LABORATORIO NO. 12: GRAFOS

1 OBJETIVOS

Utilizar el Tipo Abstracto de Datos Grafo para solucionar problemas computacionales que requieran modelar relaciones matriciales.

a) Comenzar la implementación del reto 4.
b) Utilizar algunos algoritmos básicos de recorridos sobre grafos.
c) Integrar los *grafos* con las otras estructuras de datos vistas en el curso.

2 DESARROLLO

En los grupos previamente definidos sigan los siguientes pasos para el laboratorio de hoy.

2.1 CARGAR EL PROYECTO DE ARRANQUE ISIS1225-RETO 4

Asegúrense que todos los miembros del equipo tienen acceso al ejemplo proyecto de arranque. Para ello utilicen el URL https://github.com/ISIS1225DEVS/Reto4-202020-Template y sigan el proceso acostumbrado: Fork del proyecto a su espacio en GitHub y clone a su máquina local.

En la sección unificada del curso, en la sección de Datos, encontrarán la sección *"Ejemplo CitiBike"*. En este ejemplo encuentran 4 archivos **.CSV* de diferentes tamaños. Descárguelos y ténganlos a la mano para ir probando el proyecto.

2.2 CONSTRUIR EL MENÚ DE OPCIONES

En el archivo view.py cree un menú para todas las opciones del reto. Mantenga la misma convención. La opción 1 es para crear las estructuras de datos y la opción 2 para leer información; las siguientes opciones para cada uno de los requerimientos del reto.

Al finalizar cada opción del reto, se debe informar el tiempo que tomó la operación (similar a como se realiza en el proyecto de ejemplo SampleGraph).

2.3 DEFINICIÓN DE LAS ESTRUCTURAS DE DATOS

Lo primero que deben hacer es definir en el archivo *model.py* el grafo que será utilizado para guardar la información de los viajes en bicicleta.

Por ejemplo, una definición del grafo podría ser así:

```
  citibike['graph'] = gr.newGraph(datastructure='ADJ_LIST',
                                  directed=True,
                                  size=1000,
                                  comparefunction=compareStations)
```

Ahora es necesario implementar los métodos que adicionan un viaje entre dos estaciones:

El siguiente es un ejemplo de cómo podría ser agregar un viaje:

```
def addTrip(citibike, trip):
    """
    """
    origin = trip['start station id']
    destination = trip['end station id']
    duration = int(trip['tripduration'])
    addStation(citibike, origin)
    addStation(citibike, destination)
    addConnection(citibike, origin, destination, duration)
```

Cada estación se agrega como un vértice al grafo, si es que aún no existe.

```
def addStation(citibike, stationid):
    """
    Adiciona una estación como un vertice del grafo
    """
    if not gr.containsVertex(citibike ['graph'], stationid):
            gr.insertVertex(citibike ['graph'], stationid)
    return citibike
```

Se adiciona un arco al grafo que representa un viaje entre dos estaciones, el peso es la duración del mismo. Tenga en cuenta que, en el reto, pueden llegar muchos viajes entre un par de estaciones y debe guardar el promedio de las duraciones como peso del arco. Por ahora como ejemplo puede probar solo con un valor.

```
def addConnection(citibike, origin, destination, duration):
    """
    Adiciona un arco entre dos estaciones
    """
    edge = gr.getEdge(citibike [graph], origin, destination)
    if edge is None:
        gr.addEdge(analyzer[graph], origin, destination, duration)
    return citibike
```

NOTA: Este código es solo un ejemplo de cómo se puede realizar, usted puede y debe modificarlo para el propósito del reto 4.

Implemente la opción 1 del menú, para inicializar el grafo; es decir, conecte la vista, el controlador y el modelo.

2.4 CARGUE DE LOS ARCHIVOS

Para el reto 4, se deben cargar todos los archivos *.CSV que se encuentren en el directorio Data del proyecto (sin importar su nombre). Para esto, en el controlador puede utilizar funciones como las que se presentan en este ejemplo:

```
def loadTrips(citibike):
    for filename in os.listdir(cf.data_dir):
        if filename.endswith('.csv'):
            print('Cargando archivo: ' + filename)
            loadFile(analyzer, filename)
    return analyzer


def loadFile(citibike, tripfile):
    """
    """
    tripfile = cf.data_dir + tripfile
    input_file = csv.DictReader(open(tripfile, encoding="utf-8"),
                                delimiter=",")
    for trip in input_file:
        model.addTrip(citibike, trip)
    return citibike
```

NOTA: Este es solo un ejemplo de cómo se pueden leer todos los archivos de un directorio. Usted puede modificarlo o implementarlo como considere necesario. (cf.data_dir está definido en el archivo config.py. – import config as cf)

En este punto pruebe inicialmente con uno de los 4 archivos CSV que descargaron del sitio del curso.

2.5 Cálculo de componentes conectados

Una vez que tengan el grafo con las estaciones y rutas cargadas, el siguiente paso es calcular el número de componentes fuertemente conectados del grafo.

Para ello deben utilizar el algoritmo de Kosaraju, el cual pueden importar de la siguiente manera:

```
from DISClib.Algorithms.Graphs import scc
```

Para utilizar el algoritmo de Kosaraju, puede utilizar las siguientes funciones. numSCC invoca el algoritmo de Kosaraju para informar cuántos componentes fuertemente conectados se encontraron.

La segunda función informa si dos estaciones están en el mismo componente conectado.

```
def numSCC(graph, sc):
    sc = scc.KosarajuSCC(graph)
    return scc.connectedComponents(sc)

def sameCC(sc, station1, station2)
    return scc.stronglyConnected(sc, station1, station2)
```

Pruebe con identificadores de estaciones que hayan sido cargadas en los archivos de prueba para saber si están en el mismo componente conectado.

NOTA: Este es solo un ejemplo para ilustrar el uso del algoritmo de Kosaraju.

2.6 Pruebas con varios archivos

Ahora prueben con 2, 3 y 4 archivos CSV en el directorio Data. Prueben cuántos componentes conectados tiene el grafo cada vez que incrementan en un archivo más.

2.7 Cálculo del promedio

Ahora recuerden que cada vez que se carga un viaje entre dos estaciones, el peso es el tiempo; pero si esa ruta se cargó previamente, se debe modificar el peso por el promedio de los pesos de dicha ruta. Modifique el proyecto para que se guarde ahora el promedio de los tiempos asociados a los viajes reportados entre dos estaciones.

Pregunta 1: Presente esta tabla con la siguiente información

No de archivos cargados	No viajes cargados	No Vértices en el Grafo	No Arcos en el Grafo	No de componentes fuertemente conectados
1	9,999	530	1571	521
2	30,001	614	4124	593
3	315,000	738	42872	518
4	718,996	768	107622	10

2.8 COMPARTIR EL PRODUCTO DE LA PRACTICA CON LOS EVALUADORES

El resultado de este laboratorio es la implementación del requerimiento 1 del reto 4. Para entregar exitosamente sus resultados de este laboratorio, por favor recuerde las siguientes indicaciones:

a) Invitar al profesor y monitores del laboratorio asignados.
b) Incluir en el **README** del repositorio los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
c) Incluir en la carpeta *Docs* un documento en formato PDF que indique lo siguiente:
   i. Datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
   ii. La respuesta a la pregunta del laboratorio, en un documento PDF llamado respuestas-lab12.pdf, marcado con el nombre de los integrantes del grupo.

Generar una versión del repositorio llamada "Entrega Final – laboratorio 12" *(git commit -m "Entrega Final – laboratorio 12")* en el depósito de GitHub antes de la media noche (11:59 pm) del miércoles 11 de noviembre.

Recuerden que cualquier documento solicitado durante las actividades debe incluirse en el repositorio GIT y que solo se calificará hasta el último **COMMIT** realizado dentro de la fecha límite del miércoles 11 de noviembre de 2020, antes de la media noche (11:59 pm).