Question:(Binary Tree)

Raju was listening to music. He creates a playlist and play the playlist on shuffle. The music tracks are been played in a shuffle list. There are totally 5 music tracks playing currently. Raju wants the music track to be played in a order after sometimes. So help Raju to get his music tracks been played in order and the playlist to be changed from shuffle to inorder traversal.

Input:

A provided nodes of the preorder shuffled playlist

Output:

An ordered playlist to be played.

Sample Testcase:

Input:

3

2

1

4

5

Output:

1 2 3 4 5

Explanation:

The music playlist has been ordered according to the user input.

Solution:

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

def inorderTraversal(self, root):

res = []

if root:

res = self.inorderTraversal(root.left)

res.append(root.data)

res = res + self.inorderTraversal(root.right)

return res

root = Node(int(input()))

root.insert(int(input()))

root.insert(int(input()))

root.insert(int(input()))

root.insert(int(input()))

print(\*root.inorderTraversal(root))

Test cases:

Test case (1) : Getting binary tree node input

Test case(2) : Complexity O(log n)

Test case(3) : Pre order node traversal input is defined

Test case(4) : Traversing through nodes

Test case(5) : Printing the nodes in a list

Test case.py:

import sys

import unittest

from bst import Node, BinarySearchTree

class BstTest(unittest.TestCase):

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(BstTest, self).\_\_init\_\_(\*args, \*\*kwargs)

self.tree = BinarySearchTree()

self.arr = [int(x) for x in input().split()]

for x in self.arr:

self.tree.iterative\_insert(x)

def test\_search(self):

self.assertTrue(self.tree.search(self.tree.get\_root(), 4))

self.assertFalse(self.tree.search(self.tree.get\_root(), 12345))

def test\_size(self):

self.assertEqual(self.tree.size(self.tree.get\_root()), 5)

def test\_depth(self):

self.assertEqual(self.tree.depth(self.tree.get\_root()), 4)

def test\_min(self):

self.assertEqual(self.tree.min(self.tree.get\_root()), 1)

def test\_max(self):

self.assertEqual(self.tree.max(self.tree.get\_root()), 3)

def test\_display(self):

print "Inorder Traversal: "

self.tree.inorder(self.tree.get\_root())

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

sys.exit(0)