

RETO #4

Juanita Gil j.gila2@uniandes.edu.co - 202111556

Gabriela Carvajal g.carvajal@uniandes.edu.co - 202111058

Complejidad temporal:

Req 1:

<pre>170 def inter_dirigido(catalog): 171 lst_vertices = gr.vertices(catalog['gd_aero_ruta']) 172 mapa = om.newMap(omaptype='BST', comparefunction=compare2) 173 for element in lt.iterator(lst_vertices): 174 arcos_llegada = int(gr.indegree(catalog['gd_aero_ruta'], element)) 175 arcos_salida = int(gr.degree(catalog['gd_aero_ruta'], element)) 176 suma = int(arcos_llegada + arcos_salida) 177 if om.contains(mapa,element)== False: 178 om.put(mapa, element, suma) 179 else: 180 pass 181 llaves_mapa= om.keySet(mapa) 182 size = om.size(mapa) 183 lst_while = lt.newList() 184 for element in lt.iterator(llaves_mapa): 185 lst_elemento = lt.newList() 186 pareja = om.get(mapa, element) 187 valor = me.getValue(pareja) 188 conec = str('conecctions: '+ str(valor)) 189 lt.addLast(lst_elemento, element) 190 lt.addLast(lst_elemento, conec) 191 lt.addLast(lst_while, lst_elemento) 192 lst_final = lt.subList(lst_while, 1, 5) 193 return lst_final, size</pre>	2O(n), ya que se cumplen ambos for
--	------------------------------------

Req 2:

<pre>205 def connectedComponents(catalog, aero1, aero2): 206 """ 207 Calcula los componentes conectados del grafo dirigido 208 Se utiliza el algoritmo de Kosaraju 209 """ 210 catalog['components'] = scc.KosarajuSCC(catalog['gd_aero_ruta']) 211 numsc = int(scc.connectedComponents(catalog['components'])) 212 aeros_cluster = [scc.stronglyConnected(catalog['components'], aero1, aero2)] 213 if aeros_cluster == False: 214 str(print("Los aeropuertos no pertenecen al mismo componente")) 215 else: 216 str(print("Si pertenecen al mismo componente")) 217 return numsc 218</pre>	O(V + E), ya que esa es la complejidad temporal del algoritmo kosaraju
--	--

Req 3:

<pre>230 def rutamascorta(catalog, origen, destino): 231 orig= mp.get(catalog['city'], origen) 232 dest=mp.get(catalog['city'], destino) 233 return (orig, dest)</pre>	
--	--

```

234 def selecruta(catalog, opcionCiudad, opcionCiudad2, orig, dest):
235     for linea in orig:
236         if linea['country']== opcionCiudad:
237             latitudciudad=linea['lat']
238             longitudciudad=linea['lng']
239     for linea in dest:
240         if linea['country']== opcionCiudad2:
241             latitudciudad=linea['lat']
242             longitudciudad=linea['lng']
243     latitudciudad=radians(latitudciudad)
244     longitudciudad=radians(longitudciudad)
245     latitudciudad=radians(latitudciudad)
246     longitudciudad=radians(longitudciudad)
247
248     x=10
249     aeropuertosCiudadOrigen=lt.newList()
250     aeropuertosCiudadDestino=lt.newList()
251     lleno=False
252     while lleno == False:
253         for linea in catalog['aeropuertos']:
254             lati= linea['Latitude']
255             longi=linea['Longitude']
256             distancia0= acos(sin(latitudciudad)*sin(lati)+cos(latitudciudad)*cos(lati)*cos(longitudciudad-longi)
257             distanciaD= acos(sin(latitudciudad)*sin(lati)+cos(latitudciudad)*cos(lati)*cos(longitudciudad-longi)
258             if distancia0 <= x:
259                 lt.addLast(aeropuertosCiudadOrigen, linea['Name'])
260             if distanciaD <= x:
261                 lt.addLast(aeropuertosCiudadDestino, linea['Name'])
262
263             if lt.size(aeropuertosCiudadOrigen) >=1 and lt.size(aeropuertosCiudadDestino >=1):
264                 lleno= True
265         else:
266             x+=10
267
268     return (aeropuertosCiudadOrigen,aeropuertosCiudadDestino)

```

Req 4:

```

292 def millas(catalog, origen, millasDisp):
293     km_plan_millas = millasDisp*1.60
294     estructura_search = prim.PrimMST(catalog['g_una_ruta'])
295     arcos_relajados = prim.prim(catalog['g_una_ruta'],estructura_search, origen)
296     lista_nodos = prim.weightMST(catalog['g_una_ruta'],estructura_search)
297     dfsSearch= dfs.DepthFirstSearch(catalog['g_una_ruta'], origen)
298     prim.edgesMST()
299     for element in lt.iterator(lista_nodos):
300         if element== origen:
301             pass
302         else:
303             dfs.pathTo(dfsSearch, element)
304     pass

```

Req 5:

```

328 def aeropuertoCerrado(catalog, codigoIATA):
329     aeroAfectados= gr.adjacents(catalog['gd_aero_ruta'], codigoIATA)
330     listaAfectados=lt.newList()
331     for linea in aeroAfectados:
332         for aeropuerto in catalog['aeropuertos_g']:
333             if linea == aeropuerto['IATA']:
334                 afectado=lt.newList()
335                 nombre=aeropuerto['Name']
336                 ciudad= aeropuerto['City']
337                 IATA= aeropuerto['IATA']
338                 lt.addLast(afectado, nombre)
339                 lt.addLast(afectado, ciudad)
340                 lt.addLast(afectado, IATA)
341                 lt.addLast(listaAfectados, afectado)
342     total= lt.size(listaAfectados)
343     return listaAfectados, total

```

$O(n^2)$