 3, 3, 3]
```

```
In [14]: # Test append - 2
         linked_list = LinkedList()
         linked_list.append(1)
         assert linked_list.to_list() == [1], f"list contents: {linked_list.to_list()}"
         linked_list.append(3)
         assert linked_list.to_list() == [1, 3], f"list contents: {linked_list.to_list()}"
```

Task 3. Write definition of `search()` function and test its functionality

```
In [15]: def search(self, value):
                """ Search the linked list for a node with the requested value and return the n
                if self.head is None:
                    return None

                node = self.head
```

```
                 while node:
                     if node.value == value:
                         return node
                     node = node.next

                 return node

         LinkedList.search = search
```

```
In [16]: # Test search
         linked_list.prepend(2)
         linked_list.prepend(1)
         linked_list.append(4)
         linked_list.append(3)
         assert linked_list.search(1).value == 1, f"list contents: {linked_list.to_list()}"
         assert linked_list.search(4).value == 4, f"list contents: {linked_list.to_list()}"
```

Task 4. Write definition of `remove()` function and test its functionality

```
In [18]: def remove(self, value):
             """ Remove first occurrence of value. """
             # TODO: Write function to remove here
             if self.head == None:
                 return
             elif self.head.value == value:
                 node = self.head
                 self.head = node.next
                 node.next = self.head
                 return
             else:
                 node = self.head
                 while node.next:
                     if node.next.value == value:
                         node.next = node.next.next
                         return
                     node = node.next

         LinkedList.remove = remove
```

```
In [19]: # Test remove
         linked_list.remove(1)
         assert linked_list.to_list() == [2, 1, 3, 4, 3], f"list contents: {linked_list.to_list(
         linked_list.remove(3)
         assert linked_list.to_list() == [2, 1, 4, 3], f"list contents: {linked_list.to_list()}"
         linked_list.remove(3)
         assert linked_list.to_list() == [2, 1, 4], f"list contents: {linked_list.to_list()}"
```

Task 5. Write definition of `pop()` function and test its functionality

```
In [20]: def pop(self):
             """ Return the first node's value and remove it from the list. """
             # TODO: Write function to pop here
             if self.head == None:
                 return None
             node = self.head
             item = node.value
             self.head = node.next
             node.next = self.head
             return item

         LinkedList.pop = pop
```

```
In [21]: # Test pop
         value = linked_list.pop()
         assert value == 2, f"list contents: {linked_list.to_list()}"
         assert linked_list.head.value == 1, f"list contents: {linked_list.to_list()}"
```

Task 6. Write definition of `insert()` function and test its functionality

```
In [24]: def insert(self, value, pos):
             """ Insert value at pos position in the list. If pos is larger than the
             length of the list, append to the end of the list. """

             # TODO: Write function to insert here
             if self.head == None or pos == 0:
                 self.prepend(value)
                 return
             elif pos > self.size():
                 self.append(value)
                 return
             counter = 0
             node = self.head
             while (counter + 1) < pos:
                 node = node.next
                 counter += 1
             new_node = Node(value)
             new_node.next = node.next
             node.next = new_node
             return

         LinkedList.insert = insert
```

```
In [25]: # Test insert
         linked_list.insert(5, 0)
         assert linked_list.to_list() == [5, 1, 4], f"list contents: {linked_list.to_list()}"
         linked_list.insert(2, 1)
         assert linked_list.to_list() == [5, 2, 1, 4], f"list contents: {linked_list.to_list()}"
```

```
linked_list.insert(3, 6)
assert linked_list.to_list() == [5, 2, 1, 4, 3], f"list contents: {linked_list.to_list(
```

```
    ---------------------------------------------------------------------------

    AssertionError                            Traceback (most recent call last)

    <ipython-input-25-f24118b35b9b> in <module>()
      1 # Test insert
      2 linked_list.insert(5, 0)
----> 3 assert linked_list.to_list() == [5, 1, 4], f"list contents: {linked_list.to_list()}"
      4 linked_list.insert(2, 1)
      5 assert linked_list.to_list() == [5, 2, 1, 4], f"list contents: {linked_list.to_list(


    AssertionError: list contents: [5, 5, 1, 4]
```

Task 7. Write definition of `size()` function and test its functionality

```
In [26]: def size(self):
             """ Return the size or length of the linked list. """
             # TODO: Write function to get size here
             if self.head is None:
                     return 0
             node = self.head
             counter = 0
             while node.next:
                 counter += 1
                 node = node.next
             return counter+1

         LinkedList.size = size
```

```
In [27]: # Test size function
         assert linked_list.size() == 5, f"list contents: {linked_list.to_list()}"
```

```
    ---------------------------------------------------------------------------

    AssertionError                            Traceback (most recent call last)

    <ipython-input-27-1ed45e79b803> in <module>()
      1 # Test size function
----> 2 assert linked_list.size() == 5, f"list contents: {linked_list.to_list()}"
```

```
AssertionError: list contents: [5, 5, 1, 4]
```

Hide Solution

```
In [ ]: # Solution

        #------------------------------------------------------#
        def prepend(self, value):
            """ Prepend a node to the beginning of the list """

            if self.head is None:
                self.head = Node(value)
                return

            new_head = Node(value)
            new_head.next = self.head
            self.head = new_head
        #------------------------------------------------------#
        def append(self, value):
            """ Append a node to the end of the list """
            # Here I'm not keeping track of the tail. It's possible to store the tail
            # as well as the head, which makes appending like this an O(1) operation.
            # Otherwise, it's an O(N) operation as you have to iterate through the
            # entire list to add a new tail.

            if self.head is None:
                self.head = Node(value)
                return

            node = self.head
            while node.next:
                node = node.next

            node.next = Node(value)
        #------------------------------------------------------#
        def search(self, value):
            """ Search the linked list for a node with the requested value and return the node.
            if self.head is None:
                return None

            node = self.head
            while node:
                if node.value == value:
                    return node
                node = node.next

            raise ValueError("Value not found in the list.")
```

```
#----------------------------------------------------#
def remove(self, value):
    """ Delete the first node with the desired data. """
    if self.head is None:
        return

    if self.head.value == value:
        self.head = self.head.next
        return

    node = self.head
    while node.next:
        if node.next.value == value:
            node.next = node.next.next
            return
        node = node.next

    raise ValueError("Value not found in the list.")

#----------------------------------------------------#
def pop(self):
    """ Return the first node's value and remove it from the list. """
    if self.head is None:
        return None

    node = self.head
    self.head = self.head.next

    return node.value
#----------------------------------------------------#
def insert(self, value, pos):
    """ Insert value at pos position in the list. If pos is larger than the
        length of the list, append to the end of the list. """
    # If the list is empty
    if self.head is None:
        self.head = Node(value)
        return

    if pos == 0:
        self.prepend(value)
        return

    index = 0
    node = self.head
    while node.next and index <= pos:
        if (pos - 1) == index:
            new_node = Node(value)
            new_node.next = node.next
            node.next = new_node
            return

        index += 1
        node = node.next
    else:
        self.append(value)

#----------------------------------------------------#

def size(self):
    """ Return the size or length of the linked list. """
    size = 0
    node = self.head
    while node:
        size += 1
        node = node.next

    return size
#----------------------------------------------------#

def to_list(self):
    out = []
    node = self.head
    while node:
        out.append(node.value)
        node = node.next
    return out
```