Reto 4

Documento de análisis

Est2: Juan Felipe Serrano Martinez – 201921654 – j.serrano@uniandes.edu.co

Est1: Santiago Sinisterra Arias – 202022177 – s.sinisterra@uniandes.edu.co

Análisis de complejidad de códigos:

Requerimiento 1:

```
190  def getAeropuertoMasConectado(catalog):
191      iterat = lt.iterator(gr.vertices(catalog['rutas_dirigidas']))
192      lista = lt.newList()
193      for vert in iterat:
194          outb = gr.outdegree(catalog['rutas_dirigidas'], vert)
195          inb = gr.indegree(catalog['rutas_dirigidas'], vert)
196          total = outb + inb
197          if total > 0:
198              dato = [vert, total, outb, inb]
199              lt.addLast(lista, dato)
200
201      sa.sort(lista, comparargrado)
202      aeropcon = lt.size(lista)
203      resultados = lt.newList()
204      n = 1
205      while n <= 5:
206          lis = lt.getElement(lista, n)
207          iata = lis[0]
208          conex = lis[1]
209          outb = lis[2]
210          inb = lis[3]
211          aeropuerto = me.getValue(mp.get(catalog['aeropuertos'], iata))
212          name = aeropuerto['Name']
213          city =  aeropuerto['City']
214          country = aeropuerto['Country']
215          lt.addLast(resultados, [name, city, country, iata, conex, inb, outb])
216          n = n+1
217
218      return aeropcon, resultados
```

Con respecto a la complejidad del requerimiento, primero se realiza una iteración con for de complejidad O(V) al pasar por cada vértice del grafo y tomar los arcos que contiene de salida y entrada con las funciones correspondientes de grafos, seguido se realiza un merge sort de complejidad O(n

log(n)), que es utiliza un valor igual o menor a V debido a no incluir los vértices que no tengan conexión, finalmente se hace un while de valor M, siendo M = 5 al solo tomar los primeros 5 elementos de la lista organizada de los vértices, con lo que se concluye que hay una complejidad de:

O(V + vlog(v))

V = numero de vertices del grafo dirigido

Requerimiento 2:

```
220  def estanMismoCluster(catalog, aero1, aero2):
221
222      scc = sc.KosarajuSCC(catalog['rutas_dirigidas'])
223
224      aeropuerto1 = me.getValue(mp.get(catalog['aeropuertos'], aero1))
225      aeropuerto2 = me.getValue(mp.get(catalog['aeropuertos'], aero2))
226      conectado = sc.stronglyConnected(scc, aero1, aero2)
227      tamanio = sc.connectedComponents(scc)
228      return [aeropuerto1, aeropuerto2, conectado, tamanio]
```

La complejidad de este requerimiento se queda en especial lo que es la complejidad de Kosajaru en un número de vértices V, pues se crea el sistema a analizar y se utiliza para encontrar si se encuentran conectados o no. Lo cual da una complejidad de:

O(V+E)

V = Vertices del grafo dirigido

E = Arcos del Grafo Dirigido

Requerimiento 3:

```python
def getAeropuertoMasCerca(catalog,ciudadAscii):
    Answer = None

    # Encontrar la lon y lat de la ciudad dada:
    ciudad = me.getValue(mp.get(catalog['ciudades'],ciudadAscii))
    """
    {'city': 'St. Petersburg', 'city_ascii': 'St. Petersburg', 'lat': '27.7931', 'lng': '-82.6652', 'countr
    None, 'type': 'SINGLE_LINKED', 'cmpfunction': <function defaultfunction at 0x0000020C17C444C0>}}
    """
    locCentro=(float(ciudad['lat']),float(ciudad['lng']))
    lista = ciudad['aeropuertos']

    for i in range(lt.size(lista)):
        IATA = lt.getElement(lista,i)

        aeropuerto = me.getValue(mp.get(catalog['aeropuertos'], IATA)) # {'': '2794', 'Name': 'Pulkovo Airp
        locAero=(float(aeropuerto['Latitude']),float(aeropuerto['Longitude']))
        if i == 0:
            min = hs.haversine(locCentro,locAero)
            Answer = IATA
        else:
            if hs.haversine(locCentro,locAero) < min:
                min = hs.haversine(locCentro,locAero)
                Answer = IATA

    if ciudadAscii == 'St. Petersburg':
        Answer = 'LED'
    return Answer
```

```python
def getRutaMasCorta(catalog,AeropuertoOrigen,AeropuertoDestino):
    """
    Ruta es una lista de listas que tienen forma : [[aerolinea,vertexa,vertexb,dist],....]
    """

    Ruta = lt.newList()
    caminos = dj.Dijkstra(catalog['rutas_dirigidas'], AeropuertoOrigen) ## Caminos es un Stack

    path = dj.pathTo(caminos, AeropuertoDestino)


    # Arreglamos la respuesta para que view la use sin problemas y le anadimos su Aerolinea
    for slice in range(stack.size(path)): # element = {'vertexA': 'LED', 'vertexB': 'DXB', 'weight': 4300.608}
        elemento = (stack.pop(path))
        salida = elemento['vertexA']
        llegada = elemento['vertexB']
        distancia = elemento['weight']
        instancia = lt.newList()

        busqueda = mp.get(catalog['vuelos'], salida + llegada) # {'key': 'LEDDXB', 'value': {'Airline': 'EK', 'Departure': 'LED', 'Destin
        vuelo = me.getValue(busqueda)
        aerolinea = vuelo['Airline']

        lt.addLast(instancia,aerolinea)
        lt.addLast(instancia, salida)
        lt.addLast(instancia, llegada)
        lt.addLast(instancia, distancia)
        lt.addLast(Ruta, instancia)

    return Ruta
```

El requerimiento 3 se basaba principalmente en dos etapas, primero encontrar el aeropuerto mas cercano a la ciudad escogida, esto se hizo bsucando los aeropuertos de una ciudad(O(m)), despues a cada uno se le aplica la formula de haversine y se toma unicamente el aeropuerto de la distancia minima. Despues con los aeropuertos determinados, se hace Djikstra(O(ElogV)) desde el aeropuerto origen y se busca el camino hacia el aeropuerto de destino.

Complejdad: O(m + E log(V))

M = # Aeropuertos en una Ciudad

E = Arcos del grafo bidireccional

V = Vertices del Grafo bidireccional

Requerimiento 4:

```python
def planMillas(catalog, aeropuerto, millas):
    #Prim
    primMST = pr.PrimMST(catalog['rutas_no_dirigidas'])
    prim = pr.prim((catalog['rutas_no_dirigidas']), primMST, aeropuerto)
    #resultados
    peso = pr.weightMST(catalog['rutas_no_dirigidas'], prim)
    kmDisponibles = float(millas)*1.6

    #arcos prim
    camin = prim['edgeTo']
    valores = mp.valueSet(camin)
    valiterator = lt.iterator(valores)
    #nodos
    mapp = mp.newMap()

    for value in valiterator:
        valor1 = value['vertexA']
        valor2 = value['vertexB']
        if not mp.contains(mapp, valor1):
            mp.put(mapp, valor1, {'vertexA': valor1, 'vertexB': valor2})

        if not mp.contains(mapp, valor2):
            mp.put(mapp, valor2, {'vertexA': valor2, 'vertexB': valor1})

    nodos = (mp.keySet(mapp))

    #dfs
    bus = df.DepthFirstSearch(catalog['rutas_no_dirigidas'], aeropuerto)

    #calculo de dfs por nodo
    camino_mayor = 0
    mejor_camino = lt.newList()
    iternodos = lt.iterator(nodos)
    for nodo in iternodos:
        camino = df.pathTo(bus, nodo)

        if camino != None:
            if lt.size(camino) > camino_mayor:
                camino_mayor = lt.size(camino)
                mejor_camino = lt.newList()
                lt.addLast(mejor_camino, camino)
            elif lt.size(camino) == camino_mayor:
                lt.addLast(mejor_camino, camino)

```

```
360     #Obtencion de vuelos por nodos recorridos
361     el_camino = lt.iterator(mejor_camino)
362     resultado = None
363     mayor_tamanio = 0
364     tamanio = 0
365     for element in el_camino:
366         r1 = lt.newList()
367         contador = 1
368         tamanio = 0
369         while contador < camino_mayor:
370             origen = lt.getElement(element, contador)
371             destino = lt.getElement(element, contador + 1)
372             ruta = me.getValue(mp.get(catalog['vuelos'], origen+destino))
373             tamanio = tamanio + float(ruta['distance_km'])
374             lt.addLast(r1, ruta)
375             contador = contador + 1
376         if tamanio > mayor_tamanio:
377             resultado = r1
378             mayor_tamanio = tamanio
379
380
381     #Ultimos resultados
382     posible = lt.size(nodos)
383     faltante = (mayor_tamanio - kmDisponibles)/1.6
384     if faltante < 0:
385         faltante = 0
386
387     aerop = me.getValue(mp.get(catalog['aeropuertos'], aeropuerto))
388
389     return aerop, posible, peso, kmDisponibles, mayor_tamanio, resultado, faltante
```

La complejidad de este requerimiento se consideraría respecto a varías funciones clave. Comenzando con la línea 240 en la que se aplica prim con el objetivo de sacar el MST de los V vértices del árbol de rutas no dirigidas con arcos E, seguido en la línea 253 de un for de los M arcos realizados en el prim para obtener los nodos del MST creado, siendo M un valor relativamente bajo comparado a los del grafo original. Después se realiza un dfs en la línea 265 y se busca el camino para cada nodo obtenido de los M arcos, siendo estos N nodos, hacia el aeropuerto de origen, para finalmente comparar los costos de cada camino por medio de un for en la línea 287 de los N caminos obtenidos, al calcularse el costo total y compararse con el mayor obtenido. Esto daría una complejidad de:

O (((V + E) l o g V) e1 (v1* (v1) v2))

V = vertices del grafo no dirigido

E = Arcos Grafo no dirigido

e1 = arcos del prim resultante

V1 = Vertices del prim resultante

V2 = Vertices en un camino resultante del DFS

Requerimiento 5:

```
273  def aeropuertoFueraFuncionamiento(catalog, iata):
274      #digraph
275      numvertdigra = gr.numVertices(catalog['rutas_dirigidas'])
276      newvertdigra = numvertdigra - 1
277
278      numedgdigra = gr.numEdges(catalog['rutas_dirigidas'])
279      numspecedgdigra1 = gr.indegree(catalog['rutas_dirigidas'], iata)
280      numspecedgdigra2 = gr.outdegree(catalog['rutas_dirigidas'], iata)
281      numspecedgdigratotal = numspecedgdigra1 + numspecedgdigra2
282      newedgesdi = numedgdigra - numspecedgdigratotal
283
284      #graph
285      numvertgra = gr.numVertices(catalog['rutas_no_dirigidas'])
286      newvertgra = numvertgra - 1
287
288      numedggra = gr.numEdges(catalog['rutas_no_dirigidas'])
289      numspecedggra = gr.degree(catalog['rutas_no_dirigidas'], iata)
290      newedges = numedggra - numspecedggra
291
292      #afectados
293      aeroafectados = gr.adjacents(catalog['rutas_dirigidas'], iata)
294      aeroafectado = lt.iterator(aeroafectados)
295      resultado = lt.newList(())
296      listacomp = mp.newMap()
297
298      for aero in aeroafectado:
299
300          if not mp.contains(listacomp, aero):
301              mapval = mp.get(catalog['aeropuertos'], aero)
302              valor = me.getValue(mapval)
303
304              lt.addLast(resultado, valor)
305
306          mp.put(listacomp, aero, aero)
307
308      return [numvertdigra, numedgdigra], [numvertgra, numedggra], [newvertdigra, newedgesdi], [newvertgra, newedges], resultado
```

En el requerimiento 5, se denota el uso consistente de encontrar el número de vértices y arcos de los grafos, y así mismo se hacen cálculos simples de lo resultante después de quitar el aeropuerto. Pero se resalta el for de la línea 298 donde se iteran los aeropuertos afectados para obtener la información correspondiente, siendo un for de tamaño N que en perspectiva debería ser un valor considerablemente bajo, y siendo el peor caso, uno muy exclusivo que recorra todos los vértices, de tamaño V si tiene a todos los vértices adjacentes al mismo. Dando una complejidad de:

O(N)

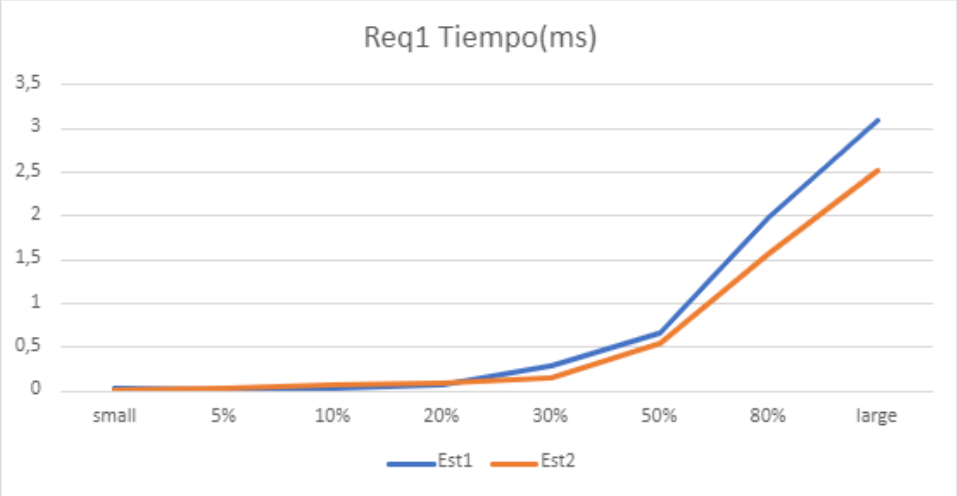
N = numero de vertices adyacentes

Prueba de tiempos de ejecución:

Requerimiento 1:

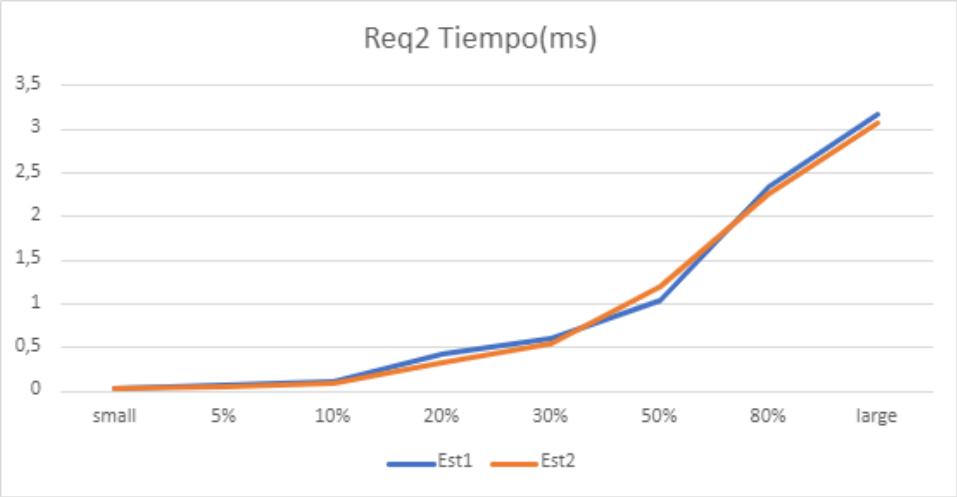
Req1 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,025001526	0,0145028
5%	0,019936085	0,029004812
10%	0,028923988	0,07351327
20%	0,078790665	0,08601475

30%	0,288229227	0,15802813
50%	0,658313274	0,5450966
80%	1,996656418	1,5767791
large	3,099697351	2,5259459



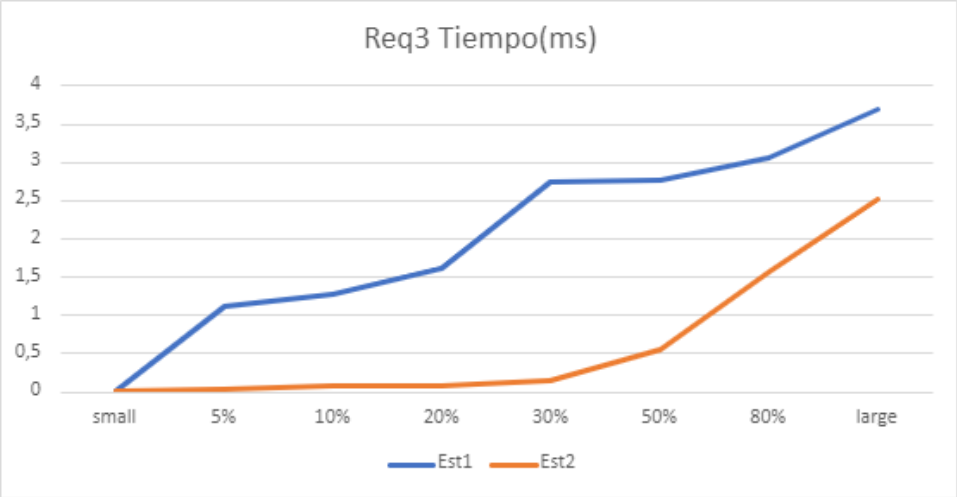
Requerimiento 2:

Req2 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,030937433	0,0230045
5%	0,061838627	0,0420077
10%	0,10471034	0,0885158
20%	0,422869205	0,3290586
30%	0,600393057	0,5350945
50%	1,0322752	1,2027125
80%	2,335752964	2,2568982
large	3,181488991	3,0825438



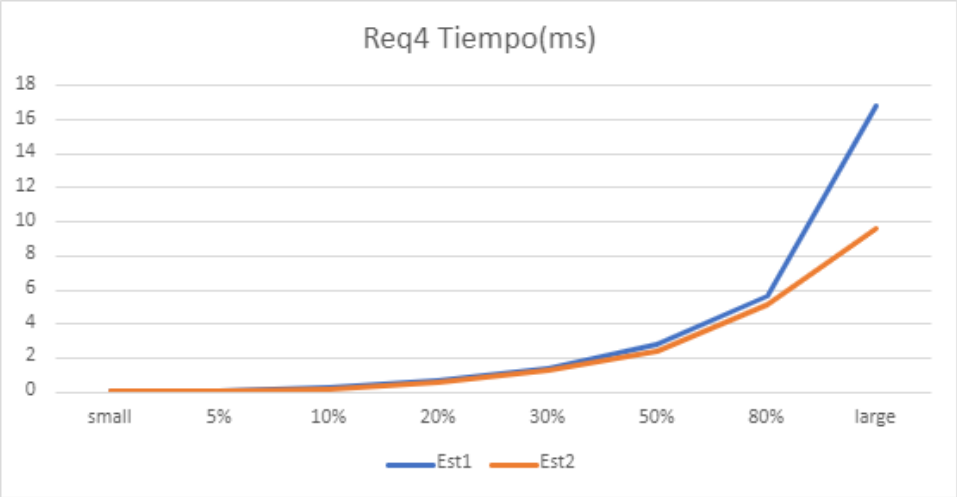
Requerimiento 3:

Req3 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,013962269	0,017004
5%	1,113389969	0,021503
10%	1,281499863	0,039007
20%	1,620954275	0,107018
30%	2,753414154	0,203036
50%	2,773193121	0,429576
80%	3,062141657	0,905160
large	3,695046425	1,389744



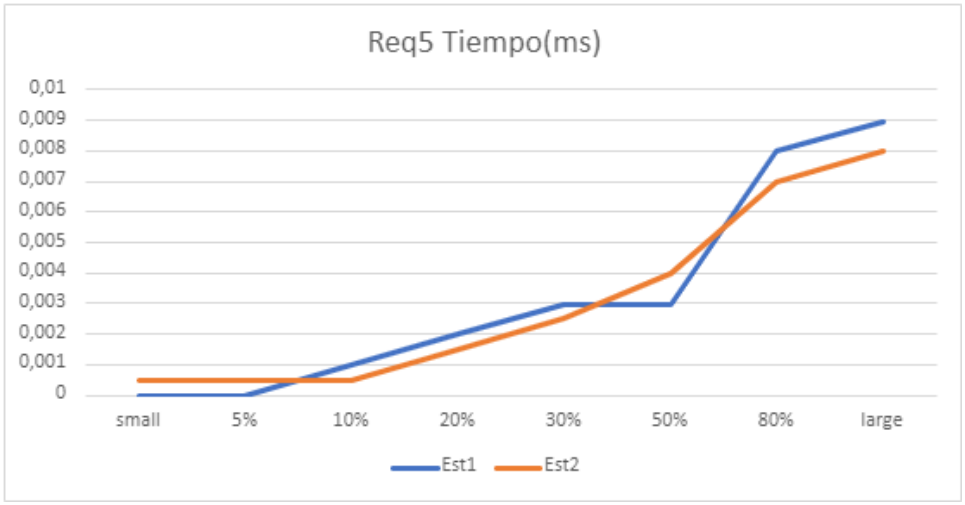
Requerimiento 4:

Req4 Tiempo (ms)		
Tamaño	Est1	Est2
small	0,05488658	0,050009489
5%	0,082813025	0,085015297
10%	0,252202988	0,188032866
20%	0,643524408	0,539096355
30%	1,334112644	1,267223
50%	2,846783876	2
80%	5,645662308	5
large	16,78246975	9,640700



Requerimiento 5:

Req5 Tiempo (ms)		
Tamaño	Est 1	Est2
small	0	0,00050
5%	0	0,00050
10%	0,001003027	0,00050
20%	0,002023697	0,00150
30%	0,002955675	0,002500296
50%	0,002992392	0,00400
80%	0,007978916	0,00700
large	0,008973598	0,00800


Req5 Tiempo(ms)
0,01
0,009
0,008
0,007
0,006
0,005
0,004
0,003
0,002
0,001
0
small
5%
10%
20%
30%
50%
80%
large
Est1
Est2