**Análisis de Complejidad**

**Nombre:** Nicolas Merchan Cuestas

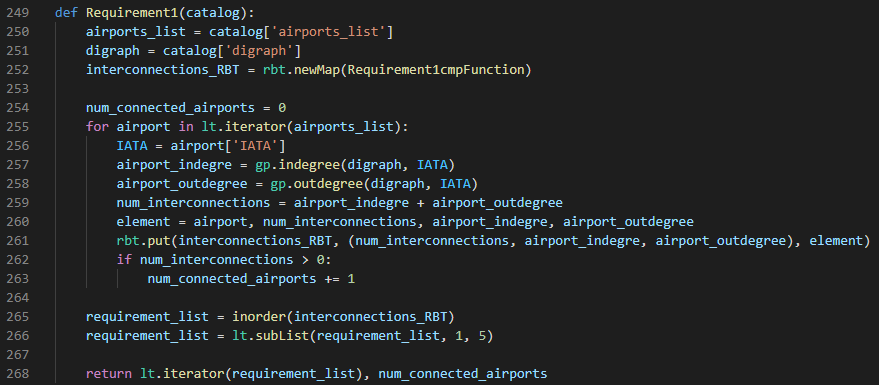
**Código:** 202112109

**Correo:** [n.merchan@uniandes.edu.co](mailto:n.merchan@uniandes.edu.co)

**Notas:**

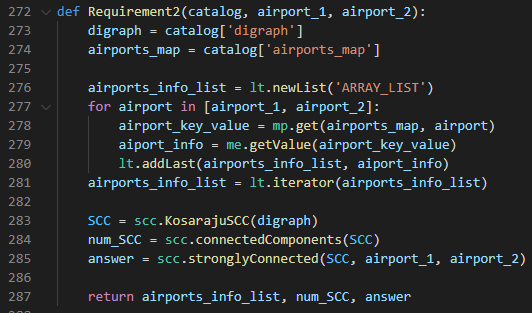
* Los resultados de las pruebas de tiempos de ejecución y las correspondientes gráficas de los mismos se encuentran en ‘*Datos - Análisis de Complejidad.xslx*’.
* El programa ‘Test\_Function.py’ realiza las pruebas de tiempos de ejecución de manera automática e ingresa los resultados en un archivo EXCEL llamado ‘Test\_Data.xlsx’.
* En el presente documento **‘E’** representará el número de arcos en los grafos considerados. Ello se debe a que en todas las pruebas se utilizaron los archivos completos de las ciudades y los aeropuertos (**worldcities-utf8.csv** y **airports-utf8-large.csv**, respectivamente). Por lo tanto, el parámetro de comparación utilizado en las pruebas fue el número de rutas registradas.

**Requerimiento 1**



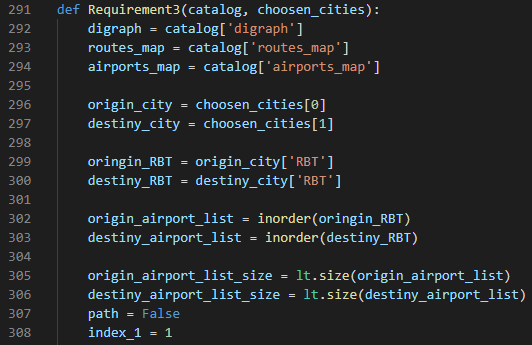
La complejidad del requerimiento 1 está dada por la función **Requirement1()**. El proceso implementado para encontrar los aeropuertos más interconectados requiere obtener el grado de todos los aeropuertos (vértices) del grafo dirigido e ingresar dichos aeropuertos en un RBT (Red-Black Tree) en base a al grado de cada uno. Después, se realiza un recorrido inorder sobre el RBT en cuestión para obtener una lista de los aeropuertos ordenados en función de su grado de manera descendente. Finalmente, la complejidad del requerimiento 1 respecto al número de rutas (arcos) es **O(1)**¸ dado que tanto el proceso de búsqueda en la lista de adyacencias, como el proceso de adición de los aeropuertos en el RBT no dependen del número de rutas.

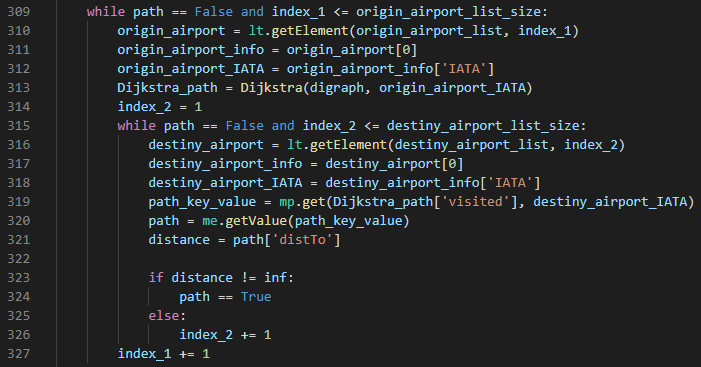
**Requerimiento 2**



La complejidad del requerimiento 2 está dada por la función **Requirement2()**. El proceso implementado para encontrar los componentes fuertemente conectados (SCC) en el grafo dirigido conformado por aeropuertos (vértices) y rutas (arcos) implica utilizar el algoritmo de Kosaraju. Así mismo, con el fin de conocer si dos aeropuertos pertenecen al mismo SCC solamente es necesario comparar los valores referentes al SCC al que pertenece cada aeropuerto y comparar si son iguales. La complejidad asociada al algoritmo de Kosaraju con respecto al número de arcos es **O(E)**, dado que es necesario recorrer todos los arcos al menos una vez en su implementación. Además, la verificación de pertenencia al mismo SCC de dos aeropuertos tiene una complejidad de **O(1)**, dado se consulta una tabla de Hash generada en el algoritmo.Finalmente, la complejidad del requerimiento 2 respecto al número de rutas es **O(E)**¸ dado que el proceso más complejo tiene dicha complejidad asociada.

**Requerimiento 3**

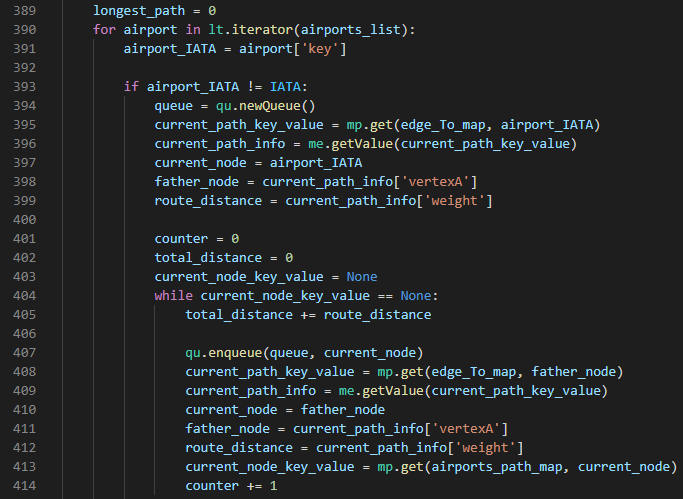


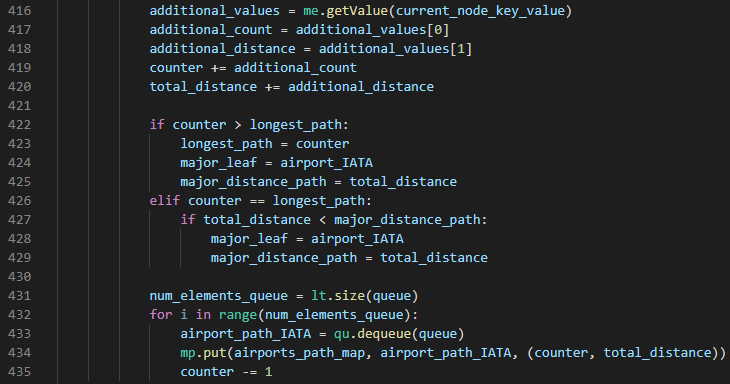


La complejidad del requerimiento 3 está dada por la función **Requirement3()**. El proceso implementado para encontrar la ruta más corta entre dos aeropuertos (vértices) por medio de rutas (arcos) en el grafo dirigido solamente implica utilizar el algoritmo de Dijkstra iniciando del aeropuerto de salida y consultando el camino generado al aeropuerto de llegada. Así mismo, antes de ejecutar el algoritmo de Dijkstra es necesario consultar las ciudades asociadas a un nombre homónimo. Ello se logra agrupando todas las ciudades homónimas en una lista que es el valor asociado a la llave del nombre de las ciudades en una tabla de Hash. La complejidad asociada al algoritmo de Dijkstra respecto al número de rutas es **O(log(E))**¸ dado que, en el caso particular de los datos analizados, el número de arcos entre aeropuertos incrementa de manera logarítmica con el incremento de rutas. Ello se debe a que existen muchas rutas en **routes-utf8-large.csv** que representan los mismos arcos en los grafos cargados. Además, la complejidad de la consulta de ciudades homónimas es **O(1)**¸ dado que solo se realiza una consulta en una tabla de Hash. Finalmente, la complejidad del requerimiento 3 respecto al número de rutas es **O(log(E))**¸ dado que el proceso más complejo tiene dicha complejidad asociada.

**Requerimiento 4**

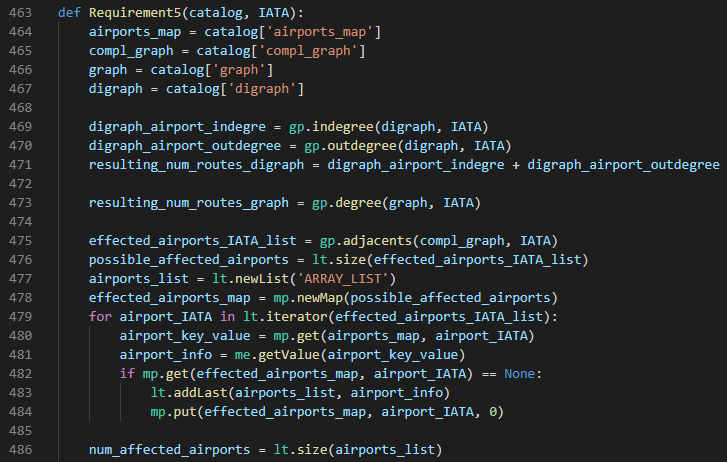






La complejidad del requerimiento 4 está dada por la función **Requirement4()**. El proceso implementado para encontrar el árbol de expansión mínima (MST) y la rama más larga de dicho MST implica utilizar el algoritmo Prim (Eager) desde el aeropuerto de salida en un grafo no dirigido conformado por aeropuertos (vértices) y rutas (arcos). Primero, se obtiene el MST respectivo, luego se obtiene una lista de los aeropuertos que hacen parte del MST y finalmente se obtiene la tabla de Hash que contiene a los aeropuertos como llave y como valor la información referente a sus padres y el peso de los arcos que los unen (distancia), todo ello se implementa entre las líneas 377 a 381. Posteriormente, se procede a buscar la longitud de cada rama del MST siguiendo la línea de sucesión dada por la tabla de Hash y utilizando la lista de aeropuertos que pertenecen al MST, todo ello se implementa entre las líneas 390 a 435. El proceso de búsqueda de la rama más larga se realiza paralelamente a la búsqueda de la longitud de cada rama. Así mismo, antes de ejecutar el algoritmo de Dijkstra es necesario consultar las ciudades asociadas a un nombre homónimo. Ello se logra agrupando todas las ciudades homónimas en una lista que es el valor asociado a la llave del nombre de las ciudades en una tabla de Hash. La complejidad asociada al algoritmo Prim (Eager) con respecto al número de rutas es Prim **O(E)**, dado que es necesario consultar la mayoría de los arcos del grafo no dirigido. También, el proceso de búsqueda de la rama más larga es **O(1)**, dado esta solo depende del número de aeropuertos. Además, la complejidad de la consulta de ciudades homónimas es **O(1)**¸ dado que solo se realiza una consulta en una tabla de Hash. Finalmente, la complejidad del requerimiento 4 respecto al número de rutas es **O(E)**¸ dado que el proceso más complejo tiene dicha complejidad asociada.

**Requerimiento 5**



La complejidad del requerimiento 5 está dada por la función **Requirement5()**. El proceso implementado para encontrar el número de rutas (arcos) afectadas por el cierre de un aeropuerto (vértice) en un grafo no dirigido implica encontrar el grado del aeropuerto cerrado y el número de aeropuertos adyacentes a dicho aeropuerto. Ello se logró creando un grafo no dirigido con todas las rutas registradas en **routes-utf8-large.csv** con el fin de facilitar la búsqueda de los aeropuertos adyacentes al aeropuerto cerrado. La complejidad del requerimiento 5 respecto al número de rutas es **O(E)**¸ dado que el número de rutas a analizar incrementa de manera lineal el tiempo de procesamiento del conteo de rutas afectadas.