**Hoja análisis de complejidad**

Gregorio Salazar

Valentina Uribe

Requerimiento 1.

def connected\_components(analyzer,landing\_name1,landing\_name2):

    #req 1

    """

    Calcula los componentes conectados del grafo

    Se utiliza el algoritmo de Kosaraju

    """

    landing\_id1=name\_to\_id(analyzer,landing\_name1)

    landing\_id2=name\_to\_id(analyzer,landing\_name2)

    cable1=lt.getElement(m.get(analyzer['landing\_points\_cables'],landing\_id1)['value'],1)

    cable2=lt.getElement(m.get(analyzer['landing\_points\_cables'],landing\_id2)['value'],1)

    verta=format\_vertex(landing\_id1,cable1)

    vertb=format\_vertex(landing\_id2,cable2)

    analyzer['components'] = scc.KosarajuSCC(analyzer['connections'])

    connected=scc.stronglyConnected(analyzer['components'],verta,vertb)

    return scc.connectedComponents(analyzer['components']),connected

La función name\_to\_id tiene complejidad O(V). Kosaraju tiene complejidad O(V+E), por lo que el requerimiento tiene complejidad O(V+E)

1631.722799999996 [ms] 4338.046875 [kb]

Requerimiento 2.

def critical\_landing\_points(analyzer):

    #req 2

    cables=analyzer['landing\_points\_cables']

    info=analyzer['landing\_points\_info']

    lista=lt.newList('ARRAY\_LIST')

    for landing\_id in lt.iterator(m.keySet(cables)):

        num\_cables=lt.size(m.get(cables,landing\_id)['value'])

        landing\_point=m.get(info,landing\_id)['value']

        country=landing\_point['country']

        name=landing\_point['name']

        entry=(name,country,landing\_id,num\_cables)

        lt.addLast(lista,entry)

    lista=merge.sort(lista,compare\_entry\_req2)

    return lista

Este requerimiento es un simple recorrido por todos los landing points, por lo que la complejidad es O(V).

137.38899999999558 [ms] 159.60546875 [kb]

Requerimiento 3.

def minimum\_path(analyzer, pais1, pais2):

    #req 3

    """

    Calcula los caminos de costo mínimo desde la estacion initialStation

    a todos los demas vertices del grafo

    """

    countries=analyzer['countries']

    capital1=m.get(countries,pais1)['value']['CapitalName']

    cable1=lt.getElement(m.get(analyzer['landing\_points\_cables'],capital1)['value'],1)

    verta=format\_vertex(capital1,cable1)

    capital2=m.get(countries,pais2)['value']['CapitalName']

    cable2=lt.getElement(m.get(analyzer['landing\_points\_cables'],capital2)['value'],1)

    vertb=format\_vertex(capital2,cable2)

    path=None

    camino=None

    total=0

    analyzer['paths'] = djk.Dijkstra(analyzer['connections'], verta)

    if djk.hasPathTo(analyzer['paths'], vertb):

        path = djk.pathTo(analyzer['paths'], vertb)

        camino=lt.newList('ARRAY\_LIST')

        for arco in lt.iterator(path):

            landing\_id1=arco['vertexA'].split('~')[0]

            landing\_name1=m.get(analyzer['landing\_points\_info'],landing\_id1)['value']['name']

            landing\_id2=arco['vertexB'].split('~')[0]

            landing\_name2=m.get(analyzer['landing\_points\_info'],landing\_id2)['value']['name']

            distance=arco['weight']

            total+=distance

            if landing\_id1!=landing\_id2:

                entry=(landing\_name1,landing\_name2,distance)

                lt.addLast(camino,entry)

    return camino,total

Primero se ejecutan varias operaciones de O(1). Luego, Dijkstra tiene una complejidad de O(E log V). Luego se reconstruye el camino de Dijkstra pero esta complejidad es O(E), por lo que finalmente la complejidad del requerimiento es O(E log V).

2115.2687999999907 [ms] 3010.1845703125 [kb]

Requerimiento 4.

def MST(analyzer):

    #req 4

    analyzer['MST']=prim.PrimMST(analyzer['connections'])

    peso=prim.weightMST(analyzer['connections'], analyzer['MST'])

    arcos = analyzer['MST']['mst']

    mincon=['','',-1]

    maxcon=['','',-1]

    x=0

    for arco in lt.iterator(arcos):

        distance=arco['weight']

        landing\_id1=arco['vertexA'].split('~')[0]

        landing\_name1=m.get(analyzer['landing\_points\_info'],landing\_id1)['value']['name']

        landing\_id2=arco['vertexB'].split('~')[0]

        landing\_name2=m.get(analyzer['landing\_points\_info'],landing\_id2)['value']['name']

        if landing\_name1!=landing\_name2:

            if mincon[2]==-1 or distance<mincon[2]:

                mincon[2]=distance

                mincon[0]=landing\_name1

                mincon[1]=landing\_name2

            if maxcon[2]==-1 or distance>maxcon[2]:

                maxcon[2]=distance

                maxcon[0]=landing\_name1

                maxcon[1]=landing\_name2

    numvertices=gr.numVertices(analyzer['connections'])

    ans=numvertices,peso,mincon,maxcon

    return ans

Se ejecuta el algoritmo de prim que tiene una complejidad de O(E log V). Luego se recorren los arcos, lo cual tiene una complejidad de O(E), que es menor. Finalmente la complejidad del requerimiento es O(E log V).

2627.517600000021 [ms] 5757.61328125 [kb]

Requerimiento 5.

def countries\_to\_landing\_point(analyzer,landing\_name):

    #req 5

    landing\_id=name\_to\_id(analyzer,landing\_name)

    country\_origin=m.get(analyzer['landing\_points\_info'],landing\_id)['value']['country']

    lstcables=m.get(analyzer['landing\_points\_cables'],landing\_id)['value']

    mapa=m.newMap()

    for cable in lt.iterator(lstcables):

        vertex=format\_vertex(landing\_id,cable)

        arcos=gr.adjacentEdges(analyzer['connections'],vertex)

        for arco in lt.iterator(arcos):

            landing\_id2=arco['vertexB'].split('~')[0]

            distance=arco['weight']

            country=m.get(analyzer['landing\_points\_info'],landing\_id2)['value']['country']

            if country==country\_origin:

                distance=0

            entry=(country,distance)

            if not m.contains(mapa,country):

                m.put(mapa,country,entry)

            else:

                entry2=m.get(mapa,country)['value']

                if distance<entry2[1]:

                    m.put(mapa,country,entry)

    lista=merge.sort(m.valueSet(mapa),compare\_entry\_req5)

    numcountries=lt.size(lista)

    return numcountries, lista

Solamente se recorren una vez cada arco de los vertices del landing point, por lo que la complejidad del recorrido es O(E). La complejidad del requerimiento es O(E).

25.15089999997872 [ms] 11.78125 [kb]