Análisis de Reto 4 – Entrega Final

Integrantes:

- Juan David Vasquez Hernández - jd.vasquezh@uniandes.edu.co – 201914782
- Briseth Rodríguez Tovar – b.rodriguezt@uniandes.edu.co – 202116910

Complejidad de cada requerimiento:

Los códigos implementados para la solución de los requerimientos se muestran a continuación acompañados de sus respectivas complejidades.

En el caso de todos los algoritmos se usará a, r y c para representar el tamaño de airports, routes y cities.

Req1:

```
def top5Interconected(analyzer):
    digraph = analyzer['connections_free']
    airport_map = analyzer['airports']
    airports = gr.vertices(digraph)
    airports_connections = om.newMap(omaptype='RBT',comparefunction=
compareconnections)
    airport_network = 0
    for airport in lt.iterator(airports):
        outbound = gr.outdegree(digraph,airport)
        inbound = gr.indegree(digraph,airport)
        connections = outbound+inbound
        if connections > 0:
            airport_network += 1
        airportinfo = m.get(airport_map,airport)['value']
        airportinfo['outbound'] = outbound
        airportinfo['inbound'] = inbound
        if not om.contains(airports_connections,connections):
            airports_list = lt.newList()
            lt.addLast(airports_list,airportinfo)
            om.put(airports_connections,connections,airports_list)
        else:
            airports_list =
om.get(airports_connections,connections)['value']
            lt.addLast(airports_list,airportinfo)
    top5 = lt.newList()
    maxKey = om.maxKey(airports_connections)
    airports_max = om.get(airports_connections,maxKey)['value']

    while lt.size(top5)<5:
```

```python
        i = 1
        while lt.size(top5) < 5 and i <= lt.size(airports_max):
            airportinfo = lt.getElement(airports_max,i)
            lt.addLast(top5,airportinfo)
            i += 1

        om.deleteMax(airports_connections)
        maxKey = om.floor(airports_connections,maxKey)
        airports_max = om.get(airports_connections,maxKey)['value']

    return airport_network, top5
```

Complejidad: $O(a)+5*O(2loga_1)$

Para la función top5Interconected( ) la complejidad es $O(a)+5*O(loga_1)$ en el caso en el que el árbol se encuentre balanceado. En caso contrario, la complejidad de búsqueda vendría siendo $O(a)+5*O(2loga_1)$ dado que se usa un árbol RBT. Esto dado a que debemos realizar un recorrido de la lista de aeropuertos completa y luego un 5 veces un recorrido sobre la lista de llaves del árbol de aeropuertos con conexiones.

Req 2:

```python
def clusterCalculation(analyzer,IATA1,IATA2):
    """
    Calcula los componentes conectados del grafo usando el algoritmo de
Kosaraju
    """
    analyzer['components'] = scc.KosarajuSCC(analyzer['connections'])
    numCluster = scc.connectedComponents(analyzer['components'])
    sameCluster = scc.stronglyConnected(analyzer['components'],IATA1,IATA2)
    return numCluster,sameCluster
```

Complejidad: O(2*a)

Dado que la función clusterCalculation() debe realizar un doble recorrido sobre los elementos del grafo, la complejidad de este algoritmo es O(2*a), siendo que debemos recorrer dos veces la totalidad de la lista de los elementos en la lista de aeropuertos. Las funciones diferentes aplicadas de la librería scc corresponden a valores ya calculados en la propia implementación del algoritmo.

Req 3:

```python
#############################################################
```

```
def requirement_three(analyzer, city_departure, city_destiny):
    try:
        getCitiesByCity_1=getCitiesByCity1(analyzer, city_departure)
        getCitiesByCity_2=getCitiesByCity2(analyzer, city_destiny)
        return (getCitiesByCity_1, getCitiesByCity_2)
    except Exception as exp:
        error.reraise(exp, 'model:requirement_three')
#############################################################
def getCitiesByCity1(analyzer, city):
    try:
        existence = m.contains(analyzer['cities'], city)
        if existence:
            cities = m.get(analyzer['cities'], city)["value"]
            return cities
        return None
    except Exception as exp:
        error.reraise(exp, 'model:getCitiesByCity1')
#############################################################
def getCitiesByCity2(analyzer, city):
    try:
        existence = m.contains(analyzer['cities'], city)
        if existence:
            cities = m.get(analyzer['cities'], city)["value"]
            return cities
        return None
    except Exception as exp:
        error.reraise(exp, 'model:getCitiesByCity2')
#############################################################
def getCoordinates(analyzer, in_put_departure, in_put_destiny,
cities_departure, cities_destiny):
    try:
        choice_1= int(in_put_departure)
        count1= 1
        for element in lt.iterator(cities_departure):

            if count1==choice_1:

city_departureinfo=float(element["lat"]),float(element["lng"])
                break
            count1 += 1
        choice_2= int(in_put_destiny)
        count2=1
        for element in lt.iterator(cities_destiny):
            if count2==choice_2:
                city_destinyinfo=float(element["lat"]),float(element["lng"])
```

```
                break
            count2 += 1

        H1= haversine_r3(analyzer, city_departureinfo)
        H2= haversine_r3(analyzer, city_destinyinfo)
        I_need_all= route_short(analyzer, H1[0], H2[0])
        return (I_need_all, H1, H2)

    except Exception as exp:
        error.reraise(exp, 'model:getCoordinates')
#############################################################
def haversine_r3(analyzer, city_departureinfo):
    """
        Calculate the great circle distance in kilometers between two points
        on the earth (specified in decimal degrees)
        """
    try:
        min= 99**(19)
        info= ""
        cities= analyzer['airports']
        valueSet_cities= m.valueSet(cities)
        for element in lt.iterator(valueSet_cities):
            latitude_longitude=float(element["Latitude"]),
float(element["Longitude"])
            Haversine= haversine(city_departureinfo, latitude_longitude,
unit=Unit.KILOMETERS)
            if Haversine<min:
                min=Haversine
                info= element
        return info, min
    except Exception as exp:
        error.reraise(exp, "model:haversine_r3")
#############################################################
def route_short(analyzer, H1, H2):
    try:
        digraph= analyzer["connections"]
        airport_H1= H1["IATA"]
        airport_H2= H2["IATA"]
        route_s= djk.Dijkstra(digraph, airport_H1)
        distance_airports=djk.distTo(route_s, airport_H2)
        if djk.hasPathTo(route_s, airport_H2):
            I_need_all= djk.pathTo(route_s, airport_H2)
            return  I_need_all, distance_airports
    except Exception as exp:
        error.reraise(exp, "model:route_short")
```

Complejidad del algoritmo: O($a*loga_v$)

Para el 3er requerimiento, la función más importante corresponde a la aplicación del algoritmo Dijkstra, la cual posee una complejidad O($a*loga_v$) donde a_v corresponde a los arcos correspondientes a los vértices del grafo.

Req 4:

```
def createMST(analyzer):
    analyzer['search'] = pm.PrimMST(analyzer['airports_directed'])
    analyzer['prim'] =
pm.prim(analyzer['airports_directed'],analyzer['search'],'LIS')
```

Complejidad: O(a_v)

Siendo que para este requerimiento se debe usar la función primMST, se posee una complejidad O(a_v) basada en el recorrido de cada uno de los arcos del grafo para producir el MST correspondiente.

Req 5:

```
def addAirportAffected(analyzer, IATA):
    """
    Adiciona aeropuertos afectados a la estructura de almacenamiento para
estos
    """
    try:
        airports = analyzer['airports_affected']
        IATAcode = IATA
        m.put(airports,IATAcode,{'Digraph':0,'Graph':0})
    except Exception as exp:
        error.reraise(exp, 'model:addAirportInfo')

def addAirportAffectedValue(analyzer, routeinfo,routes):
    """
    Adiciona información de los aeropuertos afectados
    """
    try:
        airports = analyzer['airports_affected']
        IATAcode1 = routeinfo['Departure']
        IATAcode2 = routeinfo['Destination']
        values1 = m.get(airports,IATAcode1)['value']
```

```python
        values1['Digraph'] += 1
        if (routeinfo['Destination']+'-'+routeinfo['Departure'] in routes)
and (routeinfo['Departure']+'-'+routeinfo['Destination'] not in routes):
            values1['Graph'] += 1

        values2 = m.get(airports,IATAcode2)['value']
        values2['Digraph'] += 1
        if (routeinfo['Destination']+'-'+routeinfo['Departure'] in routes)
and (routeinfo['Departure']+'-'+routeinfo['Destination'] not in routes):
            values2['Graph'] += 1
    except Exception as exp:
        error.reraise(exp, 'model:addAirportInfo')
def evaluateClosureEffect(analyzer,IATA):
    digraph = analyzer['connections_free']
    airports = analyzer['airports']
    airports_affected = analyzer['airports_affected']
    degrees_digraph = m.get(airports_affected,IATA)['value']['Digraph']
    degrees_graph = m.get(airports_affected,IATA)['value']['Graph']
    airports_affected = gr.adjacents(digraph,IATA)
    if lt.size(airports_affected) > 0:
        known_airports = []
        airports_map = om.newMap(omaptype='RBT',comparefunction=compareIATA)
        for airport in lt.iterator(airports_affected):
            if airport not in known_airports:
                known_airports.append(airport)

om.put(airports_map,airport,m.get(airports,airport)['value'])

        ans_airports_affected = lt.newList()
        if om.size(airports_map) > 6:
            lowest_IATA = om.minKey(airports_map)
            highest_IATA = om.maxKey(airports_map)

lt.addLast(ans_airports_affected,om.get(airports_affected,lowest_IATA)['valu
e'])

lt.addLast(ans_airports_affected,om.get(airports_affected,highest_IATA)['val
ue'])
            t = 1
            for i in range(2):
                om.deleteMin(airports_map)
                om.deleteMax(airports_map)
                lowest_IATA = om.minKey(airports_map)
                highest_IATA = om.maxKey(airports_map)
```

```
lt.insertElement(ans_airports_affected,om.get(airports_map,lowest_IATA)['val
ue'],i)

lt.addLast(ans_airports_affected,om.get(airports_map,highest_IATA)['value'],
lt.size(ans_airports_affected)-t)
                t += 1
        else:
            for key in lt.iterator(om.keySet(airports_map)):

lt.addLast(ans_airports_affected,om.get(airports_map,key)['value'])

    return degrees_digraph,degrees_graph,ans_airports_affected
```

Complejidad: O(a)+O(loga)

Para esta función, la complejidad consiste en un valor de O(a)+O(loga) dado que se realiza un único recorrido sobre la carga de datos para obtener la información, y luego se realiza un segundo recorrido sobre el árbol RBT para la búsqueda de los valores.