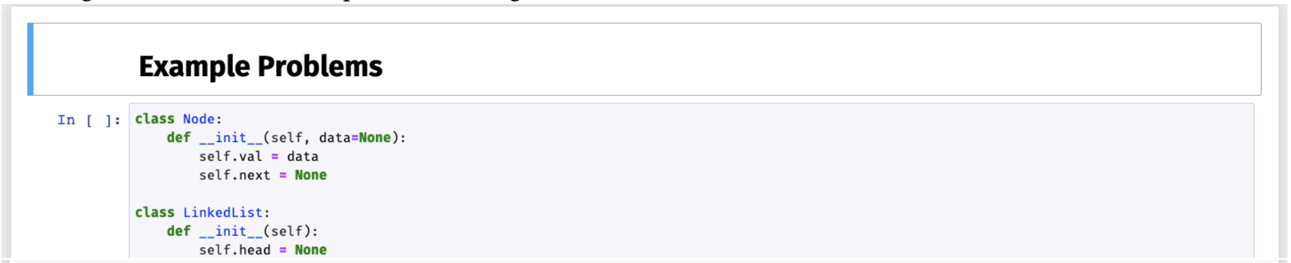
CS218 - Data Structures
FAST NUCES Peshawar Campus
Dr. Nauman (recluze.net)

September 17, 2019

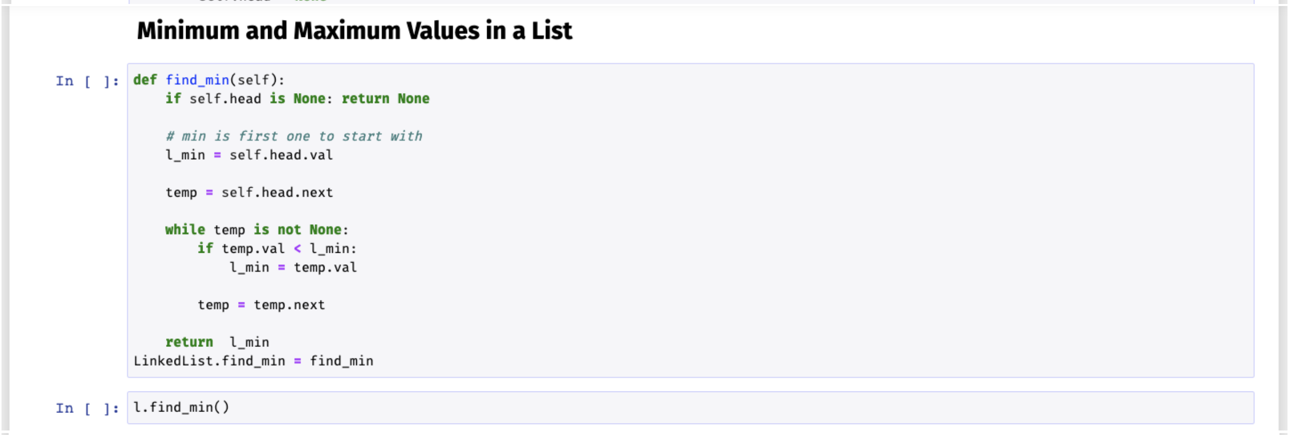
1 Example Problems

Raster images of the notebook 11-problem-solving-01



Example Problems

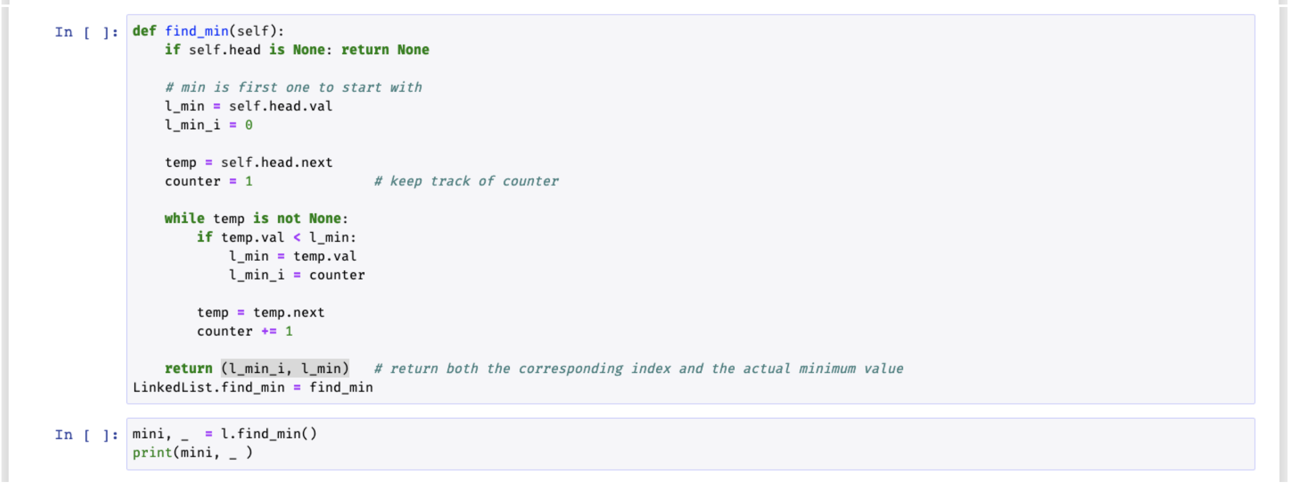
```
In [ ]: class Node:
            def __init__(self, data=None):
                self.val = data
                self.next = None

        class LinkedList:
            def __init__(self):
                self.head = None
```



Minimum and Maximum Values in a List

```
In [ ]: def find_min(self):
            if self.head is None: return None

            # min is first one to start with
            l_min = self.head.val

            temp = self.head.next

            while temp is not None:
                if temp.val < l_min:
                    l_min = temp.val

                temp = temp.next

            return  l_min
        LinkedList.find_min = find_min
```

```
In [ ]: l.find_min()
```



```
In [ ]: def find_min(self):
            if self.head is None: return None

            # min is first one to start with
            l_min = self.head.val
            l_min_i = 0

            temp = self.head.next
            counter = 1             # keep track of counter

            while temp is not None:
                if temp.val < l_min:
                    l_min = temp.val
                    l_min_i = counter

                temp = temp.next
                counter += 1

            return (l_min_i, l_min)   # return both the corresponding index and the actual minimum value
        LinkedList.find_min = find_min
```

```
In [ ]: mini, _  = l.find_min()
        print(mini, _ )
```

Remove the Minimum from a List

In [ ]:
```
def remove_min(self):
    if self.head is None: return
    l_min_i = self.find_min()[0]

    self.remove_at(l_min_i)

LinkedList.remove_min = remove_min
```

In [ ]:
```
print(l)
```

In [ ]:
```
l.remove_min()
print(l)
```

In [ ]:
```
l.remove_min()
print(l)
```

You can do the exact same thing for maximum instead of a minimum

Find Third Highest

In [ ]:
```
l = [101, 202, 303, 404, 5, 6, 10, 20, 1001]
```

In [ ]:
```
def find_three_highest(l):
    if len(l) < 3: return None

    h1 = l[0]
    h2 = l[0]
    h3 = l[0]

    for i in l:
        if i >= h1:
            h3 = h2      # scootch over everybody!
            h2 = h1
            h1 = i

        elif i >= h2:
            h3 = h2
            h2 = i

        elif i >= h3:
            h3 = i


    return (h1, h2, h3)
```

In [ ]:
```
find_three_highest(l)
```

In [ ]:
```
def find_third_highest(l):
    return find_three_highest(l)[2]
```

In [ ]:
```
find_third_highest(l)
```

Reverse a Linked List

This is an important interview question and good for logic building

In [ ]:
```
def rev_list(self):
    # empty list or one element list is already reversed
    if self.head is None: return
    if self.head.next is None: return

    # at least two nodes
    new_head = self._get_last()
    processing = new_head

    for i in range(self.len() - 1):  # loop n-1 times

        temp = self.head
        while temp.next != processing:
            temp = temp.next

        processing.next = temp
        # print(processing.val, " -> ", temp.val)
        processing = processing.next # move "backwards"

    self.head.next = None # this is now the tail
    self.head = new_head


LinkedList.rev_list = rev_list
```

In [ ]:
```
l = LinkedList()
l.push(100)
l.push(200)
l.push(300)
l.push(40)
l.push(500)
print(l)
```

In [ ]:
```
l.rev_list()
print(l)
```

Reversing a Doubly

Reversing a doubly connected linked list is easy! Just set the prev to next and next to prev for all nodes! But make sure you keep track of head at the very beginning!

Most Common Value in a List

We don't have to write the whole thing from scratch. We can use a data structure we already have!

In [ ]:
```
l = LinkedList()
l.push({'age': 15})
l.push({'age': 10})
# l.push(3)
# l.push(1)
# l.push(1)
# l.push(2)
# l.push(500)
# l.push('the')
print(l)
```

In [ ]:
```
def get_counts(self):
    from collections import Counter
    cnt = Counter()

    temp = self.head

    while temp is not None:
        to_count = temp.val['age']
        cnt[to_count] += 1
        temp = temp.next

    return cnt.most_common()

LinkedList.get_counts = get_counts
```

In [ ]:
```
l.get_counts()
```

In [ ]:
```
l.get_counts()[0][0]  # and the most common one is on the top of this list
```

Append One List to Another

If you think before you leap, this is quite easy too.

In [ ]:
```
l = LinkedList()
l.push(1)
l.push(2)
l.push(3)
print(l)

m = LinkedList()
m.push(4)
m.push(5)
m.push(6)
print(m)
```

In [ ]:
```
def append_list(self, lst):
    if self.head is None:
        self.head = lst.head

    last = self._get_last()
    last.next = lst.head

LinkedList.append_list = append_list
```

In [ ]:
```
l.append_list(m)
```

In [ ]:
```
print(l)
```

In [ ]:
```
print(m)
```

In [ ]:
```
m.pop()
```

In [ ]:
```
print(l)   # this might or not be what you want to do!
```

Perform an Operation Over All Elements of a List

This might seem obvious but it's extremely important!

In [ ]:
```
def some_op(self, fn):
    temp = self.head

    while temp is not None:
        print(fn(temp.val))
        temp = temp.next

LinkedList.some_op = some_op
```

In [ ]:
```
from math import sqrt
l.some_op(sqrt)
```