Daily Coding Problem Blog

Daily Coding Problem #256

Problem

This problem was asked by Fitbit.

Given a linked list, rearrange the node values such that they appear in alternating `low -> high -> low -> high ...` form. For example, given `1 -> 2 -> 3 -> 4 -> 5`, you should return `1 -> 3 -> 2 -> 5 -> 4`.

Solution

Let's take a look at the example input and see if we can derive an algorithm. One straightforward method is to examine each consecutive pair of nodes, and perform a swap if they do not alternate as required. We would carry out the following steps:

- `1 < 2`? Yes, so we proceed with `1 -> 2`
- `2 > 3`? No, so we swap these values to end up with `1 -> 3 -> 2`
- `2 < 4`? Yes, so we proceed with `1 -> 3 -> 2 -> 4`
- `4 < 5`? No, so we swap these values to end up with `1 -> 3 -> 2 -> 5 -> 4`

In order to implement this, we must know at any given time whether a node's value should be less than or greater than that of its successor. To do this we can use a variable that is `True` at even nodes, and `False` at odd ones.

```
class LinkedList:
    def __init__(self, data):

        self.data = data
        self.next = None

def alternate(ll):
    even = True
    curr = ll

    while curr.next:
        if curr.data > curr.next.data and even:
            curr.data, curr.next.data = curr.next.data, curr.data

        elif curr.data < curr.next.data and not even:
            curr.data, curr.next.data = curr.next.data, curr.data

        even = not even
        curr = curr.next

    return ll
```

While this works, the use of even is somewhat inelegant. Note that in order for the node values to alternate in this way, it must be true that every odd node's value is greater than its preceding and succeeding values. So an alternative algorithm would be to check every other node, and perform the following swaps:

- If the previous node's value is greater, swap the current and previous values.
- If the next node's value is greater, swap the current and next values.

Instead of using a variable for parity, we can use two pointers that jump forward two steps after each check.

```
def alternate(ll):
    prev = ll
    curr = ll.next

    while curr:
        if prev.data > curr.data:
            prev.data, curr.data = curr.data, prev.data

        if not curr.next:

            break
        if curr.next and curr.next.data > curr.data:
            curr.next.data, curr.data = curr.data, curr.next.data

        prev = curr.next
        curr = curr.next.next

    return ll
```

Both of these algorithms use `O(N)` time and `O(1)` space, since we must traverse the entire linked list, and we are only tracking one or two nodes at a time.