CS-UY 1134 - Spring 2017 1134 Midterm 2

Justin Huang

TOTAL POINTS

55 / 75

QUESTION 1

1 Question 1 15 / 15

- 0 Correct

QUESTION 2

2 Question 2 5 / 15

- 10 used methods of class to accomplish

QUESTION 3

3 Question 3 10 / 20

- 10 Inaccurate implementation - not accumulating the returned list correctly

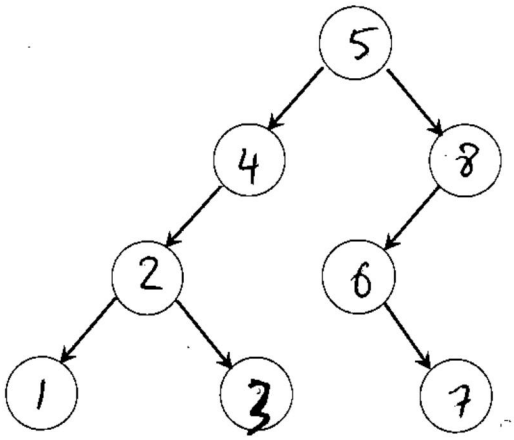
QUESTION 4

4 Question 4 25 / 25

- 0 Correct

Question 1 (15 points)

a. Fill the nodes of the following tree with the numbers 1, 2, 3, 4, 5, 6, 7, 8, so that the resulting tree would be a binary search tree.



b. Let `bst` be an empty binary search tree. Give a sequence of keys to insert to `bst`, so that the resulting tree would contain 1, 2, ..., 8 and it would have the structure of the tree given in section (a).

1st insert: 5

2nd insert: 4

3rd insert: 2

4th insert: 1

5th insert: 3

6th insert: 8

7th insert: 6

8th insert: 7

Question 2 (15 points)

In this question, we will implement `add_second(self, data)`, a new method in the class `DoublyLinkedList`. When called, it adds `data` to the list in the second position, or raises an exception if the list doesn't have any elements (therefore there is no second position).

Complete the code of the definition of the method `add_second` given below.

Implementation requirements: You are **not allowed to use any of the other methods** of the class `DoublyLinkedList`. You need to update the links of the nodes in the list to reflect the insertion.

```
def add_second(self, data):
    if (self.is_empty()):
        raise IndexError("list is empty, can't add second")
    new_node = DoublyLinkedList.Node(data)
    self.insert(1, new_node)
    self.size += 1
    return new_node
```

Question 3 (20 points)

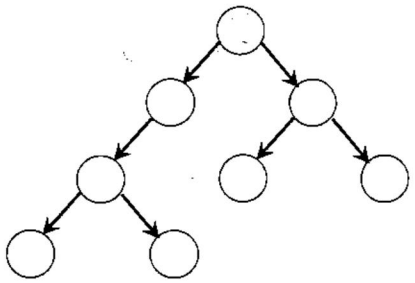
Give a **recursive** implementation of the following method in the `LinkedBinaryTree` class:

```
def subtree_children_dist(self, curr_root):
```

When given `curr_root`, a reference to a root of some subtree, it will return a list, of length 3, which represents the distribution of the number of children of all nodes in the subtree rooted with `curr_root`. That is:

- The first element in the list should be the number of leaves (nodes with no children)
- The second element in the list should be the number of nodes with a single child
- The third element in the list should be the number of nodes with two children

For example, let `curr_root` be a reference the root of the tree below:



The call to `subtree_children_dist` with `curr_root` should return `[4, 1, 3]` (there are 4 leaves, 1 node with a single child, and 3 nodes with 2 children).

Note: Full grade for this question will be given for a linear time solution.

Implement the function on the next page.

```
def subtree_children_dist(self, curr_root):
    leaves = 0
    1ch = 0
    2ch = 0
    if curr_root is None:
        return 0
    else:
        if curr_root.left is not None and curr_root.right is not None:
            2ch += 1
            subtree_children_dist(self, curr_root.left)
            subtree_children_dist(self, curr_root.right)
        elif curr_root.left is not None and curr_root.right is None:
            1ch += 1
            subtree_children_dist(self, curr_root.left)
        else:
            leaves += 1
        return [leaves, 1ch, 2ch]
```

Question 4 (25 points)

In this question, we will suggest a data structure to implement a *boost queue ADT*. A *boost queue* is like a queue, but it also supports a special "boost" operation, that moves the element in the back of the queue a few steps forward.

The operations of the *boost queue ADT* are:

- `__init__(self)`: creates an empty `BoostQueue` object.
- `__len__(self)`: returns the number of elements in the queue
- `is_empty(self)`: returns `True` if and only if the queue is empty
- `enqueue(self, elem)`: adds `elem` to the back of the queue.
- `dequeue(self)`: removes and returns the element at the front of the queue. If the queue is empty an exception is raised.
- `first(self)`: returns the element in the front of the queue without removing it from the queue. If the queue is empty an exception is raised.
- `boost(self, k)`: moves the element from the back of the queue k steps forward. If the queue is empty an exception is raised. If k is too big (greater or equal to the number of elements in the queue) the last element will become the first.

For example, your implementation should provide the following behavior:

```
>>> boost_q = BoostQueue()
>>> boost_q.enqueue(1)
>>> boost_q.enqueue(2)
>>> boost_q.enqueue(3)
>>> boost_q.enqueue(4)
>>> boost_q.boost(2)
>>> boost_q.dequeue()
1
>>> boost_q.dequeue()
4
>>> boost_q.dequeue()
2
>>> boost_q.dequeue()
3
```

Implementation requirements:

1. `BoostQueue` objects may only use a doubly linked list (of type `DoublyLinkedList`) as a data member to store the queue's elements. In additional to that your object can only use $\theta(1)$ extra space.
2. All queue operations should run in $\theta(1)$ worst case, besides the `boost(k)` operation, that should run in $\theta(k)$ worst case.

```
import doubly_linked_list
class Empty(Exception):
    pass
class BoostQueue:
    def __init__(self):
        self.data = None

    def __len__(self):
        return len(self)

    def is_empty(self):
        if len(self) == 0:
            return True
        else:
            return False
    def enqueue(self, elem):
        self.append(elem)

    def first(self):
        if (self.is_empty()):
            raise Empty("Boost queue is empty")
        else:
            return self.remove(0)
            return self(0)
```

```
def dequeue(self):
    if (self.is_empty()):
        raise Empty("Boost queue is empty")
    else:
        del self(0)
        n = self(0)
        return n
        del n
```

```
def boost(self, k):
    if (self.is_empty()):
        raise Empty("Boost queue is empty")
    else:
        last_elem = self[len(self)-1]
        boost_ind = (len(self)-1)-k
        self
        if k <= len(self):
        # last_elem = self[len(self)-1]
            boost_ind = (len(self)-1)-k
            self.insert(boost_ind, last_elem)
        else:
            last_elem = self[len(self)-1]
            self.insert(0, last_elem)
```

EXTRA PAGE IF NEEDED

Note question numbers of any questions or part of questions that you are answering here.
Also, write "ANSWER IS ON LAST PAGE" near the space provided for the answer.