Laboratory practice No. 4: 3D representation of space in data structures (tree and/or hash table).

Juan David Echeverri Villada
Universidad Eafit
Medellín, Colombia
jdecheverv@eafit.edu.co

Octavio Vásquez Zapata
Universidad Eafit
Medellín, Colombia
ovasquezz@eafit.edu.co

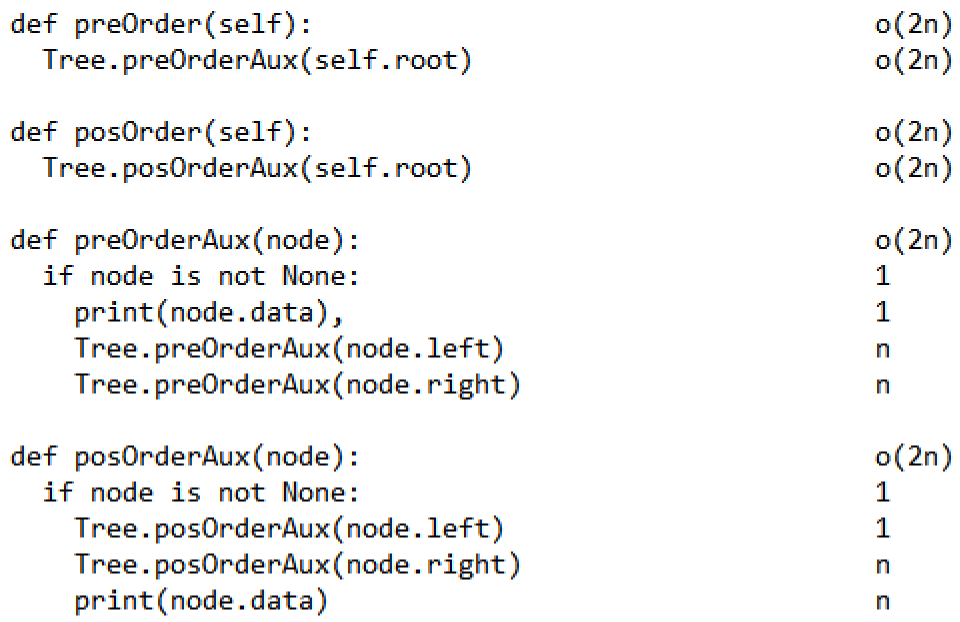
3) Practice for final project defense presentation

3.1 To calculate the collisions between bees we implemented the data structure called octree, which is a tree where each node has between 0 to 8 child nodes. We choose this data structure because with it is possible to represent a tridimensional space as a cube, and in each recursive call of the tree we can divide this cube into smaller cubes, allowing us to encapsulate these bees into these little cubes and letting us know if two or more bees share the same space, indicating a collision. Calculating the collision is not a problem, because we have all the collisions added into an array, and we just have to walk through it, making its worst-case complexity O(n), but due to the fact that we need to build the octree to add the collisions into the array that we need to walk through, we also need its complexity. The complexity of building the octree is (x*y*z), being x, y and z the space dimensions that we will be using in our octree. But since x y z are the attributes of our bee, and our complexity is in terms of bees, the complexity of building the octree is finally O(n), being n the quantity of bees. In summary, building the octree O(n) plus the complexity of walking through the array of collisions O(n) makes our final complexity O(2n), and simplifying, just O(n).

3.4 O(8 + 4n + n^2)
O(4n + n^2)
O(n^2)

```
class Main:
  def main ():                                                  o(8 + 4n + n^2)
    test = [50,30,24,5,28,45,98,52,60]                          o(1)
    arbolitou = Tree()                                          o(1)
    arbolitou.buildingTree(test)                                o(n^2)
    print("Para el recorrido en preorden de un arbol")          o(1)
    arbolitou.preOrder()                                        o(2n)
    print("\nEl PosOrder es: ")                                 o(1)
    arbolitou.posOrder()                                        o(2n)

class Node:
  def __init__(self, data , left = None, right = None):
    self.left = None
    self.right = None
    self.data = data

class Tree:
  def __init__(self, root = None):
    self.root = root

  def buildingTree (self, preOrder):              o(n^2)
    self.root=Node(preOrder[0])                   1
    for i in range (1, len(preOrder)):            n
      self.insert(self.root, preOrder[i])         n

  def insert (self, node, data):                           o(n)
    if data < node.data and node.left is None:            1
      node.left = Node(data)

    elif data > node.data and node.right is None:         1
      node.right = Node(data)

    elif data < node.data and node.left is not None:      n
      self.insert(node.left, data)

    elif data > node.data and node.right is not None:     n
      self.insert(node.right, data)
```

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473



```
def preOrder(self):                                       o(2n)
  Tree.preOrderAux(self.root)                             o(2n)

def posOrder(self):                                       o(2n)
  Tree.posOrderAux(self.root)                             o(2n)

def preOrderAux(node):                                    o(2n)
  if node is not None:                                    1
    print(node.data),                                     1
    Tree.preOrderAux(node.left)                           n
    Tree.preOrderAux(node.right)                          n

def posOrderAux(node):                                    o(2n)
  if node is not None:                                    1
    Tree.posOrderAux(node.left)                           1
    Tree.posOrderAux(node.right)                          n
    print(node.data)                                      n
```

3.5 In the complexity of the of the previous exercise there is no variable m, and the variable n is the number of elements in the given array that will allow us to build the tree

4) Practice for midterms

4.1 B, D
4.2 C
4.3 false, suma == 0, (a.left, suma-node.data), (a.right, suma-node.data)
4.9 A
4.13 raiz.id, D

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473