Module 9 Homework: Binary Search Trees

We provide the code for a working binary search tree (BST) mapping. Inherit the provided `BSTMap` and `BSTNode` classes into subclasses `MyBSTMap` and `MyBSTNode` and add 2 pieces of functionality:

1) Equality - Return `True` if two trees share the same key:value pairs and shape
2) Tree reconstruction - return a BST equal to a tree used to create a list of (key, value) pairs via pre- or postorder traversal.

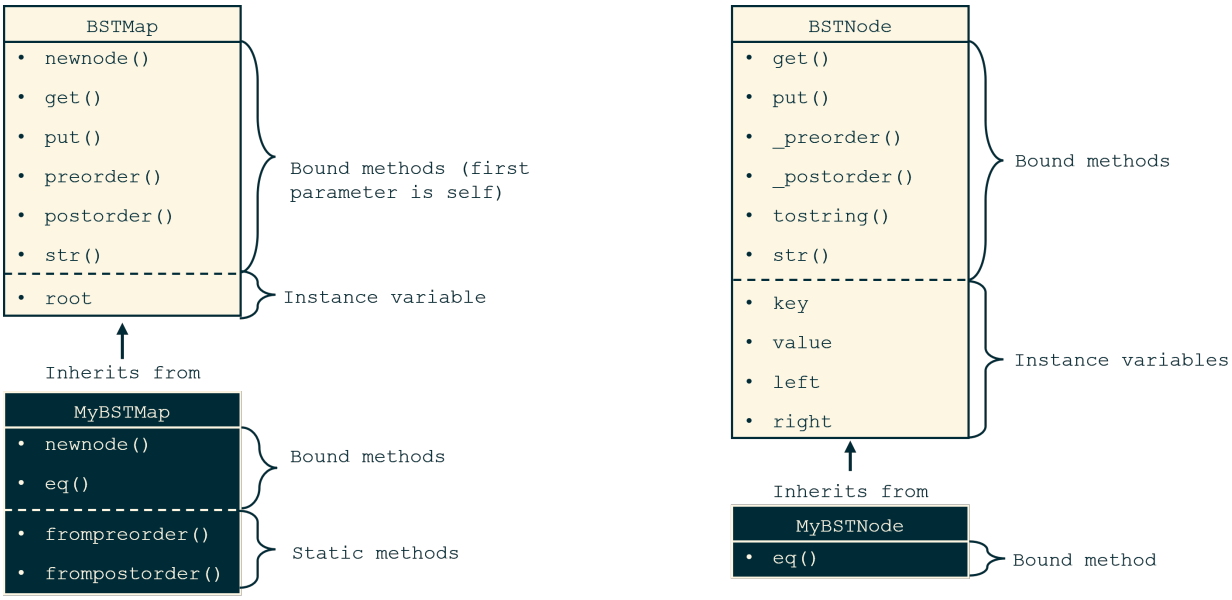

Figure 1: Class diagrams for this assignment. The dark-background boxes are the classes you will be coding. The light-background boxes are already complete.

Part 1 - Equality

Implement a method to compare two BSTs. Two BSTs are equal if both contain exactly the same k-v pairs **and** have the same shape:

```
      3:'3'              3:'3'
     /                  /
   1:'1'      ==      1:'1'
  /    \             /    \
0:'0'  2:'2'       0:'0'  2:'2'

      3:'3'              1:'1'
     /                  /    \
   1:'1'      !=      0:'0'   3:'3'
  /    \                      /
0:'0'  2:'2'                2:'2'
```

As is typical for Trees comprising Nodes, the user will call equality on the tree (`bst1 == bst2`). The equality function in the tree will then call equality on the root nodes, which will recursively call equality on their left and right children.

Most of the work here is recursively defining equality for a node - two nodes are equal if:

1) They have the same key:value pairs
2) The subtrees rooted at their left and right children are equal

Unittests

Include at a minimum the following unittests:

- 2 empty trees (should be equal)
- 2 equal trees with several levels of nodes
- 2 unequal trees with several layers of nodes:
  a) same key:value pairs, different shapes
  b) same shapes, different key:value pairs

Use ascii art in the test cases to denote what the trees should look like for each of the above.

Part 2 - Tree Reconstruction

Consider the following tree of `k:v` pairs, where the values are all `str(k)`:

```
      3:'3'
     /
   1:'1'
  /    \
0:'0'  2:'2'
```

Traversals of this tree yields the following lists of `k:v` tuples:

- pre-order: `[(3,'3'), (1,'1'), (0,'0'), (2,'2')]`
- post-order: `[(0,'0'), (2,'2'), (1,'1'), (3,'3')]`

Implement two functions:

- `frompreorder` - generates a BST from a list of pre-ordered `k, v` tuples
  - takes the list as input
  - returns the generated BST
- `frompostorder` - as above, but for postorder.

Examples

Examples are illustrative, not exhaustive. Do not use these examples for your own tests - create your own.

```
>>> from MyBSTMap import MyBSTMap
>>> bst1 = MyBSTMap()
>>> for k in [3, 1, 2, 0]: # Build the tree shown above
...     bst1.put(k, str(k))
...
>>> L = [(k, v) for (k, v) in bst1.preorder()] # construct preorder list
>>> bst2 = MyBSTMap.frompreorder(L)            # reconstruct the original bst
>>> bst1 == bst2                               # verify trees are equal
True
```

Unittests

Start with a small tree (~3 levels), and include ascii art of the tree as above. Do not use any of the trees or shapes given in this assignment.

Then, compare two large random trees (at least 100 nodes)

a) create a tree using randomly generated keys (use the `random` module)
b) create a list of k:v pairs using preorder traversal
c) create a new tree using that list `MyBSTMap.frompreorder()`
d) verify the two trees are equal
e) modify one of the trees
f) verify the two trees are no longer equal.

Repeat the above for `frompostorder`.

Submitting

At a minimum, submit the following files:

- `BSTMap.py` (unmodified)
- `MyBSTMap.py`
- `TestMyBSTMap.py`

Students must individually by the due date (Tuesday at 11:59 pm) to receive credit.

Grading

This assignment is 100% manually graded.

- 30 - Equality
  - 20 - functionality
  - 10 - tests
- 35 - `frompreorder`
  - 15 - functionality
  - 20 - tests
- 35 - `frompostorder`
  - 15 - functionality
  - 20 - tests

Feedback

If you have any feedback on this assignment, please leave it here.