DOCUMENTO DE ANÁLISIS RETO 4

Nicholas Barake 202020664

Jesed Domínguez 202011992

Requerimiento 1

```
def inter_points(catalog):
    Airports = lt.newList(datastructure="ARRAY_LIST")
    i_connections = lt.newList(datastructure="ARRAY_LIST")
    routes = catalog['DirectedConnections']
    vertexs = gr.vertices(routes)
    for vertex in lt.iterator(vertexs):
        actualDegree = gr.indegree(routes, vertex) + gr.outdegree(routes, vertex)
        element = {}
        element["key"] = vertex
        element["value"] = actualDegree
        lt.addLast(Airports, element)
    ms.sort(Airports, cmpDegree)
    for i in range(1, 6):
        index= (lt.size(Airports)+1)-i
        IATA= lt.getElement(Airports, index)
        lt.addLast(i_connections, IATA["key"])
    lista_final= lt.subList(Airports,1,5)
    return lista_final, lt.size(i_connections)
```

Complejidad temporal calculada: O(nlogm)

Conseguir la lista de vértices de un grafo es o(n), n siendo el número de vértices, ya que los vértices están guardados en una lista de adyacencia y se debe recorrer para meterlos a una lista. Luego, si recorremos eso estamos haciendo n*n. Agregando un mergesort, la complejidad total es O(nlogm), m siendo el tamaño de la lista de los aeropuertos interconectados, y n el número de vértices del grafo.

Requerimiento 2

```
def clusters(catalog,iata1,iata2):
    SCC= strong_c.KosarajuSCC(catalog['DirectedConnections'])
    all_SCC= SCC["components"]
    id_SCC= SCC["idscc"]
    same_cluster = None
    value1= mp.get(id_SCC, iata1)["value"]
    value2= mp.get(id_SCC, iata2)["value"]
    if value1 == value2 and value1 != None:
        same_cluster= True
    else:
        same_cluster= False

    return all_SCC, same_cluster
```

Complejidad temporal calculada: O(V+E)

La complejidad es la misma para el algoritmo de Kosaraju debido a que es la única función en el requerimiento que toma complejidad temporal. Ya que la función get de los mapas es constante, la complejidad se queda como O(V+E), donde V es el número de vértices en el grafo y E, el número de arcos.

Requerimiento 5

```
def itsclosed(catalog,iata):
    gr.removeVertex(catalog["DirectedConnections"], iata)
    gr.removeVertex(catalog["No_DirectedConnections"], iata)
    damage = gr.adjacents(catalog["DirectedConnections"], iata)
    lst_damage = lt.newList(datastructure="ARRAY_LIST")
    for IATA in lt.iterator(damage):
        if IATA not in lst_damage:
            lt.addLast(lst_damage, IATA)
    size_damaged = lt.size(lst_damage)
    return catalog, size_damaged, lst_damage
```

Complejidad temporal calculada: O((v+e)*n)

La complejidad que presentaría sería O((v+e)*n) refiriéndose a n como la cantidad de veces que se itera el for in. Esto simplemente se multiplica por v+e debido a que es la complejidad que ofrece el remover un vértice de un grafo.