Lab 9 – CSCI 112

Due Date: Wednesday, March 23 at 11:59pm EST

Information

- This lab is intended to be completed **individually.**
- The files must be submitted with the exact file name provided in this file. If the file names do not match you will receive **zero** points for that file.
- Before you submit, make sure that your code runs. Any code which does not run without errors will receive **zero** points.
- Do not share your work with anyone other than Professor Khan or the TAs. You may discuss algorithms, approaches, ideas, but **NOT** exact code.
- If you submit work after a second past the due date **WILL** be locked out from submission.

Assignment

In this project, you will complete the tree traversal methods and also add some methods to a binary search tree class to optimize its runtime behavior. To test your implementations, you use the code in `testtree.py.`

Task 1 – Tree Traversal

Begin by completing the traversal methods in `linkedbst.py`. For this part, you may initially use recursive implementations for the various traversals, as discussed in class. The strategy for a level order traversal is mentioned on page 301 of the textbook, which you may use as a reference.

Task 2 – Improving Traversals

One issue with the recursive tree traversals is that each build a list from the tree and return an iterator on this list. This approach wastes running time and memory, and does not catch attempts to mutate the tree within the context of a traversal. To improve this, one can use a stack to yield items as they are visited in the tree. Use this insight to make the methods `preorder` and `inorder` run more efficiently.

Here is the pseudocode for `inorder` with a stack:

```
stack = new stack
probe = root
while True
    if probe != None
```

```
        stack.push(probe)
        probe = probe.left
    elif stack not empty
        probe = stack.pop
        yield probe's data
        probe = probe.right
    else
        break
```

Task 3 – Balancing the Tree [7 points]

Recall that the performance of the access methods of a binary search tree depends on the shape of the tree. As the tree becomes linear in shape, these methods acquire linear running times. In contrast, a balanced tree supports worst-case logarithmic behavior for searches and insertions.

The shape of a binary search tree can be quantified as a relationship between its height (the longest path from the root to a leaf node) and its length (the total number of nodes). Ideally, the tree's height will be no greater than $\log_2(\text{length} + 1) - 1$. However, optimal behavior for large trees can be achieved when the tree's height is no greater than twice that amount. Periodic rebalancing of unbalanced BSTs can dramatically improve the performance of their access methods.

Now complete the following three methods in your `LinkedBST` class:

```
t.height()              # Returns the height of the tree

t.isBalanced()          # Returns True if the tree is balanced or False
                        # otherwise

t.rebalance()           # Rebalances the tree if necessary
```

A short test function (`testbalance`) appears in `testtree.py`. Use this function to test your new methods.

The `height` method should contain a simple recursive function on a node. The height of an empty node is 0. The height of a nonempty node is 1 plus the maximum of the heights of its two subtrees. If the tree is not empty, the result returned by this recursive function is deceremented by 1.

The `isBalanced` method calls the `height` method and the `len` function to determine whether the tree is balanced, using the criteria described above.

The `rebalance` method obtains a sorted list of the tree's items and then clears the tree. The `rebalance` method then calls a recursive helper method, named `rebuild`, to add the items from the list to the tree. The helper method visits the midpoint of the list to add an item, then recurses with the left half of the list and with the right half of the list. This is kind of like a quicksort where the pivot is always at the midpoint. The process stops when the left and right index values cross over.

What To Turn In

For this lab, you should only submit the completed **`linkedbst.py`** file on Canvas.