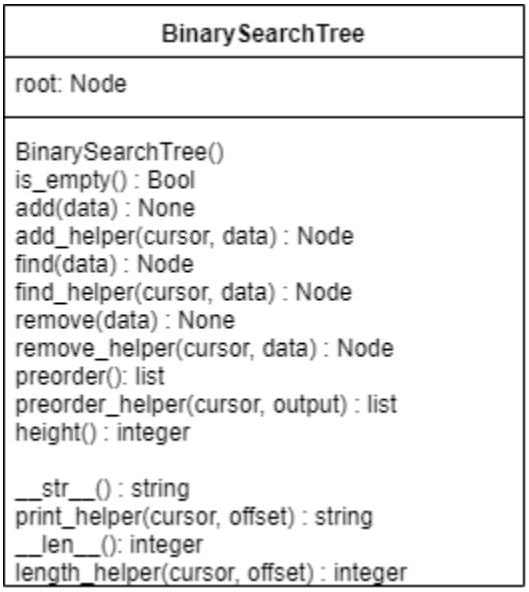
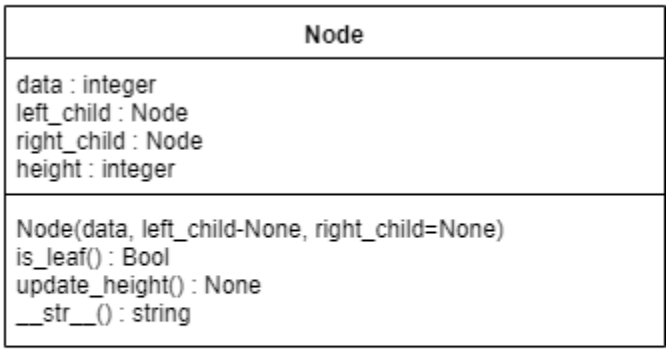
Binary Search Tree

Purpose:

This project will give you experience with Binary Search Trees. The next project builds of this one project, so be sure you document your code well and understand well how it works.

Implement the two classes defined in the following UML diagrams





Binary Search Tree ADT (binarysearchtree.py)

You will implement a BST that supports the following operations:

- is_empty: Return True if empty, False otherwise.
- __len__: Return the number of items in the tree.
- height: Return the height of the tree
- __str__: Return a string that shows the shape of the tree when printed
- add(item): Add item to its proper place in the tree
- remove(item): Remove item from the tree.
- find(item): Return the matched item. If item is not in the tree, return None.
- inorder: Return an iterator that performs an inorder traversal of the tree
- height: returns the height of the tree (height of the root node)

Recursion

The following operations must be implemented recursively:

- add
- find
- remove
- preorder
- __str__
- __len__

These functions should all use a "helper" function to do the actual recursion and work. So that the UnitTest can verify that recursion was properly used, at the beginning of every "helper" function, you must add the following line:

```
''' recursrion helper. DO NOT CALL DIRECTLY '''
RecursionCounter()
```

And at the beginning of the module you will have to add:

```
from recursioncounter import RecursionCounter # place at the beginning of the mod
ule
```

Main Program (main.py)

You must implement a main function that does the following:

1. Insert the following numbers into the tree in this order: 21,26,30,9,4,14,28,18,15,10,2,3,7
2. Print a preorder traversal of the tree
3. Print the tree
4. Remove the following data from the tree: 21,9,4,18,15,7
5. Print the tree

The output should look similar to Figure 1.

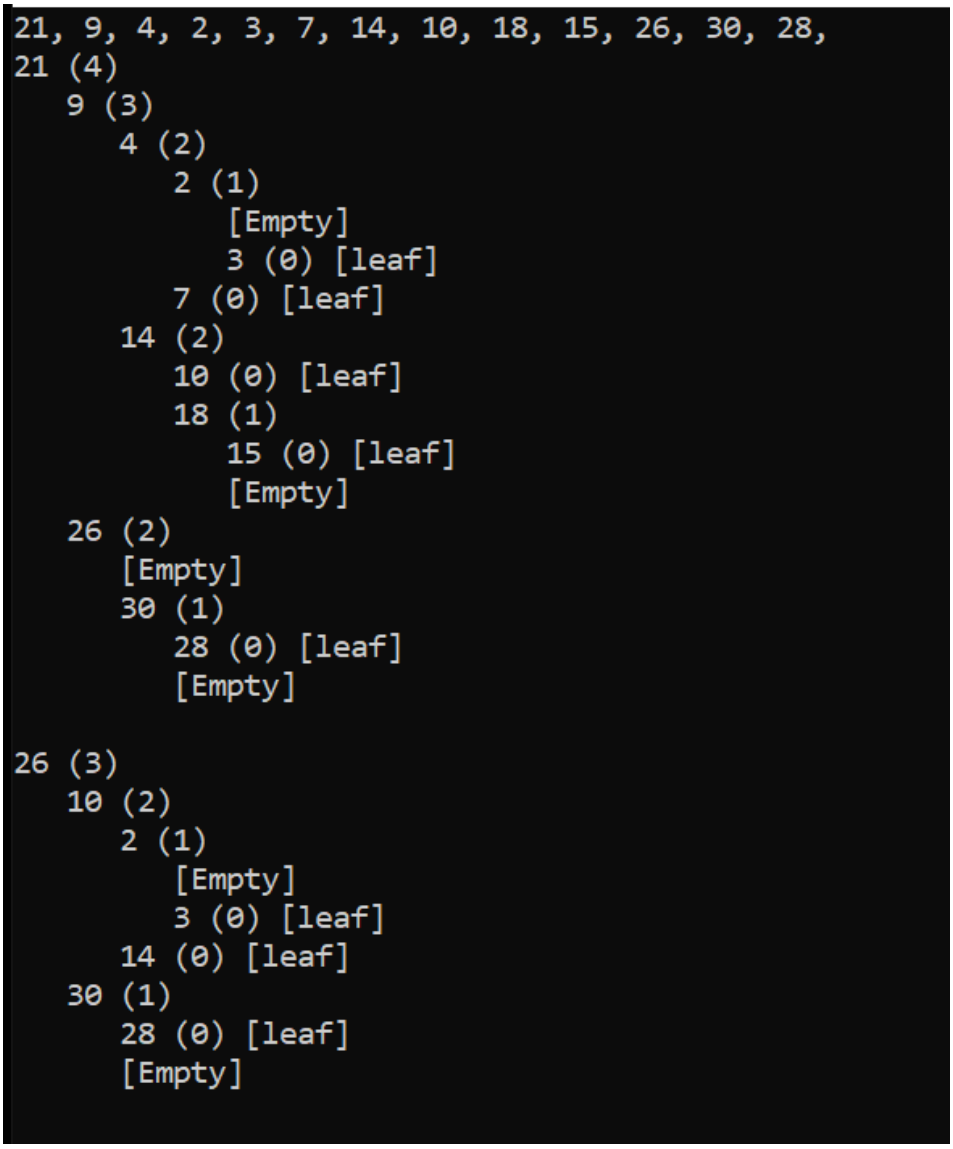


Figure 1. Example output showing preorder traversal and printing the tree.

Test Cases

Following is the set of assertion-based test cases that your program must pass, and by which your code will be graded. *You will be given the pytest unit test code to run against your code as you develop it. This allows you to learn how test are written, and to know what your score is going to be when the code is graded before you submit it*

Test Empty BST

Create an empty BST

assert size == 0 assert is_empty is True

Test Size

Create an empty tree

seed random number generator with 0

Insert 123 random integers in range 1, 400 into the tree assert size == 123

Test Height

Create an empty tree Insert value 123 into the tree

assert height == 0

Insert 12 and 2 into the tree

assert height == 1

Test Insert and Find

Create an empty tree

seed random number generator with 0

Generate a list of 123 random integers in range 1, 400

Insert each integer into the list in the order they appear in the list above

assert find(first item in list) == first item in list

assert find(401) == None

Test Remove

Create an empty tree

seed random number generator with 0

Generate a list of 10 random integers in range 1, 100

Insert each integer into the list in the order they appear in the list above

remove(first item in list)

assert find(first item in list) == None

Test Preorder Traversal

Create an empty tree

seed random number generator with 0

Generate a list of 123 random integers in range 1, 400

Insert them in the list in the order they appear in the list above

assert returned value is a list of length 123

do preorder traversal assert that the first element of the random integers is the first element of the returned list

Test String

Implement scenario given in Figure 1 and implemented in the main program, but as a test case.

Test Recursion

the following functions will be tested to verify that they use recursion to do their work

- add
- find
- remove
- preorder
- __str__
- __len__

(remember to have the line RecursionCounter() at the beginning of every recursive helper function)

Test Code Quality

PyLint will be run on main.py and binarysearchtree.py. Both must have at least an 8.5 PyLint score.

Grading (100 points)

- Passes TestNode 10
- Passed TestBinarySearchTree 35
- Passed RecursionTests 30
- passed Coding Standards 25

Turn in to Canvas:

- main.py

- binarysearchtree.py