Testing-for-Coverage Review

Venkatesh-Prasad Ranganath
Kansas State University

Problem Statement

Jane has implemented a binary search tree (BST), and she'd like your help to write testcases to test this implementation. Specifically, she wants the tests to test for correctness and achieve 100% line coverage of the implementation.

For this purpose, she has provided you with her BST implementation in impl.pyPreview the documentView in a new window.

Since Jane is not sure if her implementation is correct, she plans to modify it. So, she wants your tests to catch bugs that she may accidentally introduce into the implementation while tweaking it. However, she is fine if the tests do not achieve 100% line coverage of the implementation after she has modified it. In terms of modifications, she will not modify the name of the classes, the signature of class members whose name does not start with underscores, and the signature of __init__ methods.

Your task is to write tests to help Jane test her BST implementation.

Properties to Test*

1. If bst.insert(x) is True, then bst.search(x) should be True
   - If bst.insert(x) is True, then a following bst.search(x) should be True if there is no intervening bst.delete(x)
2. If bst.delete(x) is True, then bst.search(x) should be False
   - If bst.delete(x) is True, then a following bst.insert(x) should be False if there is no intervening bst.insert(x)
3. If x is in bst, then bst.insert(x) should be False
   - If bst.search(x) is True, then a following bst.insert(x) should be False if there is no intervening bst.delete(x)
4. If x is in bst, then bst.delete(x) should be True
   - If bst.search(x) is True, then a following bst.delete(x) should be True if there is no intervening bst.delete(x)

Properties to Test*

5. If x is not in bst, then bst.delete(x) should be False
    - If bst.search(x) is False, then following bst.delete(x) should be False if there is no intervening bst.insert(x)
6. If x is not in bst, then bst.insert(x) should be True
    - If bst.search(x) is False, then bst.insert(x) should be True if there is no intervening bst.insert(x)
7. If we insert S distinct values into a new BST bst and delete the same S distinct values, then bst.root should be None
8. Upon creation, bst.root should be None

Properties to Test*

9. If bst is not empty and x < bst.root.key, then x should be inserted into the left child tree of bst.root (implementation specific?)
10. If bst is not empty and x > bst.root.key, then x should be inserted into the right child tree of bst.root (implementation specific?)
11. If x is inserted to an empty BST, then bst.root.key should be x and both bst.root.right and bst.root.left should be None
12. If bst is not empty, then bst.root should return root

- We can test this if restated as *if bst is not empty, then bst.root should not be None.*

Properties to Test*

13. If we delete the key at node n with a left child but no right child, then n'.key should be equal to n.left.key (where n' is n after deletion) (implementation specific?)
14. If we delete the key at node n with a right child but no left child, then n'.key should be equal to n.right.key (where n' is n after deletion) (implementation specific?)
15. If starting from a new BST bst, inserting S values, inserting k (not in S), deleting S values, then bst.search(k) should be True

Properties to Test*

16. For each node n in the tree, n.left.key < n.key < n.right.key (subsumed by 17.5)
17. A BST is a BST
    1. Each node should have at most two children
    2. Each non-root node should have only one parent
    3. Root node should have no parent
    4. There should be no cycles in the tree
    5. For each node n in the tree, all keys of left descendants of n should be less than n.key and n.key should be less than all keys of right descendants of n