Programming and Data Structures with Python 2025

Assignment 5

12 Nov 2025, due 20 Nov 2025

Setting

Our goal is to build a (not necessarily balanced) binary search tree for a collection of integers in which all the values in the collection are actually stored at the leaves. As usual, we assume there is at most one copy of any value stored as a leaf in the tree. Values may repeat as search keys in internal nodes.

The intermediate nodes help us navigate to the leaf where a value is stored (or should be stored, in case it does not exist in the tree). Since values are at the leaves, the interpretation of an intermediate node with a value v is as follows:

- All values strictly less than v (i.e., $< v$) are in the left subtree
- All values greater than or equal to v (i.e., $\geq v$) are in the right subtree

An example is given on the next page.

Task

1. Define a class `Node` with a modifed structure, as follows:
   - Each node has four fields: `tag`, `value`, `left`, `right`
   - `tag` is a string with value `"I"` (i.e., `"Internal"`) for non-leaf nodes and `"L"` (i.e., `"Leaf"`) for leaf nodes.
   - `value` is an integer in the collection. For internal nodes, this is used to navigate left or right, as described above. For leaf nodes, this represents the actual integer in the collection.
   - As usual, `left` and `right` point to the roots of the left and right subtrees of a node. If there is no subtree, use the value `None`. A leaf will therefore have both `left` and `right` set to `None`.
   - The empty tree is denoted by a single leaf node with value `None`. In a non-empty tree, no node should have value `None`. A collection containing a single value v is represented by a single leaf node with `value` set to v.
2. Like the binary search trees discussed in class, the constructor `__init__` should allow creation of an empty tree or a leaf containing a single value.
3. The data structure should support the usual functions `find(v)`, `insert(v)` and `delete(v)`.
4. The function `__str__` should generate a string that lists *all* the nodes in the tree using inorder traversal (recursively list the left subtree, then the root, then the right subtree). Each node should be represented in the list as a pair `(tag,value)`.
5. Submit a Python file with a definition for the class `Node` that implements the functions above. You may use auxiliary functions as needed. You may also add extra fields within a node.

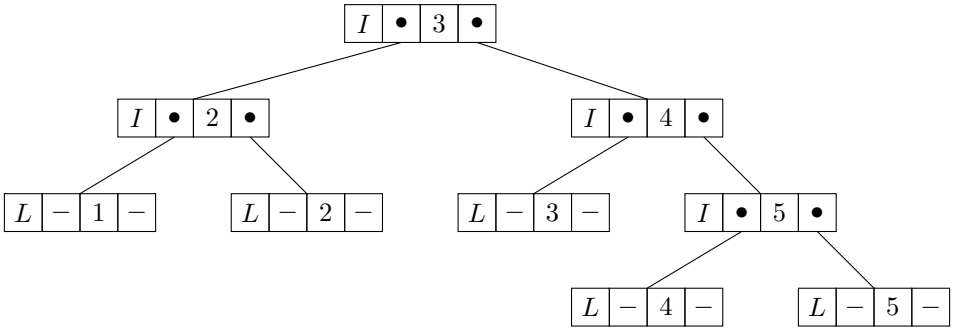
Implementation notes

- Any internal node should have two children.
  - If it has only one child which is a leaf, copy that value to the current node and convert it into a leaf node.
  - If it has only one child that is an internal node, promote that child to replace the current node.
- The values stored in intermediate nodes should be values present in the collection. You can verify that the value at any intermediate node should be the minimum value in the right subtree of the node.

- All deletions happen at the leaves. This may require promoting a node on the other branch and adjusting values at higher levels to restore the previous two invariants.

Example

Here is one possible search tree on the values `{1,2,3,4,5}`. Inside each node, we have shown the fields in the order `(tag,left,value,right)`, so that `value` comes between `left` and `right`, for better visualization.



The `__str__` function would display this tree as `[("L",1), ("I",2), ("L",2), ("I",3), ("L",3), ("I",4), ("L",4), ("I",5), ("L",5)]`.

Instructions

- Submit your solution through Moodle as a single Python notebook
- Add documentation to explain at a high level what your code is doing

---