**Documento análisis reto 4**

Hecho por: Daniel Gomez y Jenifer Arce

**Contenido**

## Observaciones generales

- Link repositorio reto github: Reto4-G02
- Usamos la librería Prettytable() para mostrar resultados en el view
- Para visualizar los bono de follium utilizamos la ventana interactiva de VS Code (Link one drive video)
- Las variables de entorno usadas para el API fueron llamadas AMADEUS_CLIENT_ID y AMADEUS_CLIENT_SECRET

## Estructuras de datos realizadas

### Grafos

- ***Grafo no dirigido:***

Con el grafo no dirigido sabemos las rutas en dos direcciones.

Usado en requerimientos: 4,5

- ***Grafo dirigido:***

En este grafo tenemos todas las rutas del archivo

Usado en requerimientos: 2,3

**Ver aclaración construcción de grafos en comentarios al final del documento.**

### Tablas de símbolos

- ***Tabla de símbolos para los aeropuertos:***

En esta tabla estamos guardando toda la información relacionada a los aeropuertos, para así poder acceder a esta en tiempo O(1)

Usado en requerimientos: Todos

- ***Tabla de símbolos para las ciudades:***

Con estaba tabla tenemos la información de las ciudades y podemos acceder a estas en tiempo O(1)

Usado en requerimientos: 3,4,7

### Árboles y relacionados

- ***Árbol RBT con los grados de los vértices de los aeropuertos:***
- Con este árbol clasificamos los aeropuertos por su nivel de conectividad (arcos adyacentes de entrada + arcos adyacentes de salida, es decir, queda un árbol con el grado de los vértices).

- Lista ordenada con los nodos del anterior árbol de menor a mayor. Esta se usa para acceder a las últimas posiciones de manera eficiente al ejecutar el requerimiento1

Usado en requerimientos: 1

**Requerimiento 1**

Para este requerimiento hicimos un árbol RBT que agrupó a los aeropuertos por sus grados (degrees). Es así como para obtener el top 5 de los aeropuertos más conectados teníamos que recorrer con in-order (que tiene complejidad O(N), donde N son el número de nodos) para obtener los grados de menor a mayor. A partir de esto podíamos acceder a las últimas posiciones para conocer a los aeropuertos con mayor conectividad.

Cabe resaltar, que este árbol RBT queda con un tamaño de 366 nodos con el archivo large, por lo tanto, el recorrido que se hace en esta estructura es menor que la cantidad de aeropuertos de ambos grafos (hay 9075 aeropuertos/vértices).

La complejidad de este requerimiento es:

- Haciendo el recorrido inorder directamente desde la carga del catálogo: O(K)
- Recorrido inorder dentro de la función + acceder a los últimos aeropuertos: O(N) + O(K)

**Requerimiento 2**

```
    sccClusters=scc.KosarajuSCC(catalog["AeropuertosRutasGraph"])
    aeropuertosPertenecen=scc.stronglyConnected(sccClusters,
                          aeropuerto1,aeropuerto2)
    componentesConectados=sccClusters["components"]
```

Para este requerimiento se utilizó el algoritmo de Kosaraju para conocer los componentes conectados del grafo dirigido. La complejidad de este requerimiento es:

- Utilizar la función KosarajuSCC() : O(3(V+E))
- Número de componentes O(1) (accedemos a una llave del resultado anterior)
- Utilizar la función stronglyConnected para conocer si dos aeropuertos pertenecen al mismo clúster : O(K) ≈ O(1), en vista que se está accediendo a keys de mapas y se comparan los resultados

Total: O(3(V+E))

**Requerimiento 3**

Para este requerimiento primero se despliega las opciones para que el usuario seleccione la ciudad de origen y ciudad de destino que quiere. En caso de que la ciudad sea homónima se despliega un menú para que el usuario seleccioné la ciudad que desee. Como se accede al nombre de una ciudad usando la tabla de símbolos tiene una complejidad O(1). Si la ciudad es homónima en el peor de los casos esta búsqueda también es O(1) debido a la construcción del modelo, en vista que se retorna las opciones para el usuario y esté tiene que ingresar un número haciendo referencia a la posición de la ciudad.

Así mismo, debido a la construcción del catálogo al acceder a cualquier ciudad ya se sabe cual es el aeropuerto [1]más cercano.


```
{'city': 'Lisbon', 'city_ascii': 'Lisbon', 'lat':
38.7452, 'lng': -9.1604, 'country': 'Portugal',
'iso2': 'PT', 'iso3': 'PRT', 'admin_name':
'Lisboa', 'capital': 'primary', 'population':
'506654', 'id': '1620619017', 'AeropuertoCercano':
{'IATA': 'LIS', 'Distancia': 4.540734086311949}}
```


*Figura 1. Información de una ciudad*

Ya con esto conocemos los aeropuertos. Es así como para encontrar la ruta más corta se utilizó el algoritmo de Dijkstra.

- Usar jk.Dijkstra tiene una complejidad de O(logV)
- Después de esto se usa djk.distTo para conocer la distancia hasta la ciudad de llegada. Al revisar la disclib, esta función tiene una complejidad de O(1) pues lo que se hace es acceder a una key en una tabla de símbolos
- Finalmente, para conocer la ruta de aeropuertos se utiliza djk.pathTo, el cual tiene una complejidad en el peor caso de O(V)

Total: O(logV) + O(V) = O(V)

**Requerimiento 4**

Para este requerimiento fue necesario obtener un MST con el algoritmo de PRIM.

- Al revisar la Disclib podemos observar que el algoritmo de PRIM que es utilizado es PRIM Eager, por lo tanto la complejidad de hacer una búsqueda inicial es O(ElogV).
- Seguido a esto se utiliza Prim para relacionar con el vértice de origen.
- Con esto se utiliza una función modificada de edgesMSTeditada(), para conocer el camino del vértice. La complejidad en el peor caso será O(V)
- A partir de esto se ordena la lista anterior, lo cual tiene una complejidad de O(V)

[1] Cuando una ciudad no tiene aeropuerto la llave de 'AeropuertoCercano' queda de la siguiente forma {'IATA': None, Distancia: -100}

La complejidad será: O(ElogV) + 2 * O(V)

## Requerimiento 5

```
def efectoSuspension(catalog,aeropuerto):
    adyacentesAfectados=gr.adjacents(catalog["AeropuertosRutasGraph"],aeropuerto)
    dirigido=mp.get(catalog["AeropuertosTabla"],aeropuerto)["value"]["connections"]
    nodirigido=gr.degree(catalog["AeropuertosRutasDoblesGraph"],aeropuerto)
    respuestaLista=lt.newList("ARRAY_LIST")
    for aeropuertoAfectado in lt.iterator(adyacentesAfectados):
        info=mp.get(catalog["AeropuertosTabla"],aeropuertoAfectado)["value"]
        lt.addLast(respuestaLista,info)
    return respuestaLista,dirigido,nodirigido
```

- Tomar los adyacentes de un grafo tiene una complejidad de O(1), dado que se accede a una llave de una tabla de símbolos y se toman los adyacentes.
- Conocer el número de aeropuertos que son afectados en el grafo dirigido es una operación O(1), debido a que en la construcción del catálogo se crearon llaves que representaran las rutas totales que tiene un aeropuerto.

```
{'id': '2781', 'Name': 'Yuzhno-Sakhalinsk Airport',
'City': 'Yuzhno-sakhalinsk', 'Country': 'Russia',
'IATA': 'UUS', 'Latitude': '46.88869858','Longitude': '142.7180023',
 'connections': 6, 'inbound': 3, 'outbound': 3}
```

- Para conocer el impacto del no dirigido se calcula el grado del vértice, lo cual tiene una complejidad de O(1), pues se calcula el tamaño de la lista de los adyacentes.
- Por último, recorremos todos los adyacentes para tener una lista con la información comleta. Esto tiene una complejidad de O(N), donde N es el tamaño de los adyacentes de ese vértice
- Total: O(N) ; (donde N es el tamaño de los adyacentes)

## Requerimiento 6

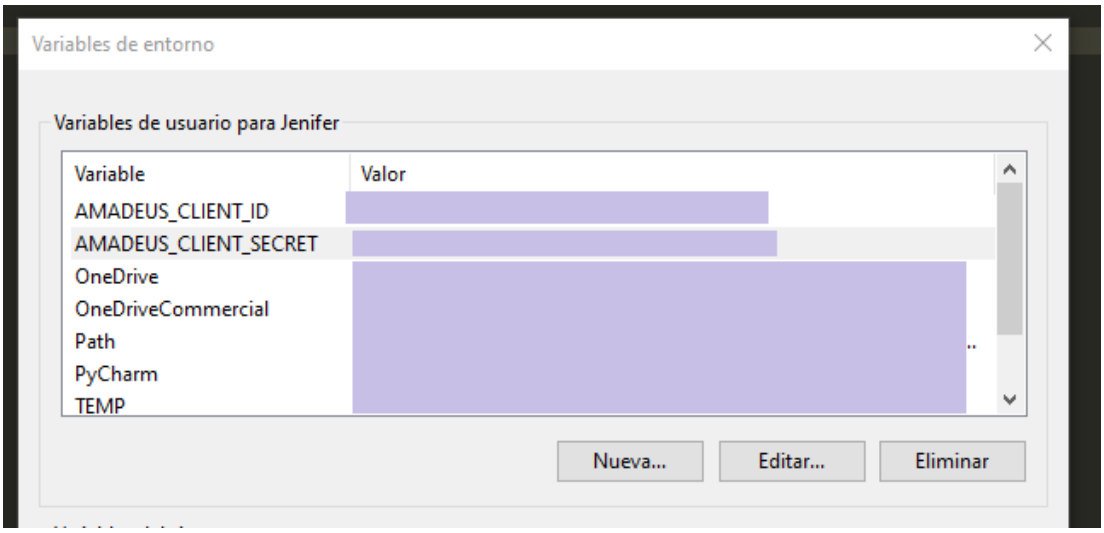
Los mapas de folium de cada uno de los requerimientos anteriores se visualizan inmediatamente después de obtener los resultados. Producir un mapa de folium tiene una complejidad de O(K). Para poner markers en los mapas de folium tiene una complejidad de O(N) dependiendo de la lista del requerimiento, por ejemplo, en el requerimiento 1 será una lista de 5 elementos, mientras que en los requerimientos como lo son el 5, en donde se tiene que poner markers de los aeropuertos afectados esto tendrá un tamaño mayor.

## Requerimiento 7 (Bono API)

**Aclaración:**

Para proteger las credenciales de acceso se crearon unas variables de entorno que tenían como valor las API key y secret de Amadeus.

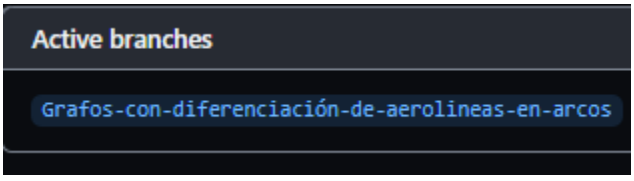
```
amadeus = Client(client_id=os.getenv('AMADEUS_CLIENT_ID'),
                 client_secret=os.getenv('AMADEUS_CLIENT_SECRET'))
```




La complejidad de este requerimiento es ligada al funcionamiento del API. Por lo tanto, la búsqueda de ciudades en el API es O(K). A partir de esto se obtienen valores relevantes como lo son la distancia del aeropuerto a la ciudad, y los códigos IATA de los aeropuertos. Con estos datos podemos aplicar la función del requerimiento 3, que tiene una complejidad O(V).

**Aclaración construcción arcos de grafos**

¡!! Hay una rama con el grafo diferenciando con aerolíneas en los arcos. Esta Branch se llama: Grafos-con-diferenciación-de-aerolineas-en-arcos



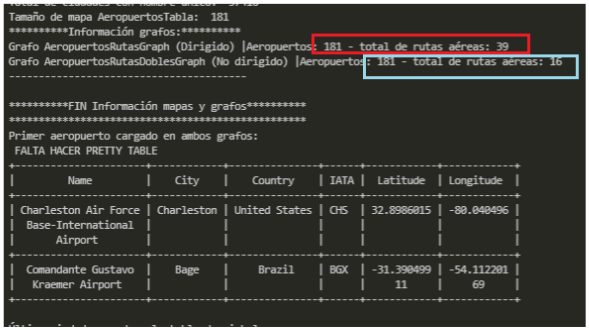

Para la construcción de los arcos se hizo una diferenciación de las aerolíneas. Cada arco entonces posee 4 llaves, que son:

```
edge = {'vertexA': va,
        'vertexB': vb,
        'weight': weight,
        "lineaA":linea}
```

Donde lineaA representa la aerolínea que hace esa ruta aérea. Para realizar estos ajustes de los grafos se añadieron unas funciones modificadas a la librería de DiscLib, las cuales son:

- newEdgeLinea (crea un arco con las llaves originales de la disclib pero con una llave adicional que es la línea aérea)
- addEdgeLinea (para añadir el arco con la modificación de la aerolínea)
- getEdgeLinea (para acceder a un arco que tenga un aeropuertoA, aeropuertoB, y una aerolínea especifica)

En las siguientes imágenes se muestra que nuestro modelo funciona tanto con los resultados del pdf actualizado como del anterior.




Reto nuestro - carga de datos con diferenciación de aerolíneas

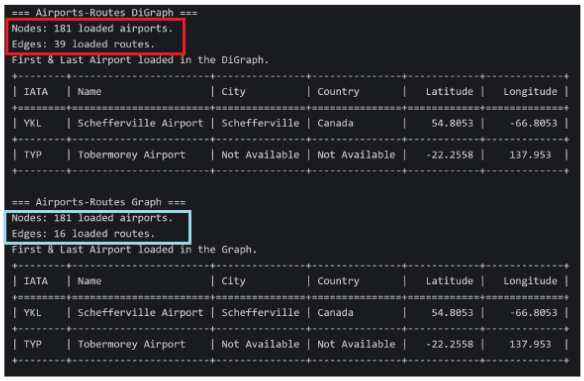



Foto del pdf del reto - Versión antes del 8 de diciembre


Reto nuestro - carga de datos con diferenciación de aerolíneas

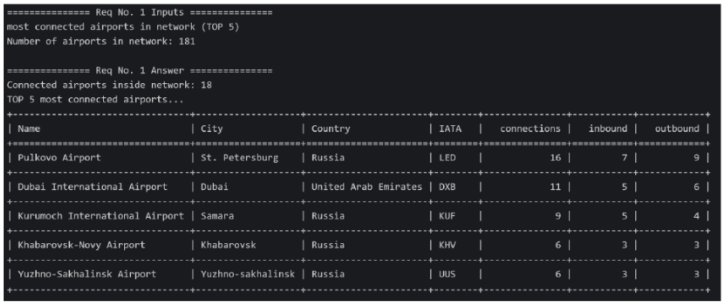
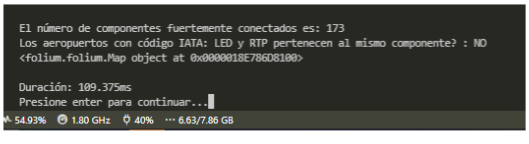



Foto del pdf del reto - Versión antes del 8 de diciembre




Reto nuestro - carga de datos con diferenciación de aerolíneas

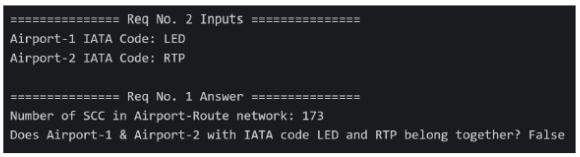
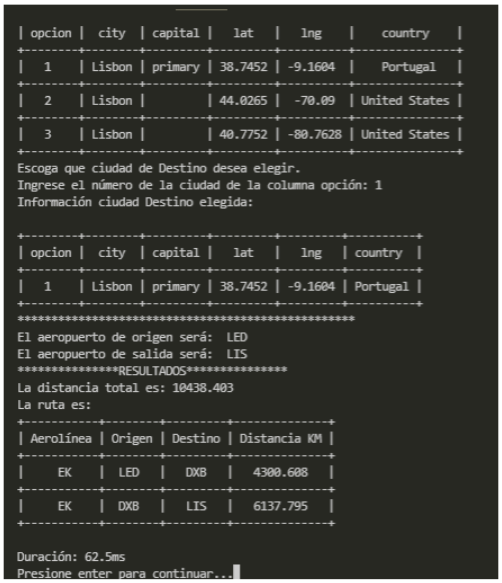



Foto del pdf del reto - Versión antes del 8 de diciembre



| opcion | city | capital | lat | lng | country |
|---|---|---|---|---|---|
| 1 | Lisbon | primary | 38.7452 | -9.1604 | Portugal |
| 2 | Lisbon | | 44.0265 | -70.09 | United States |
| 3 | Lisbon | | 40.7752 | -80.7628 | United States |

```
Escoga que ciudad de Destino desea elegir.
Ingrese el número de la ciudad de la columna opción: 1
Información ciudad Destino elegida:
```

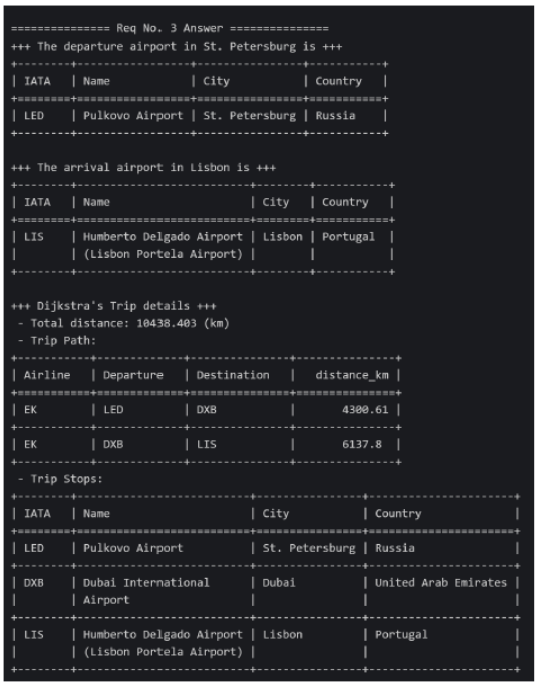
| opcion | city | capital | lat | lng | country |
|---|---|---|---|---|---|
| 1 | Lisbon | primary | 38.7452 | -9.1604 | Portugal |

```
**********
El aeropuerto de origen será:  LED
El aeropuerto de salida será:  LIS
**********RESULTADOS**********
La distancia total es: 10438.403
La ruta es:
```

| Aerolínea | Origen | Destino | Distancia KM |
|---|---|---|---|
| EK | LED | DXB | 4300.608 |
| EK | DXB | LIS | 6137.795 |

```
Duración: 62.5ms
Presione enter para continuar...
```

Reto nuestro - carga de datos con diferenciación de aerolíneas



```
=============== Req No. 3 Answer ===============
+++ The departure airport in St. Petersburg is +++
```

| IATA | Name | City | Country |
|---|---|---|---|
| LED | Pulkovo Airport | St. Petersburg | Russia |

```
+++ The arrival airport in Lisbon is +++
```

| IATA | Name | City | Country |
|---|---|---|---|
| LIS | Humberto Delgado Airport (Lisbon Portela Airport) | Lisbon | Portugal |

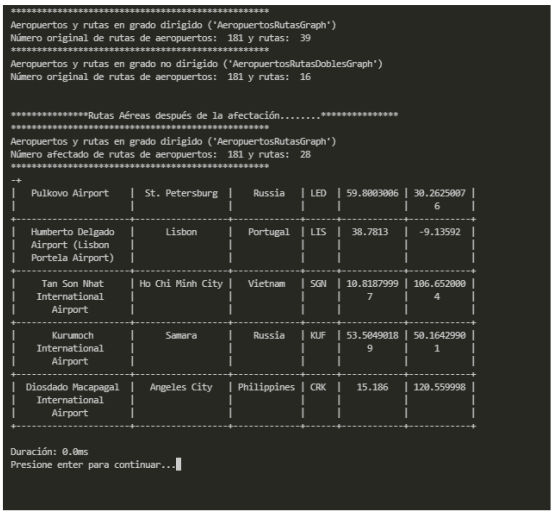
```
+++ Dijkstra's Trip details +++
 - Total distance: 10438.403 (km)
 - Trip Path:
```

| Airline | Departure | Destination | distance_km |
|---|---|---|---|
| EK | LED | DXB | 4300.61 |
| EK | DXB | LIS | 6137.8 |

```
 - Trip Stops:
```

| IATA | Name | City | Country |
|---|---|---|---|
| LED | Pulkovo Airport | St. Petersburg | Russia |
| DXB | Dubai International Airport | Dubai | United Arab Emirates |
| LIS | Humberto Delgado Airport (Lisbon Portela Airport) | Lisbon | Portugal |

Foto del pdf del reto - Versión antes del 8 de diciembre



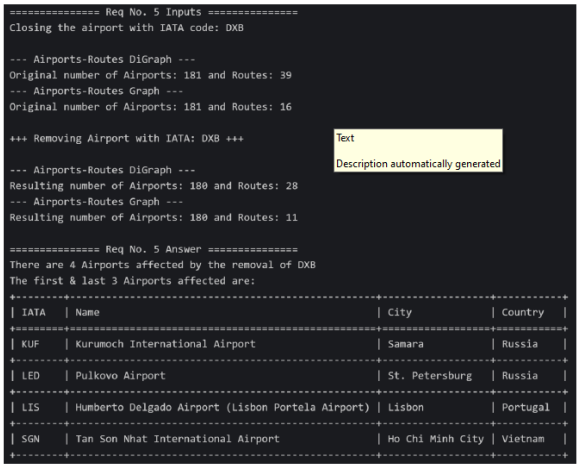
```
**************************************************
Aeropuertos y rutas en grado dirigido ('AeropuertosRutasGraph')
Número original de rutas de aeropuertos:  181 y rutas:  39
**************************************************
Aeropuertos y rutas en grado no dirigido ('AeropuertosRutasDoblesGraph')
Número original de rutas de aeropuertos:  181 y rutas:  16


***************Rutas Aéreas después de la afectación........***************
**************************************************
Aeropuertos y rutas en grado dirigido ('AeropuertosRutasGraph')
Número afectado de rutas de aeropuertos:  181 y rutas:  28
**************************************************
--+
```

| | | | | | | |
|---|---|---|---|---|---|---|
| Pulkovo Airport | St. Petersburg | Russia | LED | 59.8003006 | 30.2625007 | 6 |
| Humberto Delgado Airport (Lisbon Portela Airport) | Lisbon | Portugal | LIS | 38.7813 | -9.13592 | |
| Tan Son Nhat International Airport | Ho Chi Minh City | Vietnam | SGN | 10.8187999 | 106.652000 | 7 / 4 |
| Kurumoch International Airport | Samara | Russia | KUF | 53.5049018 | 50.1642990 | 9 / 1 |
| Diosdado Macapagal International Airport | Angeles City | Philippines | CRK | 15.186 | 120.559998 | |

```
Duración: 0.0ms
Presione enter para continuar...
```

Reto nuestro - carga de datos con diferenciación de aerolíneas



```
=============== Req No. 5 Inputs ===============
Closing the airport with IATA code: DXB

--- Airports-Routes DiGraph ---
Original number of Airports: 181 and Routes: 39
--- Airports-Routes Graph ---
Original number of Airports: 181 and Routes: 16

+++ Removing Airport with IATA: DXB +++

--- Airports-Routes DiGraph ---
Resulting number of Airports: 180 and Routes: 28
--- Airports-Routes Graph ---
Resulting number of Airports: 180 and Routes: 11

=============== Req No. 5 Answer ===============
There are 4 Airports affected by the removal of DXB
The first & last 3 Airports affected are:
```

| IATA | Name | City | Country |
|---|---|---|---|
| KUF | Kurumoch International Airport | Samara | Russia |
| LED | Pulkovo Airport | St. Petersburg | Russia |
| LIS | Humberto Delgado Airport (Lisbon Portela Airport) | Lisbon | Portugal |
| SGN | Tan Son Nhat International Airport | Ho Chi Minh City | Vietnam |

Text

Description automatically generated

Foto del pdf del reto - Versión antes del 8 de diciembre