Code Sample for Nathan Cooper Jones

AVL Tree

This was an extra credit lab for my data structures class, which we wrote on our own to show our mastery of data structures. Here, I create a self-balancing binary search tree and test it in the final cell, showing that no matter what order I add the elements into the tree, it will always stay balanced.

In [41]:

```
class AVLTree:
    class Node:
        def __init__(self, val, left=None, right=None):
            self.val = val
            self.left = left
            self.right = right

        def rotate_right(self):
            n = self.left
            self.val, n.val = n.val, self.val
            self.left, n.left, self.right, n.right = n.left, n.right, n

        def rotate_left(self):
            n = self.right
            self.val, n.val = n.val, self.val
            self.right, n.right, self.left, n.left = n.right, n.left, n

        @staticmethod
        def height(n):
            if not n:
                return 0
            else:
                return max(1+AVLTree.Node.height(n.left), 1+AVLTree.Nod

    def __init__(self):
        self.size = 0
        self.root = None

    @staticmethod
    def rebalance(t):
        if AVLTree.Node.height(t.left) > AVLTree.Node.height(t.right):
            if AVLTree.Node.height(t.left.left) >= AVLTree.Node.height(
                # left-left
                # print('left-left imbalance detected')
                t.rotate_right()
            else:
                # left-right
                # print('left-right imbalance detected')
                t.left.rotate_left()
                t.rotate_right()
        else:
            # right branch imbalance tests needed
            if AVLTree.Node.height(t.right.right) >= AVLTree.Node.heigh
                # right-right
                # print('right-right imbalance detected')
                t.rotate_left()
            else:
                # right-left
                # print('right-left imbalance detected')
                t.right.rotate_right()
                t.rotate_left()

    def add(self, val):
        assert(val not in self)
        def add_rec(node):
            if not node:
```

```
57                 return AVLTree.Node(val)
58             elif val < node.val:
59                 node.left = add_rec(node.left)
60             else:
61                 node.right = add_rec(node.right)
62             if abs(AVLTree.Node.height(node.left) - AVLTree.Node.height
63                 AVLTree.rebalance(node)
64             return node
65         self.root = add_rec(self.root)
66         self.size += 1
67 
68     def __delitem__(self, val):
69         assert(val in self)
70         def delitem_rec(node):
71             if val < node.val:
72                 node.left = delitem_rec(node.left)
73             elif val > node.val:
74                 node.right = delitem_rec(node.right)
75             else:
76                 if not node.left and not node.right:
77                     return None
78                 elif node.left and not node.right:
79                     return node.left
80                 elif node.right and not node.left:
81                     return node.right
82                 else:
83                     stack = []
84                     # remove the largest value from the left subtree (t
85                     # for the root value of this tree
86                     t = node.left
87                     stack.append(t)
88                     if not t.right:
89                         node.val = t.val
90                         node.left = t.left
91                     else:
92                         n = t
93                         while n.right.right:
94                             n = n.right # passing through nodes here
95                             stack.append(n) # append it to a 'stack' -
96                         t = n.right
97                         n.right = t.left
98                         node.val = t.val
99                     while stack:
100                        temp_node = stack.pop()
101                        if abs(AVLTree.Node.height(temp_node.left) - AV
102                            AVLTree.rebalance(temp_node)
103            if abs(AVLTree.Node.height(node.left) - AVLTree.Node.height
104                AVLTree.rebalance(node) # rebalance
105            return node
106        self.root = delitem_rec(self.root)
107        self.size -= 1
108
109    def __contains__(self, val):
110        def contains_rec(node):
111            if not node:
112                return False
113            elif val < node.val: # which path down the tree should we g
```

```
                return contains_rec(node.left)
            elif val > node.val:
                return contains_rec(node.right)
            else:
                return True
        return contains_rec(self.root)

    def __len__(self):
        return self.size

    def __iter__(self):
        def iter_rec(node):
            if node:
                yield from iter_rec(node.left)
                yield node.val
                yield from iter_rec(node.right)
        yield from iter_rec(self.root)

    def pprint(self, width=64): # CODE FROM INSTRUCTOR
        """Attempts to pretty-print this tree's contents."""
        height = self.height()
        nodes  = [(self.root, 0)]
        prev_level = 0
        repr_str = ''
        while nodes:
            n,level = nodes.pop(0)
            if prev_level != level:
                prev_level = level
                repr_str += '\n'
            if not n:
                if level < height-1:
                    nodes.extend([(None, level+1), (None, level+1)])
                repr_str += '{val:^{width}}'.format(val='-', width=widt
            elif n:
                if n.left or level < height-1:
                    nodes.append((n.left, level+1))
                if n.right or level < height-1:
                    nodes.append((n.right, level+1))
                repr_str += '{val:^{width}}'.format(val=n.val, width=wi
        print(repr_str)

    def height(self): # CODE FROM INSTRUCTOR
        """Returns the height of the longest branch of the tree."""
        def height_rec(t):
            if not t:
                return 0
            else:
                return max(1+height_rec(t.left), 1+height_rec(t.right))
        return height_rec(self.root)
```

In [42]:
```
import random

TestTree = AVLTree() # establish our testing tree

TestTree.add(18)
TestTree.add(17)
TestTree.add(1)
TestTree.add(49)
TestTree.add(33)
TestTree.add(5)
TestTree.pprint()
print('\n')

del TestTree[1]
TestTree.pprint()
print('\n')
del TestTree[17]
TestTree.pprint()
print('\n')

for x in TestTree:
    print(x)
print('\n')

lst = list(range(55))
random.shuffle(lst)
print("Randomized list of elements: ", lst, '\n')

for x in lst:
    if x not in TestTree:
        TestTree.add(x)
    else:
        del TestTree[x]
TestTree.pprint()
print('\n')

for x in TestTree:
    print(x) # elements still in order
```

```
                        17
                1                               33
        -               5               18              49


                        17
                5                               33
        -               -               18              49


                        33
                5                               49
        -               18              -               -
```

```
5
18
33
49
```

```
Randomized list of elements:  [30, 38, 7, 2, 17, 49, 24, 37, 36, 18, 35,
 5, 50, 46, 12, 28, 43, 40, 42, 34, 13, 19, 20, 4, 33, 54, 44, 41, 53, 3
1, 21, 25, 29, 6, 52, 47, 1, 27, 11, 15, 10, 48, 23, 8, 3, 14, 16, 9, 45,
 0, 32, 51, 26, 39, 22]
```

```
                                30
                17                                      43
        7               24                      37              50
    2       12      20      28      35      40      46      53
 1   4   10  14  19  22  26  29  32  36  38  42  44  47  52  54
0 - 3 6 8 111315- - 21232527- - 3134- - - 3941- - 45- 4851- - -
---------9-----16-------------------------------------------------
```

```
0
1
2
3
4
6
7
8
9
10
11
12
13
14
15
16
17
19
20
21
22
23
```

24
25
26
27
28
29
30
31
32
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
50
51
52
53
54