HOME WORK № 4

CSCI B505 Applied Algorithms 09/23/2022

Instructions

1. Under any circumstances the deadline will NOT be extended.
2. Code should be written in python language in a Jupyter notebook .
3. Unless mentioned in the questions, feel free to use built-in python functions
4. Upload Jupyter notebook and html file of the Jupyter notebook in canvas. No other forms submissions is considered as valid submission. Make sure to have the output of the required cells of the jupyter notebook and its html version before making submission.
5. Plagiarism is unacceptable and we have ways to find it. So do not do it.
6. There is one problem in this assignment for 100 points.
7. The code should readable with variables named meaningfully.
8. Write test cases wherever required so that they cover all scenarios.

Problem 1

Fill in the methods of the Tree class. All the methods that change the tree should maintain the Binary Search Tree (BST) invariant.

```
Note:

  1. Diameter of a tree is the length of the longest path in the
     tree.

  2. The __contains__ "dunder" method returns True if an
     element is present in the tree and False otherwise.
     By writing the __contains__ "dunder" method you can use
     Python's ‘in‘ syntax: ‘7 in tree‘ and ‘11 not in tree‘.

  3. insert() and delete() don't return anything--they just
     update the tree. In Python, returning None corresponds to
     returning nothing.
```

4. You may assume all inputs are valid, and you don’t have to perform any error handling. In partiular, you may assume that all elements inserted into the tree are distinct.

Example 1:

```
>>> nodel = Node(5, Node(1), Node(7))
>>> noder = Node(15, Node(12), Node(17))
>>> t = BST (Node(10, nodel, noder))



        10
       /    \
     5       15
    / \     /  \
   1   7   12   17

>>> t.max()
17

>>> 15 in t
True

>>> 100 not in t
True

>>> t.height()
2

>>> t.diameter()
4

>>> t.inorder()
[1, 5, 7, 10, 12, 15, 17]

>>> t.preorder()
[10, 5, 1, 7, 15, 12, 17]

>>> t.postorder()
[1, 7, 5, 12, 17, 15, 10]
```

```
>>> t.level_order()
[10, 5, 15, 1, 7, 12, 17]



Example 2:

t2 = BST (Node (10, None, Node(15)))

    10
      \
       15

t2.insert(12)

    10
      \
       15
       /
     12

>>> t2.insert(16)
>>> t2.insert(9)
>>> t2.insert(11)

   10
  /  \
 9   15
    /  \
   12   16
  /
 11

>>> t2.level_order()
[10, 9, 15, 12, 16, 11]

>>> t2.inorder()
[9, 10, 11, 12, 15, 16]

>>> t2.preorder()
```

```
[10, 9, 15, 12, 11, 16]

>>> t2.postorder()
[9, 11, 12, 16, 15, 10]

>>> t2.diameter()
4

>>> t2.delete(10)

   11
  /  \
 9   15
    /  \
   12  16
```

Write code following the below format:

```python

class Node:

  def __init__(self, data, left=None, right=None):
    self.data = data
    self.left = left
    self.right = right


class BST:

  def __init__(self, node):
    self.root = node

  def __contains__(self, data):
    pass

  def insert(self, data):
    pass

  def delete(self, data):
    pass

  def preorder(self):
    pass

  def postorder(self):
    pass

  def inorder(self):
    pass

  def level_order(self):
    pass

  def diameter(self):
    pass

  def height(self):
    pass

  def max(self):
    pass
```