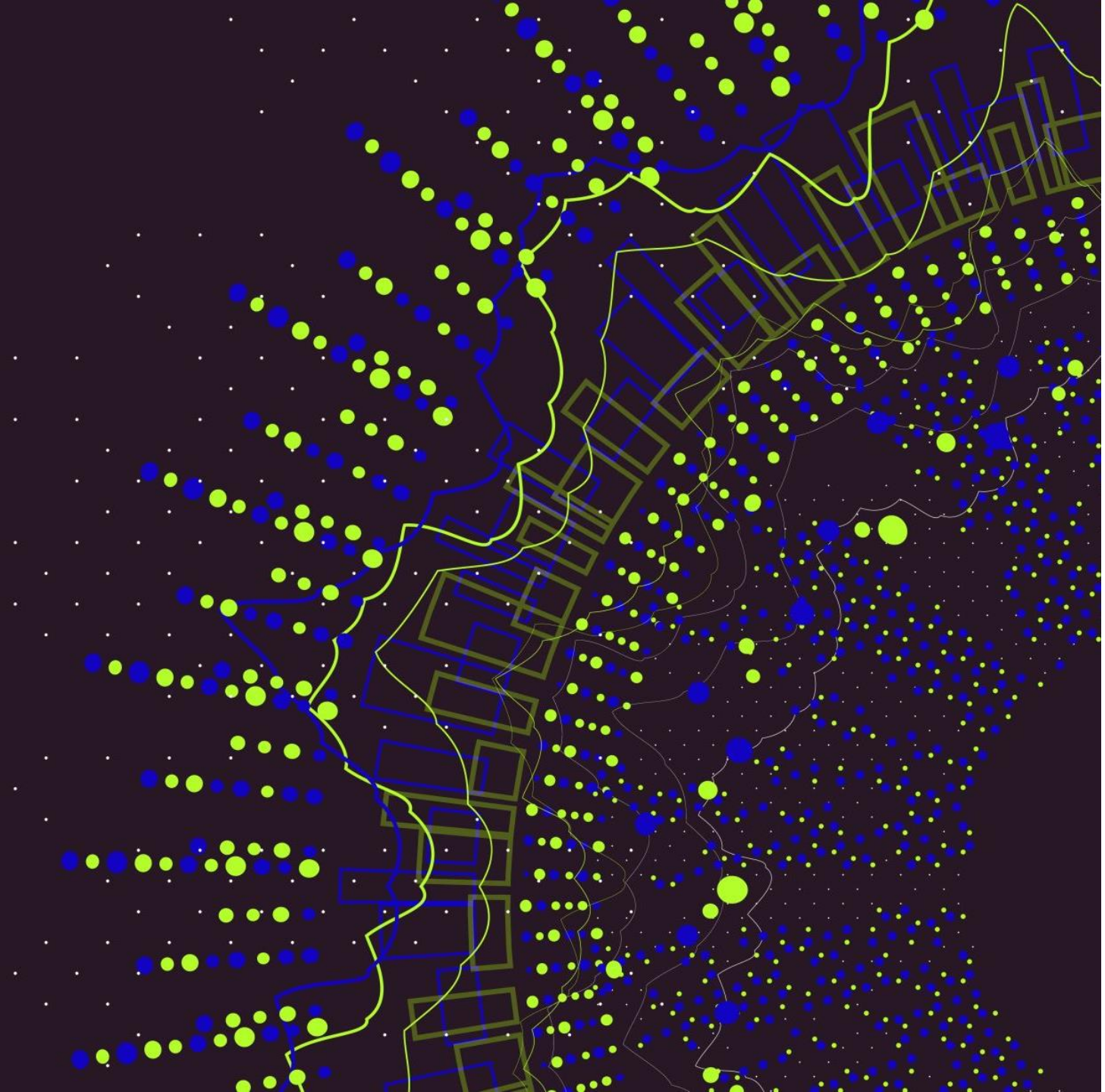

Análisis de complejidad reto 4

- JULIANA SOFIA AHUMADA ARCOS

201921471

J.AHUMADAA@UNIANDES.EDU.CO

- DANIELA PARRA MARTÍNEZ

202013036

D.PARRAM2@UNIANDES.EDU.CO

Requerimiento 1

```python
# REQUERIMIENTO 1 (ENCONTRAR PUNTOS DE INTERCONEXIÓN AÉREA)
def InterAerea(catalog):
    lista = lt.newList('ARRAY_LIST') #O(1)
    lista_c = lt.newList('ARRAY_LIST') #O(1)
    lista_f = lt.newList('ARRAY_LIST') #O(1)
    vertices = gr.vertices(catalog['Dirigido']) #O(1)
    for i in lt.iterator(vertices): #O(n)
        in_d = gr.indegree(catalog['Dirigido'], i) #O(1)
        out_d = gr.outdegree(catalog['Dirigido'], i) #O(1)
        total = in_d + out_d #O(1)
        if total != 0: #O(1)
            lt.addLast(lista, (total, i, str(in_d), str(out_d)))#O(1)
    orden = ordenamiento(lista) #O(1)
    mayores = lt.subList(orden, 1, 5) #O(1)

    for j in lt.iterator(mayores):#O(n)
        IATA = j[1] #O(1)
        entry = mp.get(catalog['Aeropuertos'], IATA) #O(1)
        value = me.getValue(entry) #O(1)
        lt.addLast(lista_f, (j[0], j[2], j[3], value)) #O(1)
        lt.addLast(lista_c, value) #O(1)

    cuantos = lt.size(orden) #O(1)
    Visualizar(lista_c, 'Requerimiento 1.html')
    return cuantos, lista_f['elements']
```

Complejidad:

- O(n)

Justificación:

```python
for i in lt.iterator(vertices): #O(n)
    in_d = gr.indegree(catalog['Dirigido'], i) #O(1)
    out_d = gr.outdegree(catalog['Dirigido'], i) #O(1)
    total = in_d + out_d #O(1)
    if total != 0: #O(1)
        lt.addLast(lista, (total, i, str(in_d), str(out_d)))#O(1)
orden = ordenamiento(lista) #O(1)
mayores = lt.subList(orden, 1, 5) #O(1)

for j in lt.iterator(mayores):#O(n)
    IATA = j[1] #O(1)
    entry = mp.get(catalog['Aeropuertos'], IATA) #O(1)
    value = me.getValue(entry) #O(1)
    lt.addLast(lista_f, (j[0], j[2], j[3], value)) #O(1)
    lt.addLast(lista_c, value) #O(1)
```

Requerimiento 2

```python
# REQUERIMIENTO 2 (ENCONTRAR CLÚSTERES DE TRÁFICO AÉREO)
def ClusterAereo(catalog, IATA_1, IATA_2):
    componentes = scc.KosarajuSCC(catalog['Dirigido']) # O(V+E)
    cuantos = scc.connectedComponents(componentes)
    pertenecen = scc.stronglyConnected(componentes, IATA_1, IATA_2)
    return cuantos, pertenecen
```

- **Complejidad:**
- O(V+E)

V= vértices

E= arcos

- **Justificación:**

```python
componentes = scc.KosarajuSCC(catalog['Dirigido']) # O(V+E)
```

Requerimiento 3

```
def RutaCorta(catalog, origen, destino, lista_1, lista_2):
    d_o = lt.newList('ARRAY_LIST') # O(1)
    d_d = lt.newList('ARRAY_LIST') # O(1)
    origen = lt.getElement(lista_1, int(origen)) # O(1)
    destino = lt.getElement(lista_2, int(destino)) # O(1)
    pais_o = origen['country'] # O(1)
    ln_o = origen['lng'] # O(1)
    lt_o = origen['lat'] # O(1)
    pais_d = destino['country'] # O(1)
    ln_d = destino['lng'] # O(1)
    lt_d = destino['lng'] # O(1)
    aeropuertos = mp.keySet(catalog['Aeropuertos']) # O(1)
    for i in lt.iterator(aeropuertos): # O(n)
        entry = mp.get(catalog['Aeropuertos'], i) # O(1)
        value =  me.getValue(entry) # O(1)
        if value['Country'] == pais_o: # O(1)
            lat = value['Latitude'] # O(1)
            lon = value['Longitude'] # O(1)
            rta = Haversine(float(lt_o), float(ln_o), float(lat), float(lon)) # O(1)
            lt.addLast(d_o, (rta, value['IATA'])) # O(1)
        if value['Country'] == pais_d: # O(1)
            lat = value['Latitude'] # O(1)
            lon = value['Longitude'] # O(1)
            rta = Haversine(float(lt_d), float(ln_d), float(lat), float(lon)) # O(1)
            lt.addLast(d_d, (rta, value['IATA'])) # O(1)

    orden_o = lt.lastElement(ordenamiento(d_o)) # O(1)
    orden_d = lt.lastElement(ordenamiento(d_d)) # O(1)

    aeropuerto_o = me.getValue(mp.get(catalog['Aeropuertos'], orden_o[1])) # O(1)
    aeropuerto_d = me.getValue(mp.get(catalog['Aeropuertos'], orden_d[1])) # O(1)

    Arbol = dj.Dijkstra(catalog['Dirigido'], aeropuerto_o['IATA']) # O(n**2) n= #vertices
    camino = dj.pathTo(Arbol, aeropuerto_d['IATA'])
    costo = dj.distTo(Arbol, aeropuerto_d['IATA'])

    return aeropuerto_o, aeropuerto_d, camino, costo, orden_o, orden_d
```

- **Complejidad:**
- O(n**2)
- n= número de vértices
- **Justificación:**

```
Arbol = dj.Dijkstra(catalog['Dirigido'], aeropuerto_o['IATA']) # O(n**2) n= #vertices
camino = dj.pathTo(Arbol, aeropuerto_d['IATA'])
costo = dj.distTo(Arbol, aeropuerto_d['IATA'])
```

Requerimiento 4

```python
def MillasViajero(catalog, millas, origen):
    cuantos = 0 # O(1)
    vertices = lt.newList('ARRAY_LIST') # O(1)
    lista = lt.newList('ARRAY_LIST') # O(1)
    millas = float(millas) * 1.60 # O(1)
    nuevo_grafo = mst(catalog) # O(n**2)
    x = dfs.DepthFirstSearch(nuevo_grafo, origen) # O(V+E)
    d = x['visited']['table']['elements'] # O(1)
    for i in d: # O(n)
        if i['key'] is not None and i['value']['edgeTo'] is not None: # O(1)
            lt.addLast(vertices, (i['value']['edgeTo'], i['key'])) # O(1)

    for j in lt.iterator(vertices): # O(n)
        peso = gr.getEdge(nuevo_grafo, j[0], j[1])['weight'] # O(1)
        cuantos += peso # O(1)
        tupla = j[0], j[1], peso # O(1)
        lt.addLast(lista, tupla) # O(1)

    nodos = gr.numVertices(nuevo_grafo) # O(1)
    faltantes = (cuantos - millas)/1.60 # O(1)

    return nodos, cuantos, lista, faltantes, millas
```

- **Complejidad:**
- O(n)
- **Justificación:**

La complejidad es O(n) por que en esa función se está usando el algoritmo de prim que tiene complejidad O(n**2) y el dfs que tiene complejidad O(V+E) lo cual sería O(n+n) y O(n**2) y en el peor caso sería O(n)

Requerimiento 5

```python
def AeropuertoCerrado(catalog, cerrado):
    n = lt.newList('ARRAY_LIST') # O(1)
    lista = lt.newList('ARRAY_LIST') # O(1)
    AD = gr.adjacents(catalog['Dirigido'], cerrado) # O(1)
    for i in lt.iterator(AD): # O(n)
        if lt.isPresent(n, i) is 0: # O(1)
            lt.addLast(n, i) # O(1)
            entry = mp.get(catalog['Aeropuertos'], i) # O(1)
            value = me.getValue(entry) # O(1)
            lt.addLast(lista, value) # O(1)

    cuantos = lt.size(lista) # O(1)

    primeros_3 = lt.subList(lista, 1, 3) # O(1)
    ultimos_3 = lt.subList(lista, len(lista) - 3, 3) # O(1)

    Visualizar(lista, 'Requerimiento 5.html')

    return cuantos, primeros_3, ultimos_3
```

- **Complejidad:**
- O(n)
- **Justificación:**

```python
for i in lt.iterator(AD): # O(n)
    if lt.isPresent(n, i) is 0: # O(1)
        lt.addLast(n, i) # O(1)
        entry = mp.get(catalog['Aeropuertos'], i) # O(1)
        value = me.getValue(entry) # O(1)
        lt.addLast(lista, value) # O(1)
```

Bono 7

```
def Visualizar(lista, nombre_mapa):
    aeropuertos = lista
    mapa = folium.Map()
    tooltip = '¿Cuál aeropuerto es?: ¡Click para ver!'
    for i in lt.iterator(aeropuertos):
        nombre = i['Name'],(i['IATA'])
        folium.Marker([i['Latitude'], i['Longitude']], tooltip=tooltip,
                      popup=nombre).add_to(mapa)
    mapa.save(nombre_mapa)
```

- **Complejidad y Justificación:**

La librería Folium normalmente están muy optimizadas y tienden hacer constantes.