CPE 202 Fall 2018

Lab5: Binary Search Tree

Due: 10/29

Partially Implement Binary Search Tree (BST)

1. Implement the following operations for a Binary Search Tree class starting from the template provided. Use the Class TreeNode that is provided. You may implement helper methods that make your code easier to write, read, and understand. You should write test cases of your own as you develop the methods. You may use iterative or recursive functions in your implementation. (BStree.py and BStreeTests.py)

You will likely want to add setters and getters for the tree node fields other than the key field. Changing the key of a node is equivalent to removing and inserting it and that is the safer way to do the implementation rather than trying to move the node to reflect the change in the key.

To implement BST use two classes, because you must be able to create and work with a BST that is empty. The class BinarySearchTree has a reference to the class TreeNode that is the root of the BST. The class TreeNode can provide many helper functions that make implementation in class BinarySearchTree much easier. You need to include necessary getters and setters. You can add extra instance variables if you need, but make sure it will not affect our test cases.

```
class TreeNode:
    def __init__(self, key, data=None, left=None, right=None):
        self.key = key
        self.data = data
        self.left = left
        self.right = right

class BinarySearchTree:
    # Returns empty BST
    def __init__(self):
        self.root = None

    #returns True if tree is empty, else False
    def is_empty(self):
        pass

    # returns True if key is in a node of the tree, else False
    def search(self, key):
        pass

    # inserts new node w/ key and data
    def insert(self, key, data=None):
        # On insert, can assume key not already in BST
        # Example node creation: temp = TreeNode(key, data)
        pass

    # deletes node containing key – can assume the node exists
    def delete(self, key):
        # Will need to consider all cases – will likely want helper functions
        # like find_successor() and splice_out()
        pass

    # returns node with min key in the BST - can assume at least one node in BST
    def find_min(self):
        pass

    # returns node with max key in the BST - can assume at least one node in BST
    def find_max(self):
        pass

    # return Python list of BST keys representing inorder traversal of BST
    def inorder_list(self):
        pass

    # return Python list of BST keys representing preorder traversal of BST
    def preorder_list(self):
         pass

    # return the height of the tree
    def tree_height(self):
         pass
```

Submit the following file to PolyLearn

2. BStree.py
3. BStreeTests.py

Write two tests for each function. Each function must have its own Test function, such as test_inorder, test_height, etc.