CSCI-SHU 210 Data Structures - 100 Points

HOMEWORK ASSIGNMENT 7 - BINARY SEARCH TREES

PROBLEM 1 – LONGEST DISTANCE - 30 POINTS

Implement the member *diameter(self)*, which returns the maximum distance between two nodes. The distance between two nodes can be defined as the following function in which LCA is the Lowest common ancestor of *node1* and *node2*: *diameter = depth(node1) + depth(node2) −2 ∗ depth(LCA)*

Requirements

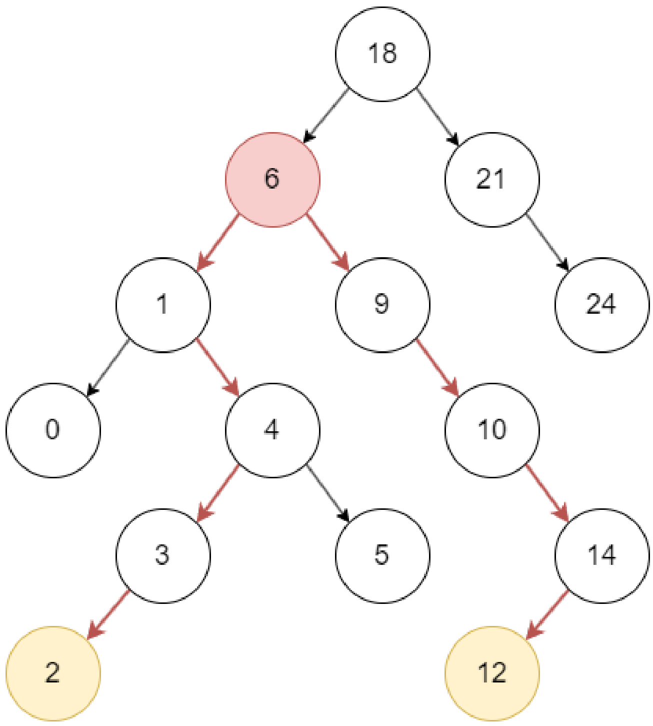
- The time complexity requirement of this method is at most **O(n)**.
- You cannot use Python lists or any other built-in data structures.
- You can not change the provided Binary Search Tree.

Important

- You can expect that each tree has only one longest path.
- You can expect that given trees have at least one node.
- You can define any helper functions.

Example

Longest distance between two nodes (diameter) in the following Binary Search Tree is 8. The nodes of the longest path in this example are the nodes with elements 2 and 12. The lowest common ancestor of these nodes is the node with element 6.



Problem 2 – Delete Range in BST - 15 Points

Implement the member function *delete_range(start, end)* in the binary search tree class. The member function deletes all nodes in the range from *start* to *end*. Your solution has to be iterative.

Requirements

- You are not allowed to alter the provided classes. **(-15 Points)**
- You are not allowed to insert new nodes **(-15 Points)**
- Your solution can not be recursive **(-15 Points)**

Example

```
bst = BinarySearchTree()
bst.insert(18)
bst.insert(6)
bst.insert(21)
bst.insert(24)
bst.insert(1)
bst.insert(9)
bst.insert(0)
bst.insert(4)
bst.insert(10)
bst.insert(14)
bst.insert(12)
bst.insert(3)
bst.insert(5)
bst.insert(2)

print("Before deletion:")
print(list(bst))  # [0, 1, 2, 3, 4, 5, 6, 9, 10, 12, 14, 18, 21, 24]

bst.delete_range(5, 10)

print("After deletion:")
print(list(bst))  # [0, 1, 2, 3, 4, 12, 14, 18, 21, 24]
```

Problem 3 – Find Sum in BST - 30 Points

Implement the recursive member function *pairs(self, sum1)*, which finds node pairs for a provided sum in a binary search tree. Return the first possible pair if a pair exists. Return *None* otherwise.

Please note that the binary search tree member functions *before* and *after* have an amortised time complexity of *O(1)*. A node's successor or predecessor usually is its direct child or parent node. Only in exceptional cases is a subtree upward/downward traversal of *O(h)* required, where *h* is the tree's height.

Requirements

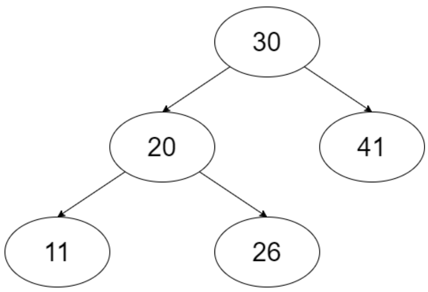
- The time complexity requirement of this method is at most $O(n)$. **(10 Points)**
- The space complexity requirement of this method is at most $O(1)$. **(10 Points)**
- You cannot use Python lists or any other built-in data structures. **(-20 Points)**
- Your function has to be recursive.

Example

$res\ = Solution().pairs(41)$
$print(res)\ \ \#\, Should\ print\ (30,\ 11)\ or\ (11,\ 30)$

$res\ = Solution().pairs(100)$
$print(res)\ \ \#\, Should\ print\ None$

$res\ = Solution().pairs(37)$
$print(res)\ \ \#\, Should\ print\ (11,\ 26)\ or\ (26,\ 11)$



Problem 4 – Same Tree - 25 Points

Implement the member function *same(self, i1, i2)*, which verifies whether two sets of keys build the same binary search tree without building a binary search tree. Return *True* if both sets describe the same tree. Return *False* otherwise.

Requirements

- The time complexity requirement of this method is at most **O(n^2)**.
- The space complexity requirement of this method is at most **O(n^2)**.
- You are not allowed to build a Tree of nodes.

Example

```
# You can find this tree on the right
i1 = [15, 25, 20, 22, 30, 18, 10, 8, 9, 12, 6]
i2 = [15, 10, 12, 8, 25, 30, 6, 20, 18, 9, 22]

res = Solution().same(i1,i2)
print(res)  # Should print true
```