Module6 - Critical Thinking - Option1

David Edwards

CSC565 - Design and Analysis of Algorithms

Colorado State University-Global Campus

Dr. Isaac K. Gang

April 25, 2021

- Build a Node class. It is should have attributes for the data it stores as well as its left and right children. As a bonus, try including the Comparable module and make nodes compare using their data attribute.
- Build a Tree class which accepts an array when initialized. The Tree class should have a root attribute which uses the return value of #build_tree which you'll write next.
- Write a #build_tree method which takes an array of data (e.g. [1, 7, 4, 23, 8, 9, 4, 3, 5, 7, 9, 67, 6345, 324]) and turns it into a balanced binary tree full of Node objects appropriately placed (don't forget to sort and remove duplicates!). The #build_tree method should return the level-1 root node.
- Write an #insert and #delete method which accepts a value to insert/delete.

To complete this project, I borrowed liberally from the ZyBooks AVL Tree implementation, with tree printing code from http://krenzel.org/articles/printing-trees.

```
tree = Tree()
# insert with duplicates
tree.build_tree([1, 7, 4, 23, 8, 9, 4, 3, 5, 7, 9, 67, 6345, 324])
print(tree)
```

```
            6345
        324
    67
            23
        9
8
        7
    5
            4
        3
            1
```

This shows several things. First, this is a sideways representation of a binary tree, with `8` in the root, and leaves of `6345, 23, 4, 1`. This allows a quick view of the tree to verify things are working properly.

Second, this shows that the build_tree implementation properly removes duplicates (using the `set()` function). Per the description of a Tree class, you can pass the array of values directly in when initializing, or you can also use the `build_tree()` method after the Tree is created. That's what this example shows.

Third, this shows a properly balanced (but not complete) tree. The nodes `9` and `7` have no children, which they would need to be complete.

Next, we'll try inserting a duplicate:

```
tree.insert(7)
print(tree)
```

```
            6345
        324
    67
            23
        9
8
        7
    5
            4
        3
            1
```

This shows that no duplicate `7` was inserted. Now we'll try to imbalance the tree.

```
tree.insert(232)
print(tree)
```

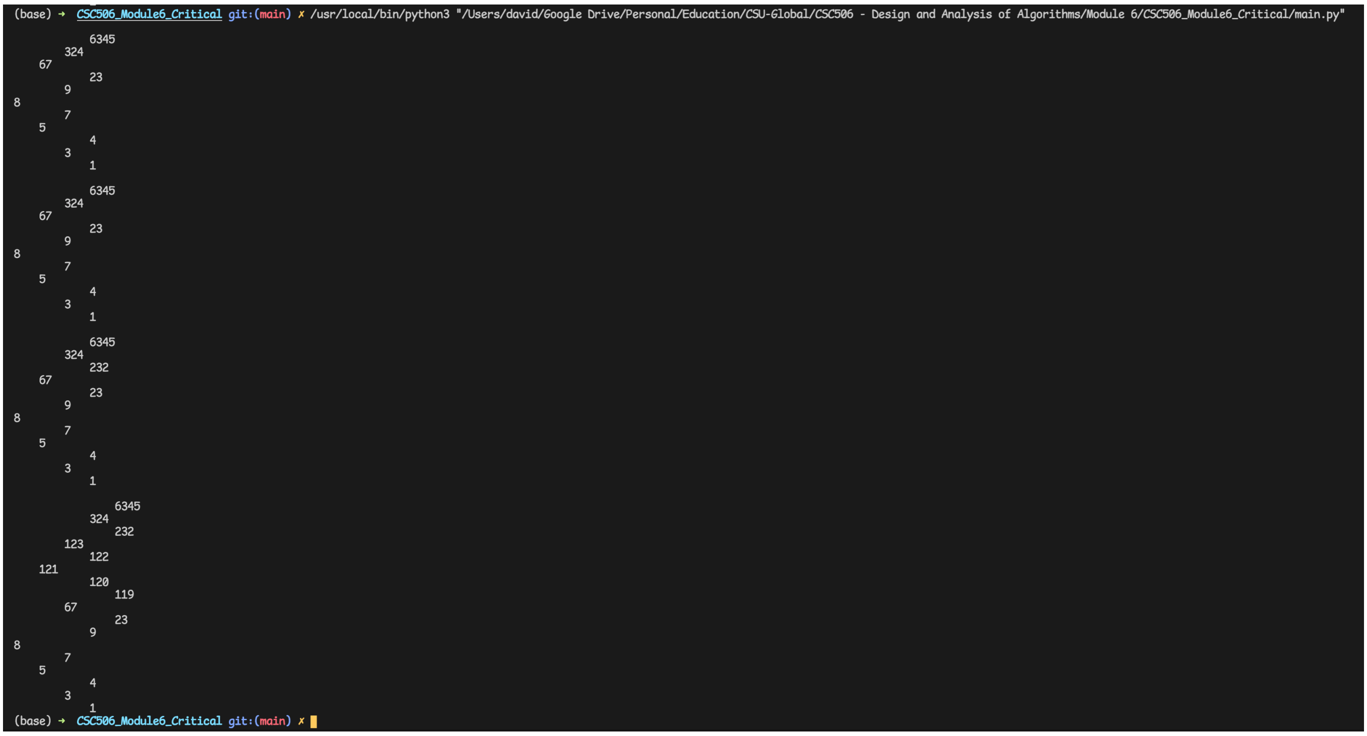
```
            6345
        324
            232
    67
            23
        9
8
        7
    5
            4
        3
            1
```

If we continue adding items that would appear to the left of `232` we will find if our balancing works:

```
tree.insert(123)
tree.insert(122)
tree.insert(121)
tree.insert(120)
tree.insert(119)
print(tree)
```

```
                6345
            324
                232
        123
            122
    121
            120
                119
        67
                23
            9
8
        7
    5
            4
        3
            1
```

This shows the tree being properly rebalanced. Here is the full execution screen shot:



```
(base) ➜  CSC506_Module6_Critical git:(main) ✗ /usr/local/bin/python3 "/Users/david/Google Drive/Personal/Education/CSU-Global/CSC506 - Design and Analysis of Algorithms/Module 6/CSC506_Module6_Critical/main.py"
            6345
        324
    67
            23
        9
8
        7
    5
            4
        3
            1
            6345
        324
    67
            23
        9
8
        7
    5
            4
        3
            1
            6345
        324
            232
    67
            23
        9
8
        7
    5
            4
        3
            1
                6345
            324
                232
        123
            122
    121
            120
                119
        67
                23
            9
8
        7
    5
            4
        3
            1
(base) ➜  CSC506_Module6_Critical git:(main) ✗ ▌
```