**Proyecto Final del Curso**

# Requerimiento 1: Reporte de información de compañías y taxis

## ¿Qué TAD utilizaron en la solución del requerimiento?

def newAnalyzer():

    analyzer={  'servicioIndex':None,

                'companias':None,

                'taxiIndex':None,

                'CompaniasConServicios':None,

                'CompaniasConTaxis':None }

    analyzer['servicioIndex']=lt.newList('SINGLE\_LINKED',compareIds)

    analyzer['companias']=om.newMap(omaptype='RBT',comparefunction=compareServicio)

    analyzer['taxiIndex']=om.newMap(omaptype='RBT',comparefunction=compareTaxi)

    analyzer['CompaniasConTaxis']=om.newMap(omaptype='RBT',comparefunction=compareTaxi)

    analyzer['CompaniasConServicios']=om.newMap(omaptype='RBT',comparefunction=compareTaxi)

    return analyzer

En línea con lo anterior, a continuación, se relacionan las estructuras y sus respectivas llaves usadas en las mismas:

* **Analyzer['servicioIndex']:** aquí se agregaron los datos generarles de todos y cada uno de los servicios, variable pivote “service”, lo cual fue usada como parámetro para las demás funciones con los campos de interés.
* **Analyzer['companias']:** La llave usada en este mapa fue service["company"]
* **Analyzer['taxiIndex']:** La llave usada en este mapa fue service['taxi\_id']
* **Analyzer['CompaniasConTaxis']:** La llave usada en este mapa fue service["company"] y el valor service["idTaxi”]
* **Analyzer['CompaniasConServicios']:** La llave usada en este mapa fue service["company"] y el valor service["trip\_id”]

## ¿Por qué eligieron esa estructura de datos?

Básicamente escogimos estas estructuras las cuales son las que mas se alinean al requerimiento, toda vez que si bien los registros de entrada (23 campos para cada registro), teníamos que agruparlos en categorías, una categoría era, (a) Compañía con la cantidad de taxis afiliados y (b) Compañía con la cantidad de servicios prestados.

En este sentido, la lista que recibe los servicios prestados, fuera almacenada en una Single linked list, y fue usada cómo pivote para este requerimiento y otros requerimientos, se preservaron inicialmente los 23 campos de cada servicio.

Los demás fueron Mapas Ordenados (Tabla de símbolos ordenadas), estos mapas nos permitían almacenar de una manera eficiente las Compañías y los campos que se requerían para este requerimiento que solo fueron: Nombre de compañía (company), identificador del viaje (trip\_id) y placa de taxi (taxi\_id). Con lo anterior como no quedan duplicados las llaves, solo con el llamado del método Size, ya teníamos las cantidades respectivas de Compañías, servicios y taxis.

## ¿Cuál es la complejidad estimada del algoritmo?

En general, los métodos usados fueron sobre Mapas Ordenados (Tabla de símbolos ordenadas, om.newMap(omaptype='RBT',comparefunction=compareTaxi)), por ser Map Red Black Tree, la complejidad para las operaciones usadas fue O(log (n)), por eso los cálculos se ven casi de inmediato.

En línea con el problema, podemos expresar que la complejidad es O(log (c)), donde C responde al numero de las compañías. Ahora bien, como para el cargue se hizo necesario realizar el barrido de todos los servicios, la mejor propuesta de complejidad total para el requerimiento es S\*log (C), donde C responde al número de las compañías y S al numero de servicios prestados por las diferentes compañías.

# Requerimiento 2: SISTEMA DE PUNTOS Y PREMIOS A TAXIS

## ¿Qué TAD utilizaron en la solución del requerimiento?

No se realizo

## ¿Por qué eligieron esa estructura de datos?

No se realizo

## ¿Cuál es la complejidad estimada del algoritmo?

No se realizo

# Requerimiento 3: CONSULTA DEL MEJOR HORARIO EN TAXI ENTRE 2 “COMMUNITY AREAS”

## ¿Qué TAD utilizaron en la solución del requerimiento?

def newGraph():

    graphr={  'community\_area':None,

                'grafo':None,

                'paths':None

           }

    graph['comunity\_area'] = m.newMap(numelements=14000, maptype='PROBING', comparefunction=compare\_community\_areas)

    graph['grafo'] = gr.newGraph(datastructure='ADJ\_LIST', directed=True, size=14000, comparefunction=compare\_community\_areas)

return graph

* **Graph["community\_area”]:** en este apartado se guardaron las informaciones de los viajes por medio de la llave id.
* **Graph[“grafo”]:** En este apartado se desarrolló en si la principal estructura que se uso para solucionar el requerimiento, el grafo, se usaron como vértices las community áreas descritas en el CSV, y los arcos son el promedio del tiempo (trip duration) de los trayectos entre cada community área con otra.
* **Graph[“Paths”]:** Este apartado se usó para implementar el algoritmo de Dijkstra.

## ¿Por qué eligieron esa estructura de datos?

Seleccionamos esta estructura de datos porque es la mas adecuada para manejar y guardar los datos de todos los trayectos. Es decir, es la mejor manera de guardar todas las ubicaciones importantes y la conexión entre las mimas. Adicionalmente, esta estructura es la mas apropiada para obtener los datos de la manera mas eficiente teniendo en cuenta el objetivo del requerimiento.

## ¿Cuál es la complejidad estimada del algoritmo?

La complejidad de este requerimiento está basada en el algoritmo de Dijkstra, el cual tiene una complejidad de **e log (n)** en el peor caso.