Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO 04

Juan David Cuervo Ortiz, j.cuervoo@uniandes.edu.co, 202125304.

Julián Blanco, j.blancog@uniandes.edu.co, 202212119.

Aclaración: Uno de nuestros compañeros de grupo (Santiago Pérez) retiró todo el semestre, incluyendo la materia. Por lo que, siguiendo las instrucciones del enunciado, no se hizo ningún desarrollo sobre el requerimiento 1 y el requerimiento 5.

Requerimiento 2

```
def req_2(data_structs, initialStation, destination):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    data_structs['search']= bfs.BreadhtFisrtSearch(data_structs['grafoDir'],initialStation)
    hay_camino= bfs.hasPathTo(data_structs['search'],destination)
    num_gathering=0
    tot_dist=0
    ult_nodo= None
    if hay_camino==True:
        lista_camino= lt.newList('ARRAY_LIST')
        lista_camino2= lt.newList('ARRAY_LIST')
        camino= bfs.pathTo(data_structs['search'],destination)
        i=1
        if camino is not None:
            num_vert = st.size(camino)
            while (not st.isEmpty(camino)):
                stop = st.pop(camino)
                lt.addLast(lista_camino, stop)
            for cada_nodo in lt.iterator(lista_camino):
                i+=1
                lista_stop= []
                node_id= cada_nodo
                if ult_nodo≠ None and cada_nodo≠ destination:
                    tot_dist+= gr.getEdge(data_structs['grafoDir'],cada_nodo, lt.getElement(lista_camino, i))['weight']
                    dist= gr.getEdge(data_structs['grafoDir'],cada_nodo,lt.getElement(lista_camino, i))['weight']
                else:
                    dist= 0.0
                cant_= cada_nodo.count('_')
                if cant_==1:
                    num_gathering+=1
                    individual_count0=gr.adjacentEdges(data_structs['grafoDir'],cada_nodo)
                    individual_count= lt.size(individual_count0)
                    info_stop = mp.get(data_structs['MTPs'],cada_nodo)
                    info= me.getValue(info_stop)['elements'][0]
                    long= info['location-long']
                    lat=info['location-lat']
                    lista_ady=[]
```

```
                for ady1 in lt.iterator(individual_count0):
                    ad= ady1['vertexB']
                    info_stop=mp.get(data_structs['individualPoints'], ad)
                    info= me.getValue(info_stop)
                    ady2= info['elements'][0]['individual-id']
                    lista_ady.append(ady2)
                ady= lista_ady
            else:
                    info_stop=mp.get(data_structs['individualPoints'], cada_nodo)
                    info= me.getValue(info_stop)
                    individual_count= 1
                    long= info['elements'][0]['location-long']
                    lat=info['elements'][0]['location-lat']
                    ady= info['elements'][0]['individual-id']
            if cada_nodo≠destination:
                    edge_to= lt.getElement(lista_camino, i)
            else:
                dist= 'Unknown'
                edge_to='Unknown'
            ult_nodo= cada_nodo
            lista_stop.append(long)
            lista_stop.append(lat)
            lista_stop.append(node_id)
            lista_stop.append(ady)
            lista_stop.append(individual_count)
            lista_stop.append(edge_to)
            lista_stop.append(dist)
            lt.addLast(lista_camino2,lista_stop)
    num_track= num_vert - num_gathering
    edges= num_vert- 1
    tot_dist= round(tot_dist, 4)
    if lt.size(lista_camino2) > 10:
        lista_camino2 = getiFirstandLast(lista_camino2,5)
    else:
        lista_camino2 = lista_camino2
    return lista_camino2, num_gathering, num_vert, num_track, edges, tot_dist
```

Este requerimiento se encarga de retornar la ruta más rápida entre dos puntos de encuentro.

Descripción

Entrada	1. Identificador del punto de encuentro de partida. 2. Identificador del punto de encuentro de origen.
Salidas	1. Distancia total del camino. 2. Total de puntos de encuentro. 3. Total de nodos de seguimiento. 4. 5 primeros y últimos nodos de la ruta, incluyendo: a. Id del nodo.

	b. Longitud y latitud de cada uno. c. El número de individuos que transitan ese punto. d. 3 primeros y últimos Ids de lobos que transitan ese punto. e. Distancia al siguiente punto del camino.
Implementado (Sí/No)	Sí. Autor: David Cuervo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Hacer y guardar el BFS del grafo.	O(V+E)
Verificar si existe un camino entre dos puntos	O(V)
Obtener la ruta más corta entre dos puntos	O(V)
Sacar todos los datos de la pila del BFS a una lista	O(V)
Calcular distancia total	O(V)
Calcular los 3 primeros y últimos	O(V)
Crear lista con datos a retornar	O(7)
TOTAL	***O(V)***

Pruebas Realizadas

Las pruebas fueron realizadas con un computador con **Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz,** *Windows 10 Pro* y 16GB de RAM.

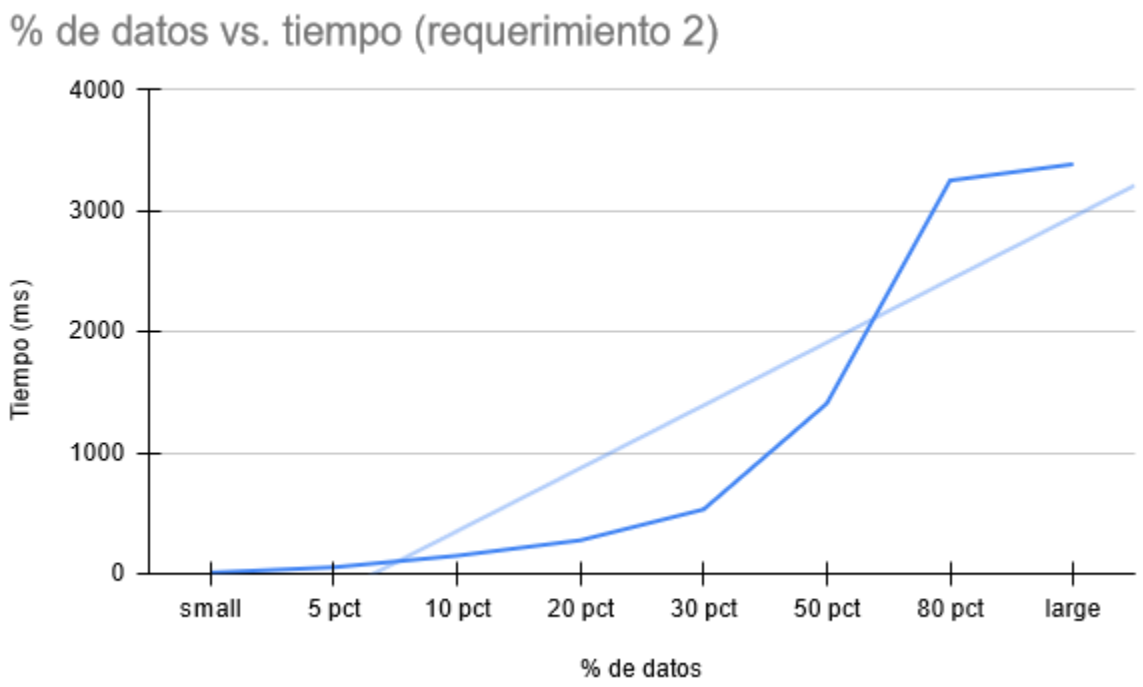
Entrada	**Tiempo (s)**
small	8.72
5 pct	53.12
10 pct	148.6
20 pct	276.02
30 pct	531.42
50 pct	1407.23
80 pct	3251.43
large	3386.24

Tablas de datos

Entrada	**Salida**	**Tiempo (ms)**

small	La ruta más rápida entre los dos puntos junto con información adicional.	8.72
5 pct	La ruta más rápida entre los dos puntos junto con información adicional.	53.12
10 pct	La ruta más rápida entre los dos puntos junto con información adicional.	148.6
20 pct	La ruta más rápida entre los dos puntos junto con información adicional.	276.02
30 pct	La ruta más rápida entre los dos puntos junto con información adicional.	531.42
50 pct	La ruta más rápida entre los dos puntos junto con información adicional.	1407.23
80 pct	La ruta más rápida entre los dos puntos junto con información adicional.	3251.43
large	La ruta más rápida entre los dos puntos junto con información adicional.	3386.24

Gráficas



Análisis

El funcionamiento de este requerimiento es conceptualmente simple independientemente de que el código pueda ser un poco extenso. Resumidamente, el requerimiento funciona de manera que, basándose en el BFS del grafo a partir del nodo de origen, se busca el camino de menor costo desde el origen hasta el destino, luego se pasan los datos de la cola que retorna el BFS a una lista en la que se itera para sacar los datos necesarios para poder obtener el resto de datos solicitados. La complejidad general de este requerimiento es O(V+E), esto debido a que para hacer el BFS del grafo es necesario tomar en cuenta los arcos.

Requerimiento 3

```
def req_3(data_structs):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    sccStruct = scc.KosarajuSCC(data_structs['grafoDir'])
    numScc = scc.connectedComponents(sccStruct)
    idSccMap = mp.newMap(maptype='PROBING')
    for nodo in lt.iterator(mp.keySet(sccStruct['idscc'])):
        idScc = mp.get(sccStruct['idscc'],nodo)['value']
        contains = mp.get(idSccMap,idScc)
        if contains == None:
            lstNodes = lt.newList('ARRAY_LIST')
            lt.addLast(lstNodes,nodo)
            mp.put(idSccMap,idScc,lstNodes)
        else:
            lstNodes = me.getValue(contains)
            lt.addLast(lstNodes,nodo)

    omSize = om.newMap()
    for idScc in lt.iterator(mp.keySet(idSccMap)):
        lstNodes = me.getValue(mp.get(idSccMap,idScc))
        if lt.size(lstNodes) > 2:
            info = mp.newMap(maptype='PROBING')
            mp.put(info,'idScc',idScc)
            mp.put(info,'nodes',lstNodes)
            om.put(omSize,lt.size(lstNodes),info)
    c =0
    FivemaxManInfo = []
    while c < 5:
        minLat = 1000
        maxLat = 0
        minLong = 0
        maxLong = -1000
        infoScc = []
        maxM= om.maxKey(omSize)
        info = me.getValue(om.get(omSize,maxM))
        sccId = me.getValue(mp.get(info,'idScc'))
        lstNodesId = me.getValue(mp.get(info,'nodes'))
        nodeIds = getIFirstandLast(lstNodesId,3)
        wolfs = lt.newList('ARRAY_LIST')
        for nodo in lt.iterator(lstNodesId):
            entry = mp.get(data_structs['MTPs'],nodo)
            if entry == None:
                entry = mp.get(data_structs['individualPoints'],nodo)
                wolf = me.getValue(entry)['elements'][0]['individual-id']
                if not lt.isPresent(wolfs,wolf):
                    lt.addLast(wolfs,wolf)
            event = me.getValue(entry)['elements'][0]
            if event['location-lat'] > maxLat:
                maxLat = event['location-lat']
            elif event['location-lat'] < minLat:
                minLat = event['location-lat']
            if event['location-long'] > maxLong:
                maxLong = event['location-long']
            elif event['location-long'] < minLong:
                minLong = event['location-long']
```

```
        infoScc.append(sccId)
        res = []
        p =0
        for Id in nodeIds:
            lst = [Id]
            res.append(lst)
            p+= 1
            if p ==3:
                pt = ['...']
                res.append(pt)
        infoScc.append(tabulate(res,tablefmt="plain"))
        infoScc.append(maxM)
        infoScc.append(minLat)
        infoScc.append(maxLat)
        infoScc.append(minLong)
        infoScc.append(maxLong)
        infoScc.append(lt.size(wolfs))
        lstOflst =[]
        if lt.size(wolfs) > 6:
            wolfs = getiFirstandLast(wolfs,3)
        for lobo in lt.iterator(wolfs):
            infoLobo = []
            details = me.getValue(mp.get(data_structs['lobos'],lobo))
            infoLobo.append(details['individual-id'])
            infoLobo.append(details['animal-taxon'])
            infoLobo.append(details['animal-sex'])
            infoLobo.append(details['animal-life-stage'])
            infoLobo.append(details['study-site'])
            lstOflst.append(infoLobo)
        wolftable = tabulate(lstOflst,headers=['indiv-id','wolf taxon','wolf sex','life-stage','study-site'],
                             tablefmt='grid',maxheadercolwidths=5,maxcolwidths=5,stralign="center")
        infoScc.append(wolftable)
        FivemaxManInfo.append(infoScc)
        om.deleteMax(omSize)
        c += 1
    return numScc, FivemaxManInfo
```

Este requerimiento se encarga de retornar información relacionada con las manadas incluidas en todo el grafo, como número de individuos y algunos miembros de cada una.

Descripción

Entrada	Este requerimiento trabaja sobre todo el grafo, por lo que no tiene parámetros de entrada.
Salidas	1. Total de manadas identificadas 2. Las 5 manadas con mayor dominio sobre el territorio junto con: a. Número de puntos de encuentro y seguimiento. b. Tres primeros y últimos puntos de encuentro. c. Número de individuos que forman la manada. d. Tres primeros y últimos miembros con: i. Id. ii. Taxonomía. iii. Ciclo de vida. iv. Sexo. v. Lugar de estudio.

	3. Longitudes y latitudes máximas y mínimas de los puntos de encuentro.
Implementado (Sí/No)	Si. Autor: Julián Blanco.

Análisis de complejidad

Pasos	Complejidad
Recorrer y obtener el resultado de Kosaraju sobre el grafo.	O(V+E)
Sacar toda la información del Kosaraju y meterla en un mapa.	O(V+E)
Crear y organizar un árbol conteniendo sólo los datos que correspondan a una manada.	O(nlogn)
Obtener las 5 manadas con mayor control en la región.	O(N^2)
Armar la lista de la manada.	O(7)
Armar la lista de los lobos.	O(N)
TOTAL	***O(N^2)***

Pruebas Realizadas

Las pruebas fueron realizadas con un computador con **Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz,** *Windows 10 Pro* y 16GB de RAM.

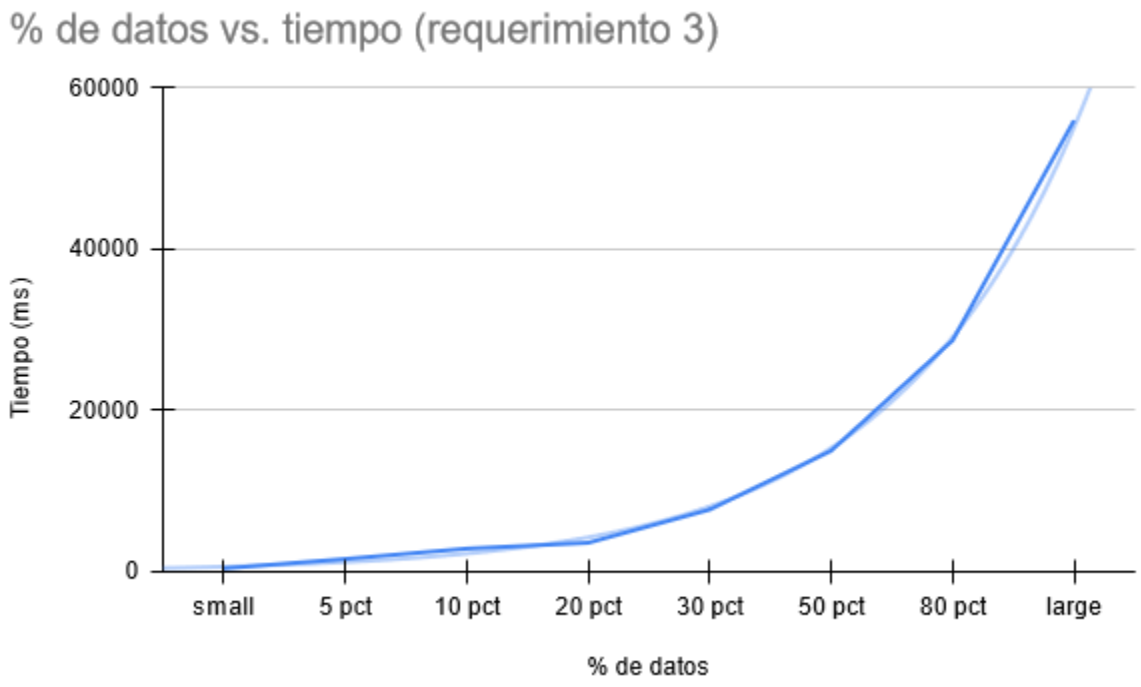
Entrada	Tiempo (ms)
small	410.85
5 pct	1577.07
10 pct	2861.09
20 pct	3582.31
30 pct	~7700
50 pct	~15000
80 pct	~28700
large	~56000

Tablas de datos

Entrada	Salida	Tiempo (ms)
small	Información relacionada con todas las manadas identificadas.	410.85
5 pct	Información relacionada con todas las manadas identificadas.	1577.07

10 pct	Información relacionada con todas las manadas identificadas.	2861.09
20 pct	Información relacionada con todas las manadas identificadas.	3582.31
30 pct	Información relacionada con todas las manadas identificadas.	~4200
50 pct	Información relacionada con todas las manadas identificadas.	~5600
80 pct	Información relacionada con todas las manadas identificadas.	~7700
large	Información relacionada con todas las manadas identificadas.	~9100

Gráficas



Análisis

El concepto del algoritmo creado para este requerimiento es relativamente simple. Básicamente, se recorre el grafo con Kosaraju para poder tener una indicación más clara donde se encuentran las manadas y cuales son, esto mediante la identificación de nodos fuertemente conectados. Luego se separan cada uno de esos y se clasifican en sus manadas correspondientes para, posteriormente, sacar las 5 manadas con mayor dominio sobre el territorio analizado.

La complejidad de las diferentes partes del algoritmo varía debido a la cantidad de pasos que se deben hacer antes de retornar el resultado, por un lado se usa un árbol para guardar datos intermedios, dando complejidad O(nlogn), sin embargo, en la parte en la que se debe obtener la información de las 5 manadas con mayor dominio de la región, se tiene una complejidad O(N^2).

Requerimiento 4

```
def req_4(data_structs, ini, fin):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    longini= float(ini[0])
    latini= float(ini[1])
    longfin= float(fin[0])
    latfin= float(fin[1])
    dist_menor_ini=9999999999
    mapa= data_structs['MTPs']
    keys= mp.keySet(mapa)
    ino=''
    dest=''
    for llave in lt.iterator(keys):
        lista_mtp= mp.get(mapa, llave)
        lista_mtp= me.getValue(lista_mtp)
        info= lt.getElement(lista_mtp, 1)
        long2=info['location-long']
        lat2= info['location-lat']
        dist= haversine(latini, longini, lat2, long2)
        if dist==0:
            continue
        if dist<dist_menor_ini:
            dist_menor_ini=dist
            ino=llave
    dist_menor_dest=999999999
```

```
    for llave in lt.iterator(keys):
        lista_mtp= mp.get(mapa, llave)
        lista_mtp= me.getValue(lista_mtp)
        info= lt.getElement(lista_mtp, 1)
        long2=info['location-long']
        lat2= info['location-lat']
        dist= haversine( lat2, long2, latfin, longfin)
        if dist==0:
            continue
        if dist<dist_menor_dest:
            dist_menor_dest=dist
            dest= llave
    info_stop = mp.get(data_structs['MTPs'],ino)
    info= me.getValue(info_stop)
    longino= info['elements'][0]['location-long']
    latino=info['elements'][0]['location-lat']
    individual_idino=gr.adjacentEdges(data_structs['grafoDir'],ino)
    lista_adyini= []
    for ady1 in lt.iterator(individual_idino):
        ad= ady1['vertexB']
        info_stop=mp.get(data_structs['individualPoints'], ad)
        info= me.getValue(info_stop)
        ady2= info['elements'][0]['individual-id']
        lista_adyini.append(ady2)
    individual_idino= lista_adyini

    info_stop = mp.get(data_structs['MTPs'],dest)
    info= me.getValue(info_stop)
    longdest= info['elements'][0]['location-long']
    latdest=info['elements'][0]['location-lat']
    individual_id_dest=gr.adjacentEdges(data_structs['grafoDir'],dest)
    lista_adyini= []
```

```
for ady1 in lt.iterator(individual_idino):
    ad= ady1['vertexB']
    info_stop=mp.get(data_structs['individualPoints'], ad)
    info= me.getValue(info_stop)
    ady2= info['elements'][0]['individual-id']
    lista_adyini.append(ady2)
individual_idino= lista_adyini

info_stop = mp.get(data_structs['MTPs'],dest)
info= me.getValue(info_stop)
longdest= info['elements'][0]['location-long']
latdest=info['elements'][0]['location-lat']
individual_id_dest=gr.adjacentEdges(data_structs['grafoDir'],dest)
lista_adyini= []
```

```
    for ady1 in lt.iterator(individual_id_dest):
        ad= ady1['vertexB']
        info_stop=mp.get(data_structs['individualPoints'], ad)
        info= me.getValue(info_stop)
        ady2= info['elements'][0]['individual-id']
        lista_adyini.append(ady2)
    individual_id_dest= lista_adyini
    paths= djk.Dijkstra(data_structs['grafoDir'], ino)
    hay_path= djk.hasPathTo(paths, dest)
    if hay_path ==True:
        path = djk.pathTo(paths, dest)
        lista_camino= lt.newList('ARRAY_LIST')
        lista_gathering= lt.newList('ARRAY_LIST')
        lista_camino2= lt.newList('ARRAY_LIST')
        dist_tot=0
        if path is not None:
            num_vert = st.size(path)
            while (not st.isEmpty(path)):
                stop = st.pop(path)
                lt.addLast(lista_camino, stop)
```

```
    for info in lt.iterator(lista_camino):
        lista= []
        src_node_id= info['vertexA']
        tgt_node_id= info['vertexB']
        dist= info[ 'weight']
        dist_tot+= info[ 'weight']
        src_count=src_node_id.count('_')
        tgt_count=tgt_node_id.count('_')
        if src_count==1:
            lista_gath= []
            info_stop = mp.get(data_structs['MTPs'],src_node_id)
            info= me.getValue(info_stop)['elements'][0]
            longsrc= info['location-long']
            latsrc=info['location-lat']
            lista_gath.append(src_node_id)
            lista_gath.append(longsrc)
            lista_gath.append(latsrc)
            lt.addLast(lista_gathering, lista_gath)
        else:
            info_stop=mp.get(data_structs['individualPoints'], src_node_id)
            info= me.getValue(info_stop)
            longsrc= info['elements'][0]['location-long']
            latsrc=info['elements'][0]['location-lat']
        if tgt_count==1:
            info_stop = mp.get(data_structs['MTPs'],tgt_node_id)
            info= me.getValue(info_stop)['elements'][0]
            longtgt= info['location-long']
            lattgt=info['location-lat']
            individual_idtgt= info['individual-id']
            lista_gath.append(tgt_node_id)
            lista_gath.append(longtgt)
            lista_gath.append(lattgt)
            lt.addLast(lista_gathering, lista_gath)
```

Este requerimiento se encarga de identificar el camino más corto entre dos puntos del hábitat mediante su localización geográfica.

Descripción

Entrada	1. Longitud y latitud del punto de origen. 2. Longitud y latitud del punto de destino.
Salidas	1. La distancia entre el punto de origen y el punto de encuentro más cercano.

	2. La distancia entre el punto de destino y el punto de encuentro más cercano. 3. La distancia entre el punto de origen y el punto de destino. 4. El total de puntos de encuentro que forman el camino. 5. El total de lobos que usan ese camino. 6. El total de segmentos (arcos) que forman el camino. 7. Los tres primeros y últimos puntos de encuentro junto con: a. Él, id del punto. b. La longitud y latitud del punto. c. El número de individuos que transitan por el punto. d. Los 3 primeros y últimos lobos que transitan el punto. e. La distancia al siguiente punto de encuentro.
Implementado (Sí/No)	Sí. Autor: David Ortiz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Sacar el Keyset del mapa.	O(V)
Encontrar el punto de encuentro más cercano a las coordenadas de origen.	O(V)
Encontrar el punto de encuentro más cercano a las coordenadas de destino.	O(V)
Recorrer el grafo con Djisktra.	O(V)
Verificar y obtener el camino más corto entre los dos puntos de encuentro.	O(V+E)
Encontrar los 3 primeros y últimos lobos que transitan ese camino.	O(V^2)
Organizar y retornar los datos	O(8)
TOTAL	***O(V^2)***

Pruebas Realizadas

Las pruebas fueron realizadas con un computador con **Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz,** *Windows 10 Pro* y 16GB de RAM.

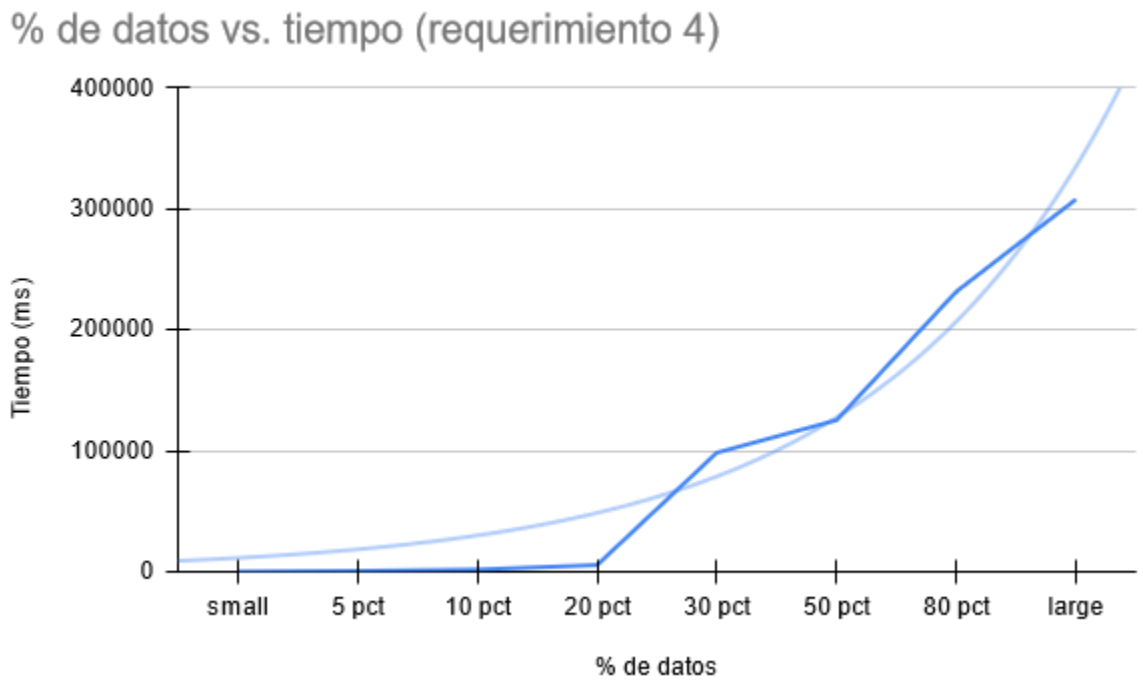
Entrada	**Tiempo (ms)**
small	24.61
5 pct	629.19
10 pct	1745.23
20 pct	5500.03
30 pct	98413.37
50 pct	125273.01

80 pct	231582.51
large	307920.16

Tablas de datos

Entrada	Salida	Tiempo (ms)
small	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	24.61
5 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	629.19
10 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	1745.23
20 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	5500.03
30 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	98413.37
50 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	125273.01
80 pct	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	231582.51
large	Puntos de encuentro más cercanos a coordenadas de origen y destino junto con información adicional.	307920.16

Gráficas



Análisis

Este requerimiento tiene una complejidad general de O(V^2), sin embargo, la mayoría de la función tiene una complejidad O(V) debido a que solo se está tratando con los lobos (nodos), aunque sea un algoritmo bastante extenso, se puede observar como no se toma en cuenta la distancia sino sólo cuando se va a calcular el camino entre los dos puntos calculados y cuando se va a recorrer el grafo dirigido mediante Dijkstra, en estos casos, la complejidad sería de O(V+E).

Requerimiento 6

```python
def req_6(data_structs,animal_sex, ini, fin):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    lista_lobos= lt.newList('ARRAY_LIST')
    lbs= data_structs['lobos']
    keys= mp.keySet(lbs)
    mapa_filt= mp.newMap(maptype='CHAINING')
    for lobo in lt.iterator(keys):
        entry= mp.get(lbs, lobo)
        info= me.getValue(entry)
        a_s= info['animal-sex']
        if a_s == animal_sex:
            lt.addLast(lista_lobos,lobo )
    mapa= data_structs['orderedData']
    for lobo in lt.iterator(lista_lobos):
        entry= mp.get(mapa, lobo)
        info= me.getValue(entry)
        for inf in lt.iterator(info):
            fecha= inf['timestamp']
            fecha= datetime.datetime.strptime(fecha,'%Y-%m-%d %H:%M')
            if fecha > ini and fecha< fin:
                if mp.contains(mapa_filt, lobo):
                    por_fecha=mp.get(mapa_filt, lobo)
                    por_fecha=me.getValue(por_fecha)
                    lt.addLast(por_fecha, inf)
                else:
                    por_anio= lt.newList('ARRAY_LIST')
                    lt.addLast(por_anio, inf)
                    mp.put(mapa_filt,lobo, por_anio)

    omDist = om.newMap()
    for wolf in lt.iterator(mp.keySet(mapa_filt)):
        distance = 0
        lstEvents = me.getValue(mp.get(mapa_filt,wolf))
        lasttrack = None
        for event in lt.iterator(lstEvents):
            if lasttrack ≠ None:
                startP = lasttrack['node-id']
                destP = event['node-id']
                distance += gr.getEdge(data_structs['grafoDir'],startP,destP)['weight']
            lasttrack = event
        om.put(omDist,distance,lstEvents)

    maxDist = om.maxKey(omDist)
    minDist = om.minKey(omDist)
    wolfMaxDist = me.getValue(om.get(omDist,maxDist))['elements'][0]['individual-id']
    wolfMinDist = me.getValue(om.get(omDist,minDist))['elements'][0]['individual-id']
    maxWolfInfo = ObtainWolfInfo(data_structs,wolfMaxDist,maxDist)
    minWolfInfo = ObtainWolfInfo(data_structs,wolfMinDist,minDist)
    #segunda parte
    pathInfoMax = pathInfo(data_structs,omDist,wolfMaxDist,maxDist)
    pathInfoMin = pathInfo(data_structs,omDist,wolfMinDist,minDist)

    return wolfMaxDist, maxWolfInfo, pathInfoMax,wolfMinDist,minWolfInfo,pathInfoMin
```

Este requerimiento se encarga de retornar información sobre el lobo que recorrió más distancia dentro de la región junto con información sobre el lobo que recorrió menor distancia dentro de la región, esto tomando en cuenta un periodo de tiempo específico y el sexo del animal.

Descripción

Entrada	1. Fecha inicial del análisis. 2. Fecha final del análisis. 3. El sexo del animal.
Salidas	1. El lobo que recorrió la mayor distancia junto con:

	a. Id. b. Taxonomía. c. Ciclo de vida. d. Sexo. e. Lugar de estudio. f. Distancia total recorrida. g. Comentarios sobre el animal. 2. La ruta más larga del lobo que recorrió la mayor distancia junto con: a. La distancia total del recorrido. b. El total de puntos que lo forman. c. El total de trayectos que lo forman. d. Los tres primeros y últimos puntos de encuentro que lo forman junto con: i. Id del punto. ii. Longitud y Latitud. iii. Número de lobos que lo transitan. iv. Los tres primeros y últimos Ids de lobos que usan ese punto. 3. El lobo que recorrió la menor distancia junto con: a. Id. b. Taxonomía. c. Ciclo de vida. d. Sexo. e. Lugar de estudio. f. Distancia total recorrida. g. Comentarios sobre el animal. 4. La ruta más larga del lobo que recorrió la mayor distancia junto con: a. La distancia total del recorrido. b. El total de puntos que lo forman. c. El total de trayectos que lo forman. d. Los tres primeros y últimos puntos de encuentro que lo forman junto con: i. Id del punto. ii. Longitud y Latitud. iii. Número de lobos que lo transitan. iv. Los tres primeros y últimos Ids de lobos que usan ese punto.
Implementado (Sí/No)	Sí. Autor: David Cuervo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Obtener el keyset de los datos de los lobos	O(1)
Obtener los datos de cada lobo	O(N)
Clasificar y escoger únicamente los lobos del sexo solicitado	O(N)
Seleccionar solo los datos de los lobos clasificados que cumplan con el periodo de tiempo requerido.	O(N)
Crear un árbol para organizar los datos.	O(1)
Organizar los datos en el árbol.	O(nlogn)
Seleccionar y retornar los resultados	O(nlogn)
Total	***O(nlogn)***

Pruebas Realizadas

Las pruebas fueron realizadas con un computador con **Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz,** *Windows 10 Pro* y 16GB de RAM.

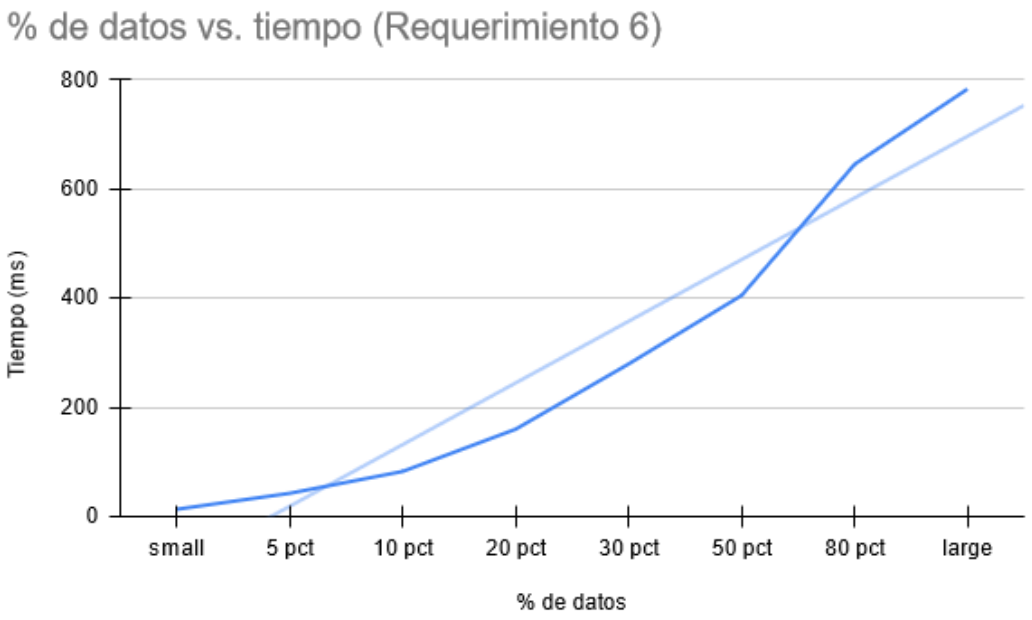
Entrada	Tiempo (ms)
small	13.24
5 pct	42.49
10 pct	82.56
20 pct	159.97
30 pct	279.52
50 pct	405.04
80 pct	645.37
large	783.19

Tablas de datos

Entrada	Salida	Tiempo (ms)
small	Información sobre los lobos con menores y mayores recorridos del estudio.	13.24
5 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	42.49
10 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	82.56
20 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	159.97
30 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	279.52

50 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	405.04
80 pct	Información sobre los lobos con menores y mayores recorridos del estudio.	645.37
large	Información sobre los lobos con menores y mayores recorridos del estudio.	783.19

Gráficas



Análisis

Este, al ser el requerimiento más complejo de todo el reto, hace uso de varias funciones auxiliares. Adicionalmente, teniendo en cuenta que DISCLIB no tiene las herramientas necesarias para realizar el requerimiento, se decidió trabajar con las herramientas proporcionadas para manejar árboles y desarrollar el requerimiento alrededor de estos, es por eso que, después de obtener los datos necesarios de los lobos, se organizan y manejan dentro de un árbol para después retornarlos. La complejidad general de este requerimiento es O(nlogn).

Requerimiento 7

```
def req_7(data_structs,dateI, dateF,tempMax,tempMin):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    mapaFilt = mp.newMap()
    for wolf in lt.iterator(mp.keySet(data_structs['orderedData'])):
        lstWolfs = lt.newList('ARRAY_LIST')
        for track in lt.iterator(me.getValue(mp.get(data_structs['orderedData'],wolf))):
            fecha = track['timestamp']
            fecha= datetime.datetime.strptime(fecha,'%Y-%m-%d %H:%M')
            if float(track['external-temperature']) ⩾ tempMin and float(track['external-temperature']) ⩽ tempMax and fecha ⩾ dateI and fecha ⩽ dateF:
                lt.addLast(lstWolfs,track)
        mp.put(mapaFilt,wolf,lstWolfs)
    mapaPosiciones = mp.newMap()
    graFilt = grafoFilt(mapaFilt,mapaPosiciones)
    for pos in lt.iterator(mp.keySet(mapaPosiciones)):
        if lt.size(me.getValue(mp.get(mapaPosiciones,pos))) ⩾ 2:
            gr.insertVertex(graFilt,pos)
            for node in lt.iterator(me.getValue(mp.get(mapaPosiciones,pos))):
                gr.addEdge(graFilt,node['node-id'],pos,0)
                gr.addEdge(graFilt,pos,node['node-id'],0)
    #para los componentes
    sccStruct = scc.KosarajuSCC(graFilt)
    numScc = scc.connectedComponents(sccStruct)
    idSccMap = mp.newMap(maptype='PROBING')
    for nodo in lt.iterator(mp.keySet(sccStruct['idscc'])):
        idScc = mp.get(sccStruct['idscc'],nodo)['value']
        contains = mp.get(idSccMap,idScc)
        if contains == None:
            lstNodes = lt.newList('ARRAY_LIST')
            lt.addLast(lstNodes,nodo)
            mp.put(idSccMap,idScc,lstNodes)
        else:
            lstNodes = me.getValue(contains)
            lt.addLast(lstNodes,nodo)
    omSize = om.newMap()
    for idScc in lt.iterator(mp.keySet(idSccMap)):
        lstNodes = me.getValue(mp.get(idSccMap,idScc))
        if lt.size(lstNodes) > 2:
            info = mp.newMap(maptype='PROBING')
            mp.put(info,'idScc',idScc)
            mp.put(info,'nodes',lstNodes)
            om.put(omSize,lt.size(lstNodes),info)
    c =0
    FivemaxManInfo = []
    while c < 3:
        minLat = 1000
        maxLat = 0
        minLong = 0
        maxLong = -1000
        infoScc = []
        maxM= om.maxKey(omSize)
        info = me.getValue(om.get(omSize,maxM))
        sccId = me.getValue(mp.get(info,'idScc'))
        lstNodesId = me.getValue(mp.get(info,'nodes'))
        nodeIds = getIFirstandLast(lstNodesId,3)
        wolfs = lt.newList('ARRAY_LIST')
        for nodo in lt.iterator(lstNodesId):
            entry = mp.get(data_structs['MTPs'],nodo)
            if entry == None:
                entry = mp.get(data_structs['individualPoints'],nodo)
                wolf = me.getValue(entry)['elements'][0]['individual-id']
                if not lt.isPresent(wolfs,wolf):
                    lt.addLast(wolfs,wolf)
            event = me.getValue(entry)['elements'][0]
            if event['location-lat'] > maxLat:
                maxLat = event['location-lat']
            elif event['location-lat'] < minLat:
                minLat = event['location-lat']
            if event['location-long'] > maxLong:
                maxLong = event['location-long']
            elif event['location-long'] < minLong:
                minLong = event['location-long']
        infoScc.append(sccId)
        res = []
        p =0
```

```
    for Id in nodeIds:
        lst = [Id]
        res.append(lst)
        p+= 1
        if p ==3:
            pt = ['...']
            res.append(pt)
    infoScc.append(tabulate(res,tablefmt="plain"))
    infoScc.append(maxM)
    infoScc.append(minLat)
    infoScc.append(maxLat)
    infoScc.append(minLong)
    infoScc.append(maxLong)
    infoScc.append(lt.size(wolfs))
    lstOflst =[]
    if lt.size(wolfs) > 6:
        wolfs = getiFirstandLast(wolfs,3)
    for lobo in lt.iterator(wolfs):
        infoLobo = []
        details = me.getValue(mp.get(data_structs['lobos'],lobo))
        infoLobo.append(details['individual-id'])
        infoLobo.append(details['animal-taxon'])
        infoLobo.append(details['animal-sex'])
        infoLobo.append(details['animal-life-stage'])
        infoLobo.append(details['study-site'])
        lstOflst.append(infoLobo)
    wolftable = tabulate(lstOflst,headers=['indiv-id','wolf taxon','wolf sex','life-stage','study-site'],
                         tablefmt='grid',maxheadercolwidths=5,maxcolwidths=5,stralign="center")
    infoScc.append(wolftable)
    FivemaxManInfo.append(infoScc)
    om.deleteMax(omSize)
    c += 1
return numScc, FivemaxManInfo
```

Este requerimiento se encarga de retornar información sobre los efectos que tiene el clima en la actividad de las manadas como ruta más larga y manadas con dominio sobre el terreno.

Descripción

Entrada	1. Fecha inicial del análisis. 2. Fecha final del análisis. 3. Temperatura ambiente mínima. 4. Temperatura ambiente máxima.
Salidas	1. El total de manadas basadas en su movimiento y puntos de encuentro dentro de los parámetros ingresados. 2. Las tres primeras y últimas manadas con mayor dominio sobre la región junto con: a. Su número de puntos de encuentro. b. Tres primeros y últimos puntos de encuentro dentro de su territorio. c. La cantidad de lobos que forman la manada. d. Los tres primeros y últimos lobos junto con: i. Id. ii. Taxonomía. iii. Ciclo de vida. iv. Sexo del animal. v. Lugar de estudio.

	e. Longitudes y latitudes máximas y mínimas de los puntos de encuentro. 3. La ruta más larga dentro de la región junto con: a. La distancia del camino. b. El total de nodos que lo conforman. c. El total de arcos que lo forman. d. Los tres primeros y últimos puntos de encuentro. e. El total de lobos que usan ese camino. f. Los tres primeros y últimos lobos que usan ese camino.
Implementado (Sí/No)	Sí. Autor: David Cuervo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Clasificar y obtener los datos que cumplan con las restricciones de tiempo y clima pasadas por parámetro.	O(V)
Obtener los datos únicamente de las manadas	O(V+E)
Organizar y escoger las 3 primeras y últimas manadas con mayor control del terreno.	O(V)
Encontrar los tres primeros y últimos puntos de encuentro de cada manada.	O(V)
Encontrar los tres primeros y últimos lobos que usan el camino.	O(V)
Organizar y retornar el resultado	O(V)
Total	***O(V+E)***

Pruebas Realizadas

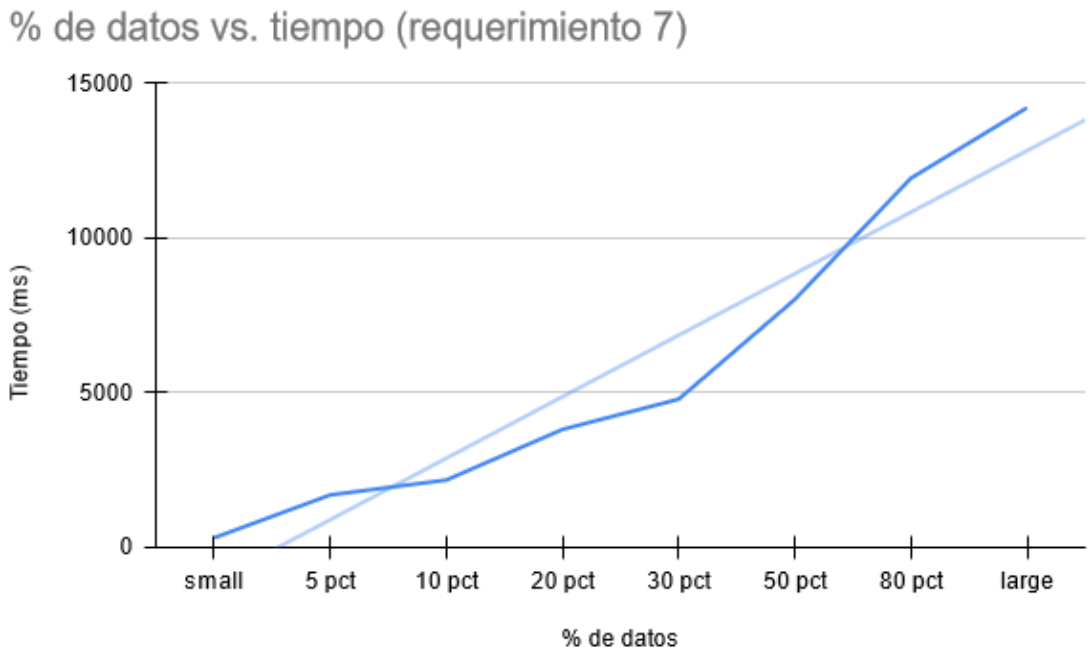
Las pruebas fueron realizadas con un computador con **Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz 2.90 GHz,** *Windows 10 Pro* y 16GB de RAM.

Entrada	**Tiempo (ms)**
small	308.22
5 pct	1695.09
10 pct	2178.24
20 pct	3814.81
30 pct	4791.26
50 pct	8016.27
80 pct	11937.95
large	14218.13

Tablas de datos

Entrada	Salida	Tiempo (ms)
small	Información de las manadas respecto a los cambios de clima.	308.22
5 pct	Información de las manadas respecto a los cambios de clima.	1695.09
10 pct	Información de las manadas respecto a los cambios de clima.	2178.24
20 pct	Información de las manadas respecto a los cambios de clima.	3814.81
30 pct	Información de las manadas respecto a los cambios de clima.	4791.26
50 pct	Información de las manadas respecto a los cambios de clima.	8016.27
80 pct	Información de las manadas respecto a los cambios de clima.	11937.95
large	Información de las manadas respecto a los cambios de clima.	14218.13

Gráficas



Análisis

Este requerimiento tiene una complejidad general de O(V+E), esto debido a que la única parte del código en la que se obtiene esta complejidad es en la que se encarga de recorrer el grafo con Kosaraju para poder identificar las mandas y consecuentemente seleccionar los datos correctos. El resto del código tiene una complejidad de O(V), esto debido a que, fundamentalmente, todo depende de la cantidad de lobos (nodos) que entren en la carga de datos.