

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA



1INF06: ESTRUCTURA DE DATOS Y PROGRAMACIÓN METÓDICA

Docente:

BELLO RUIZ, ALEJANDRO TORIBIO

H – 0582

Alumno:

Campana Yauri, Dan-el Mauricio
Palacios Coronado, Antony
Perez Cabrera, Estephany Elizabeth

20125121
20141735
20142604

Tema:

INFORME 6: Desarrollo en python 3 de una aplicación que procesa el grafo de rutas de sus domicilios a la PUCP con el algoritmo de Ordenación Topológica de un grafo.

Junio del 2018

ÍNDICE

1. INTRODUCCIÓN.....	3
2. DEFINICIONES.....	3
3. EXPLICACIÓN.....	4
4. INTERFAZ EN PYTHON 3.....	7
5. COMENTARIOS.....	8
6. PARTES TRABAJADAS POR LOS INTEGRANTES...	9
7. ANEXOS CÓDIGOS PYTHON.....	10
8. BIBLIOGRAFÍA.....	17

1. Introducción

En el presente informe se presentará el algoritmo de Ordenamiento Topológico aplicado a un grafo obtenido mediante OpenStreetMaps con el fin de hallar una secuencia de nodos que indiquen la ruta de un punto a otro en el mapa. Para ello, se tomará como puntos iniciales las casas de los integrantes del grupo y como punto final la entrada de la PUCP.

El algoritmo de Ordenamiento Topológico parte de un nodo que no tenga predecesores para empezar a procesar todos los nodos de un grafo de tal forma que cada nodo sea procesado antes que los nodos a los que apunta. De esta manera revisa cada secuencia de nodos hasta llegar a uno que ya no tiene conexión con otros nodos para empezar a guardarlos en una pila. Una vez que recorre todos los nodos, desapila los elementos y obtiene una serie de vértices organizados de manera lineal y ordenada.

2. Definiciones

Ordenamiento Topológico: Este algoritmo organiza de forma lineal una serie de vértices en desorden, para lo cual primero debe empezar de un "vértice padre", los cuales serán las ubicaciones de las casas de los integrantes, y después visitar a sus vecinos. Luego de esto, pasa a analizar otro vértice, y de igual forma identifica todos sus vecinos, y hace esto recursivamente, hasta que haya visitado a todos los vértices.

OpenStreetMap: Es un proyecto colaborativo para crear mapas libres y editables. Los mapas se crean utilizando información geográfica capturada con dispositivos GPS móviles y otras fuentes libres. Esta cartografía, tanto las imágenes creadas como los datos vectoriales almacenados en su base de datos, se distribuye bajo licencia abierta Licencia Abierta de Bases de Datos.

En octubre de 2014, en el proyecto estaban registrados alrededor de 1,840,000 usuarios, de los cuales 22,600 realizaban alguna edición en el último mes. El número de usuarios crece un 10% por mes. Por países, el

mayor número de ediciones provienen de Alemania, EE.UU., Rusia, Francia e Italia. Los usuarios registrados pueden subir sus trazas desde el GPS y crear y corregir datos vectoriales mediante herramientas de edición creadas por la comunidad OpenStreetMap.

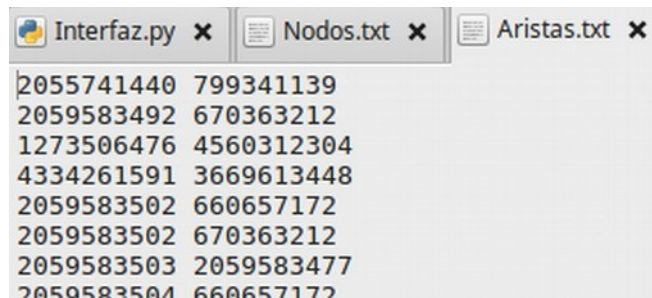
Algoritmo de Dijkstra: Fue creado por Edsger Dijkstra, también conocido como algoritmo de los caminos mínimos, es un algoritmo que determina el camino más corto desde un vértice dado hacia los vértices a los cuales apunta. En el caso de que se trate de un grafo ponderado se toma en cuenta los pesos de las aristas, y en caso contrario se toma en cuenta la cantidad de vértices que formarían parte del camino.

3. Explicación

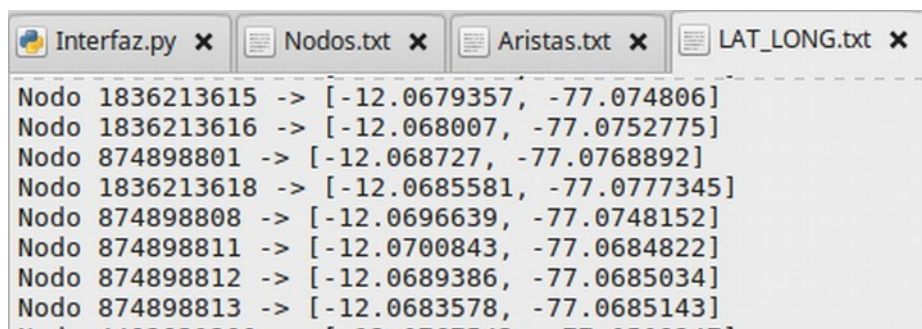
Para este trabajo se están usando aplicaciones en python que trabajan con archivos de texto. Para empezar se utilizó OpenStreetMap para seleccionar una zona, en la cual se encontraban las ubicaciones de las casas de los integrantes del grupo. Luego, se exportó este archivo con extensión .osm y se procesó en python (códigos anexados más adelante) haciendo uso de la librería "pyroute". Con esta librería se creó un archivo con todos los nodos junto con su latitud y longitud, llamado LAT-LON.txt. Después, se procesó este archivo para obtener solo los nodos y se guardó en un archivo llamado Nodos.txt. Este archivo también se procesó en python para buscar los vecinos de cada nodo, los cuales representaban las aristas y se guardaron en el archivo Aristas.txt. Este archivo también se procesó usando el algoritmo de Dijkstra, al cual se le dieron puntos iniciales y finales para que muestre el camino más corto entre los vértices dados y se guardó en un archivo llamado Camino.txt. Usando este último archivo se procesó el primer parámetro de cada línea del archivo y aplicando el método de parseo de la librería pyroute se determinaron los caminos alternos al punto final desde el punto de inicio. Estos resultados finalmente se guardaron en un archivo llamado Grafo.txt, el cual es precisamente el grafo dirigido acíclico que se necesitaba para aplicar el algoritmo de Ordenamiento Topológico.

i. Extracción de los nodos y aristas:

- Para este primer paso usaremos la librería **osmapi** y **pyroutelib3** los cuales nos servirán para extraer los datos de nodos, aristas y, latitud y longitud de cada nodo. En los archivos **Codigo.py** y **LAT_LONG.py** nos generará los archivos **Nodos.txt** y **Aristas.txt**. El modo en el cual extraeremos el mapa será en modo “**car**”, ya que de este modo se obtendrá las aristas en modo grafo dirigido.



```
Interfaz.py x Nodos.txt x Aristas.txt x
2055741440 799341139
2059583492 670363212
1273506476 4560312304
4334261591 3669613448
2059583502 660657172
2059583502 670363212
2059583503 2059583477
2059583504 660657172
```



```
Interfaz.py x Nodos.txt x Aristas.txt x LAT_LONG.txt x
Nodo 1836213615 -> [-12.0679357, -77.074806]
Nodo 1836213616 -> [-12.068007, -77.0752775]
Nodo 874898801 -> [-12.068727, -77.0768892]
Nodo 1836213618 -> [-12.0685581, -77.0777345]
Nodo 874898808 -> [-12.0696639, -77.0748152]
Nodo 874898811 -> [-12.0700843, -77.0684822]
Nodo 874898812 -> [-12.0689386, -77.0685034]
Nodo 874898813 -> [-12.0683578, -77.0685143]
```

ii. Generar camino usando Dijkstra:

- Para obtener un inicio y un fin de nuestro recorrido con los nodos usaremos el algoritmo de **Dijkstra** en el archivo **Dijkstra.txt**. Este algoritmo usará el archivo **Nodos.txt** dándole un nodo inicio (dirección de estudiante) y un nodo fin (PUCP). Estos nodos se mostrarán en orden que se debe seguir y exportados en un archivo de texto llamado **Caminos.txt**.

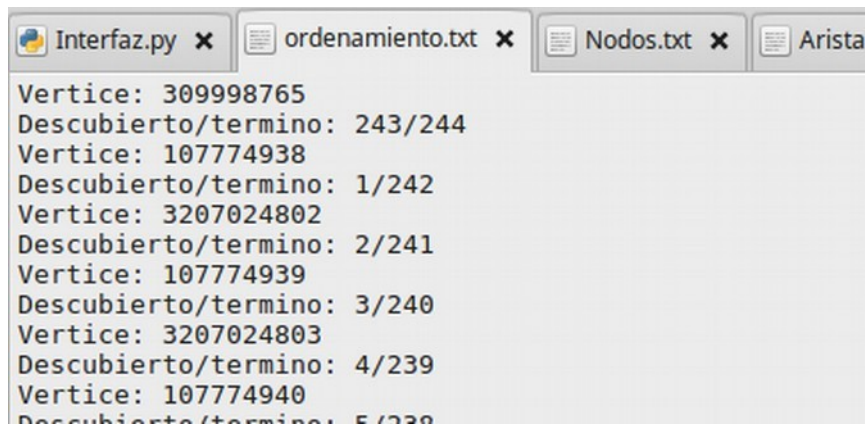
iii. Extracción de nodos con sus nodos adyacentes:

- Una vez obtenido los nodos en el archivo **Camino.txt** necesitaremos que todos esos nodos estén con sus nodos adyacentes para eso usaremos el archivo **parsear.py**. Esto debido a que nuestro algoritmo de Ordenamiento Topológico necesitamos los nodos adyacentes a

nuestro recorrido que es en profundidad; por ellos, extraeremos los otros nodos en un archivo de texto **Grafos.txt**.

iv. Implementación de algoritmo de Ordenamiento Topológico de un Grafo:

- Con el archivo **Grafos.txt** generaremos nuestra Pila de nodos, en el orden que deberíamos recorrer, donde el **top** de la pila es el inicio y el **fondo** de la pila es el destino a cual queremos llegar. Usaremos el programa **algoritmo.py**. Para una mejor visualización se generará un archivo llamado **ordenamiento.txt**.

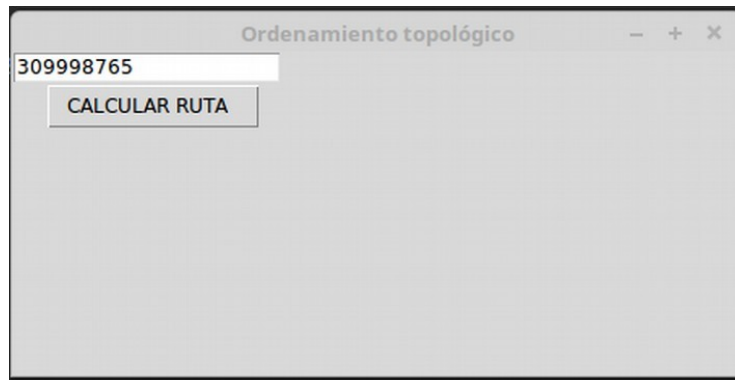


The screenshot shows a Python IDE with four tabs: 'Interfaz.py', 'ordenamiento.txt', 'Nodos.txt', and 'Arista'. The 'ordenamiento.txt' tab is active and displays the output of a topological sort algorithm. The output consists of a series of lines, each containing a vertex ID followed by its discovery and finishing times in the format 'Vertice: ID Descubierta/termino: A/B'. The vertices are listed in descending order of their finishing times.

```
Vertice: 309998765
Descubierta/termino: 243/244
Vertice: 107774938
Descubierta/termino: 1/242
Vertice: 3207024802
Descubierta/termino: 2/241
Vertice: 107774939
Descubierta/termino: 3/240
Vertice: 3207024803
Descubierta/termino: 4/239
Vertice: 107774940
Descubierta/termino: 5/238
```

4. Interfaz en Python 3

Para implementar la interfaz usaremos la librería TKinter donde colocaremos el nodo el cual queremos partir, ya que el destino para ambos casos será la PUCP. Esta interfaz usará los mismo código mencionados anteriormente en el mismo orden.



```
/home/invictus/PycharmProjects/untitled/venv/bin/python "/home/invictus/Desktop/Informes/Informe 6/Interfaz.py"  
Vertice: 309998765  
Descubierto/termino: 243/244  
Vertice: 107774938  
Descubierto/termino: 1/242  
Vertice: 3207024802  
Descubierto/termino: 2/241  
Vertice: 107774939  
Descubierto/termino: 3/240  
Vertice: 3207024803  
Descubierto/termino: 4/239  
Vertice: 107774940  
Descubierto/termino: 5/238
```

```
Descubierto/termino: 114/125  
Vertice: 1273733186  
Descubierto/termino: 115/128  
Vertice: 1273635442  
Descubierto/termino: 116/127  
Vertice: 1273499641  
Descubierto/termino: 117/126  
Vertice: 2981919235  
Descubierto/termino: 118/125  
Vertice: 2981919239  
Descubierto/termino: 119/124  
Vertice: 2981919236  
Descubierto/termino: 120/123  
Vertice: 2981919237  
Descubierto/termino: 121/122
```

El resultado nos muestra el nodo inicial del cual se parte. Y el fondo de la pila nos muestra a dónde se quiere llegar.

5. Comentarios

Uno de los problemas que encontramos al inicio era de cómo empezar con el programa y qué datos de entrada se le debía de dar. Lo primero que tuvimos que encontrar era el modo de extraer los nodos sin repetirse y los nodos adyacentes. Indagando sobre el tema, encontramos que existía una librería llamada **pyroutelib3** y **osmapi** los cuales nos facilitaron el modo de extraer los datos.

Para generar los archivos tuvimos el problema de memoria, el cual nos mostraba que no era suficiente para la cantidad de datos que teníamos; para ello, usamos el siguiente comando: **sys.setrecursionlimit(2000)**, donde aumentamos los recursos que podía usar el programa.

Una vez extraído nos encontramos con el siguiente problema, el cual era cómo darle los datos al programa, ya que para nuestro tema era necesario que se cumplan dos condiciones: que el grafo sea dirigido y sea acíclico. Para resolver este inconveniente usamos el método **Dijkstra** que nos daba un camino entre dos puntos, luego de ello, como nuestro grafo necesitaba más nodos colocamos todos los nodos con sus aristas correspondientes para simular un grafo dirigido y acíclico. Una vez obtenido esto solo necesitábamos colocar los datos en el programa y ejecutarlo.

Asimismo, al momento de parsear los archivos **Caminos.txt** y **Aristas.txt** solo se ejecutaba una vez la lectura de **Aristas.txt**. Esto debido a que al momento de hacer **file2 = open("Aristas.txt", "r")** dentro de un **for** este solo se abría una sola vez y no se volvía a leer, con lo cual solo nos mostraba una sola coincidencia que era la del primer nodo de archivo **Caminos.txt**. Para solucionar este error agregamos antes de cada **for** el comando **file2 = open("Aristas.txt", "r")** con lo cual se solucionó el problema.

6. Partes trabajadas por el grupo

En el presente trabajo, todos los integrantes trabajaron por igual, tanto en la elaboración del informe como en la implementación del código en Python3. Con respecto a la redacción del informe, se trabajó en la plataforma de Google Drive, es decir se trabajó en simultáneo. En cuanto a la interfaz, esta se fue implementando gradualmente en las reuniones grupales buscando en internet.

Código :

Campana Yauri, Dan-el Mauricio	20125121
Palacios Coronado, Antony	20141735
Perez Cabrera, Estephany Elizabeth	20142604

Informe:

Campana Yauri, Dan-el Mauricio	20125121
Palacios Coronado, Antony	20141735
Perez Cabrera, Estephany Elizabeth	20142604

Métodos Python:

Campana Yauri, Dan-el Mauricio	20125121
Palacios Coronado, Antony	20141735
Perez Cabrera, Estephany Elizabeth	20142604

Interfaz en Python:

Campana Yauri, Dan-el Mauricio	20125121
Palacios Coronado, Antony	20141735
Perez Cabrera, Estephany Elizabeth	20142604

7. Anexos Códigos Python

Para sacar los nodos con su latitud y longitud:

```
from pyrouelib3 import Router # Import the router
router = Router("car", "map2.osm") # Initialise it
archivo = open("LAT_LONG.txt", "w")
archivo.write('LAT Y LON:\n')
for i in router.data.rnodes:
    cadena = 'Nodo ' + str(i) + ' -> ' + str(router.data.rnodes[i]) +
'\n'
    archivo.write(cadena)
```

Algoritmo de Dijkstra:

```
from collections import defaultdict
from heapq import *
from pyrouelib3 import Router # Import the router
router = Router("car", "map2.osm") # Initialise it
def dijkstra(edges, f, t):
    g = defaultdict(list)
    for l,r,c in edges:
        g[l].append((c,r))
    q, seen = [(0,f,())], set()
    while q:
        (cost,v1,path) = heappop(q)
        if v1 not in seen:
            seen.add(v1)
            path = (v1, path)
            if v1 == t: return (cost, path)
            for c, v2 in g.get(v1, ()):
                if v2 not in seen:
                    heappush(q, (cost+c, v2, path))
    return float("inf")
edges=[]
for nodos, dic in router.data.routing.items():
    for item in dic:
        edges.append((nodos, item, dic[item]))
if __name__ == "__main__":
    print ("=== Dijkstra ===")
    camino = (str(dijkstra(edges,309998765,2981919237 )))
    for char in '(),\n':
        camino=camino.replace(char,' ')
    camino = camino.split()[1:]
    archivo = open("Camino.txt", "w")
    for elem in camino:
        cadena=str(elem) + '\n'
        archivo.write(cadena)
```

Parsear:

```
file1 = open("Camino.txt", "r")
file2 = open("Aristas.txt", "r")
```

```

grafo = open("Grafo.txt", "w")
'''
for item in file1:
    #print(file.readline())
    a,b = item.split(" ")
    print(a,b)
'''
cadena=str()
for item in file1:
    item=item.replace("\n","")
    file2 = open("Aristas.txt", "r")
    for lista in file2:
        n1,n2=lista.split()
        n1=n1.replace("\n","")
        if(int(item) == int(n1)):
            cadena = str(n1) + ' ' + str(n2) + '\n'
            print(cadena)
            grafo.write(cadena)

```

Código:

```

from pyrouelib3 import Router # Import the router
router = Router("car", "map2.osm") # Initialise it
archivo = open("Nodos.txt", "w")
archivo2 = open("Aristas.txt", "w")
## --- MUESTRA EN CONSOLA
for nodos, dic in router.data.routing.items():
    print(nodos, dic)
    for item in dic:
        print(nodos, item)
        #print(nodos, item, dic[item])
## --- Guarda en archivo los nodos
for nodos in router.data.routing.items():
    cadena = str(nodos[0]) + '\n'
    archivo.write(cadena)
## --- Guarda en archivo las aristas
for nodos, dic in router.data.routing.items():
    for item in dic:
        cadena2 = str(nodos) + ' ' + str(item) + '\n'
        archivo2.write(cadena2)

```

Ordenamiento Topológico:

```

import operator
from pyrouelib3 import Router # Import the router
router = Router("car", "map2.osm") # Initialise it
archivo = open("ordenamiento.txt", "w")
camino = open("Camino.txt", "r")
grafo = open("Grafo.txt", "r")
import sys
sys.setrecursionlimit(2000)
class Vertice:
    def __init__(self,n):

```

```

        self.nombre = n
        self.vecinos = list()
        self.d = 0
        self.f = 0
        self.color = 'white'
        self.pred = -1
    def agregarVecino(self, v):
        if v not in self.vecinos:
            self.vecinos.append(v)
            self.vecinos.sort()
class Grafo:
    vertices = {}
    tiempo = 0
    def agregarVertice(self, vertice):
        if isinstance(vertice, Vertice) and vertice.nombre not in self.vertices:
            self.vertices[vertice.nombre] = vertice
            return True
        else:
            return False
    def agregarArista(self, u, v):
        if u in self.vertices and v in self.vertices:
            for key, value in self.vertices.items():
                if key == u:
                    value.agregarVecino(v)
                    #if key == v: #Se comenta porque es grafo dirigido
                    #    value.agregarVecino(u)
            return True
        else:
            return False
    def imprimeGrafo(self):
        for key in sorted(list(self.vertices.keys())):
            print("Vertice: "+key )
            print("Descubierto/termino: "+str(self.vertices[key].d)+
"/"+ str(self.vertices[key].f))
    def imprimeOrdenado(self, atributo, invertida = False):
        cadena = str()
        for vertice in (sorted(self.vertices.values(),
key=operator.attrgetter(atributo), reverse= True)):
            print("Vertice: {}".format(vertice.nombre))
            cadena = cadena + str("Vertice: {}".format(vertice.nombre))
+ '\n'
            print("Descubierto/termino: {}/{}".format(vertice.d,
vertice.f))
            cadena = cadena + str("Descubierto/termino: {}/
{}".format(vertice.d, vertice.f)) + '\n'
            archivo.write(cadena)
    def dfs(self, vert):
        self.tiempo = 0
        for v in sorted(list (self.vertices.keys())):
            if self.vertices[v].color == 'white':
                self.dfsVisitar (self.vertices[v])
    def dfsVisitar(self, vert):
        self.tiempo += 1
        vert.d = self.tiempo
        vert.color = 'gris'

```

```

        for v in vert.vecinos:
            if self.vertices[v].color == 'white':
                self.vertices[v].pred = vert
                self.dfsVisitar(self.vertices[v])
        vert.color = 'black'
        self.tiempo += 1
        vert.f = self.tiempo
class Controladora:
    def main(self):
        g = Grafo()
        a = Vertice(int(309998765))
        g.agregarVertice(a)
        for item in camino:
            g.agregarVertice(Vertice(int(item)))
        for item in grafo:
            n1,n2=item.split()
            g.agregarArista(int(n1),int(n2))
        g.dfs(a)
        g.imprimeOrdenado('f', invertida = True)
obj = Controladora()
obj.main()

```

Interfaz:

```

import operator
from collections import defaultdict
from heapq import *
from pyrouelib3 import Router # Import the router
router = Router("car", "map2.osm") # Initialise it
import tkinter
from tkinter import *
import os
import sys
sys.setrecursionlimit(2000)
##----- PRIMERO SE EJECUTA EL DIJKSTRA -----
##-- Para obtener un camino hacia el destino -----
def dijkstra(edges, f, t):
    g = defaultdict(list)
    for l,r,c in edges:
        g[l].append((c,r))
    q, seen = [(0,f,())], set()
    while q:
        (cost,v1,path) = heappop(q)
        if v1 not in seen:
            seen.add(v1)
            path = (v1, path)
            if v1 == t: return (cost, path)
            for c, v2 in g.get(v1, ()):
                if v2 not in seen:
                    heappush(q, (cost+c, v2, path))
    return float("inf")
edges=[]
for nodos, dic in router.data.routing.items():
    for item in dic:
        edges.append((nodos, item, dic[item]))

```

```

## -----
##-----
##----- ALGORITMO DE ORDENACION TOPOLOGICA
-----
class Vertice:
    def __init__(self,n):
        self.nombre = n
        self.vecinos = list()
        self.d = 0
        self.f = 0
        self.color = 'white'
        self.pred = -1
    def agregarVecino(self, v):
        if v not in self.vecinos:
            self.vecinos.append(v)
            self.vecinos.sort()
class Grafo:
    vertices = {}
    tiempo = 0
    def agregarVertice(self, vertice):
        if isinstance(vertice, Vertice) and vertice.nombre not in self.vertices:
            self.vertices[vertice.nombre] = vertice
            return True
        else:
            return False
    def agregarArista(self, u, v):
        if u in self.vertices and v in self.vertices:
            for key, value in self.vertices.items():
                if key == u:
                    value.agregarVecino(v)
                #if key == v: #Se comenta porque es grafo dirigido
                #    value.agregarVecino(u)
            return True
        else:
            return False
    def imprimeGrafo(self):
        for key in sorted(list(self.vertices.keys())):
            print("Vertice: "+key )
            print("Descubierto/termino: "+str(self.vertices[key].d)+
"/"+ str(self.vertices[key].f))
    def imprimeOrdenado(self, atributo, invertida = False):
        archivo = open("ordenamiento.txt", "w")
        cadena = str()
        for vertice in (sorted(self.vertices.values(),
key=operator.attrgetter(atributo), reverse= True)):
            print("Vertice: {}".format(vertice.nombre))
            cadena = cadena + str("Vertice: {}".format(vertice.nombre))
+ '\n'
            print("Descubierto/termino: {}/{}".format(vertice.d,
vertice.f))
            cadena = cadena + str("Descubierto/termino: {}/
{}".format(vertice.d, vertice.f)) + '\n'
            archivo.write(cadena)
            archivo.close()
    def dfs(self, vert):

```

```

        self.tiempo = 0
        for v in sorted(list (self.vertices.keys())):
            if self.vertices[v].color == 'white':
                self.dfsVisitar (self.vertices[v])
        def dfsVisitar(self, vert):
            self.tiempo += 1
            vert.d = self.tiempo
            vert.color = 'gris'
            for v in vert.vecinos:
                if self.vertices[v].color == 'white':
                    self.vertices[v].pred = vert
                    self.dfsVisitar(self.vertices[v])
            vert.color = 'black'
            self.tiempo += 1
            vert.f = self.tiempo
class Controladora:
    def main(self):
        g = Grafo()
        a = Vertice(int(309998765))
        g.agregarVertice(a)
        camino = open("Camino.txt", "r")
        for item in camino:
            g.agregarVertice(Vertice(int(item)))
        grafo = open("Grafo.txt", "r")
        for item in grafo:
            n1,n2=item.split()
            g.agregarArista(int(n1),int(n2))
        g.dfs(a)
        g.imprimeOrdenado('f', invertida = True)
        grafo.close()
        camino.close()

## -----
##-----
##----- INTERFAZ-----
##-----
window = Tk()
window.geometry("450x200")
window.title("Ordenamiento topológico")
e = Entry(window)
e.grid(row = 1, columnspan=4)
b_calcular = Button(window, text = "CALCULAR RUTA", anchor="w",
width=13, command = lambda:ejecutar(int(e.get())))
b_calcular.grid(row=7, column=2)
def ejecutar(coordenada):
    #----- ("=== Dijkstra ===")
    camino = (str(dijkstra(edges, coordenada, 2981919237)))
    for char in '(),\n':
        camino = camino.replace(char, ' ')
    camino = camino.split()[1:]
    archivo = open("Camino.txt", "w")
    for elem in camino:
        cadena = str(elem) + '\n'
        archivo.write(cadena)
    archivo.close()
##-----
## --- EMPEZAMOS A PARSEAR PARA OBTENER LOS ARCHIVOS-----

```

```

file1 = open("Camino.txt", "r")
file2 = open("Aristas.txt", "r")
grafo = open("Grafo.txt", "w")
for item in file1:
    item = item.replace("\n", "")
    file2 = open("Aristas.txt", "r")
    for lista in file2:
        n1, n2 = lista.split()
        n1 = n1.replace("\n", "")
        if (int(item) == int(n1)):
            cadena = str(n1) + ' ' + str(n2) + '\n'
            # print(cadena)
            grafo.write(cadena)
grafo.close()
file1.close()
file2.close()
obj = Controladora()
obj.main()
##### MAIN
#####
if __name__ == "__main__":
    window.mainloop()

```


8. Bibliografía:

Estructura de Datos: Ordenación Topológica. Consulta 6 de Junio del 2018.

<http://ordenamientotopologico.blogspot.com/>

STACKOVERFLOW

¿Cómo puedo implementar un algoritmo de ordenación topológica en mi código de grafos?. Consulta 6 de Junio del 2018.

<https://es.stackoverflow.com/questions/58308/c%C3%B3mo-puedo-implementar-un-algoritmo-de-ordenaci%C3%B3n-topologica-en-mi-c%C3%B3digo-de-gr>

INTERACTIVEPYTHON

Ordenamiento topológico. Consulta 6 de Junio del 2018.

<http://interactivepython.org/runestone/static/pythoned/Graphs/OrdenamientoTopologico.html>

IVAN SULLIVAN

Implementation of Dijkstra in Python. Consulta 6 de Junio del 2018.

<https://www.youtube.com/watch?reload=9&v=IG1QioWSXRI>

TUTORIALSPPOINT

Python String replace() Method. Consulta 6 de Junio del 2018.

https://www.tutorialspoint.com/python/string_replace.htm

PYTHON PARA IMPACIENTES

Abrir paginas web en un navegador con webbrowser. Consulta 6 de Junio del 2018.

<https://python-para-impacientes.blogspot.com/2015/11/abrir-paginas-web-en-un-navegador-con.html>

WIKIPEDIA

OpenStreetMap. Consulta 6 de Junio del 2018.

<https://es.wikipedia.org/wiki/OpenStreetMap>