Assignment 3a - Symbolic Regression

Joel Doumit
College of Computer Science
University of Idaho
Moscow, ID 83843
Email: doum6708@vandals.uidaho.edu

1. Output

```
Proof of Full tree generation:
min(-(min(-10, -4), *(3, -5)), /(min(3, x), min(2, -4)))
=====
-0.25

=+=+=+=+=+=+=+=
Proof of Grow tree generation:
+(-1, -(min(x, 0), min(4, -2)))
=====
1

=+=+=+=+=+=+=+=
proof of filled population:

-(min(*(-3, x), *(-10, 1)), *(/(4, 5), min(1, -4)))
-(+(-(x, -4), /(10, 0)), min(*(3, -2), *(-3, -10)))
-(+(*(1, 2), +(x, 2)), /(*(-2, 10), min(x, -10)))
Proof of copy:
======
+(-(+(2, 5), min(-3, -2)), min(-(4, -3), /(4, 5)))
--------
+(-(+(2, 5), min(-3, -2)), min(-(4, -3), /(4, 5)))
--------
Proof of delete:
100
99
```

2. Code

```
#!/Library/Frameworks/Python.framework/Versions/3.4/bin/python3
# Joel Doumit
# CS472 - Evolutionary Algorithms
# Assignment 3a - Symbolic Regression
# node structure code borrowed from
# https://www.geeksforgeeks.org/evaluation-of-expression-tree/

import numpy as np
import random
```

```
operators = ["+", "-", "*", "/", "min"]
terminals = ["x", -10, -5, -4, -3, -2, 0, -1, 1, 2, 3, 4, 5, 10]

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

def evaluateTree(root, x):

    # empty tree
    if root is None:
        return 0

    if root.data == "x":
        return x

    # leaf node
    if root.left is None and root.right is None:
        return int(root.data)

    # evaluate left tree
    left_sum = evaluateTree(root.left, x)

    # evaluate right tree
    right_sum = evaluateTree(root.right, x)

    if root.data == '+':
        return left_sum + right_sum

    elif root.data == '-':
        return left_sum - right_sum

    elif root.data == '*':
        return left_sum * right_sum

    elif root.data == 'min':
        if left_sum > right_sum:
            return right_sum
        else:
            return left_sum

    else:
        if(right_sum == 0):
            return 0
        else:
            return left_sum / right_sum

def printTree(root):
    if root is None:
        print("Nothing")
    else:
        print(root.data, end='')
        if root.left is None and root.right is None:
            print("", end='')
        else:
            print("(", end='')
            printTree(root.left)
            print(", ", end='')
            printTree(root.right)
            print(")", end='')

def nodeVal(operChanceLow, operChanceHigh):
    if (np.random.choice([0,1], p=[operChanceLow, operChanceHigh])):
        return random.choice(operators)
    else:
        return random.choice(terminals)


def generateTerminal():
    return random.choice(terminals)

def generateFull(depth=1):
    if depth < 4:
        root = node(nodeVal(0, 1))
        root.left = generateFull(depth+1)
        root.right = generateFull(depth+1)
        return root
    else:
        root = node(nodeVal(1,0))
        return root

def generateGrow(depth=1):
    if depth < 4:
        root = node(nodeVal(0.1, 0.9))
        if root.data in operators:
            root.left = generateGrow(depth+1)
            root.right = generateGrow(depth+1)
        return root
    else:
        root = node(nodeVal(1,0))
        return root

def Copy(source):
    if(source):
        new = node(source.data)
        new.data = source.data
        if source.left:
            new.left = (Copy(source.left))
        if source.right:
            new.right = (Copy(source.right))
    return new

def Delete(population, index):
    del population[index]

if __name__=='__main__':

    tree1 = generateFull()
    print("Proof of Full tree generation:")
    printTree(tree1)
    print("")
    print("=====")
    print(evaluateTree(tree1, 1))
```