Data Structures

Homework Assignment 7 - Binary Search Tree

Problem 1 – Longest Distance - 25 Points
Problem 2 – Lowest Common Ancestor - 25 Points
Problem 3 – Kth Largest Element - 25 Points
Problem 4 – Same Tree - 25 Points

Notes and Requirements

- Your submission must be your effort. You can not copy other students' code.
- This worksheet is graded on performance; Implementations must be correct.
- You are encouraged to visit our office hours to ask coding questions.
- Only the latest (most recent) submission is graded.
- Late submissions are not considered for grading.
- You can not use any third-party libraries.

Some assignments on this worksheet are manually graded.

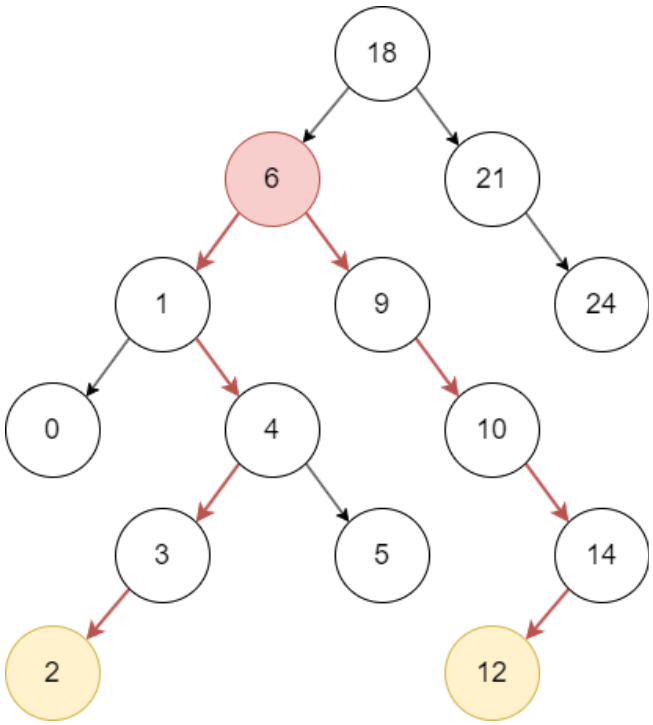
Spring 2025 - Revision 1 - *April*

Problem 1 – Longest Distance - 25 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)*

The 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.

Example



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)**.
- You cannot use Python lists or any other built-in data structures.
- You can not change the provided Binary Search Tree.

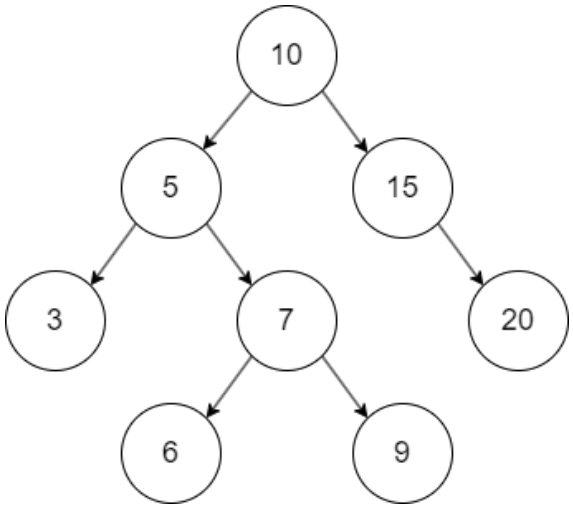
Important

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

Problem 2 – Lowest Common Ancestor - 25 Points

Implement the member `lca(self, node1, node2)`, which returns the node of the lowest common ancestor of the given nodes *node1* and *node2*.

Example




```
lca(node_6, node_9) → node_7
lca(node_6, node_7) → node_7
lca(node_3, node_20) → node_10
lca(node_3, node_15) → node_10
```

Requirements

- The time complexity requirement of this method is at most **O(n)**.
- The space complexity requirement of this method is at most **O(1)**.
- You cannot use Python lists or any other built-in data structures.
- Your function has to return a tree node, not a node's element.

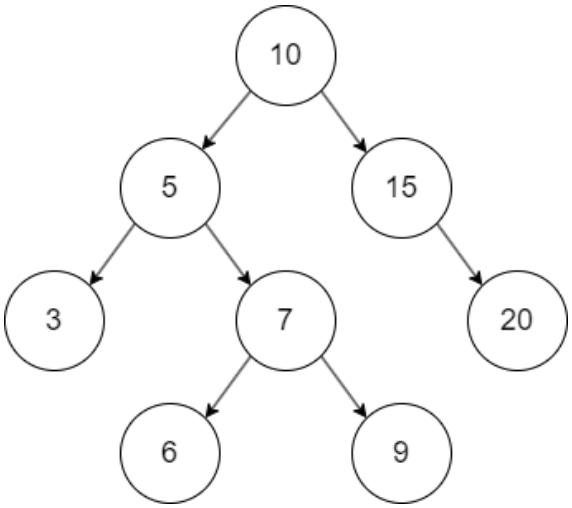
Important

- You can expect that given trees have at least one node.
- You can define any helper functions.

Problem 3 – Kth Largest Element - 25 Points

Implement the member `kth_largest(self, k)`, which returns the kth largest element in a Binary Search Tree. If k is too large, return the smallest element's node within the tree. If k is too small, return the largest element's node in the tree.

Example




```
kth_largest(1) → node_20
kth_largest(3) → node_10
```

Requirements

- The time complexity requirement of this method is **O(n log n)**.
- The space complexity requirement of this method is **O(n)**.
- Your function has to return a tree node, not a node's element.
- You cannot use Python lists or any other built-in data structures.

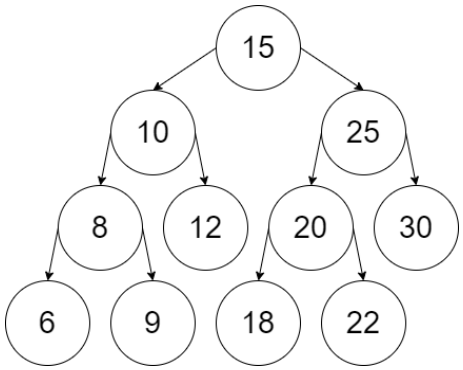
Problem 4 – Same Tree - 25 Points

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.

Example

```
# 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
```



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)**.