PRÁCTICA 8

ÁRBOLES PARTE I

## Alumno: Robles Martínez Héctor

## Número de Cuenta UNAM: 317356548

## Semestre: Tercer semestre de Ingeniería en Computación (2016)

## Semestre Actual: 2021-1

## Profesor: Jorge A. Solano G.

Objetivo:

Conocer e identificar las características de la estructura no lineal árbol.

Instrucciones:

• Implementar en lenguaje Python el recorrido de un árbol A en orden infijo, prefijo

y sufijo.

• Obtener la complejidad algorítmica de los algoritmos de recorrido en orden infijo,

prefijo y sufijo.

• Crear las gráficas de la complejidad que tienen los algoritmos de recorrido en orden

infijo, prefijo y sufijo dentro de un árbol para el mejor caso, el peor caso y el caso

promedio.

Resultados obtenidos:

class node():

name = ""

dad = ""

son = None

def \_\_init\_\_(self, name, dad = '\*', son = None):

self.name = name

self.dad = dad

self.son = str(son)

def \_\_str\_\_(self):

string = "|\t" + self.name + "\t|\t" + self.dad + "\t|\t" + self.son + "\t|"

return string

class operation():

a = None

b = None

c = None

root = None

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

self.c = c

self.root = self

def \_\_str\_\_(self, mode = 1):

string = self.a.name + self.c.name + self.b.name

return string

def string(self, mode = 1):

if mode == 1:

string = self.a.name + self.b.name + self.c.name

elif mode == 2:

string = self.a.name + self.c.name + self.b.name

elif mode == 3:

string = self.c.name + self.a.name + self.b.name

return string

class graph():

grade = []

nodes = [[None], [None]]

flag = True

def \_\_init\_\_(self, grade = 0, nodes = [[], []]):

self.nodes = nodes

self.grade = grade

def insert(self, op):

self.nodes[0].append(op)

size = len(self.nodes[0])

if size == 1:

self.nodes[1].append("")

else:

self.nodes[1].append(self.nodes[0][(size // 2) - 1])

def string(self, mode = 1, count = 0):

string = ""

for i in range(len(self.nodes[0])):

count += 1

pos = 0

ps = 0

node = self.nodes[0][i]

string += node.string() + " hijos: "

if node in self.nodes[1][pos:]:

ps = self.nodes[1].index(node)

pos += ps

string += (self.nodes[0][ps]).string(mode)

try:

string += (self.nodes[0][ps+1]).string(mode)

except:

pass

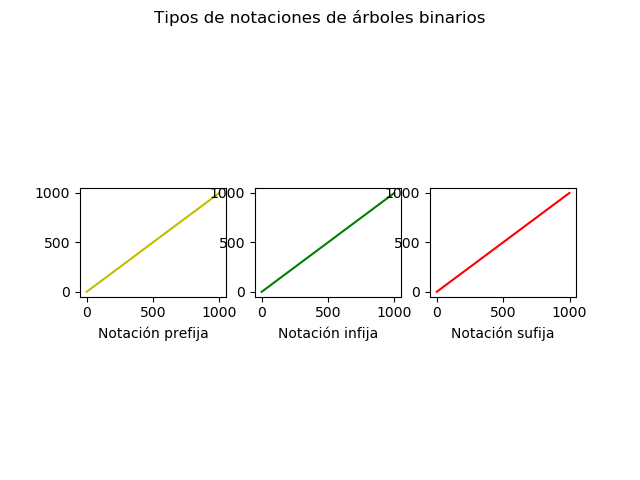
pos += 1

else:

pos = -1

string += "\n"

return string, count



Conclusiones:

Los árboles binarios son muy eficientes y sencillos de implementar.