RETO #4

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

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

Complejidad temporal:

Req 1:

```
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_map= om.keySet(mapa)
182     size = om.size(mapa)
183     lst_while = lt.newList()
184     for element in lt.iterator(llaves_map):
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
```

2O(n), ya que se cumplen ambos for

Req 2:

```
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     numscc = 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 numscc
```

O(V + E), ya que esa es la complejidad temporal del algotimo kosaraju

Req 3:

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

O(n) ya que es la complejidad del get

```python
def selecruta(catalog, opcionCiudad, opcionCiudad2, orig, dest):
    for linea in orig:
        if linea['country']== opcionCiudad:
            latitudciudado=linea['lat']
            longitudciudado=linea['lng']
    for linea in dest:
        if linea['country']== opcionCiudad2:
            latitudciudadd=linea['lat']
            longitudciudadd=linea['lng']
    latitudciudado=radians(latitudciudado)
    longitudciudado=radians(longitudciudado)
    latitudciudadd=radians(latitudciudadd)
    longitudciudadd=radians(longitudciudadd)

    x=10
    aeropuertosCiudadOrigen=lt.newList()
    aeropuertosCiudadDestino=lt.newList()
    lleno=False
    while lleno == False:
        for linea in catalog['aeorpuertos']:
            lati= linea['Latitude']
            longi=linea['Longitude']
            distanciaO= acos(sin(latitudciudado)*sin(lati)+cos(latitudciudado)*cos(lati)*cos(longitudciudado-longi))
            distanciaD= acos(sin(latitudciudadd)*sin(lati)+cos(latitudciudadd)*cos(lati)*cos(longitudciudadd-longi))
            if distanciaO <= x:
                lt.addLast(aeropuertosCiudadOrigen, linea['Name'])
            if distanciaD <= x:
                lt.addLast(aeropuertosCiudadDestino, linea['Name'])

        if lt.size(aeropuertosCiudadOrigen) >=1 and lt.size(aeropuertosCiudadDestino >=1):
            lleno= True
        else:
            x+=10

    return (aeropuertosCiudadOrigen,aeropuertosCiudadDestino)
```

o(n) multiplicado por las veces que se repita el while

Req 4:

```python
def millas(catalog, origen, millasDisp):
    km_plan_millas = millasDisp*1.60
    estructura_search = prim.PrimMST(catalog['g_una_ruta'])
    arcos_relajados = prim.prim(catalog['g_una_ruta'],estructura_search, origen)
    lista_nodos = prim.weightMST(catalog['g_una_ruta'],estructura_search)
    dfsSearch= dfs.DepthFirstSearch(catalog['g_una_ruta'], origen)
    prim.edgesMST()
    for element in lt.iterator(lista_nodos):
        if element== origen:
            pass
        else:
            dfs.pathTo(dfsSearch, element)
    pass
```

Req 5:

```python
def aeropuertoCerrado(catalog, codigoIATA):
    aeroAfectados= gr.adjacents(catalog['gd_aero_ruta'], codigoIATA)
    listaAfectados=lt.newList()
    for linea in aeroAfectados:
        for aeropuerto in catalog['aeropuertos_g']:
            if linea == aeropuerto['IATA']:
                afectado=lt.newList()
                nombre=aeropuerto['Name']
                ciudad= aeropuerto['City']
                IATA= aeropuerto['IATA']
                lt.addLast(afectado, nombre)
                lt.addLast(afectado, ciudad)
                lt.addLast(afectado, IATA)
                lt.addLast(listaAfectados, afectado)
    total= lt.size(listaAfectados)
    return listaAfectados, total
```

O(n^2)