



Alumno: Rivera Navarrete Miguel Ángel amado.

Profesor: Blancas Ruiz José Luis.

Ciclo escolar: 19-2.

Carrera: Ing. Sistemas Computacionales.

Tema: Practica 4.

**Grafico.**

**#Diccionario donde se hará el recorrido y que nodos son los padres y hacia donde ira.**

conexion = {}

conexion["S"] = {"A", "D"}

conexion["A"] = {"B", "D"}

conexion["B"] = {"E", "C"}

conexion["C"] = {"B"}

conexion["D"] = {"A", "E"}

conexion["E"] = {"D", "F"}

conexion["F"] = {"G"}

**Nodo.**

class Nodo: ‘’’aquí se declara la clase nodo’’’

'''

Esta el nodo en la búsqueda de árbol/gráfico, para saber a dónde es va a viajar

'''

def \_\_init\_\_(self, estado):

"""

Constructor , para saber qué es lo que llevaran las variables,

Se crean variable boleanas para decir si son ciertas o falsas, una variable que va estar en nulo.

"""

self.estado = estado

self.profundidad = 0

self.hijos = []

self.limite = True

self.padre = None

def agregaHijos(self, nodoHijo):

"""

Este método agrega un nodo debajo de otro nodo, aquí empieza la estructuración del árbol y el recorrido.

"""

self.hijos.append(nodoHijo)

nodoHijo.padre = self

nodoHijo.profundidad = self.profundidad + 1 ‘’’se le pone más uno para que valla recorriendo los nodos y así poder decir cuales ya estuvo’’’

def imprimeArbol(self):

"""

Este método imprime un árbol, de cómo va la estructura y cómo va el recorrido.

"""

print(self.profundidad , " - " , self.estado.lugar)

for hijo in self.hijos: ‘’’el for se ocupa para decir las posiciones en las cuales se encuentran el nodo o el apuntador’’’

hijo.imprimeArbol()

**Estado.**

from grafico2 import conexion

from collections import OrderedDict

lista\_de\_listas = []

print("Valor inicial lista de listas : ",lista\_de\_listas)

print("Variable defined outside of method automatically global? "

+ str("lista\_de\_listas" in globals()))

class Estado:

'''

Esta clase obtiene la información del estado para nuestra aplicación de búsqueda.

'''

def \_\_init\_\_(self, lugar = None):

if lugar == None:

#create initial state

self.lugar = self.obtenEstadoInicial()

else:

self.lugar = lugar

def obtenEstadoInicial(self):

"""

Este método retorna el directorio actual.

"""

#estadoInicial = "Parada Autobus"

estadoInicial = "S"

#estadoInicial = os.path.dirname(os.path.realpath(\_\_file\_\_))

return estadoInicial

def funcionSucesora(self):

"""

Esta es la función sucesora. Y genera todas las rutas

rutas posibles que pueden ser alcanzadas de la ruta actual

"""

global lista\_de\_listas

print("NODO SUCESOR ::::> ",conexion[self.lugar])

dictemp = {}

dictemp = conexion[self.lugar]

stack1 = []

print("LISTA DE LISTAS %%%%%%%%",lista\_de\_listas)

if conexion[self.lugar] in lista\_de\_listas: ''' Visita el estado visitado '''

print("ESTADO REPETIDO \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

stack1 = list(dictemp)

stack1.pop() ''' saco el valor de la lista '''

else:

stack1 = list(dictemp)

stack1.sort() ''' sort ordena una lista'''

lista\_de\_listas.append(conexion[self.lugar])

dictofWords2 = OrderedDict() ''' ordenamiento de diccionario'''

for i in range(0,len(stack1)):

dictofWords2[stack1[i]] = i+1 '''asigna el diccionario'''

#print(dictofWords2.keys())

return dictofWords2.keys() ''' ordenamiento de diccionario'''

def checaMeta(self):

"""

Este método checa si en la ruta esta el estado meta.

"""

return self.lugar == "G"

**DFsf.**

from Nodod import Nodo

from Estadod import Estado

def DFS():

"""

This function performs DFS search using a stack

"""

stack = []

'''Primeras entrdas primeras salidas VEPS / append meter elementos y pop sacar los elementos'''

visited = []

#crear el nodo raiz y agregar al stack, los valores donde estarán posicionados

estadoInicial = Estado()

raiz = Nodo(estadoInicial)

print("-- Nodo Raiz --", raiz.estado.lugar)

print("Atributos Raiz ", raiz.estado.\_\_dict\_\_,raiz.\_\_dict\_\_)

stack.append(raiz)

for x in stack:

print('Valor del Stack ', x.estado.\_\_dict\_\_)

cont = 0

# checar si hay algo en el stack a expandir

while len(stack) > 0:

#obtener el nodo siguiente

nodoActual = stack.pop()

'''saco ele elemento y lo pongo en la nueva lista'''

if nodoActual.estado.lugar not in visited:

'''implementacion de la lista visitados'''

visited.append(nodoActual.estado.lugar)

'''alimento a visited''' '''implementacion de la lista visitados'''

print("-- pop --", nodoActual.estado.lugar)

'''como agrego el valor d ela lista'''

if len(visited) == 0:

print('Valor del Visited++++++++ CERO')

for x in visited:

print('Valor del Visited\*\*\*\*\*\*\*\* ', visited)

'''imprime el valor visitado, del recorrido'''

#checar si estamos en el estado meta

if nodoActual.estado.checaMeta():

print("Meta Alcanzada")

break

#obtener los nodos hijos del nodo actual que estamos revisando

childStates = nodoActual.estado.funcionSucesora()

print('Child States ????????? ',childStates)

for childState in childStates:

childNode = Nodo(Estado(childState))

'''si no esta en la lista visitado se agregan los hijos''''

print('Nodo Hijo ',childNode.estado.lugar)

if childNode.estado.lugar not in visited:

'''aqui lo agregamos a la lista visited A y D'''

print('Nodo Hijo vs VISITED ',nodoActual.estado.lugar)

nodoActual.agregaHijos(childNode)

#agregar a los hijos en orden inverso al stack // for index in range(len(nodoActual.hijos) - 1, -1, -1):

for index in range(len(nodoActual.hijos)-1,-1,-1):

'''aqui no me vuelve a agregar los valores repetidos'''

if nodoActual.hijos[index] not in visited:

stack.append(nodoActual.hijos[index])

if len(stack) == 0:

print('Valor del Stack🡪CERO')

for x in stack:

print('Valor del Stack+++++++++++++++ ', x.estado.\_\_dict\_\_)

#print("NODOS VISITADOS ::>",visited)

#mostrar el arbol explorado hasta el momento

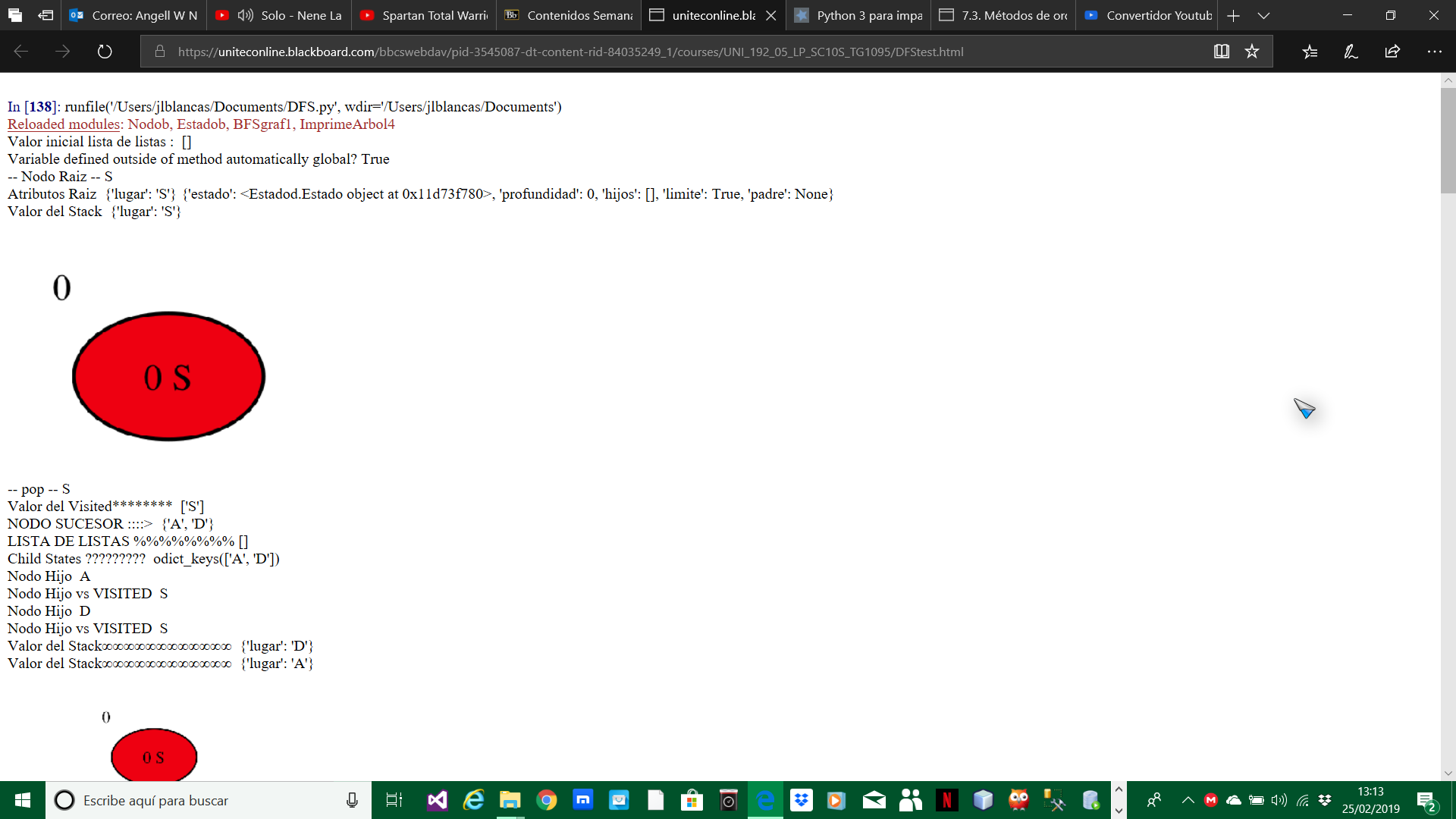
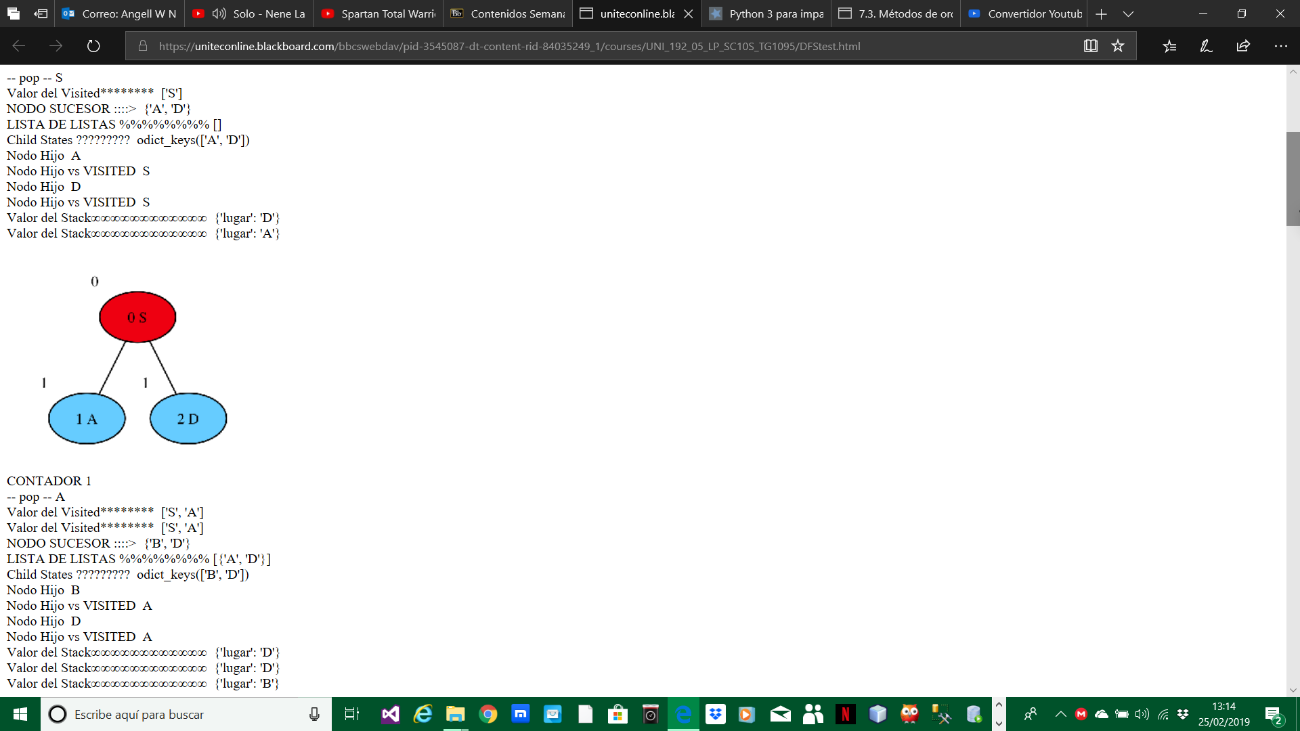
cont = cont +1

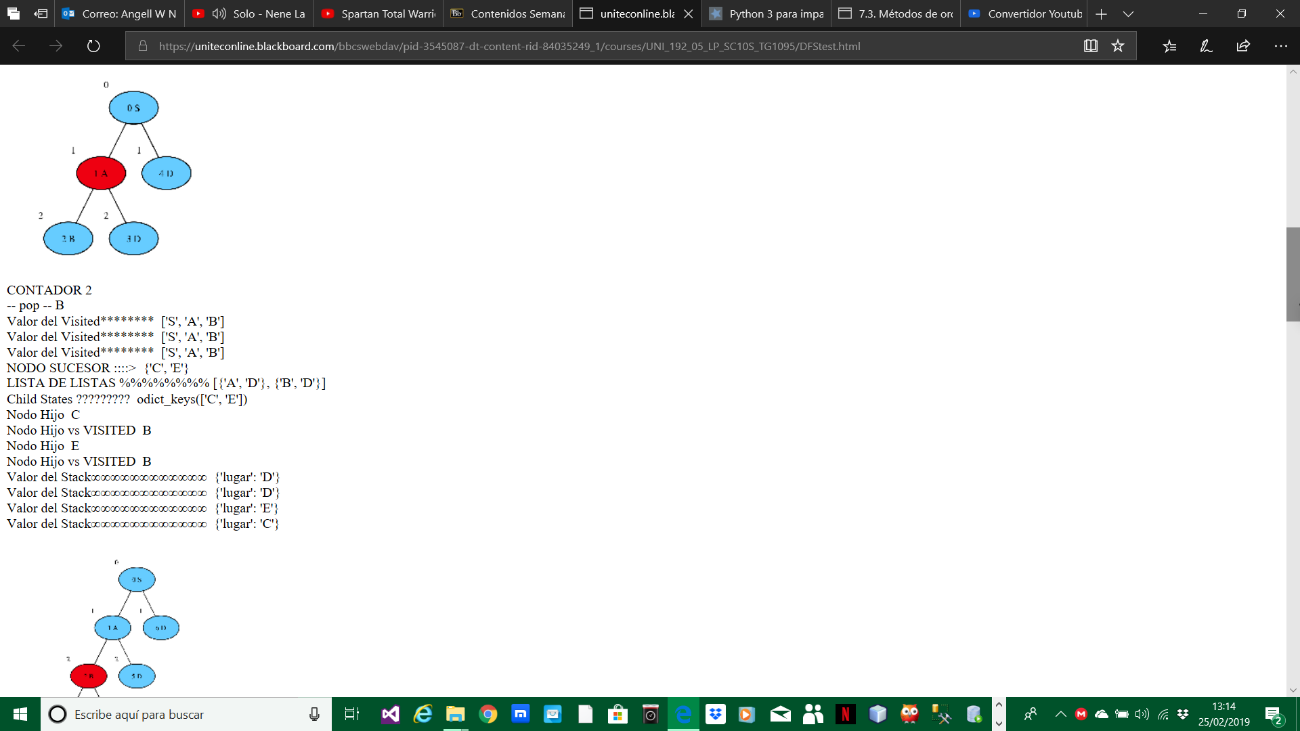
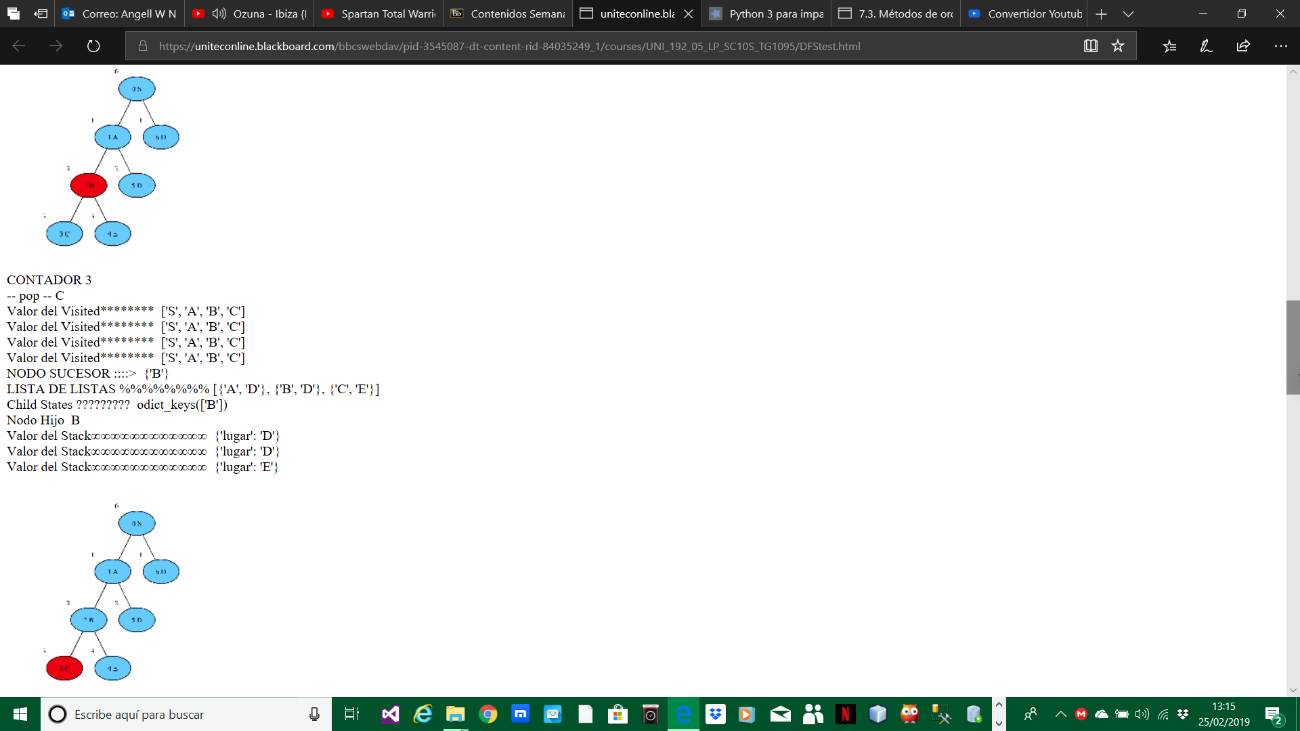
print("CONTADOR", cont)

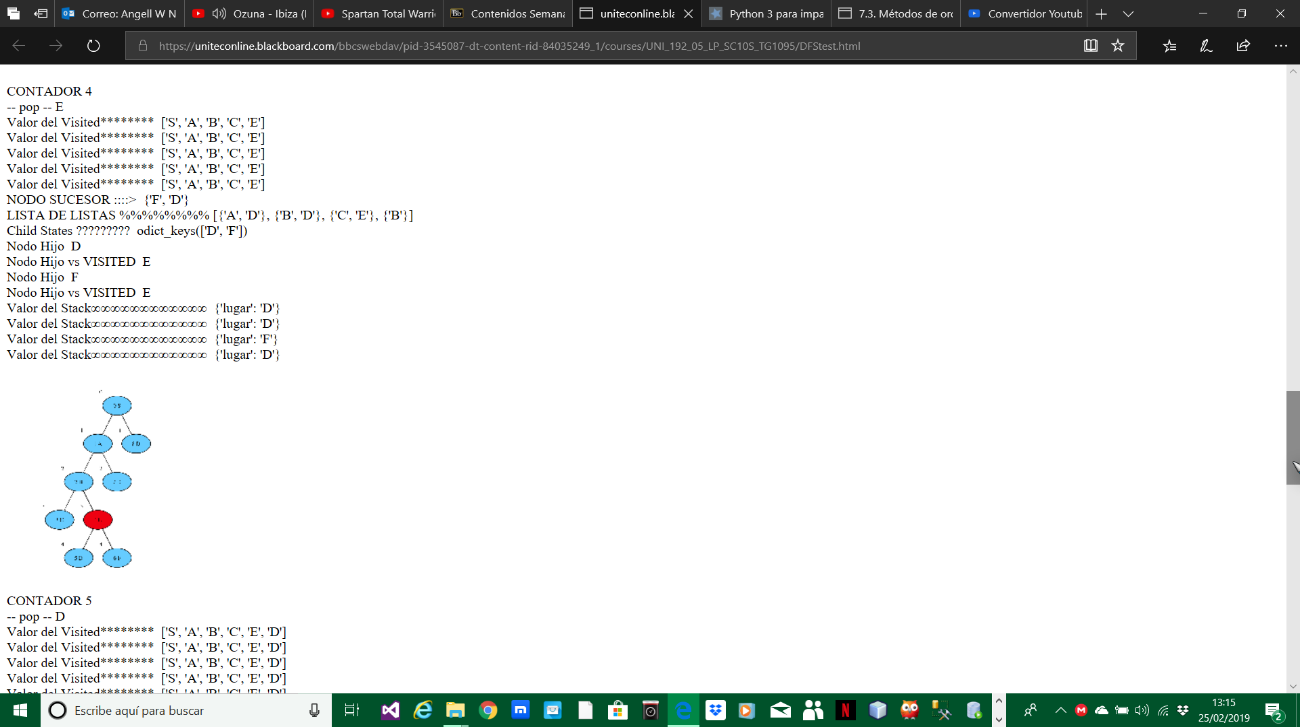
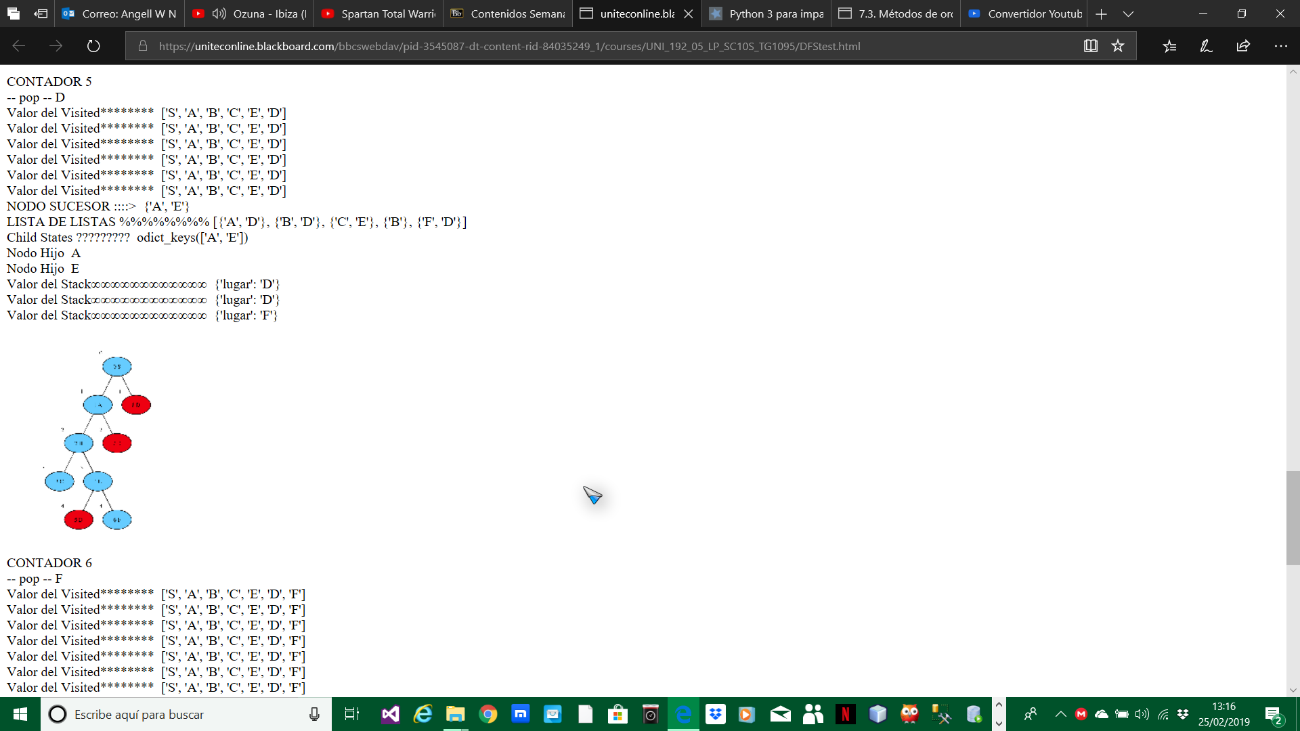
print("----------------------")

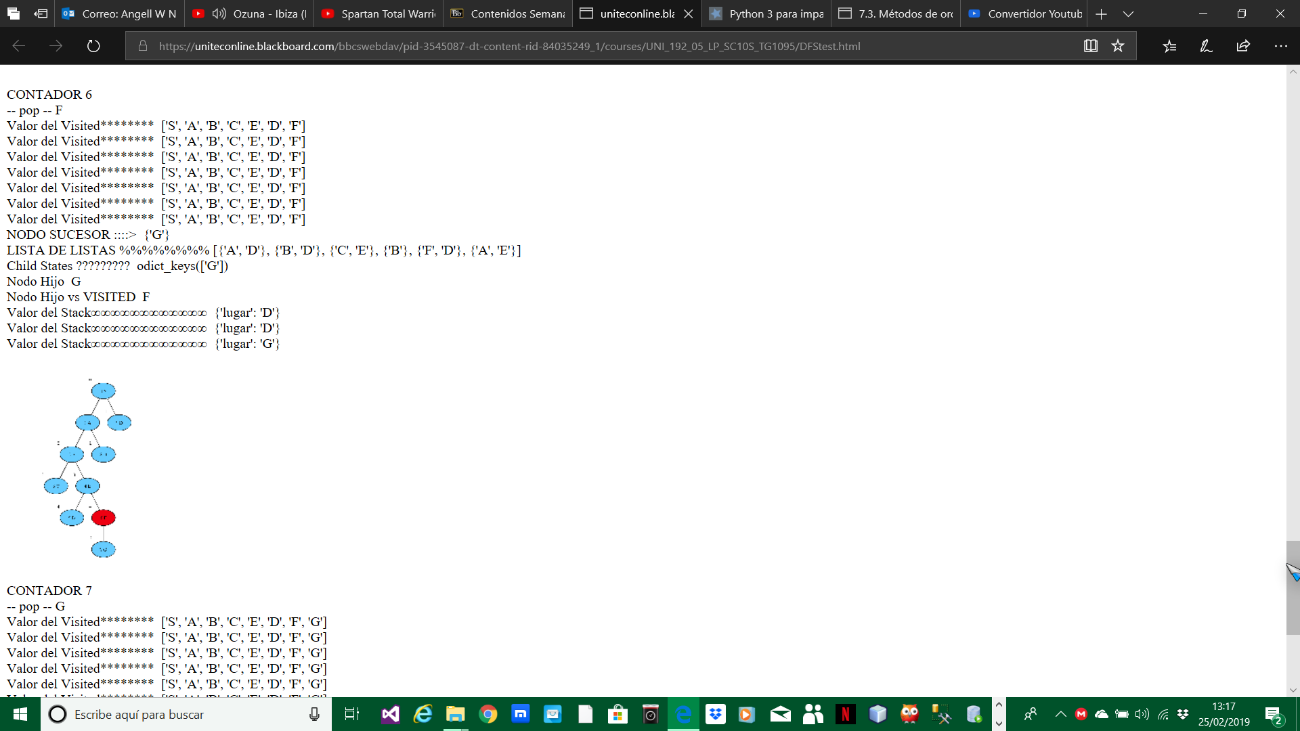
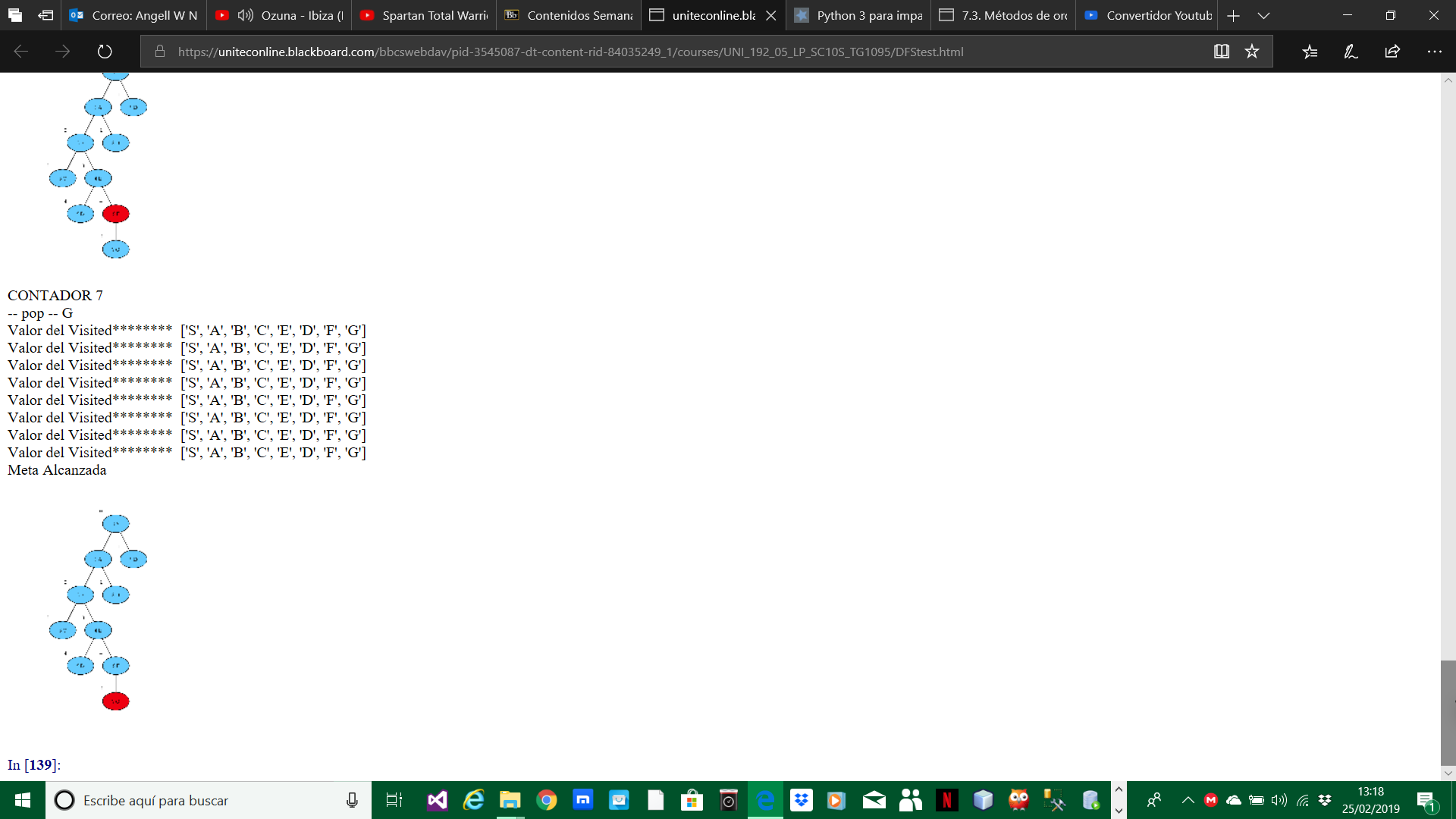
raiz.imprimeArbol()

**Impresión del árbol de cómo se verían el grafico uno y el dos.**







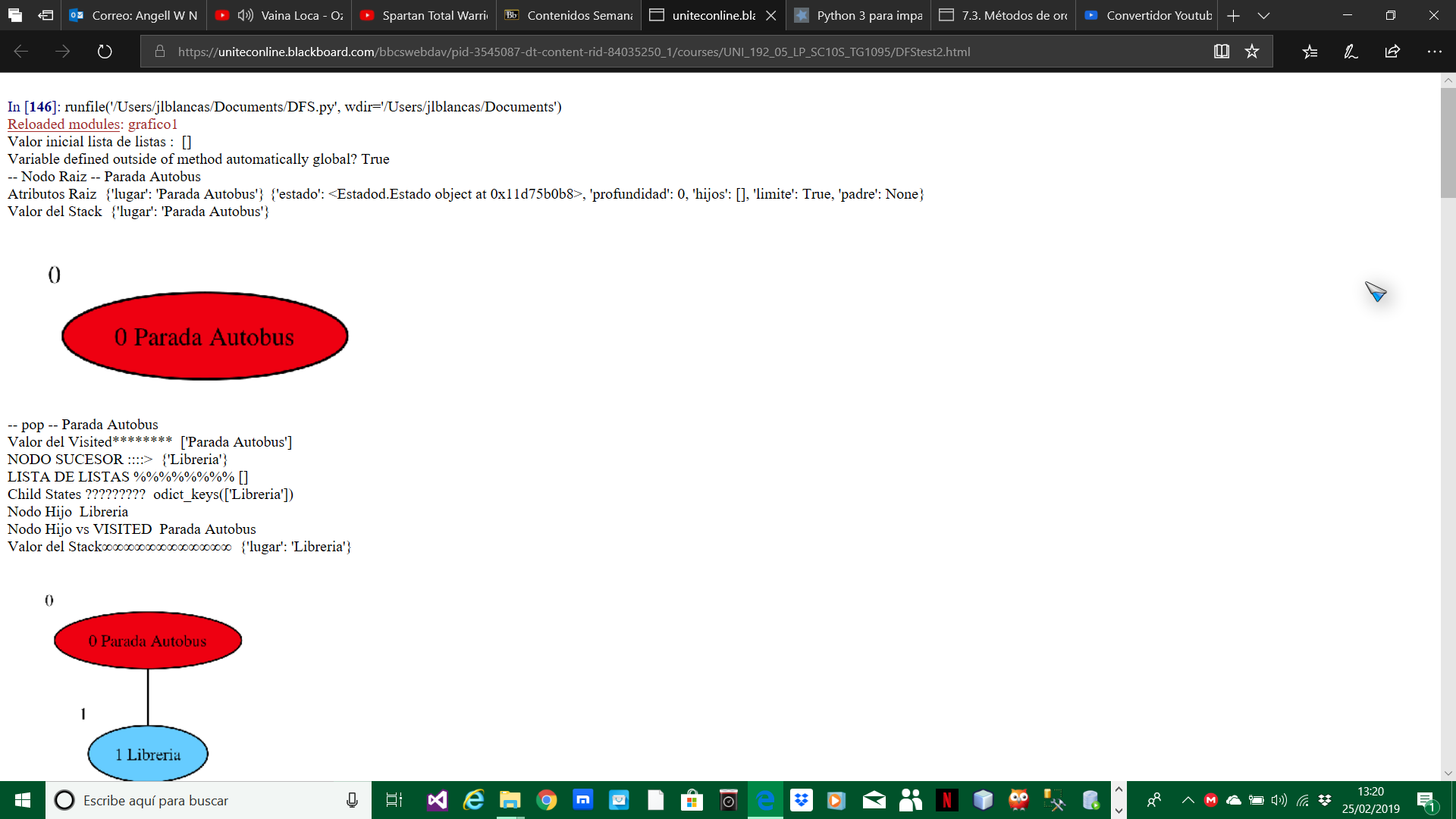


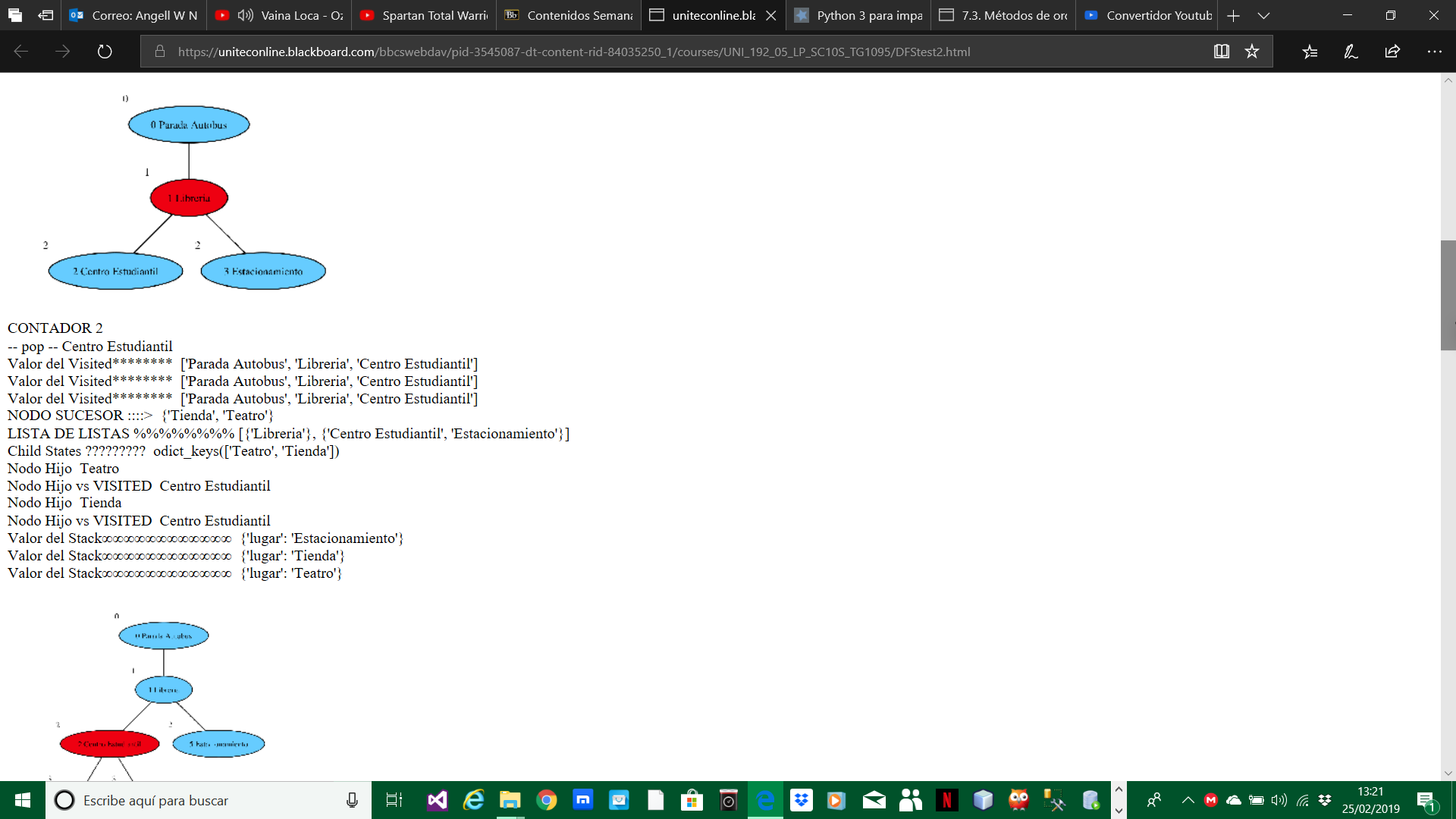
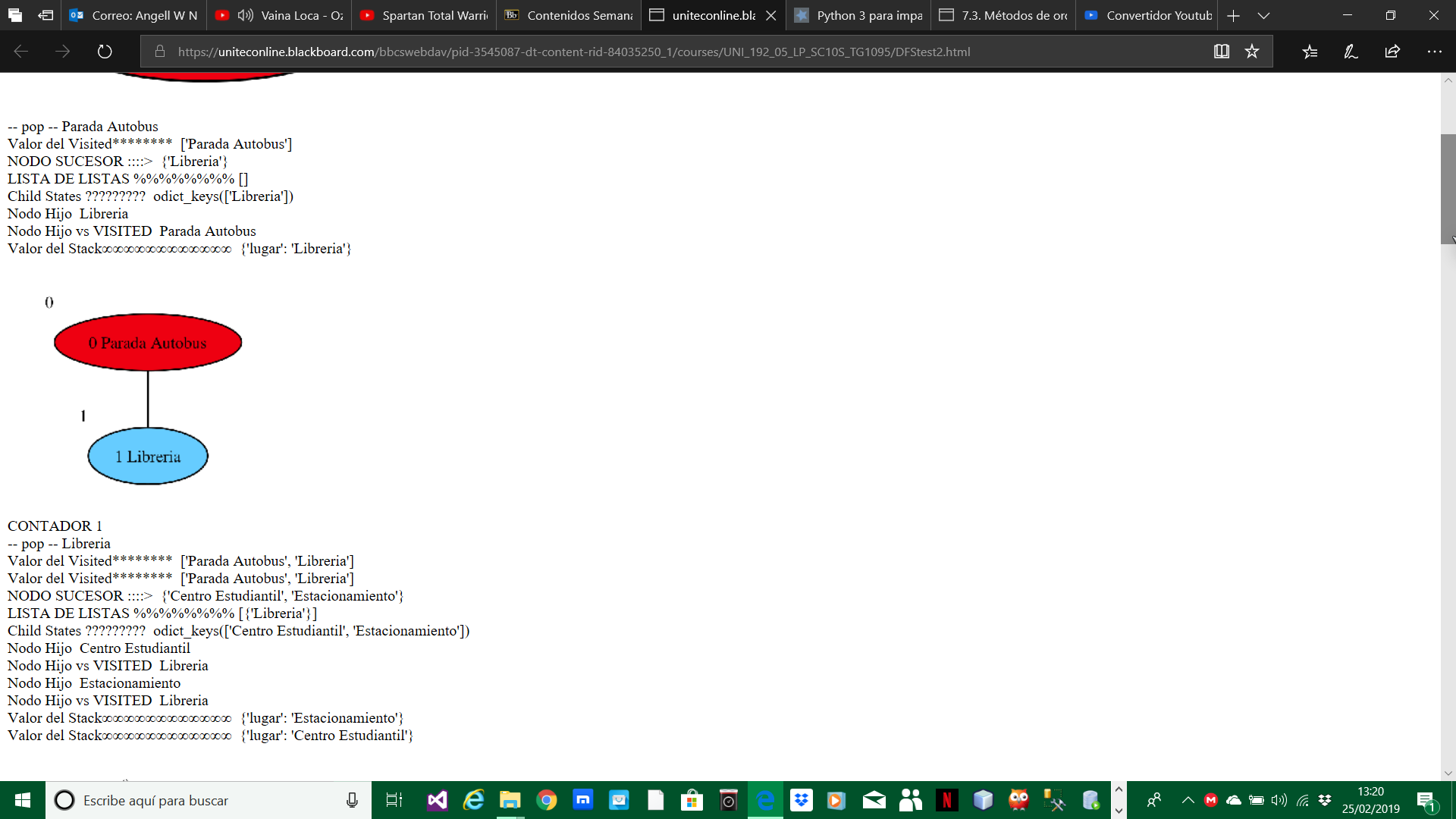
Se ponen contadores para que sepa cómo va viajando el árbol en cada nodo y así visualizar mejor el recorrido.

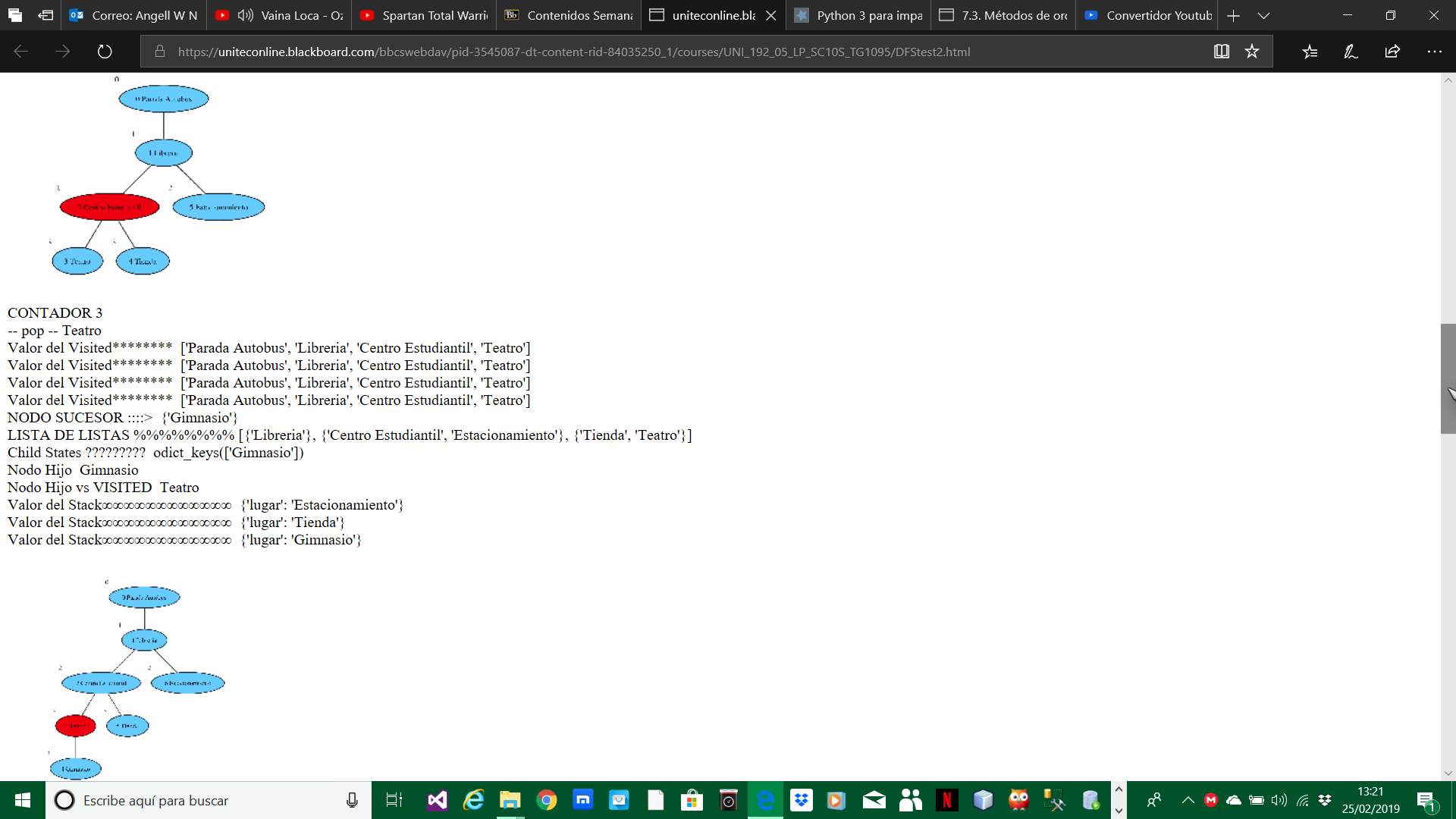
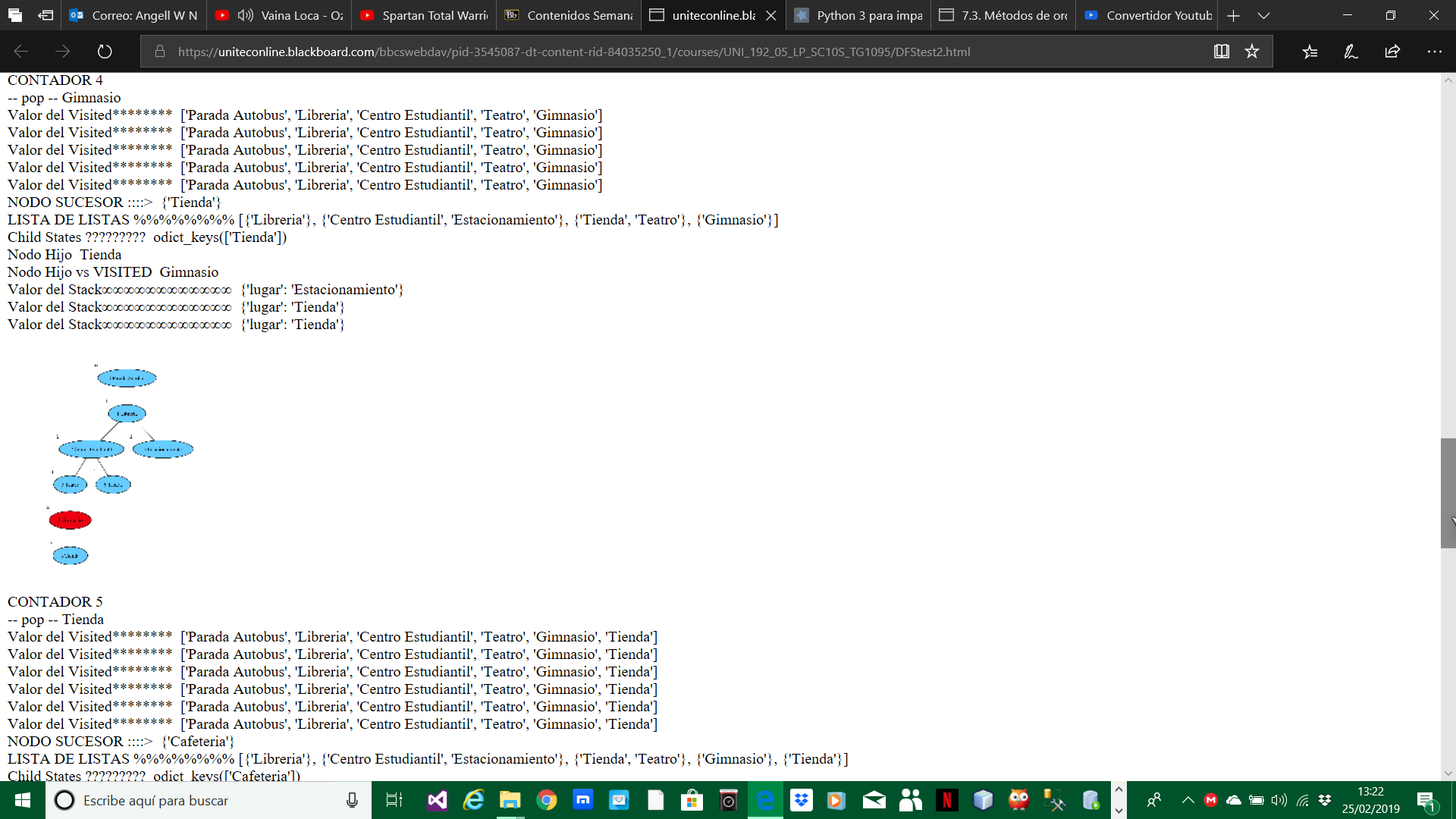
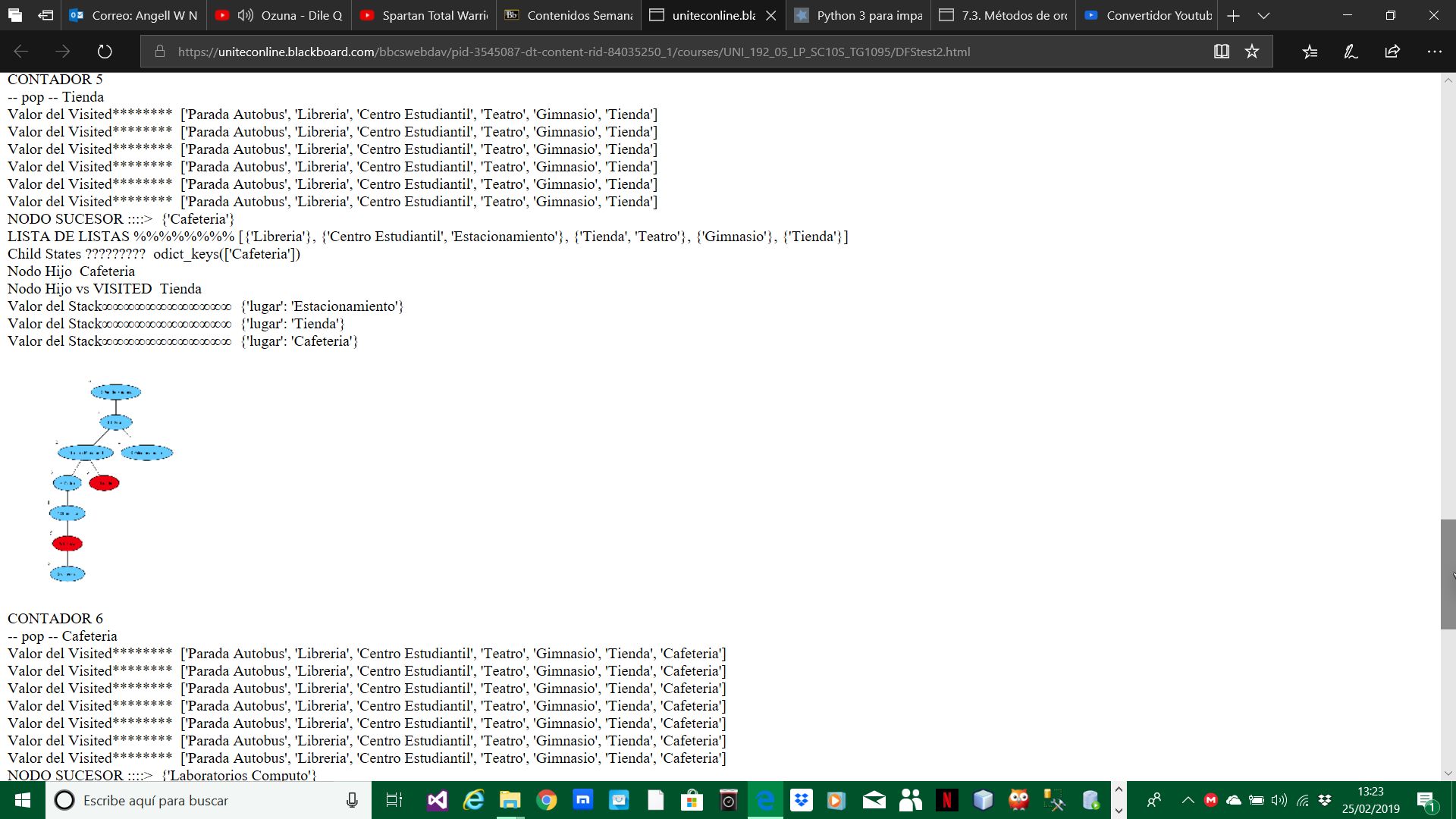
Árbol completo.

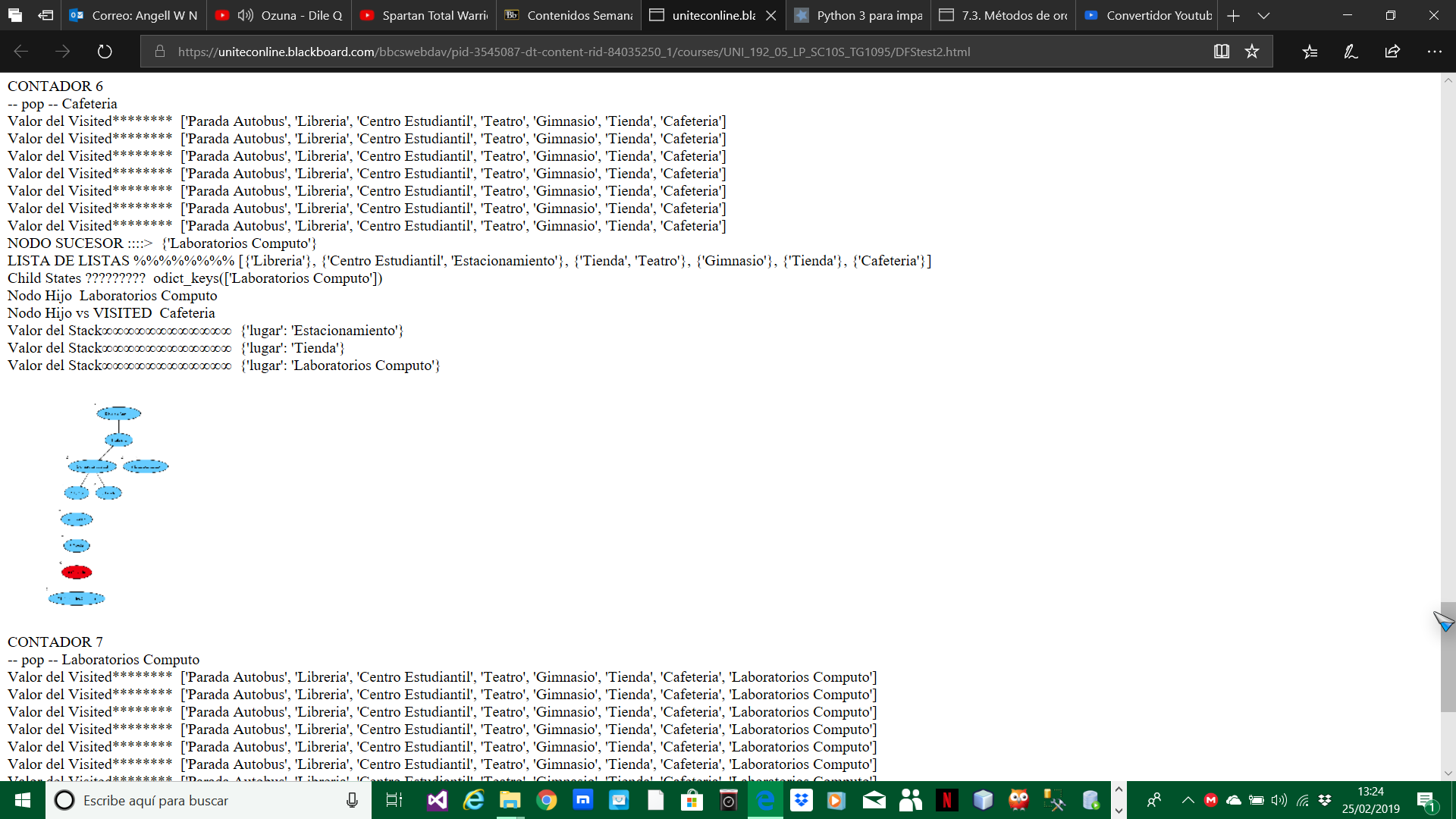


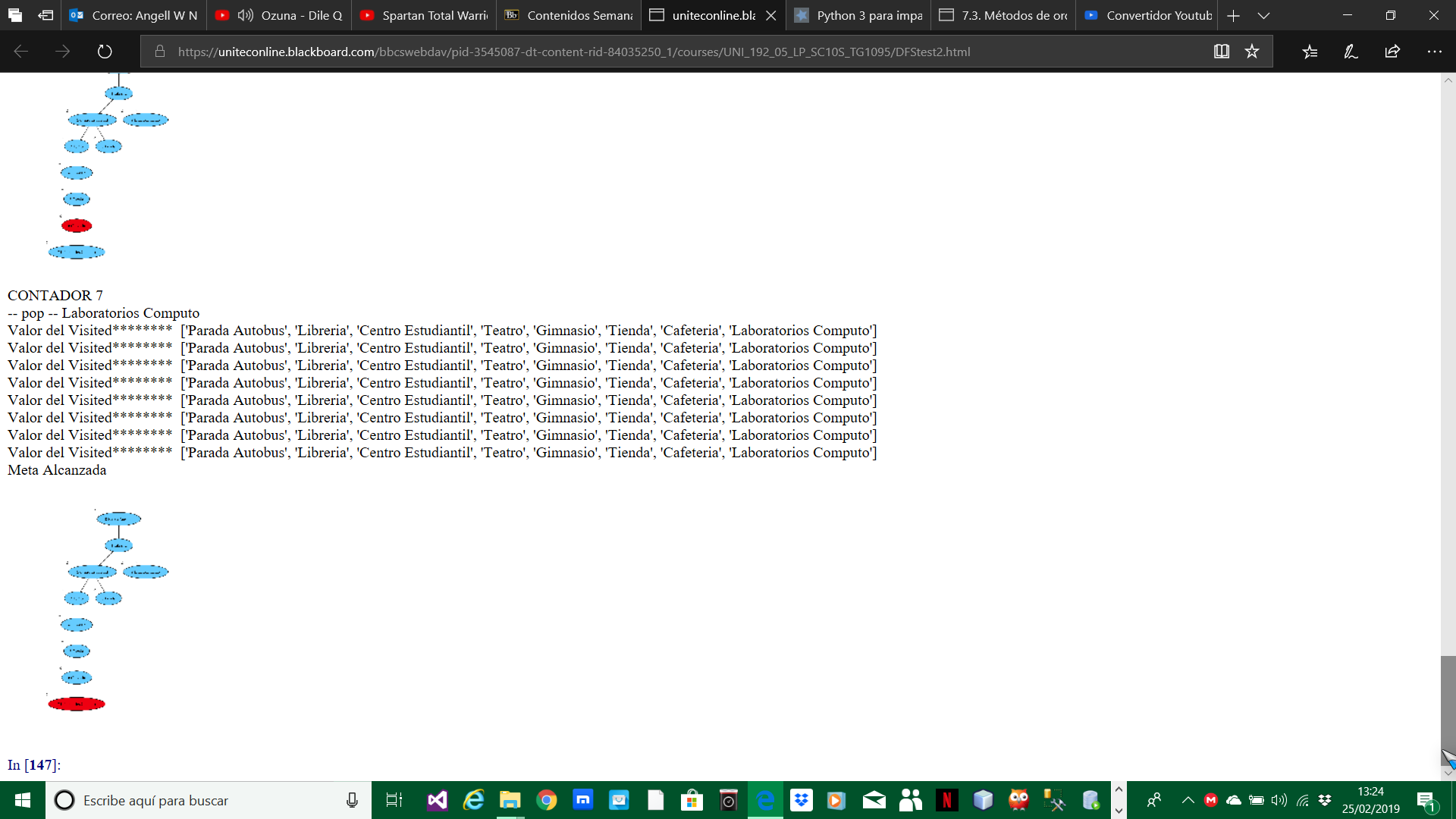
Segundo árbol por recorrer.











Árbol completo del segundo gráfico.

