DATA STRUCTURE AND ALGORITHMS

LAB 8

LUZIA MANUEL - 2021332

: > Users > Luzia Xavier Manuel > Downloads > final.py > ...

```python
class TreeNode:
    def __init__(self,key,val,left=None,right=None,parent=None):
        self.key = key
        self.payload = val
        self.leftChild = left
        self.rightChild = right
        self.parent = parent
```

CLASS TO STORE A TREE NODE

I CREATED THE CONSTRUCTOR

AFTER CREATED :

KEY

VALUE STORED IN KEY

SON ON THE LEFT

SON ON THE RIGHT

NODE DAD

```python
def hasLeftChild(self):
    return self.leftChild

def hasRightChild(self):
    return self.rightChild

def isLeftChild(self):
    return self.parent and self.parent.leftChild == self

def isRightChild(self):
    return self.parent and self.parent.rightChild == self

def isRoot(self):
    return not self.parent

def isLeaf(self):
    return not (self.rightChild or self.leftChild)
```

CHECKS IF IT HAS CHILDREN ON THE LEFT AND RIGHT
IF IT RETURNS NONE, THERE IS NO CHILD ON THE LEFT OR RIGHT

CHECKS IF THE NODE IS A CHILD TO SOMEONE'S LEFT AND SOMEONE'S RIGHT
IT HAS TO HAVE ONE IN THE PARENT AND BE A CHILD ON THE LEFT OF THIS PARENT NODE AND ON THE RIGHT

CHECK IF NODE IS ROOT
ROOT CANNOT HAVE PARENT

CHECK IF IT IS LEAF
LEAF HAS NO CHILDREN ON THE LEFT OR RIGHT

```
27      def hasAnyChildren(self):
28          return self.rightChild or self.leftChild
29
30      def hasBothChildren(self):
31          return self.rightChild and self.leftChild
32
33      def replaceNodeData(self,key,value,lc,rc):
34          self.key = key
35          self.payload = value
36          self.leftChild = lc
37          self.rightChild = rc
38          if self.hasLeftChild():
39              self.leftChild.parent = self
40          if self.hasRightChild():
41              self.rightChild.parent = self
```

CHECK IF YOU HAVE ANY CHILDREN
JUST HAVE A CHILD ON THE LEFT OR RIGHT
CHECK IF THEY BOTH HAVE CHILDREN
MUST HAVE CHILDREN ON THE LEFT AND THE RIGHT

UPDATING THE DATA OF THE NODE
NEW KEY
NEW VALUE
NEW SON ON THE LEFT
NEW SON ON THE RIGHT

```
43  class BinarySearchTree:
44      def __init__(self):
45          self.root = None
46          self.size = 0
47
48      def length(self):
49          return self.size
50
```

IMPLEMENTING THE BINARY SEARCH TREE CLASS
CREATING THE CONSTRUCTOR
CREATE EMPTY ROOT AND RETURN THE NUMBER OF NODES
IN THE TREE

```
51      def insert(self,key,val):
52          if self.root:
53              self._insert(key,val,self.root)
54          else:
55              self.root = TreeNode(key,val)
56          self.size = self.size + 1
57
58      def _insert(self,key,val,currentNode):
59          if key < currentNode.key:
60              if currentNode.hasLeftChild():
61                      self._insert(key,val,currentNode.leftChild)
62              else:
63                      currentNode.leftChild = TreeNode(key,val,parent=currentNode)
64          else:
65              if currentNode.hasRightChild():
66                      self._insert(key,val,currentNode.rightChild)
67              else:
68                      currentNode.rightChild = TreeNode(key,val,parent=currentNode)
```

```
70      def search(self,key):
71          if self.root:
72              res = self._search(key,self.root)
73              if res:
74                  return res.payload
75              else:
76                  return None
77          else:
78              return None
79
80      def _search(self,key,currentNode):
81          if not currentNode:
82              return None
83          elif currentNode.key == key:
84              return currentNode
85          elif key < currentNode.key:
86              return self._search(key,currentNode.leftChild)
87          else:
88              return self._search(key,currentNode.rightChild)
```

THIS METHOD WILL CHECK IF THE TREE ALREADY HAS A ROOT
IF NOT, A NEW NODE WILL BE CREATED AND IT WILL BE THE ROOT OF THE TREE
IF THE ROOT ALREADY EXISTS THEN THE METHOD CALLS THE INSERT FUNCTION TO
LOOK FOR THE RIGHT LOCATION OF THE ELEMENT IN THE TREE, RECURSIVELY

SEARCH FOR ELEMENT WITH KEY
CHECK IF THE TREE HAS A ROOT
CALL RECURSIVE SEARCH HELPER FUNCTION
IF IT RETURNS AN ELEMENT OTHER THAN NONE,
BECAUSE IT FOUND ITS ELEMENT, OTHERWISE IT RETURNS THE VALUE
IF CURRENT NODE DOES NOT EXIST, THERE IS NO ELEMENT
IF ELEMENT KEY EQUALS SEARCH KEY, FOUND
IF THE KEY IS SMALLER THAN THE NODE, SEARCH IN THE LEFT TREE, OTHERWISE SEARCH IN THE RIGHT TREE


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Python
Luzia Xavier Manuel@LAPTOP-BUHJMS9V MINGW64 ~/Downloads/Lab8_LuziaManuel_2021332
$ "C:/Users/Luzia Xavier Manuel/AppData/Local/Programs/Python/Python310/python.exe" "c:/Users/Luzia Xavier Manuel/Downloads/final.py"
The value is : None
The value is : 123.09
Tree root key: 43
Tree root value: 9.12
traversal in order:
None
The value is 13 is : 12.01
The value is : None
traversal in order:
None

THE OUTPUTS