Mini Project: Binary Search Tree

Joshua Canlas
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, NJ
jcanlas1@stevens.edu

I. Design and Algorithm

A. Pseudocode

Algorithm 1 buildBinarySearchTree

```
Prompt user to enter a list of numbers
for each item in list do
    Insert item in the tree
end for
Return parsed input
```

Algorithm 2 insert(val: int)

```
p ⟵ root
if p is None then
    root ⟵ Node(val)
end if
while p is not None do
    if p.val < val then
        if p.right is None then
            p.right ⟵ Node(val)
            return
        end if
        p ⟵ p.right
    else
        if p.left is None then
            p.left ⟵ Node(val)
            return
        end if
        p ⟵ p.left
    end if
end while
```

Algorithm 3 printOrder

```
Create an empty list
Call inorderTraversal while passing in root of the tree and
empty list
Print the updated list
```

Algorithm 4 inorderTraversal(root: Node, list: List)

```
if root then
    inorderTraversal(root.left, list)
    Append to the list
    inorderTraversal(root.right, list)
end if
```

Algorithm 5 insert_at(i: int, val: int)

```
Get current ordered list
if i is out of index range then
    Print error message
    return
else if i ⟵ 0 then
    if val > value at i then
        Print error message
        return
    end if
else if i ⟵ last element then
    if val < value at i then
        Print error message
        return
    end if
else
    if val > value at i or val < value at i-1 then
        Print error message
        return
    end if
end if
Clear the sorted list
Insert the value in the tree
Call inorderTraversal to create new sorted list
Print updated list
```

Algorithm 6 findNode(root: Node, x: int)

```
if root is None then
    return None
end if
if root.val ⟵ x then
    return root
end if
if root.val > x then
    return findNode(root.left, x)
end if
return findNode(root.right, x)
```

Algorithm 7 findMin(root: Node)

```
while node.left is not Node do
    node = node.left
end while
return node
```

Algorithm 8 delete(x: int)

```
if x does not exist in the tree then
    Print error message and current ordered list
else
    Call recursiveDelete
    Print out updated list
end if
```

Algorithm 9 recursiveDelete(root: Node, x: int)

```
if root is None then
    return root
end if
if root.val > x then
    root.left ⟵ recursiveDelete(root.left, x)
else if root.val < x then
    root.right ⟵ recursiveDelete(root.right, x)
else
    if root.left is None then
        return root.right
    else if root.right is None then
        return root.left
    end if
    root.val ⟵ findMin(root.right).val
    root.right ⟵ recursiveDelete(root.right, root.val)
end if
return root
```

Algorithm 10 firstCommonAncestor(root: Node, x: int, y: int)

```
if root is None then
    return None
end if
node1 ⟵ findNode(self.root, x)
node2 ⟵ findNode(self.root, y)
if either nodes don't exist in the tree then
    return None
else
    if node1.val < root.val and node2.val < root.val then
        return firstCommonAncestor(root.left, node1, node2)
    else if node1.val > root.val and node2.val > root.val
then
        return firstCommonAncestor(root.right, node1,
node2)
    else
        return root
    end if
end if
```

B. API Specifications

- Node()
- BinarySearchTree()
- BinarySearchTree.buildBinarySearchTree()
- BinarySearchTree.insert(val: int)
- BinarySearchTree.printOrder()
- BinarySearchTree.inOrderTraversal(root: Node, list: List)
- BinarySearchTree.insert_at(i: int, val: int)
- BinarySearchTree.findNode(root: Node, x: int)
- BinarySearchTree.findMin(node: Node)
- BinarySearchTree.delete(x: int)
- BinarySearchTree.recursiveDelete(root: Node, x: int)
- BinarySearchTree.firstCommonAncestor(root: Node, x: int, y: int)

II. Results

A. Example Screenshots

```
Testing buildBinarySearchTree...
Please enter a list of numbers, separated by spaces: 13 12 36 24
User input = [13, 12, 36, 24]
```

Fig. 1. buildBinarySearchTree test with user input: [13, 12, 36, 24]

```
Testing inorderTraversal...
In-order traversal: [12, 13, 24, 36]
```

Fig. 2. inorderTraversal test

```
Testing insert_at...
New tree built: [12, 13, 18, 24, 36]
```

Fig. 3. insert_at test with the following inputs: (2, 18)

```
Testing delete...
Updated tree: [12, 13, 18, 36]
30 does not exist in the tree. [12, 13, 18, 36]
```

Fig. 4. delete test showing a successful and an unsuccessful deletion

```
Testing firstCommonAncestor...
The first common ancestor value: 13
```

Fig. 5. firsCommonAncestor test with inputs: (18, 12)

B. Additional Information

I am assuming that the user input will only include integers and that the input list will be separated by spaces. If the user wants to changes the values for the other tests that require input, the user will need to go into the Python script themselves and update it. The only time that the user will be prompted for an input will be the initial list creation.

I have updated my branch on GitHub (Canlas9875). The Python script is located under the BinarySearchTree folder (Canlas9875_BinarySearchTree.py). Please contact me if you have any questions or feedback.