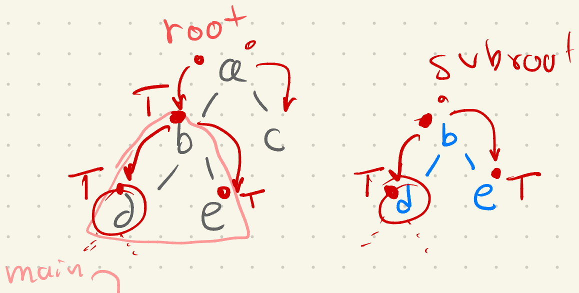
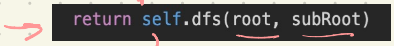
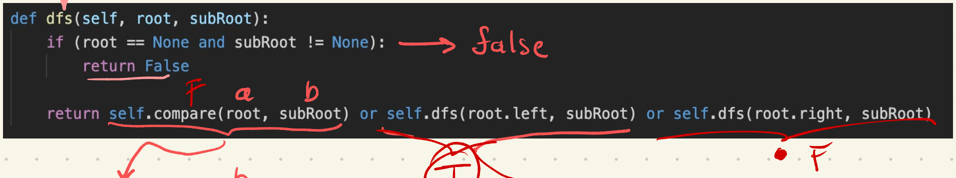


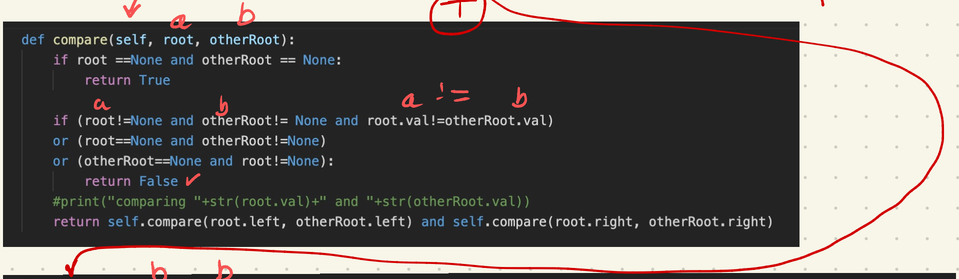


```
return self.dfs(root, subRoot)
```

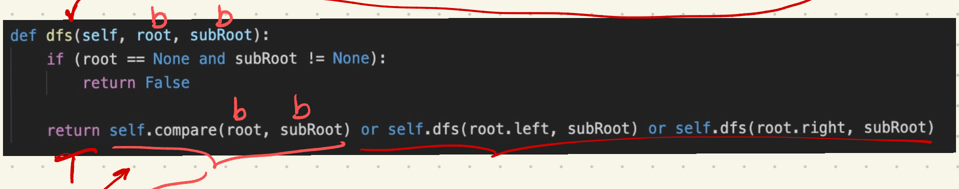
dfs (a, b)



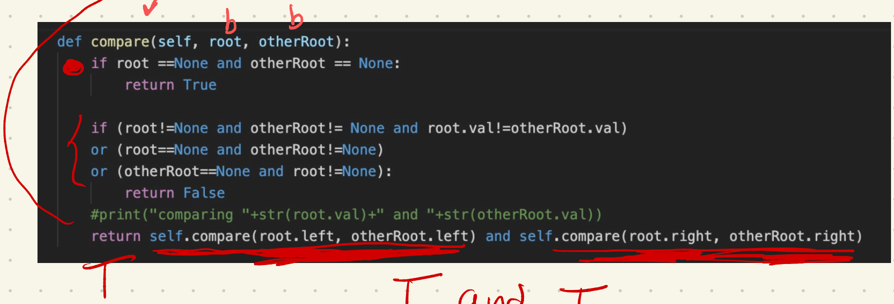
```
def dfs(self, root, subRoot):
    if (root == None and subRoot != None):
        return False

    return self.compare(root, subRoot) or self.dfs(root.left, subRoot) or self.dfs(root.right, subRoot)
```



```
def compare(self, root, otherRoot):
    if root ==None and otherRoot == None:
        return True

    if (root!=None and otherRoot!= None and root.val!=otherRoot.val)
    or (root==None and otherRoot!=None)
    or (otherRoot==None and root!=None):
        return False
    #print("comparing "+str(root.val)+" and "+str(otherRoot.val))
    return self.compare(root.left, otherRoot.left) and self.compare(root.right, otherRoot.right)
```

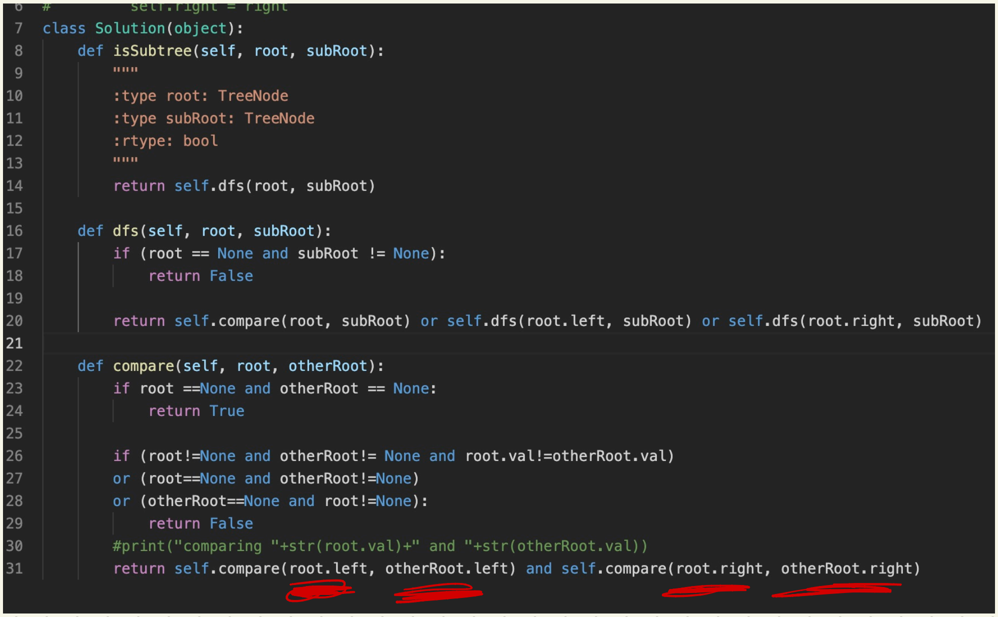


```
def dfs(self, root, subRoot):
    if (root == None and subRoot != None):
        return False

    return self.compare(root, subRoot) or self.dfs(root.left, subRoot) or self.dfs(root.right, subRoot)
```



```
def compare(self, root, otherRoot):
    if root ==None and otherRoot == None:
        return True

    if (root!=None and otherRoot!= None and root.val!=otherRoot.val)
    or (root==None and otherRoot!=None)
    or (otherRoot==None and root!=None):
        return False
    #print("comparing "+str(root.val)+" and "+str(otherRoot.val))
    return self.compare(root.left, otherRoot.left) and self.compare(root.right, otherRoot.right)
```

T and T



```python
#         self.right = right
class Solution(object):
    def isSubtree(self, root, subRoot):
        """
        :type root: TreeNode
        :type subRoot: TreeNode
        :rtype: bool
        """
        return self.dfs(root, subRoot)

    def dfs(self, root, subRoot):
        if (root == None and subRoot != None):
            return False

        return self.compare(root, subRoot) or self.dfs(root.left, subRoot) or self.dfs(root.right, subRoot)

    def compare(self, root, otherRoot):
        if root ==None and otherRoot == None:
            return True

        if (root!=None and otherRoot!= None and root.val!=otherRoot.val)
        or (root==None and otherRoot!=None)
        or (otherRoot==None and root!=None):
            return False
        #print("comparing "+str(root.val)+" and "+str(otherRoot.val))
        return self.compare(root.left, otherRoot.left) and self.compare(root.right, otherRoot.right)
```