ANÁLISIS RETO 4

David Leonardo Almanza Márquez – 202011293 – d.almanza@uniandes.edu.co
Laura Daniela Arias Flórez – 202020621 – l.ariasf@uniandes.edu.co

COMPLEJIDAD REQUERIMIENTO 1

```
def getClusters(catalog, LP1, LP2):
    #Kosaraju, O(V+E)
    SCCc = scc.KosarajuSCC(catalog['connections'])
    numComponentes1 = scc.connectedComponents(SCCc)
    boolean = False
    infoLP1 = me.getValue(mp.get(catalog['landing_points'], LP1))
    infoLP2 = me.getValue(mp.get(catalog['landing_points'], LP2))
    for cable1 in lt.iterator(infoLP1['cables']):
        cable_LP1 = formatVertex(LP1, cable1['cable_name'])
        for cable2 in lt.iterator(infoLP2['cables']):
            #Doble loop, O(N*M) donde N y M son los tamanos de las listas de cables de lo
s landing points preguntados
            cable_LP2 = formatVertex(LP2, cable2['cable_name'])

            boolean = scc.stronglyConnected(SCCc, cable_LP1, cable_LP2)
            if boolean:
                break

    return numComponentes1, boolean
```

Uso extra de memoria dependiente de la implementación en la librería del algoritmo de kosaraju. Esta función toma alrededor de 2 segundos en ejecutarse

COMPLEJIDAD REQUERIMIENTO 2

```
def Req2(catalog):
    #O(N) donde N es la cantidad de landing points
    valores = mp.valueSet(catalog['landing_points'])
    for valor in lt.iterator(valores):
        if lt.size(valor['cables']) > 1:
            try:
                print('El landing point en ', valor['info']['name'], ' con identificador
', valor['info']['landing_point_id'], ' tiene ', lt.size(valor['cables']), ' cables.')
            except:
                print('El landing point en ', valor['info']['name'], ' con identificador
', '----', ' tiene ', lt.size(valor['cables']), ' cables.')
```

No hay uso extra de memoria. Este algoritmo toma menos de un segundo en ejecutarse.

COMPLEJIDAD REQUERIMIENTO 3

```
def Req3(catalog, pais1, pais2):
    #dijkstra: O(E*log(V))
    capital1 = me.getValue(mp.get(catalog['countries'], pais1))['info']['CapitalName']

    cables1 = me.getValue(mp.get(catalog['landing_points'], capital1))['cables']
    cable1 = lt.getElement(cables1, 1)['cable_name']

    capital2 = me.getValue(mp.get(catalog['countries'], pais2))['info']['CapitalName']
    cables2 = me.getValue(mp.get(catalog['landing_points'], capital2))['cables']
    cable2 = lt.getElement(cables2, 1)['cable_name']

    LP1 = formatVertex(capital1, cable1)
    LP2 = formatVertex(capital2, cable2)
    search = dijsktra.Dijkstra(catalog['connections'], LP1)
    distancia = dijsktra.distTo(search, LP2)

    path = dijsktra.pathTo(search, LP2)
    return path, distancia
```

Uso extra de memoria dependiente de la implementación en la librería del algoritmo de dijkstra. Esta función toma alrededor de 3 segundos en ejecutarse.

COMPLEJIDAD REQUERIMIENTO 5

```
def getAffectedCountries(catalog, landingpoint):
    lp = me.getValue(mp.get(catalog['landing_points'], landingpoint))
    affectedcountries = lt.newList('ARRAY_LIST')
    for cable in lt.iterator(lp['cables']):
        #O(N) donde N es la cantidad de cables que se conectan con ese landing point
        vertexname = landingpoint + "-" + cable['cable_name']
        affectedvertices = gr.adjacents(catalog['connections'], vertexname)
        for vertex in lt.iterator(affectedvertices):
            #O(M) donde M es la la cantidad de vertices adyacentes al nodo específico
            vertexcountry = me.getValue(mp.get(catalog['landing_points'], vertex.split("-
")[0]))['info']["name"].split(", ")[-1]
            if not lt.isPresent(affectedcountries, vertexcountry):
                lt.addLast(affectedcountries, vertexcountry)
    return affectedcountries
```

Uso de memoria extra por parte de las listas affectedcountries y affectedvertices. Esta función toma alrededor de 2 segundos en ejecutarse