BinaryTree

February 27, 2017

In [1]:
```
class TreeNode(object):
            def __init__(self):
                self.data = None
                self.left = None
                self.right = None
            def set_left(self, left_node):
                self.left = left_node
            def get_left(self):
                return self.left
            def get_right(self):
                return self.right
            def set_right(self, right_node):
                self.right = right_node
            def set_data(self, data):
                self.data = data
            def get_data(self):
                return self.data
```

In [79]:
```
class BST(object):
             def __init__(self):
                 self.root = None
             def insert(self, node):
                 if node == None:
                     print "node passed is None"
                 node_data = node.get_data()
                 if self.root == None:
                     self.root = node
                 else:
                     curr = self.root
                     pred = None
                     while curr != None:
                         if node_data < curr.get_data():
                             pred = curr
                             curr = curr.get_left()
                         elif node_data > curr.get_data():
                             pred = curr
                             curr = curr.get_right()
```

```
            elif node_data == curr.get_data():
                node.set_left(curr.get_left())
                curr.set_left(node)
                return
        assert curr == None
        if pred.get_data() >= node_data:
            pred.set_left(node)
        else:
            pred.set_right(node)

def display_preorder(self, node):
    queue = list()
    queue.append(node)
    while len(queue) > 0:
        curr = queue[0]
        print curr.get_data()
        queue = queue[1:]
        left = curr.get_left()
        if left:
            queue.append(curr.get_left())
        right = curr.get_right()
        if right:
            queue.append(curr.get_right())

def display_postorder(self, node):
    curr = node
    visited = set()
    stack = list()
    # curr gets visited for the first time means curr gets touched
    # if curr has a left NOT in visited, then visit left
    # if curr has right NOT in visited, then visit right
    # if curr has no left and no right, then print curr and add curr to visited
    # unstack and check if left is visited, if left is not visited, then visit left
    while curr != None or len(stack) > 0:
        stack.append(curr)
        left = curr.get_left()
        right = curr.get_right()
        #Base condition: leaf node
        if ((left == None) or left in visited) and ((right == None) or right in vis
            visited.add(curr)
            print curr.get_data()
            if len(stack) > 1:
                stack.pop()
                curr = stack.pop()
                continue
            else:
                return
        if (left != None) and (left not in visited):
```

```
            curr = left
        if (right != None) and (right not in visited):
            curr = right


def display(self, node):
    curr = node
    if curr != None:
        self.display(curr.left)
        print curr.get_data()
        self.display(curr.right)
def display_non_recursive(self, node):
    stack = list()
    curr = node
    while True:
        while curr != None:
            stack.append(curr)
            curr = curr.get_left()
        if len(stack) == 0:
            return
        curr = stack.pop()
        print curr.get_data()
        curr = curr.get_right()
def display_nr2(self, node):
    stack = list()
    curr = node
    while True:
        while curr != None:
            stack.append(curr)
            curr = curr.get_left()
        if len(stack) == 0:
            return
        curr = stack.pop()
        print curr.get_data()
        curr = curr.get_right()

def get_root(self):
    return self.root

def height(self, root):
    count = 0
    if root == None:
        return count
    elif (root.get_left() != None) and (root.get_right() == None):
        return 1 + self.height(root.get_left())
    elif (root.get_left() == None) and (root.get_right() != None):
        return 1 + self.height(root.get_right())
    else:
```

```
                    return 1 + max(self.height(root.get_left()), self.height(root.get_right()))
```

In [104]:
```
nodes = [2, 1, 3, 7, 2.5]
#nodes = [1, 0, 2, 12, -3, -14, 14, 23, 21, 22, 2]
```

In [105]:
```
bst = BST()
for node_data in nodes:
    node = TreeNode()
    node.set_data(node_data)
    bst.insert(node)
```

In [106]:
```
bst.display(bst.get_root())
```

```
1
2
2.5
3
7
```

In [107]:
```
bst.display_non_recursive(bst.get_root())
```

```
1
2
2.5
3
7
```

In [108]:
```
bst.display_nr2(bst.get_root())
```

```
1
2
2.5
3
7
```

In [109]:
```
bst.display_preorder(bst.get_root())
```

```
2
1
3
2.5
7
```

In [110]:
```
bst.display_postorder(bst.get_root())
```

```
7
2.5
3
1
2
```

```
In [111]: bst.height(bst.get_root())
```

```
Out[111]: 3
```