DVA245 Lab Assignment 5
Binary Search Tree

Anna Friebe

January 2022

1 Introduction

In this lab, you will add functionality to the Binary Search Tree provided with the book `https://kentdlee.github.io/CS2Plus/build/html/chap6/chap6.html#binary-search-trees`. A zip file with a code skeleton and test program is on Canvas.

2 Binary Search Tree

A Binary Search Tree (BST) is a binary tree, where all keys in the left subtree of a node are smaller than the key of the node, and all keys in the right subtree of a node are larger than the key of the node. One example of a BST is found in fig. 1.

The Binary Search Tree is represented by the `BinarySearchTree` class. It has a *nested* `__Node` class, with three data members, `val` that holds the value associated with the node, `left` that references the left subtree of the node, and `right` that references the right subtree of the node. Since this is a binary search tree, all values in the left subtree are lower than `val` and all values in the right subtree are higher than `val`. The `BinarySearchTree` class has one data member, that is the root node, from which all nodes in the tree can be reached.

You will add the following functionality to the binary search tree:

- `min(self)`, that returns the minimum value in the BST.

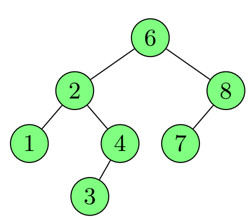

Figure 1: A Binary Search Tree example

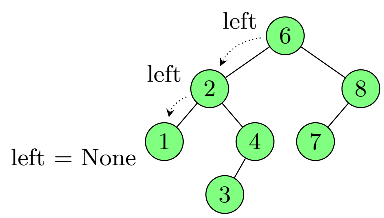

Figure 2: Illustration of finding the minimum key of the BST.

- `max(self)`, that returns the maximum value in the BST.
- `remove(self, val)`, that removes `val` from the BST.
- `__contains__(self, val)` that is called when using the operator `in`, that returns `True` if `val` is in the BST and `False` otherwise.

These shall use recursive helper functions that do the same function but for a subtree:

- `__min(root)`, that returns the minimum value in the subtree starting at root.
- `__max(root)`, that returns the maximum value in the subtree starting at root.
- `__remove(root, val)`, that returns the subtree at root with `val` removed.
- `__contains(root, val)`, that returns `True` if `val` is in the subtree starting at root and `False` otherwise.

When you have implemented these functions, the tests in `bst_test.py` should pass.

2.1 Min and max

The min and max functions are simple. For the min case, we follow the left references of the nodes in the tree, as illustrated in fig. 2. When there is no more left reference, we have found the minimum value of the tree. The max function is similar, but with the right references. Since the `__min` and `__max` functions are recursive, you need to think about what is your base case, and how to progress towards the base case if that is not reached. The `min`, `max`, `__min` and `__max` functions should be implemented in less than 5 lines each.

2.2 Remove

The most complex function is `__remove()`. It is illustrated in fig. 3 Here, we can see a couple of base cases:

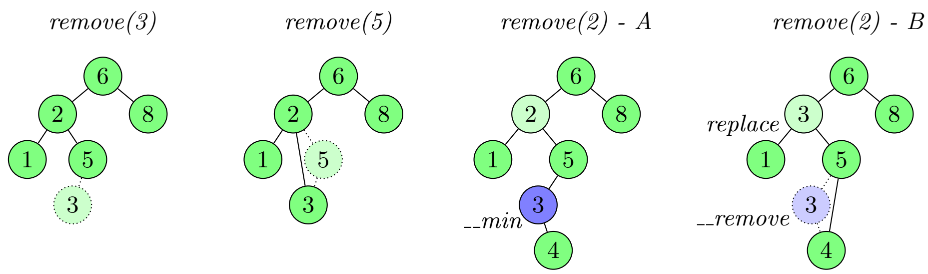

Figure 3: Illustration of removal of a leaf node (3), a node with one child (5) and a node with two children (2).

- The subtree is empty, that is the `root` argument into `__remove()` is `None`. In this case `val` does not exist in the tree, and `__remove()` can return `None`.
- The value of the `root` node of the subtree equals `val`, and has at most one child. If there is no child, `__remove()` can return `None`, the subtree turns into an empty tree from the deletion. If the left reference is None, `__remove()` can return the right reference and vice versa.

When we have not yet reached the base cases, there are two possibilities. Either we have not yet found the value to remove, or we have found it at the root of the current subtree, but this node has two children. In the first case, we expect to find `val` in the right subtree if it is larger than the value at the root, or in the left subtree if it is smaller. We can call `__remove()` for the appropriate subtree, and replace it with the returned tree. In the second case, we can copy the minimum value of the right subtree to the root where we wanted to perform the deletion. Then we can proceed to remove the minimum value from the right subtree. We know that this has at most one child, since the minimum value has no left reference. (The maximum value of the left subtree could be used in the same way.) We provide pseudocode for the `__remove(root, val)` method in algorithm 1.

2.3 Contains

The `__contains__` method is illustrated in fig. 4.

The recursive `__contains` function has two base cases.

1. If `root` is `None`, the subtree is empty. The key we are looking for is not in the subtree and we can return `False`.
2. If the value of `root` equals `val`, we have found the value and return `True`.

If we have not reached the base case, we proceed to return the value of a call to `__contains` for the left subtree if `val` is lower than the value of `root` or for the right subtree if `val` is higher than the value of `root`, as illustrated in fig. 4.

Algorithm 1 Pseudocode for the `__remove(root, val)` function that removes val from the subtree rooted at root.

```
function __REMOVE(root, val) returns the resulting subtee after removal
    inputs:
        val, the value to remove
        root the subtree root node
    local variables:
        replaceVal, the value to replace the node's value with, in the case where we want to remove it but it has two children
        newSubTree, a resulting subtree with a node removed
    if root is the empty tree then
        return the empty tree
    end if
    if val = root.GETVAL then
        if root.GETLEFT is the empty tree then
            return root.GETRIGHT
        end if
        if root.GETRIGHT is the empty tree then
            return root.GETLEFT
        end if
        replaceVal ← __MIN(root.GETRIGHT)
        root.SETVAL(replaceVal)
        newSubTree ← __REMOVE(root.GETRIGHT, replaceVal)
        root.SETRIGHT(newSubTree)
        return root
    end if
    if val > root.GETVAL then
        newSubTree ← __REMOVE(root.GETRIGHT, val)
        root.SETRIGHT(newSubTree)
    else
        newSubTree ← __REMOVE(root.GETLEFT, val)
        root.SETLEFT(newSubTree)
    end if
    return root
end function
```

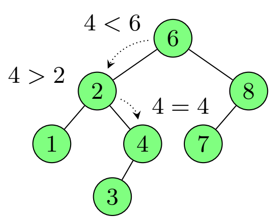

Figure 4: Illustration of finding key 4 in the BST.

Although `__iter__` can be used to find the value, you must implement the more efficient version that uses the BST structure to find the value more efficiently.

The recursive `__contains` function should be less than 10 lines of code, and the `__contains__` function can be just one line.

3 How to display

You need to run the tests in `bst_test.py`,and show that they all pass.