Mastering Data Structures and Algorithms
In Class Exercise 3

Yiying Peng (Christine)

March 27, 2019

Problem

Given a singly linked list with nodes with the following structure:

```
Struct node_t {
    Struct node_t * next;
    Struct node_t next_highest;
    Int val;
}
```

Write a function that takes a single linked list of nodes with next_highest initially null and sets that pointer to point to the next node with the highest val following that node in the current list.

Repeat the problem above but now making the extra pointer point to next higher:
Given a singly linked list with nodes with the following structure:

```
Struct node_t {
    Struct node_t * next;
    Struct node_t next_highest;
    Int val;
}
```

Sol. My Python algorithm implementation is as follows.

```
import collections


class LinkedList:
    def __init__(self, lst):
        head = Node(-1)
        self.root = head # this is a null head
        for e in lst:
            head.next = Node(e)
            head = head.next

    def iterate_next_print_val(self):
        head = self.root.next # skip null head
        while head:
            print('Current value: {CUR_VAL}'.format(
                CUR_VAL=head.val,
            ))
            head = head.next
```

```python
    def iterate_next_print_next_highest(self):
        head = self.root.next # skip null head
        while head:
            print('Current value: {CUR_VAL}; Next highest value: {NXT_HIGHEST_VAL}'.format(
                CUR_VAL=head.val,
                NXT_HIGHEST_VAL=head.next_highest.val if head.next_highest else head.next_highest,
            ))
            head = head.next

    def loop_find_next_higher(self):
        head = self.root.next # skip null head
        stack = []
        while head:
            while len(stack) and stack[-1].val < head.val:
                stack[-1].next_highest = head
                stack.pop()
            stack.append(head)
            head = head.next

    def iterate_next_print_next_higher(self):
        head = self.root.next # skip null head
        while head:
            print('Current value: {CUR_VAL}; Next higher value: {NXT_HIGHEST_VAL}'.format(
                CUR_VAL=head.val,
                NXT_HIGHEST_VAL=head.next_highest.val if head.next_highest else head.next_highest,
            ))
            head = head.next


class Node:
    def __init__(self, val):
        self.next = None
        self.next_highest = None
        self.val = val

    def recursive_find_next_highest(self):
        if self.next is None:
            return self
        else:
            self.next_highest = self.next.recursive_find_next_highest()
            if self.next_highest.val < self.val:
                return self
            else:
                return self.next_highest


def main():
    lst = [3, 1, 4, 5, 2]

    # sanity check
    llst = LinkedList(lst)
    llst.iterate_next_print_val()

    # point to next highest
    llst = LinkedList(lst)
    llst.root.next.recursive_find_next_highest()
    llst.iterate_next_print_next_highest()
    # point to nexr higher one
    llst = LinkedList(lst)
    llst.loop_find_next_higher()
    llst.iterate_next_print_next_higher()


if __name__ == '__main__':
    main()
```

For the first problem, I use recursive function to find out the largest number and propagate that number back, which takes $O(n)$ time and $O(n)$ extra space because of the usage of function stack.

For the second problem, since I only need to find the next larger number for one node in the following linked list, I can use stack to achieve it, which takes $O(n)$ time and $O(n)$ extra space. □