Implementing and traversing a linked list

May 3, 2020

1 Implementing and traversing a linked list

In this notebook we'll get some practice implementing a basic linked list—something like this:

Note - This notebook contains a few audio walkthroughs of the code cells. If you face difficulty in listening to the audio, try reconnecting your audio headsets, and use either Chrome or Firefox browser.

1.1 Key characteristics

First, let's review the overall abstract concepts for this data structure. To get started, click the walkthrough button below.

Walkthrough

1.2 Exercise 1 - Implementing a simple linked list

Now that we've talked about the abstract characteristics that we want our linked list to have, let's look at how we might implement one in Python.

Walkthrough

```
In [ ]:
```

Step 1. Once you've seen the walkthrough, give it a try for yourself:

- Create a `Node` class with `value` and `next` attributes
- Use the class to create the `head` node with the value `2`
- Create and link a second node containing the value `1`
- Try printing the values (`1` and `2`) on the nodes you created (to make sure that you can access them!)

```
In [ ]:
```

Hide Solution

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

```
head = Node(2)
head.next = Node(1)

print(head.value)
print(head.next.value)
```

At this point, our linked list looks like this:

Our goal is to extend the list until it looks like this:

To do this, we need to create three more nodes, and we need to attach each one to the `next` attribute of the node that comes before it. Notice that we don't have a direct reference to any of the nodes other than the `head` node!

See if you can write the code to finish creating the above list:

Step 2. Add three more nodes to the list, with the values `4`, `3`, and `5`

```
In [ ]:
```

Show Solution

Let's print the values of all the nodes to check if it worked. If you successfully created (and linked) all the nodes, the following should print out `2`, `1`, `4`, `3`, `5`:

```
In [ ]: head.next.next = Node(4)
        head.next.next.next = Node(3)
        head.next.next.next.next = Node(5)
```

```
In [ ]: print(head.value)
        print(head.next.value)
        print(head.next.next.value)
        print(head.next.next.next.value)
        print(head.next.next.next.next.value)
```

1.3 Exercise 2 - Traversing the list

We successfully created a simple linked list. But printing all the values like we did above was pretty tedious. What if we had a list with 1,000 nodes?

Let's see how we might traverse the list and print all the values, no matter how long it might be.

Walkthrough

```
In [ ]:
```

Once you've seen the walkthrough, give it a try for yourself. #### Step 3. Write a function that loops through the nodes of the list and prints all of the values

```
In [ ]:
```

Hide Solution

```
In [ ]: def print_linked_list(head):
            current_node = head

            while current_node is not None:
                print(current_node.value)
                current_node = current_node.next

        print_linked_list(head)
```

1.4 Creating a linked list using iteration

Previously, we created a linked list using a very manual and tedious method. We called `next` multiple times on our `head` node.

Now that we know about iterating over or traversing the linked list, is there a way we can use that to create a linked list?

We've provided our solution below—but it might be a good exercise to see what you can come up with first. Here's the goal:

Step 4. See if you can write the code for the `create_linked_list` function below

- The function should take a Python list of values as input and return the `head` of a linked list that has those values
- There's some test code, and also a solution, below—give it a try for yourself first, but don't hesitate to look over the solution if you get stuck

```
In [ ]: def create_linked_list(input_list):
            """
            Function to create a linked list
            @param input_list: a list of integers
            @return: head node of the linked list
            """
            head = None
            return head
```

Test your function by running this cell:

```
In [ ]: ### Test Code
        def test_function(input_list, head):
            try:
                if len(input_list) == 0:
                    if head is not None:
                        print("Fail")
                        return
                for value in input_list:
                    if head.value != value:
                        print("Fail")
                        return
                    else:
                        head = head.next
            print("Pass")
        except Exception as e:
            print("Fail: "  + e)



    input_list = [1, 2, 3, 4, 5, 6]
    head = create_linked_list(input_list)
    test_function(input_list, head)

    input_list = [1]
    head = create_linked_list(input_list)
    test_function(input_list, head)

    input_list = []
    head = create_linked_list(input_list)
    test_function(input_list, head)
```

Below is one possible solution. Walk through the code and make sure you understand what each part does. Compare it to your own solution—did your code work similarly or did you take a different approach?

Hide Solution

```
In [ ]: def create_linked_list(input_list):
            head = None
            for value in input_list:
                if head is None:
                    head = Node(value)
                else:
                # Move to the tail (the last node)
                    current_node = head
                    while current_node.next:
                        current_node = current_node.next

                    current_node.next = Node(value)
            return head
```

1.4.1 A more efficient solution

The above solution works, but it has some shortcomings. In this next walkthrough, we'll demonstrate a different approach and see how its efficiency compares to the solution above.

Walkthrough

```
In [ ]:
```

Step 5. Once you've seen the walkthrough, see if you can implement the more efficient version for yourself

```
In [22]: def create_linked_list_better(input_list):
             head = None
             # TODO: Implement the more efficient version that keeps track of the tail
             return head
```

```
In [15]: ### Test Code
         def test_function(input_list, head):
             try:
                 if len(input_list) == 0:
                     if head is not None:
                         print("Fail")
                         return
                 for value in input_list:
                     if head.value != value:
                         print("Fail")
                         return
                     else:
                         head = head.next
                 print("Pass")
             except Exception as e:
                 print("Fail: "  + e)



         input_list = [1, 2, 3, 4, 5, 6]
         head = create_linked_list_better(input_list)
         test_function(input_list, head)

         input_list = [1]
         head = create_linked_list_better(input_list)
         test_function(input_list, head)

         input_list = []
         head = create_linked_list_better(input_list)
         test_function(input_list, head)
```

```
Pass
Pass
Pass
```

Hide Solution

```
In [ ]: def create_linked_list_better(input_list):

            head = None
            tail = None

            for value in input_list:
```

```python
        if head is None:
            head = Node(value)
            tail = head # when we only have 1 node, head and tail refer to the same node
        else:
            tail.next = Node(value) # attach the new node to the `next` of tail
            tail = tail.next # update the tail

    return head
```