CSC111 Lecture 9: Introduction to Binary Search Trees

Hisbaan Noorani

February 8, 2021

Contents

1 Exercise 1: Implementing `BinarySearchTree.insert`

In this exercise, we will implement a method to insert items into a binary search tree. Here is the docstring for the `BinarySearchTree.insert` method:

```
class BinarySearchTree:
    def insert(self, item: Any) -> None:
        """Insert <item> into this tree.

        Do not change positions of any other values.

        >>> bst = BinarySearchTree(10)
        >>> bst.insert(3)
        >>> bst.insert(20)
        >>> bst._root
        10
        >>> bst._left._root
        3
        >>> bst._right._root
        20
        """
```

When we implement the `insert` method, we must ensure that we maintain the representation invariants of the `BinarySearchTree` class:

```
class BinarySearchTree:
    """Binary Search Tree class.

    Representation Invariants:
      - (self._root is None) == (self._left is None)
      - (self._root is None) == (self._right is None)
      - (BST Property) if self._root is not None, then
```

```
        all items in self._left are <= self._root, and
        all items in self._right are >= self._root
    """
```

1. First, we will implement the base case for the method.

   (a) Suppose you have an empty binary search tree. Based on the representation invariants, which of `_root`, `_left`, and `_right` are `None`?

   For an empty tree, all three are `None`.

   (b) Suppose you have a binary search tree with only a single value. Based on the representation invariants, what are the values of `_root`, `_left`, and `_right` are `None`?

   None of the three are `None`. `_root` will be the value, `_left` and `_right` will be `BinarySearchTree(None)`.

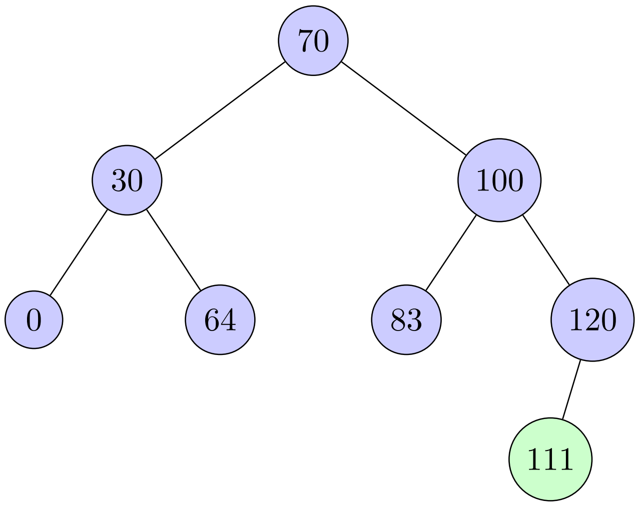
   (c) Complete the base case for `BinarySearchTree.insert`.

```
class BinarySearchTree:
    def insert(self, item: Any) -> None:
        if self.is_empty():
            self._root = item
            self._left = BinarySearchTree(None)
            self._right = BinarySearchTree(None)
        else:
            ...
```

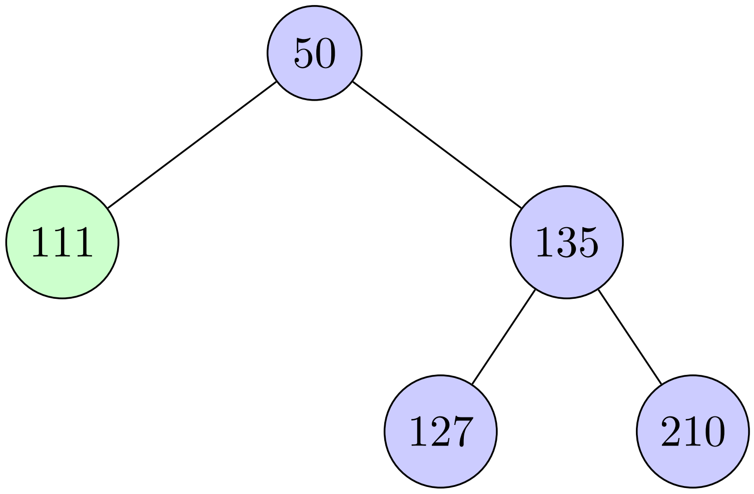
2. Before we write any code for the recursive step, we will look at some examples. In our implementation of `BinarySearchTree.insert`, we will insert a new item into the tree by creating a new *leaf* with the desired value.

   For each of the following binary search trees, insert the value `111` by adding a new leaf, ensuring that the resulting tree still satisfies the BST Property. There may be more than one correct answer!

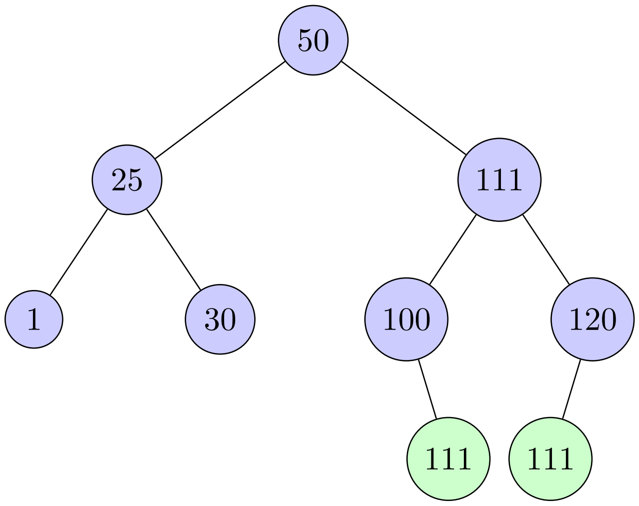
   (a) Insert to the left of `120`



(b) Insert on hte left side of `120`.



(c) Insert on the right of `100` or on the left of `120`.



3. Suppose we want to insert `111` into a non-empty binary search tree by creating a new leaf with the value `111`. We must traverse through one of the subtrees of the binary search tree until we reach an empty binary search tree (the `_left` or `_right` attribute of one of the existing leaves).

   (a) If the `_root` of the binary search tree is less than `111`, in which subtree should the new leaf be created?

   (b) If the `_root` of the binary search tree is greater than `111`, in which subtree should the new leaf be created?

   (c) If the `_root` of the binary search tree is equal to `111`, in which subtree should the new leaf be created?

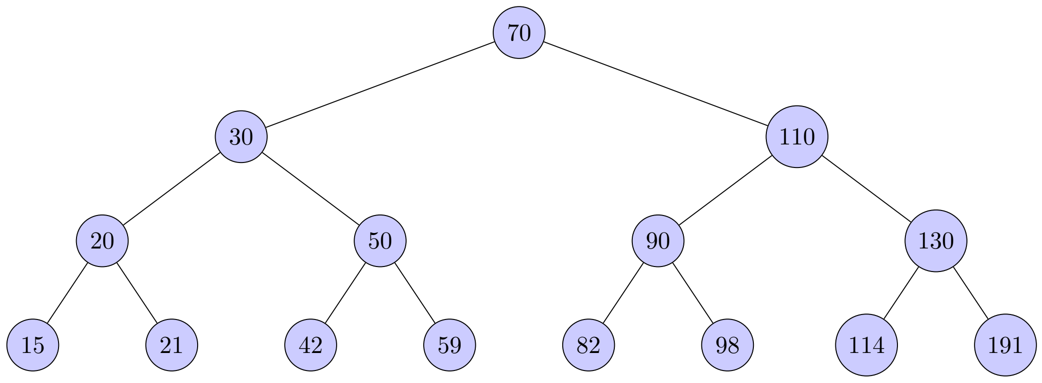
(d) Complete the recursive step for `insert` (you can use a nested if statement, or you can use an `elif` like you saw in `BinarySearchTree.__contains__`).

```python
class BinarySearchTree:
    def insert(self, item: Any) -> None:
        """...."""
        if self.is_empty():
            self._root = item
            self._left = BinarySearchTree(None)
            self._right = BinarySearchTree(None)
        elif item == self._root:
            self._left.insert(item)
        elif item < self._root:
            self._left.insert(item)
        elif item > self._root:
            self._right.insert(item)
```

2 Exercise 2: Deletion from a binary search tree (preview)

Before we write any code for BST deletion, we will look at an example to gain some intuition about how we could delete values from a binary search tree.

1. Suppose you have to delete a value from the BST below. What would be an extremely easy value to delete, without changing the rest of the tree?



   One of the leafs!

2. Suppose instead you have to delete the tree's root, 70. Ugh. One strategy is to keep most of the tree structure as is, but to find another value in the tree that can be used to replace the 70.

   (a) Could 110 replace the 70?
   No. $90 < 110$

   (b) Could 20 replace the 70?
   No. $30 > 20$

(c) Could 98 replace the 70?
No. $90 < 98$

(d) Exactly which values can replace the 70?
$59 \leq x \leq 82$