NYU, Tandon School of Engineering
CS-UY 1134 Data Structures and Algorithms — Summer Session II 2020

Homework #5

Due by Tuesday 8/11, 11:55 pm

Submission instructions:

a) For this assignment, you should turn in 3 files:
   a. 3 '.py' files, one for each question 1-3. Name your files: YourNetID_hw5_q2.py' and 'YourNetID_hw5_q3.py', etc.

      Note: your netID follows an abc123 pattern, not N12345678.

b) You should submit your homework via Gradescope.
   - Name all classes, functions, and methods exactly as they are in the assignment specifications.
   - Make sure there are no print statements in your code. If you have a tester code, please put it in a "main" function and do not call it.

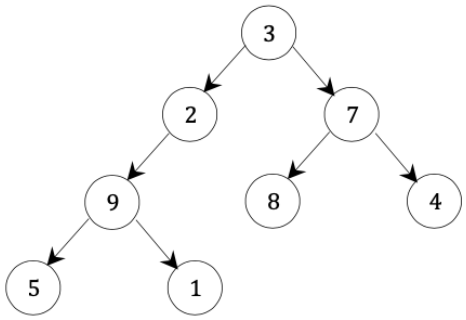
For Binary Tree related questions, use LinkedBinaryTree.py file.
For Binary Search Tree related questions, use BinarySearchTree.py file.

Question 1:

Define the following function:

```
def min_and_max(bin_tree):
```

When called on a `LinkedBinaryTree`, containing numerical data in all its nodes, it will **return a tuple**, containing the maximum and minimum values in the tree. For example, given the following tree:



Calling `min_and_max` on the tree above, should return (1, 9).

Implementation requirements:

1. Define one additional, **recursive**, helper function:

   ```
   def subtree_min_and_max(root)
   ```

   That is given `root`, a reference to a `TreeNode` in a `LinkedBinaryTree`. When called, it should return the minimum and maximum tuple for the subtree rooted by `root`.

2. In your implementations, you are not allowed to use any method from the `Tree` class. Specifically, you are not allowed to iterate over the tree, using any of the traversals.

3. Your function should run in **linear time.**

4. Since the maximum and minimum are not defined on an empty set of elements, if the function is called on an empty tree this should raise an exception.

Question 2:

Add the following method to the `LinkedBinaryTree` class:

```
def leaves_list(self)
```

When called on a tree, it will create and return a list, containing the values stored at the leaves of the tree, ordered from left to right. For example, if called on the tree from question 1, it should return: `[5, 1, 8, 4]`

Implementation requirements:

1. Your method should run in **linear time**.
   Hint: To meet this requirement you may want to define a generator that yields the desired values. You could then turn it to a list, using the `list` constructor.
2. In your implementations, you are not allowed to use any method from the `Tree` class. Specifically, you are not allowed to iterate over the tree, using any of the traversals.

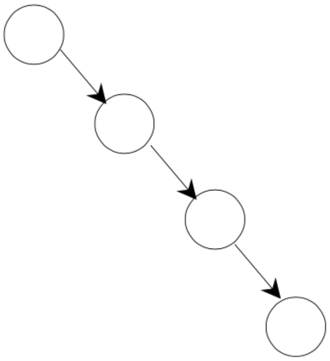
Question 3:

a) Implement the following function:

```
def create_chain_bst(n)
```

This function gets a positive integer `n`, and returns a binary search tree with `n` nodes containing the keys *1*, *2*, *3*, …, *n*. The structure of the tree should be one long chain of nodes leaning to the right.

For example, the call `create_chain_bst(4)` should create a tree of the following structure (with the values *1*, *2*, *3*, *4* inside its nodes in a valid order):



Implementation requirement: In order to create the desired tree, your function has to construct an empty binary search tree, and can then only make repeated calls to the insert method, to add entries to this tree.

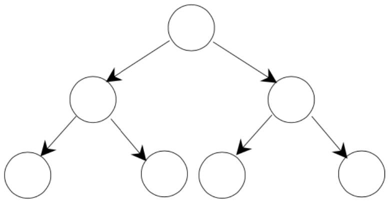
b) In this section, you will show an implementation of the following function:

```
def create_complete_bst(n)
```

`create_complete_bst` gets a positive integer `n`, where `n` is of the form $n=2^k-1$ for some non-negative integer *k*.
When called it returns a **binary search tree** with `n` nodes, containing the keys *1*, *2*, *3*, …, *n*, structured as a **complete** binary tree.

Note: The number of nodes in a complete binary tree is 2^k-1, for some non- negative integer *k*.

For example, the call `create_complete_bst(7)` should create a tree of the following structure (with the values *1*, *2*, *3*, *4*, *5*, *6*, *7* inside its nodes in a valid order):



You are given the implementation of `create_complete_bst`:

```
def create_complete_bst(n):

    bst = BinarySearchTree()

    add_items(bst, 1, n)

    return bst
```

You should implement the function:

```
def add_items(bst, low, high)
```

This function is given a binary search tree `bst`, and two positive integers `low` and `high`(`low ≤ high`). When called, it adds all the integers in the range `low` ... `high` into `bst`.

Note: Assume that when the function is called, none of the integers in the range `low` ... `high` are already in `bst`.

Hints:

- Before coding, try to draw the binary search trees (structure and entries) that `create_complete_bst(n)` creates for `n=7` and `n=15`.
- It would be easier to define `add_items` recursively.

c) Analyze the runtime of the functions you implemented in sections (a) and (b)