CS532: Homework 2 (version 0)

Question 1: Running Code from Lecture Notes

On the course website, I collected all the code from the lecture notes on binary search trees. I changed InOrderWalk so that it collects its result in a string, and returns the string.
Write some test code to show that Insert, Delete, Search, Min, Succ, and InOrderWalk work. You can use a constant for the tree, but do not use a variable or constant in creating any of the nodes. Make sure that you test your code on a tree of a non-trivial size. Make sure that InOrderWalk, Min, Search work both from the root node and some other node.
Hand in the code you used to test your routine as well as the output.

From Tree class:

```
    def Insert_test(self):
        print("testing insert into empty tree\nTest: ",end='')
        self.root = None
        self.Insert(Node(1))
        if str(self) != "1":
            print("Failed")
            return 0
        print("Passed")

        self.even_tree()
        print("basic tree: " + str(self))
        if str(self) != "((2)4(6))8((10)12(14))":
            print("error w/ testing string")
            return 0

        print("Inserting:  14")
        self.Insert(Node(14))
        test = str(self)
        print(test + "\nTest:",end='')
        if test != "((2)4(6))8((10)12(14(14)))":
            print("Failed")
            return 0
        else:
            print("Passed")

        print("Inserting:  4")
        self.Insert(Node(4))
        test = str(self)
        print(test + "\nTest:",end='')
```

```
          if test != "((2)4((4)6))8((10)12(14(14)))":
               print("Failed")
               return 0
          else:
               print("Passed")

          return 1
     ###############
     tree = Tree()
     if tree.Insert_test(): print("All tests Passed")
```

Output:

```
     spencer@spencer-rBS:~/OHSU/532/hw2$ !p
     python3 hw2.py
     testing insert into empty tree
     Test: Passed
     basic tree: ((2)4(6))8((10)12(14))
     Inserting:  14
     ((2)4(6))8((10)12(14(14)))
     Test:Passed
     Inserting:  4
     ((2)4((4)6))8((10)12(14(14)))
     Test:Passed
     All tests Passed
```

From Tree class:

```
     @staticmethod
     def Delete_test():
          tree = Tree()
          tree.basic_tree()
          string = str(tree)
          print("Delete Test: \n" + string)
          if string !=
     "(((1)2(3))4((5)6(7)))8(((9)10(11))12((13)14(15)))":
               print("Error building tree")
               return False

          print("deleting:  root")
          tree.Delete(tree.root)
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string ==
     "(((1)2(3))4((5)6(7)))9((10(11))12((13)14(15)))":
               print("Passed")
          else: print("Failed"); return 0;

          print("deleting:  root")
          tree.Delete(tree.root)
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string == "(((1)2(3))4((5)6(7)))10((11)12((13)14(15)))":
               print("Passed")
          else: print("Failed"); return 0;

          print("deleting:  min")
          tree.Delete(tree.root.Min())
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string == "((2(3))4((5)6(7)))10((11)12((13)14(15)))":
               print("Passed")
          else: print("Failed"); return 0;

          print("deleting:  min")
          tree.Delete(tree.root.Min())
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string == "((3)4((5)6(7)))10((11)12((13)14(15)))":
               print("Passed")
          else: print("Failed"); return 0;

          print("deleting:  max")
          tree.Delete(tree.root.Max())
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string == "((3)4((5)6(7)))10((11)12((13)14))":
               print("Passed")
          else: print("Failed"); return 0;

          print("deleting:  max")
          tree.Delete(tree.root.Max())
```

```
          string = str(tree)
          print (string + "\nTest:  ",end='')
          if string == "((3)4((5)6(7)))10((11)12(13))":
               print("Passed")
          else: print("Failed"); return 0;

          print("finishing...")
          while tree.root:
               tree.Delete(tree.root)
               print(tree)

          return True
```

Output:

```
spencer@spencer-rBS:~/OHSU/532/hw2$ !p
python3 hw2.py
Delete Test:
(((1)2(3))4((5)6(7)))8(((9)10(11))12((13)14(15)))
deleting:  root
(((1)2(3))4((5)6(7)))9((10(11))12((13)14(15)))
Test:  Passed
deleting:  root
(((1)2(3))4((5)6(7)))10((11)12((13)14(15)))
Test:  Passed
deleting:  min
((2(3))4((5)6(7)))10((11)12((13)14(15)))
Test:  Passed
deleting:  min
((3)4((5)6(7)))10((11)12((13)14(15)))
Test:  Passed
deleting:  max
((3)4((5)6(7)))10((11)12((13)14))
Test:  Passed
deleting:  max
((3)4((5)6(7)))10((11)12(13))
Test:  Passed
finishing...
((3)4((5)6(7)))11(12(13))
((3)4((5)6(7)))12(13)
Traceback (most recent call last):
  File "hw2.py", line 282, in <module>
    if Tree.Delete_test():
  File "hw2.py", line 192, in Delete_test
    tree.Delete(tree.root)
  File "hw2.py", line 46, in Delete
    y.right.parent = y
AttributeError: 'NoneType' object has no attribute 'parent'
```

The Tree class Delete method does not account for the case where `z.right.Min() == z.right`.

From Node class:

```
@staticmethod
def Search_test():
     tree = Tree()
     tree.basic_tree()
     if tree.root.Search(8) == None:
          return False
     if tree.root.right.Search(8) != None:
          return False
     if tree.root.Search(1) == None:
          return False
     if tree.root.left.Search(1) == None:
          return False
     if tree.root.right.Search(1) != None:
          return False
     if tree.root.Search(2) == None:
          return False
     if tree.root.Search(-1) != None:
          return False
     if tree.root.Search(12) == None:
          return False
     if tree.root.right.Search(12) == None:
          return False
     if tree.root.Search(10) == None:
          return False
     if tree.root.Search(11) == None:
          return False
     if tree.root.Search(7) == None:
          return False
     return True
```

###############################

```
if Node.Search_test(): print("Search Test Passed")
else: print("Search Test Failed")
```

Output:

```
    spencer@spencer-rBS:~/OHSU/532/hw2$ !p
    python3 hw2.py
    Search Test Passed
```

From Node class:

```
    @staticmethod
    def Min_test():
        tree = Tree()
        tree.basic_tree()
        if tree.root.Min().key != 1:
            return False
        if tree.root.left.Min().key != 1:
            return False
        if tree.root.right.Min().key != 9:
            return False
        return True
###################
if Node.Min_test(): print("Min Test Passed")
else: print("Min Test Failed")
```

Output:

```
    spencer@spencer-rBS:~/OHSU/532/hw2$ !p
    python3 hw2.py
    Min Test Passed
```

From Node class:

```
    @staticmethod
        def Succ_test():
        tree = Tree()
        tree.basic_tree()
        if tree.root.Succ().key != 9:
            return False
        if tree.root.right.Succ().key != 13:
            return False
        if tree.root.left.Succ().key != 5:
            return False
        if tree.root.Min().Succ().key != 2:
            return False
```

```
          if tree.root.Search(9).Succ().key != 10:
               return False
          if tree.root.Max().Succ() != None:
               return False
          return True
#######################
if Node.Succ_test(): print("Succ Test Passed")
else: print("Succ Test Failed")
```

Output:

```
     spencer@spencer-rBS:~/OHSU/532/hw2$ !p
     python3 hw2.py
     Succ Test Passed
```

From Node class:

```
     @staticmethod
     def InOrderWalk_test():
          tree = Tree()
          tree.basic_tree()
          if tree.root.InOrderWalk() != "123456789101112131415":
               return False
          if tree.root.left.InOrderWalk() != "1234567":
               return False
          if tree.root.right.InOrderWalk() != "9101112131415":
               return False
          if tree.root.Min().InOrderWalk() != "1":
               return False
          return True
####################################
if Node.InOrderWalk_test(): print("InOrderWalk Test Passed")
else: print("InOrderWalk Test Failed")
```

Output:

```
     spencer@spencer-rBS:~/OHSU/532/hw2$ !p
     python3 hw2.py
     InOrderWalk Test Passed
```

Question 2: str Method

Create a magic method for str. This will work similar to InOrderWalk, except that you should show the hierarchy of the tree using brackets. Consider the following tree:

```
     6
```

```
    / \
   3   9
  / \ / \
 2   5 7 11
```

str should output ((2)3(5))6((7)9(11))

Hand in a copy of your str code.

```
def __str__(self):
     if self.root:
          return self.__to_string(self.root.left) \
               + str(self.root.key) \
               + self.__to_string(self.root.right)
     return ""

def __to_string(self, root):
     if root is None:
          return ""
     return ("("
          + self.__to_string(root.left)
          + str(root.key)
          + self.__to_string(root.right)
          + ")")
```

Question 3: Insert

In class, we discussed how the Insert code from the textbook hid what happens when we insert into an empty tree. Change the code for Insert so that it deals with the empty tr
ee case first. Make sure you do not leave any of the existing code that deals with the empty case. Turn in your code.

```
def Insert(self,z):
     if not self.root:
          self.root = z
          return
     y = None
     x = self.root
     while x is not None:
          y = x
          if z.key < x.key:
               x = x.left
          else:
               x = x.right
     z.parent = y
     if z.key < y.key:
          y.left = z
     else:
          y.right = z
```

Question 4: Insert Order

Say you have the numbers 1 through 7.
What order must you insert them to get a tree with all elements strictly on the rightmost branch?
1, 2, 3, 4, 5, 6, 7

What order must you insert them to get a tree with all elements strictly on the leftmost branch?
7, 6, 5, 4, 3, 2, 1

What order must you insert them to get a strictly balanced tree?
4, 2, 6, 1, 3, 5, 7

Question 5: Height of Tree

Create a node method that determines the length of the longest path from the node to a leaf node. We will refer to this as the height of the node. To get the height of the tree, simply pass to it the root of the tree.
The height of a leaf node (no right or left child) will be 1.
Hand in a copy of your code.

In Tree class:

```
     def height(self):
          if self.root:
               return self.root.height()
          return 0
```

In Node class:

```
     def height(self):
          if self.left:
                left = self.left.height()
            else:
                left = 0
            if self.right:
                right = self.right.height()
            else:
                return left + 1
            return (left if left > right else right) + 1
```

If a tree has n nodes in it, and has a height of h, how long will the code take?
Is this a Θ, O, or Ω? You do not have to prove your answer.

Even though h will be the result, every node must be visited to obtain this result.
$\Theta(n)$

Question 6: Randomly built trees

If you have a tree of size 1023 ($2^{10} - 1$), what is the smallest height it can have?

$n = 1023$
$\log_2(n + 1) = 10$

Build a tree from the numbers 1 to 1023, such that each number just occurs once, and the order that the numbers are inserted is random.
Do this 1000 times and determine the average height of the trees that are built. Comment on how much this differs from the optimal.

Test:

```
        @staticmethod
        def avg_height_test():
            i = 1000
            sum_ = 0

            while i:
                lst = list(range(0, 1024)) #exclusive bounds
                tree = Tree()
                j = 1024
                while j:
                    j -= 1
                    tree.Insert(Node(lst.pop(random.randint(0,j))))
               sum_ += tree.height()
               i -= 1

          return sum_ / 1000.0

     #####################################

print("Average height test: " + str(Tree.avg_height_test()))
```

Output:

```
     Average height test: 22.182
```

All the test I ran fell around 22.1-22.2. While this is definitely worse than the optimal case of *h*=10, the difference is not too extreme compared to worst case (1023).