CSCI-SHU 210 Data Structures

Homework Assignment7 Binary Search Trees

*Assignment 7 tasks are located at line 486 in BinSearchTrees.py

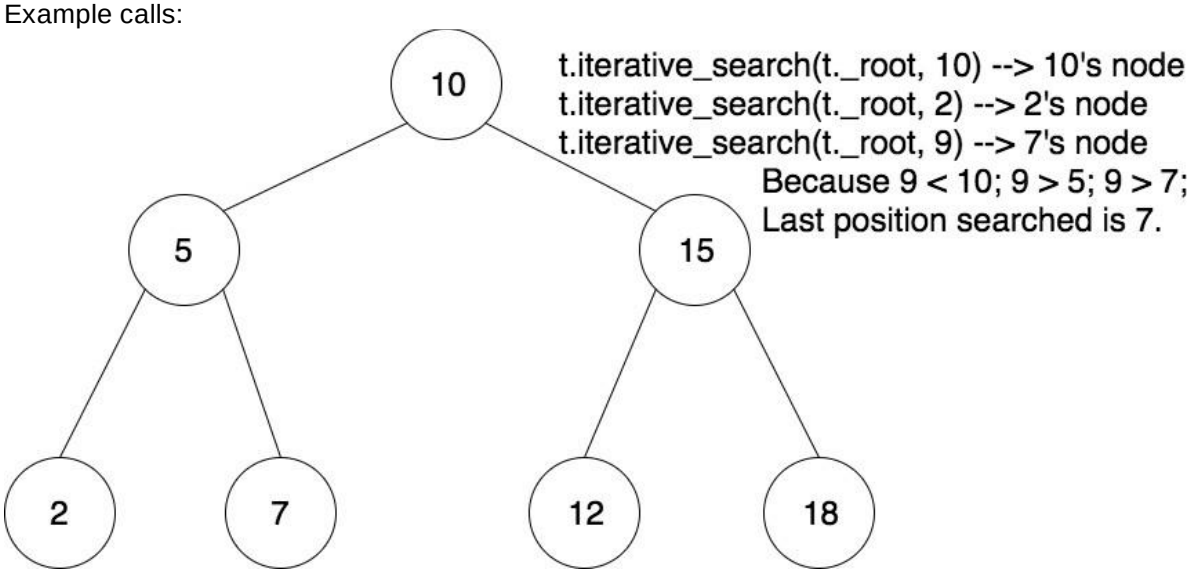
Problem 1: Iterative search in BST

In class BinarySearchTree, our search function `_subtree_search(self, node, v)` is implemented recursively.

"""Return the node having value v, or last node searched."""

Your task: Implement function `iterative_search(self, node, v)`, which performs same job as _subtree_search, but iteratively.

Example calls:



Important:

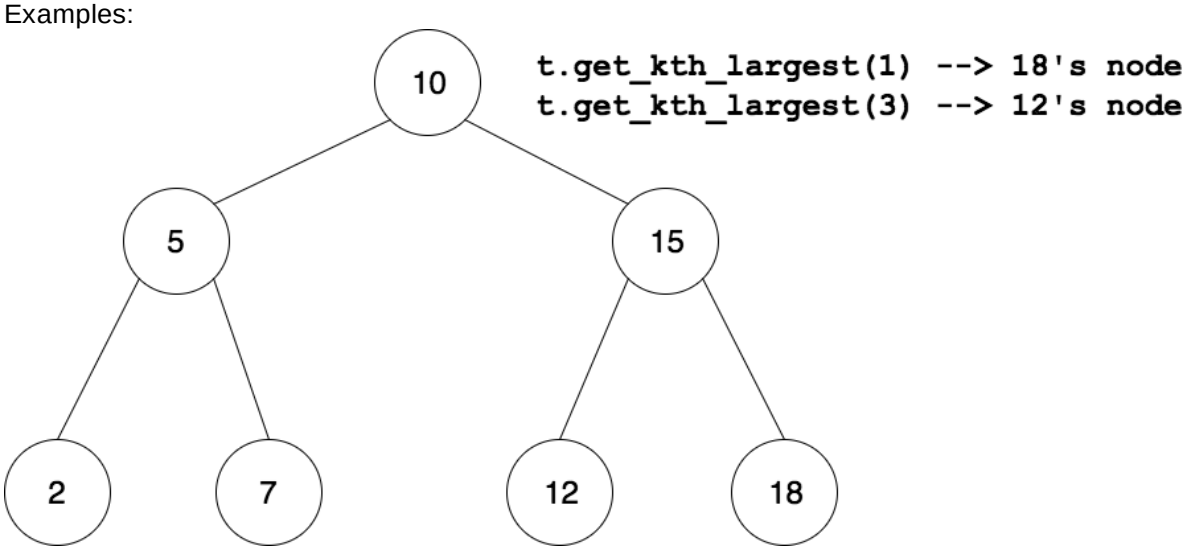
- Same job means, for the same tree, same parameters are given, `iterative_search` should return the exact same **TreeNode** as `_subtree_search.`
- Your function should return a **TreeNode**!
- You cannot use recursion.
- No other data structures allowed (in specific python lists). In other words, space complexity O(1).
- Runtime Complexity: O(h) in the worst case where h is the height of the BST
- You can reuse any method from Binary Tree, or any method from Binary Search Tree.
  - Please try to use the methods in BST in your implementation

Problem 2: Find k^{th} largest element in BST

Implement function `get_kth_largest(self, k)`, which returns the k-th largest node within self 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 within the tree.

Examples:



Important:

- Your function should return a **TreeNode**!

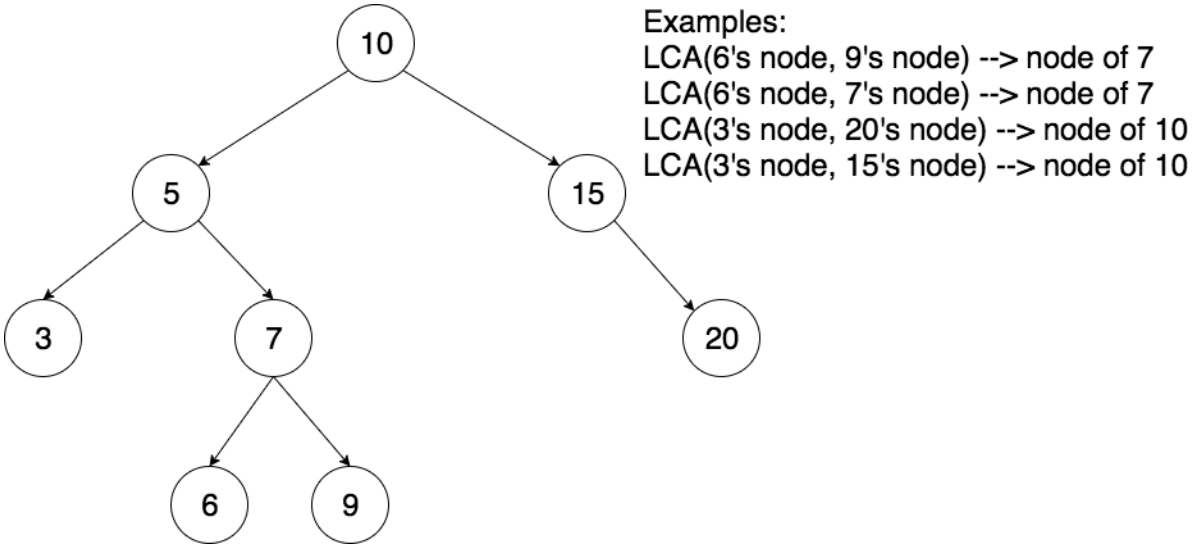
- You can reuse any method from Binary Tree, or any method from Binary Search Tree.
  - Please try to use the methods in BST in your implementation

- No other data structures allowed (in specific python lists). In other words, space complexity O(1).

Problem 3: Lowest common ancestor in BST

Your task is to solve C-8.58:

Let T be a tree with N nodes. Define the **lowest common ancestor** (LCA) between two nodes p and q as the lowest node in tree T that has both p and q as descendants (where we allow a node to be a descendant of itself). Given two nodes p and q, describe an efficient algorithm for finding the LCA of p and q.



Implement function `LCA(self, node1, node2)`. When called, it should return the **TreeNode** of the lowest common ancestor.

Important:

- Make sure your return type is **TreeNode**, so I can call **TreeNode**._element to test your code.
- No other data structures allowed (in specific python lists). In other words, space complexity O(1).
- Runtime Complexity: O(N) in the worst case
- You can reuse any method from Binary Tree, or any method from Binary Search Tree.
  - Please try to use the methods in BST in your implementation
- You can define helper functions if you need.

Problem 4: Longest Distance in BST

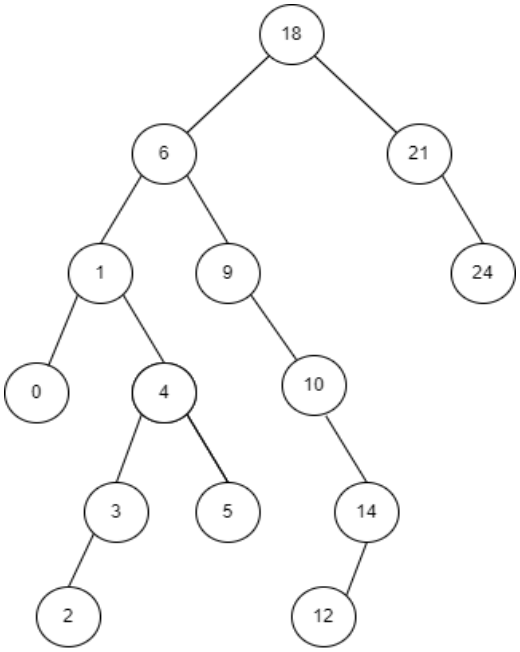
Let T be a binary search tree with N nodes. For a `node1` in T, let `depth(node1)` be its depth. The *distance* between two nodes `node1` and `node2` in T is defined as

$$\mathrm{d}(\mathrm{node1}, \mathrm{node2}) = \mathrm{depth}(\mathrm{node1}) + \mathrm{depth}(\mathrm{node2}) - 2 * \mathrm{depth}(\mathrm{LCA})$$

In where `LCA` is the lowest common ancestor of `node1` and `node2`.

The *diameter* of T is the maximum distance between two nodes in T. Implement an efficient method `diameter(self)` for finding the diameter of BST `self`. When called `self.diameter()`, it should return a non-negative integer.

Notice, your algorithm does not need to rely on the above definition.



Example:

Suppose `self` is the BST on the left.

Then the longest path in `self` is from 2's node to 12's node.

`self.diameter()` = 8

Important:

- If `self` contains root only, return 0.
  - You may assume the `self` is non-empty.
- Runtime complexity: O(N) in the worst case
- No other data structures allowed (in specific python lists).
- You can reuse any method from Binary Tree, or any method from Binary Search Tree.
  - Please try to use the methods in BST in your implementation
- Hint: draw the tree. One way to start solving it is to try post-order traversals.
- You can define helper functions if you need.