Departamento de Ingeniería de Sistemas y Computación

Estructuras de Datos y Algoritmos ISIS-1225

ANÁLISIS DEL RETO 3

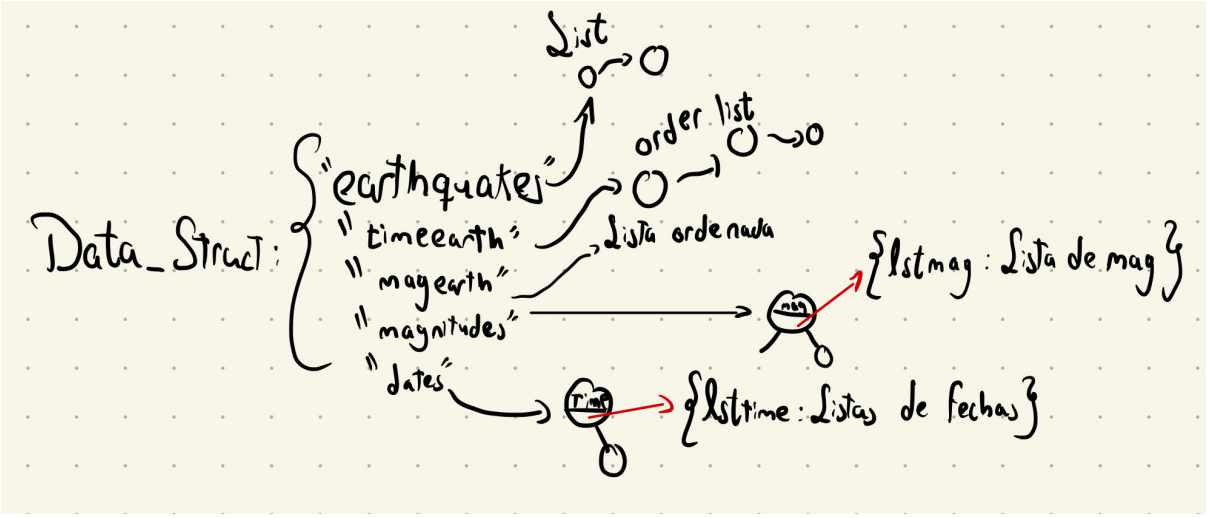
R5 Nicolás González, 202310041, n.gonzalez112@uniandes.edu.co

R4 Nikol de la Cruz , 202317231, n.delacruz@uniandes.edu.co

R3 David Esteban Quiroga Alfaro, 202310820, de.quiroga2@uniandes.edu.co

Carga de datos

Descripción:
En la carga de datos creamos dos RBT con claves de magnitudes y fechas. Estos árboles se crean al estilo del laboratorio con sus 2 funciones externas que ayudan a actualizar los árboles a medida que entran datos. Aparte de esto, tenemos tres listas, una con los sismos y otras 2 estan ordenadas por magnitud y fecha



Requerimiento <<1>>.

Descripción:

Este requerimiento retorna los N últimos partidos de un equipo en una condición específica, puede ser local, visitante o indiferente. Este algoritmo contiene un map con llaves las cuales

son los partidos y cada llave contiene una lista que incluye otra lista para cada condición, indiferente, local o visitante. Estas listas se representan con las posiciones 1, 2, 3 respectivamente lo que hace a este algoritmo eficiente, buscando únicamente la posición según la condición

Entrada	Las entradas son los datos generales para luego sacarle el árbol que ordena los sismos por fechas, además pedimos por parámetro el intervalo de fechas donde se quiere realizar la búsqueda
Salidas	Saca una lista con todos los sismos ocurridos durante las fechas establecidas de tipo single linked, además se le agrega una variable para que vaya contando cada sismo y asi solo pasar al controller y al view 2 variables de return
Implementado (Sí/No)	Si. Implementado por Grupo

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Om.Values	O(log N)
Nueva single linked lista	O(N)
Iteración en lista	O(N)
Sacar datos de cada sismo	O(N)
AddFirst en enlazada	O(N)
Contador de sismos	O(1)

Pruebas Realizadas

La prueba de tiempos se hará como parámetros, pais: Colombia, condición: indiferente, número: 100 partidos

Procesadores	**AMD Ryzen 3 3300U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	**16 GB**
Sistema Operativo	**Windows 10 Home**

TIEMPOS

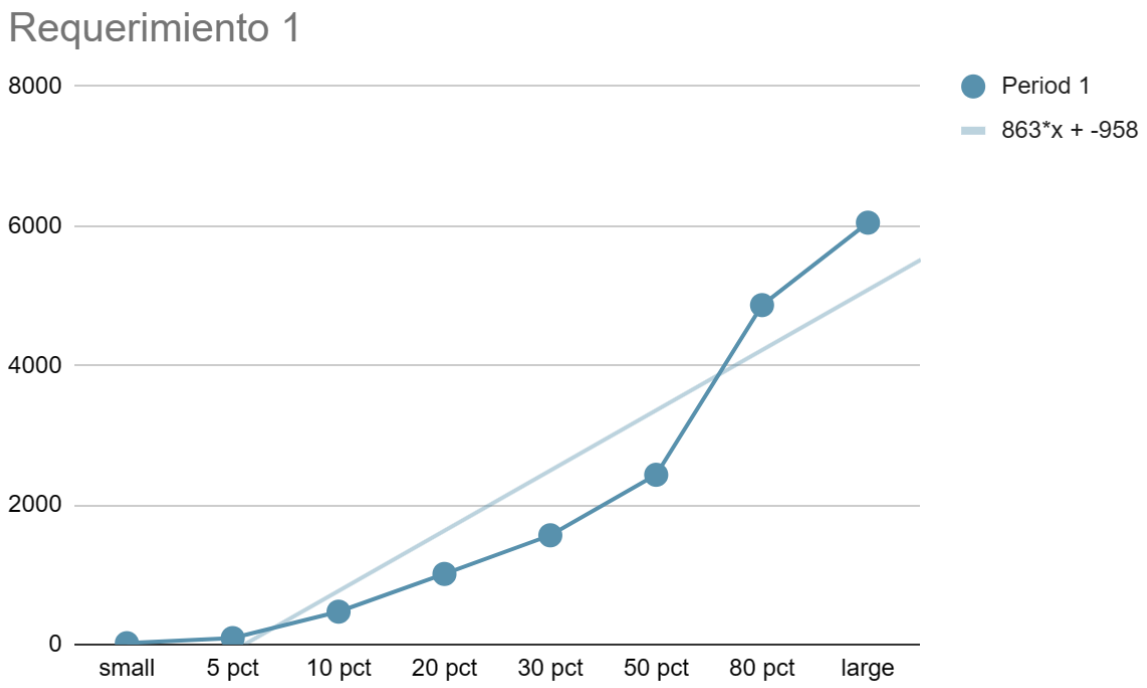
Entrada	**Tiempo (ms)**
small	18.926
5 pct	91.708
10 pct	470.664
20 pct	1011.383
30 pct	1565.442
50 pct	2432.918
80 pct	4865.326
large	6047.346

Tablas de datos

Muestra	**Entrada**	**Tiempo (ms)**
small	2000-01-01T01:00, 2013-01-01T01:00	18.926
5 pct	2000-01-01T01:00, 2013-01-01T01:00	91.708
10 pct	2000-01-01T01:00, 2013-01-01T01:00	470.664
20 pct	2000-01-01T01:00, 2013-01-01T01:00	1011.383
30 pct	2000-01-01T01:00, 2013-01-01T01:00	1565.442

50 pct	2000-01-01T01:00, 2013-01-01T01:00	2432.918
80 pct	2000-01-01T01:00, 2013-01-01T01:00	4865.326
large	2000-01-01T01:00, 2013-01-01T01:00	6047.346

Gráficas



Análisis

El árbol implementado para este requerimiento organiza todos los sismos por fechas utilizando un árbol RBT para mayor comodidad. Gracias a esto tenemos complejidades no tan grandes y tiempos muy buenos. estos árboles nos ayudan a no tener que iterar toda una lista y ordenarla ni hacer otro tipo de cosas, estos arboles ya implementan funciones avanzadas de búsqueda y filtrado

Requerimiento <<2>>.

Descripción:

```python
def req_2(map,magI,magF):
    """
    Función que soluciona el requerimiento 2
    """
    earthquakes = om.values(map,magI,magF)
    resultados=lt.newList("SINGLE_LINKED")
    tmag = lt.size(earthquakes)
    teve=0
    for earth in lt.iterator(earthquakes):
        datos={"mag":None,"events":None,"details":None}
        eventos= earth["lstmags"]
        d1 = lt.firstElement(eventos)
        magnitud= d1["mag"]
        datos["mag"]= magnitud
        datos["events"] = lt.size(eventos)
        datos["details"] = eventos
        lt.addFirst(resultados,datos)
        e=lt.size(eventos)
        teve+= e
    return resultados,tmag,teve
```

En este requerimiento se retorna los eventos sísmicos dentro de un rango de magnitudes dado por el usuario. Principalmente con la función Values de árboles RBT, se saca las listas del rango necesitado y se les hace las estadísticas necesarias

Entrada	Árbol RBT, de magnitudes y los rangos insertados por el usuario
Salidas	Tabla con las estadísticas de cada magnitud para mostrar al usuario.
Implementado (Sí/No)	Si. Implementado por Grupo

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Om.Values	O(log N)
Nueva lista	O(N)
Iteración en lista	O(N)
Tamaños de mapas y listas	O(1)
AddFirst en enlazada	O(N)
Conteo estadistica	O(N)
TOTAL	O(N)

Pruebas Realizadas:

La prueba de tiempos se hará como parámetros, magnitud inferior= 3.5, superior=6.5

Procesadores	**Intel(R) Core(™) i7-8550U CPU @ 1.80GHz, 1992 Mhz**
Memoria RAM	**8 GB**
Sistema Operativo	**Windows 10**

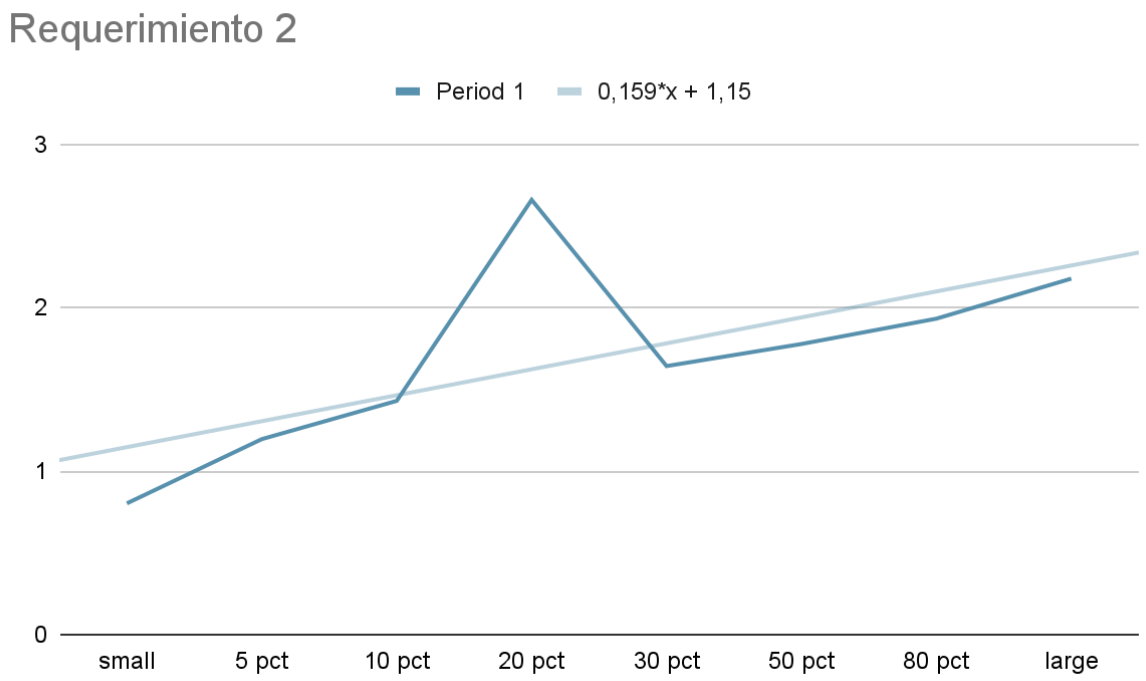
TIEMPOS

Entrada	**Tiempo (ms)**
small	**0.803**
5 pct	**1.196**
10 pct	**1.430**
20 pct	**2.660**
30 pct	**1.643**
50 pct	**1.778**
80 pct	**1.933**
large	**2.178**

Tablas de datos:

Muestra	Entrada	Tiempo (ms)
small	3.5, 6.5	**0.803**
5 pct	3.5, 6.5	**1.196**
10 pct	3.5, 6.5	**1.430**
20 pct	3.5, 6.5	**2.660**
30 pct	3.5, 6.5	**1.643**
50 pct	3.5, 6.5	**1.778**
80 pct	3.5, 6.5	**1.933**
large	3.5, 6.5	**2.178**

Gráfica



Análisis:

Gracias a que solo se tuvo que filtrar los valores de un map por magnitud, facilitó a que la búsqueda de los datos y realizar las estadísticas fuera rápido, solamente se itero la lista de Values y por eso se esperaba que el requerimiento tuviera esa complejidad de O(N); y a partir de la gráfica llegamos a lo esperado en la teoría, ya que la línea de tendencia es lineal

Requerimiento <<3>>.

Descripción:

```
def req_3(countries, pais, fecha_ini, fecha_fin, goalscorers):
    """
    Función que soluciona el requerimiento 3
    """
    fecha_ini = datetime.strptime(fecha_ini, "%Y-%m-%d")
    fecha_fin = datetime.strptime(fecha_fin, "%Y-%m-%d")
    home = 0
    away = 0
    presente = mp.contains(countries,pais)
    if presente:
        match = lt.newList("ARRAY_LIST")
        entry = mp.get(countries,pais)
        partidos = me.getValue(entry)
        npaises = mp.size(countries)
        nequipo = lt.size(partidos)
        for partido in lt.iterator(partidos):
            fecha = datetime.strptime(partido["date"], "%Y-%m-%d")
            if fecha >= fecha_ini and fecha <= fecha_fin:
                lt.addLast(match,partido)
            if partido["home_team"] == pais:
                home += 1
            elif partido["away_team"] == pais:
                away += 1
```

```
final = lt.newList("ARRAY_LIST")
for partido in lt.iterator(match):
    partido["penalty"] = "Unknown"
    partido["own_goal"] = "Unknown"
    date_result = datetime.strptime(partido["date"], "%Y-%m-%d")
    results = busqueda_partido(match_condition,date_result,partido["home_team"],partido["away_team"],pais)
    if results != -1:
        if results["own_goal"] == "True" or results["own_goal"] == "False":
            partido["own_goal"] = results["own_goal"]
        if results["penalty"] == "True" or results["penalty"] == "False":
            partido["penalty"] = results["penalty"]
    lt.addLast(final, partido)

return final, npaises, nequipo, home, away
```

Este requerimiento retorna los 10 eventos sísmicos más recientes segun una magnitud y una profundidad, para esto se filtran los datos y luego se meten en un arbol para organizarlos según la fecha

Entrada	La entrada se compone de los datos generales como una lista para iterarla y filtrar segun los datos pertinentes, además pedimos por parámetro la magnitud a superar y el limite de profundidad para buscar
Salidas	Saca una lista con todos los eventos sismicos que tienen esos parametros para luego a esos n datos, sacarles los 10 primeros y a esos 10 primeros sacar los 3 primeros y tres ultimos
Implementado (Sí/No)	**Si. Implementado por David Quiroga**

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear RBT	O(1)
Iterar en la lista y filtrar datos	O(N)
Añadir datos a un RBT, a partir de la iteración	O(NlogN)
Añadir datos a lista	O(1)
om.Values	O(logN)
Iterar valores del arbol	O(N)
Añadir a lista final	O(1)
Sacar el tamaño de listas y RBT	O(1)
TOTAL	O(NlogN)

Pruebas Realizadas:

La prueba de tiempos se hará como parámetros, pais: Argentina, fecha inicial: 1963-01-01, fecha final: 2019-12-31

Procesadores	**AMD Ryzen 3 3300U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	**16 GB**
Sistema Operativo	**Windows 10**

TIEMPOS

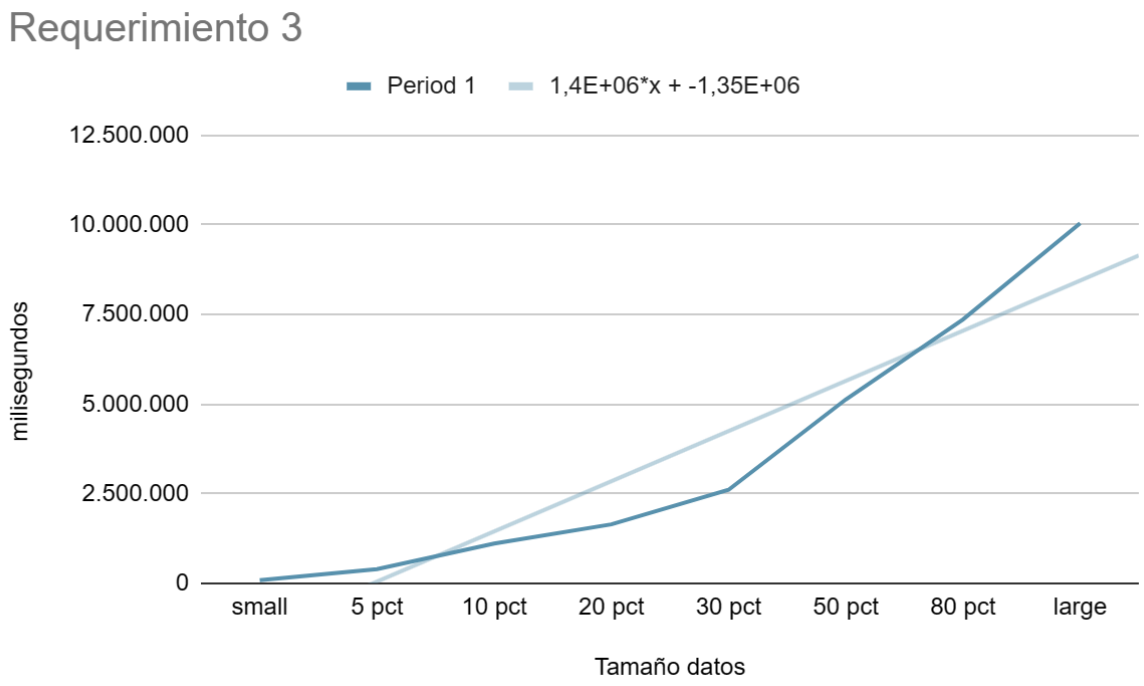
Entrada	**Tiempo (ms)**
small	87.486
5 pct	395.370
10 pct	1112.564
20 pct	1644.960
30 pct	2608.104
50 pct	5129.128
80 pct	7361.934
large	10043.285

Tablas de datos:

Muestra	**Entrada**	**Tiempo (ms)**
small	4.500, 10.000	87.486
5 pct	4.500, 10.000	395.370
10 pct	4.500, 10.000	1112.564
20 pct	4.500, 10.000	1644.960
30 pct	4.500, 10.000	2608.104
50 pct	4.500, 10.000	5129.128

80 pct	4.500, 10.000	7361.934
large	4.500, 10.000	10043.285

Gráficas:



Análisis:

Al ya tener la lista con todos los datos sísmicos, se itero esa lista para filtrar según la magnitud y la profundidad, luego de eso se creó un árbol y se insertaron los eventos sísmicos que pasaron el filtro para ser ordenados por fecha ya que así lo query esta parte. Luego en el view nos encargamos de sacar los 10 datos más recientes y a estos 10 datos sacar los 3 primeros y los 3 últimos. En conclusión este requerimiento implementa un arbol dentro del mismo requerimiento, no fue necesario crear uno global para hacerlo

Requerimiento <<4>>.

```python
def req_4(analyzer, minsig, maxgap):
    """
    Función que soluciona el requerimiento 4
    """
    lista_sismos = analyzer["earthquakes"]
    sismos_ordenados = merg.sort(lista_sismos, sort_criteria)
    resultado = lt.newList("ARRAY_LIST")

    for sismo in lt.iterator(sismos_ordenados):
        if sismo["cdi"] == "":
            sismo["cdi"] = "Unavailable"
        if sismo["mmi"] == "":
            sismo["mmi"] = "Unavailable"
        gap = sismo["gap"]
        sig = sismo["sig"]
        if gap != "" and sig != "":
            gap = sismo["gap"]
            sig = sismo["sig"]
            if float(sig) >= minsig and float(gap) <= maxgap:
                sismo["mag"]= sismo["mag"]
                sismo["lat"] = sismo["lat"]
                sismo["long"] = sismo["long"]
                sismo["depth"] = sismo["depth"]
                sismo["sig"] = sismo["sig"]
                sismo["gap"] = sismo["gap"]
                sismo["nst"] = sismo["nst"]
                sismo["title"] = sismo["title"]
                sismo["cdi"] = sismo["cdi"]
                sismo["mmi"]= sismo["mmi"]
                sismo["magType"] = sismo["magType"]
                sismo["type"] = sismo["type"]
                sismo["code"] = sismo["code"]
                lt.addLast(resultado, sismo)

    cantidad_de_sismos = lt.size(resultado)
    return lt.subList(resultado,1,15), cantidad_de_sismos
    # TODO: Realizar el requerimiento 4
    pass
```

Descripción:

La función toma como parámetros una estructura de datos, un número mínimo de significancia del evento y la distancia azimutal máxima del mismo. Primero se llama una lista sacada de la carga de datos, se ordena usando un sort criteria y se crea una nueva lista que llamamos resultado. Dentro de la lista de resultados se guardará la información de los sismos que cumplan con los criterios estipulados, cuando se le saca información a cada uno de los sismos tal como los valores de “gap” y “sig”. Por último se hace el calculo de cuantos sismos cumplen con los requerimientos y se retorna una sublista que contiene la cantidad de los 15 primeros elementos de la lista de resultados.

Entrada	Organización que contiene las estructuras de los datos del modelo inicializadas anteriormente, una significancia mínima sobre la que el usuario desee indagar y la distancia azimutal máxima que se quiera sobre los eventos
Salidas	El total de eventos sísmicos que fueron registrados según el cumplimiento de los criterios pasados por parámetro, una tabla de los 15 primeros eventos que cumplen con todas las características estipuladas.
Implementado (Sí/No)	Si. Implementado por Nikol De La Cruz

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Ordenamiento de sismos	O(n log n)
Iteración sobre los resultados ordenados	O(n)
Comparación de propiedades	O(n)
Insertar resultados a una lista	O(n)
TOTAL	O(n log n)

PRUEBAS REALIZADAS:

Para la medida de los datos se usaron los parámetros: Significancia mínima: “300.0” y distancia azimutal máxima: “45.0”.

Procesadores	**Intel(R) Pentium(R) Silver N6000 @ 1.10GHz 1.11 GHz**
Memoria RAM	**4,00 GB**
Sistema Operativo	**Windows 11**

TIEMPOS:

Entrada	**Tiempo (ms)**
small	297.6322
5 pct	1955.8379
10 pct	18147.070
20 pct	16236.621
30 pct	39060.274
50 pct	6742.36904

80 pct	440115.37290
large	204064.7665

Tablas de datos:

Muestra	**Entrada**	**Tiempo (ms)**
small	300.0 , 45.0	297.6322
5 pct	300.0 , 45.0	1955.8379
10 pct	300.0 , 45.0	18147.070
20 pct	300.0 , 45.0	16236.621
30 pct	300.0 , 45.0	39060.274
50 pct	300.0 , 45.0	6742.36904
80 pct	300.0 , 45.0	440115.37290
large	300.0 , 45.0	204064.7665

Gráfica

Análisis:

El análisis de complejidad nos dice que la complejidad principal está asociada con el proceso de ordenamiento que se puede observar en el código. La función posee una complejidad de O(n log n), lo que indica que el tiempo de procesamiento de datos aumentará de manera logarítmica en relación con el tamaño de la entrada. A pesar de que esto nos permite deducir que la función es eficiente tanto para conjuntos pequeños como para grandes, el tiempo tomado de las pruebas hechas por consola no es el mejor, ya que llega a demorarse más tiempo del que podría si en la implementación del código se hiciese uso de los ADT adecuados como los árboles.

Requerimiento <<5>>.

Descripción

```python
def req_5(lista,mindepth,minnst):
    """
    Función que soluciona el requerimiento 5
    """
    dates = om.newMap(omaptype="RBT")
    resultados = lt.newList("SINGLE_LINKED")
    teve=0
    for earth in lt.iterator(lista):
        if earth["depth"]=="":
            depth = 0
        else:
            depth= float(earth["depth"])
        if earth["nst"]=="":
            nst = 1
        else:
            nst= float(earth["nst"])
        if depth >= mindepth and nst>= minnst:
            updateDate5(dates, earth)
    valores= om.valueSet(dates)
    tdate = lt.size(dates)
    for ear in lt.iterator(valores):
        datos={"time":None,"events":None,"details":None}
        eventos= ear["lstdate"]
        d1 = lt.firstElement(eventos)
        tiempo= d1["time"]
        datos["time"]= tiempo
        datos["events"] = lt.size(eventos)
        datos["details"] = eventos
        lt.addFirst(resultados,datos)
        e=lt.size(eventos)
        teve+= e

    return resultados,tdate,teve
```

Este requerimiento, pide retornar los 20 eventos sísmicos que superen los datos mínimos de profundidad y radares de detección que inserta el usuario. Principalmente en el requerimiento se crea un RBT de clave date a partir de la lista de sismos y que superen los filtros insertados por el usuario.

Entrada	Lista de sismos, valores mínimos de profundidad y radares
Salidas	Listas de cada fecha con estadísticas solicitadas a partir de los valores mínimos insertados.
Implementado (Sí/No)	Si. Implementado por Nicolás González

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Crear RBT	O(1)
Iterar en la lista	O(N)
Añadir datos a un RBT, a partir de la iteración	O(NlogN)
Iterar en una lista de jugadores	O(N)
Comparaciones	O(1)
om.Values	O(logN)
Sacar el tamaño de listas y RBT	O(1)
Añadir elementos al principio de una enlazada	O(1)
Conteos	O(N)
TOTAL	O(N log N)

Pruebas Realizadas:

La prueba de tiempos se hará como parámetros, depth=23 y nst=38

Procesadores	**Intel(R) Core(™) i7-8550U CPU @ 1.80GHz, 1992 Mhz**
Memoria RAM	**8 GB**
Sistema Operativo	**Windows 10**

TIEMPOS:

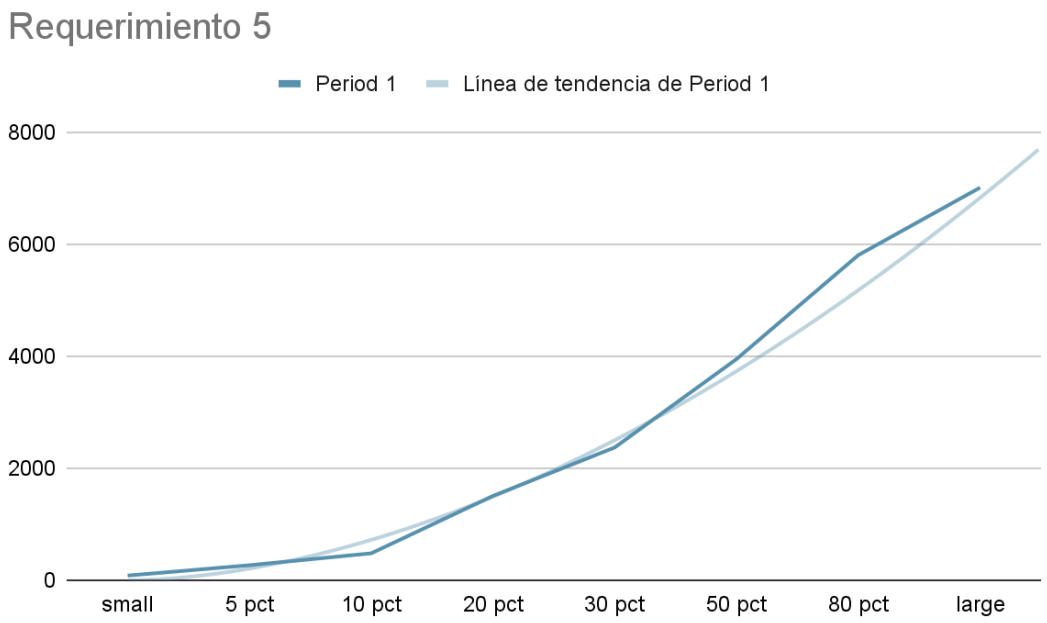
Entrada	**Tiempo (ms)**
small	83.212
5 pct	267.87
10 pct	480.27
20 pct	1505.106
30 pct	2370.141

50 pct	3947.366
80 pct	5803.044
large	7008.364

Tablas de datos

Muestra	**Entrada**	**Tiempo (ms)**
small	23 km, 38	83.212
5 pct	23 km, 38	267.87
10 pct	23 km, 38	480.27
20 pct	23 km, 38	1505.106
30 pct	23 km, 38	2370.141
50 pct	23 km, 38	3947.366
80 pct	23 km, 38	5803.044
large	23 km, 38	7008.364

Gráficas:



Análisis:

En este requerimiento lo que provocó la complejidad teórica, fue el hecho de crear mapas y añadir valores a este, a pesar de esto y con la nueva estructura, esto facilitó a la búsqueda de lo que pedía el requerimiento con una complejidad de O(N log N). Esto se puede comprobar en la gráfica ya que su tendencia es línea rítmica como lo que se esperaba.

Requerimiento <<6>>.

```python
def req_6(lista,anio,latitud,longitud,radio,N):
    """
    Función que soluciona el requerimiento 6
    """
    signi= om.newMap(omaptype="RBT")
    dates= om.newMap(omaptype="RBT")
    resultados= lt.newList("SINGLE_LINKED")
    teve=0
    for earthquake in lt.iterator(lista):
        date= datetime.strptime(earthquake["time"], "%Y-%m-%dT%H:%M:%S.%fZ")
        num= datetime.strftime(date, "%Y")
        if earthquake["lat"] =="":
            lat=0
        else:
            lat= float(earthquake["lat"])
        if earthquake["long"] =="":
            lon=0
        else:
            lon= float(earthquake["long"])
        punto1= (lat,lon)
        punto2= (latitud,longitud)
        hav=haversine(punto1,punto2)
        earthquake["hav"]=hav
        if hav <= radio and num == anio:
            updateSig(signi,earthquake)
            updateDate5(dates, earthquake)
    teve=om.size(dates)
    masSig= om.maxKey(signi)
    sigval= om.get(signi,masSig)
    listasig=me.getValue(sigval)
    for i in lt.iterator(listasig["lstsig"]):
        fecha= datetime.strptime(i["time"], "%Y-%m-%dT%H:%M:%S.%fZ")
        datos_significativo=i
    tamañoDates= om.size(dates)
    valores= om.valueSet(dates)
    muestra=lt.newList("SINGLE_LINKED")
    if tamañoDates <= N:
        muestra=valores
    else:
        datval= om.get(dates,fecha)
        datval=me.getValue(datval)
        lt.addLast(muestra,datval["lstdate"])
        antes= om.floor(dates,fecha-timedelta(seconds=1))
        despues=om.ceiling(dates,fecha+timedelta(seconds=1))
        while N > lt.size(muestra):

            if antes == None and despues != None:
                datdesp = om.get(dates,despues)
                datdesp=me.getValue(datdesp)
                lt.addFirst(muestra,datdesp["lstdate"])
                despues = om.ceiling(dates,despues+timedelta(seconds=1))
            elif antes != None and despues == None:
                datant = om.get(dates,antes)
                datant=me.getValue(datant)
                lt.addLast(muestra,datant["lstdate"])
                antes = om.floor(dates,antes-timedelta(seconds=1))

            elif antes!= None and despues != None:
                delta_antes= fecha-antes
                delta_despues= despues-fecha
                if delta_antes > delta_despues:
                    datdesp = om.get(dates,despues)
                    datdesp=me.getValue(datdesp)
                    lt.addFirst(muestra,datdesp["lstdate"])
                    despues = om.ceiling(dates,despues+timedelta(seconds=1))
                elif delta_antes < delta_despues:
                    datant = om.get(dates,antes)
                    datant=me.getValue(datant)
                    lt.addLast(muestra,datant["lstdate"])
                    antes = om.floor(dates,antes-timedelta(seconds=1))
                elif delta_despues==delta_antes:
                    datdesp = om.get(dates,despues)
                    datdesp=me.getValue(datdesp)
                    lt.addFirst(muestra,datdesp["lstdate"])
                    despues = om.ceiling(dates,despues+timedelta(seconds=1))
                    datant = om.get(dates,antes)
                    datant=me.getValue(datant)
                    lt.addLast(muestra,datant["lstdate"])
                    antes = om.floor(dates,antes-timedelta(seconds=1))
    for m in lt.iterator(muestra):
        datos= {"time":None,"events":None,"details":None}
        d1= lt.firstElement(m)
        tiempo= d1["time"]
        datos["time"]= tiempo
        datos["events"] = lt.size(m)
        datos["details"] = m
        lt.addLast(resultados,datos)
    return datos_significativo, resultados,teve
```

Descripción:

La función retorna los N eventos próximos al sismo más significativo dentro de un radio dada su longitud y latitud en un año determinado. Principalmente la función itera en una lista y crea dos RBT con claves del dato sig y date. Con sig y la máxima llave del árbol, se saca el evento más significativo y con el de dates se busca los eventos próximos de ese evento; se hacen las estadísticas y retorna una lista.

Entrada	Lista de sismos, año, latitud, longitud, radio, número de proximidad
Salidas	Diccionario del evento significativo y lista de los datos próximos a ese evento
Implementado (Sí/No)	Si. Implementado por grupo

Análisis de complejidad:

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear RBT	O(1)
Iterar en lista sismos	O(N)
Comparaciones y cálculos como Haversine	O(1)
Añadir elementos a un RBT	O(N log N)
maxKey	O(N)
om.Values	O(log N)
.get	O(N)
Añadir elementos a lista	O(1)
Ceiling and Floor	O(log N)
Iteración lista final	O(N)
Conteo estadistica	O(N)
size list y RBT	O(N)
TOTAL	O(N log N)

PRUEBAS REALIZADAS:

Para la medida de los datos se usaron los parámetros: año= 2022, latitud= 4.674, longitud= -74.068, radio= 3000, N eventos= 7

Procesadores	**Intel(R) Core(™) i7-8550U CPU @ 1.80GHz, 1992 Mhz**
Memoria RAM	**8 GB**
Sistema Operativo	**Windows 10**

TIEMPOS:

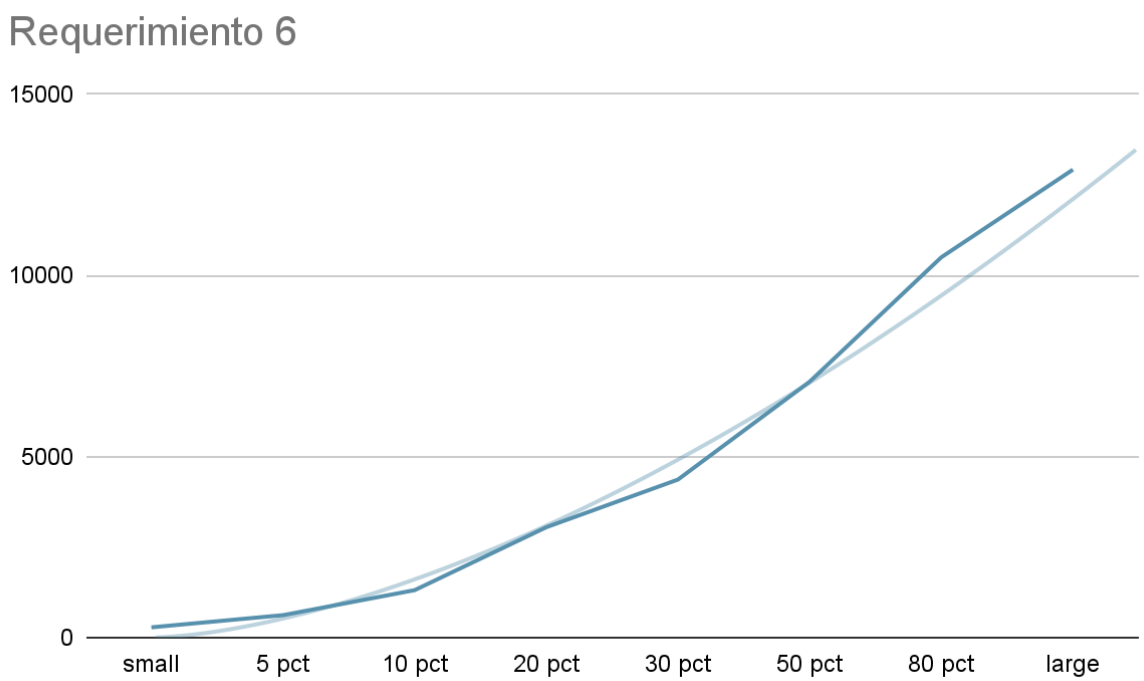
Entrada	**Tiempo (ms)**

small	**282.946**
5 pct	619.51
10 pct	1309.863
20 pct	3047.176
30 pct	4363.51
50 pct	7065.004
80 pct	10499.799
large	12920.875

Tablas de datos:

Muestra	Entrada	Tiempo (ms)
small	**2022, 4.674, -74.068, 3000, 7**	**282.946**
5 pct	**2022, 4.674, -74.068, 3000, 7**	619.51
10 pct	**2022, 4.674, -74.068, 3000, 7**	1309.863
20 pct	**2022, 4.674, -74.068, 3000, 7**	3047.176
30 pct	**2022, 4.674, -74.068, 3000, 7**	4363.51
50 pct	**2022, 4.674, -74.068, 3000, 7**	7065.004
80 pct	**2022, 4.674, -74.068, 3000, 7**	10499.799
large	**2022, 4.674, -74.068, 3000, 7**	12920.875

Gráfica :

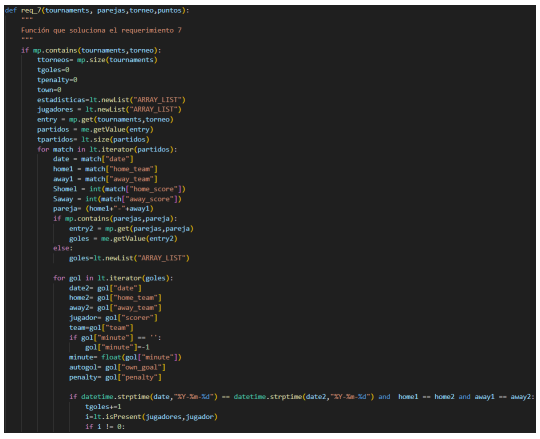


Análisis:

En este requerimiento lo que produjo la complejidad fue crear los dos RBT, pero esto facilitó después la búsqueda de los datos, además solo se itero después los valores de los mapas lo que no conllevo tanta complejidad, es por esto que la complejidad teórica di O(N log N). Esto se puede comprobar en la gráfica ya que tiende a ser línea rítmica, que es el resultado que se esperaba.

Requerimiento <<7>>.

Descripción:



```
def req_7(tournaments, parejas,torneo,puntos):
    """
    Función que soluciona el requerimiento 7
    """
    if mp.contains(tournaments,torneo):
        ttorneos= mp.size(tournaments)
        tgoles=0
        tpenalty=0
        town=0
        estadisticas=lt.newList("ARRAY_LIST")
        jugadores = lt.newList("ARRAY_LIST")
        entry = mp.get(tournaments,torneo)
        partidos = me.getValue(entry)
        tpartidos= lt.size(partidos)
        for match in lt.iterator(partidos):
            date = match["date"]
            home1 = match["home_team"]
            away1 = match["away_team"]
            Shome1 = int(match["home_score"])
            Saway = int(match["away_score"])
            pareja= (home1+"-"+away1)
            if mp.contains(parejas,pareja):
                entry2 = mp.get(parejas,pareja)
                goles = me.getValue(entry2)
            else:
                goles=lt.newList("ARRAY_LIST")

            for gol in lt.iterator(goles):
                date2= gol["date"]
                home2= gol["home_team"]
                away2= gol["away_team"]
                jugador= gol["scorer"]
                team=gol["team"]
                if gol["minute"] == '':
                    gol["minute"]=-1
                minute= float(gol["minute"])
                autogol= gol["own_goal"]
                penalty= gol["penalty"]

                if datetime.strptime(date,"%Y-%m-%d") == datetime.strptime(date2,"%Y-%m-%d") and  home1 == home2 and away1 == away2:
                    tgoles+=1
                    i=lt.isPresent(jugadores,jugador)
                    if i != 0:
```

```
                resultado = lt.getElement(estadisticas,i)
                resultado["total_goals"]+=1
                if penalty == "True":
                    resultado["penalty_goals"]+=1
                    tpenalty+=1

                if autogol == "True":
                    resultado["own_goals"]+=1
                    town+=1

                resultado["avg_time"] = ((resultado["avg_time"]*(resultado["total_goals"]-1))+minute)/resultado["total_goals"]

                if team == home1:
                    if Shome1 > Saway:
                        resultado["scored_in_wins"]+=1
                    elif Shome1 < Saway:
                        resultado["scored_in_losses"]+=1
                    else:
                        resultado["scored_in_draws"]+=1
                if team == away1:
                    if Shome1 > Saway:
                        resultado["scored_in_losses"]+=1
                    elif Shome1 < Saway:
                        resultado["scored_in_wins"]+=1
                    else:
                        resultado["scored_in_draws"]+=1
                resultado["total_points"]= resultado["total_goals"]+resultado["penalty_goals"]-resultado["own_goals"]
            else:
                resultado={"scorer":jugador,
                           "total_points":0,
                           "total_goals":1,
                           "penalty_goals":0,
                           "own_goals":0,
                           "avg_time":minute,
                           "scored_in_wins":0,
                           "scored_in_losses":0,
                           "scored_in_draws":0,
                           "last_goal": [date,torneo,home1,away1,Shome1,Saway,minute,penalty,autogol]
                           }
                if penalty == "True":
                    resultado["penalty_goals"]+=1
                    tpenalty+=1
```

Este requerimiento retorna una tabla con los eventos sísmicos más recientes y más antiguos de un año especifico, todo esto se complementa con el histograma de frecuencia el cual nos dice cuántas veces se repite un evento sísmico según una propiedad que le demos, ya sea magnitud, profundidad o sig

Entrada	Se dan los datos generales como parámetro y se le pide al usuario que ingrese el año que quiere buscar, la propiedad que quiere mirar en el histograma y un título relacionado con el evento sísmico, en la gran mayoría de casos el país pero puede variar segun la informacion de cada dato en este apartado
Salidas	Retorna una lista con los datos que iran en la tabla y otra lista con los datos pertinentes para hacer el histograma
Implementado (Sí/No)	Si. Implementado por grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear RBT	O(1)
Iterar en lista sismos y filtro de datos	O(N)
Añadir a una lista	O(1)
Añadir elementos de lista a un RBT	O(N log N)

om.ValuesSet	O(N)
Formar una lista con los diccionarios iniciales	O(1)
Sacar elementos según arbol	O(N)
Iterar para sacar los diccionarios	$O(N^2)$
Insertarlos en una lista	O(1)
TOTAL	$O(N^2)$

Pruebas Realizadas:

La prueba de tiempos se hará como parámetros, torneo: FIFA World Cup, 7

Procesadores	**AMD Ryzen 3 3300U with Radeon Vega Mobile Gfx 2.10 GHz**
Memoria RAM	**16 GB**
Sistema Operativo	**Windows 10**

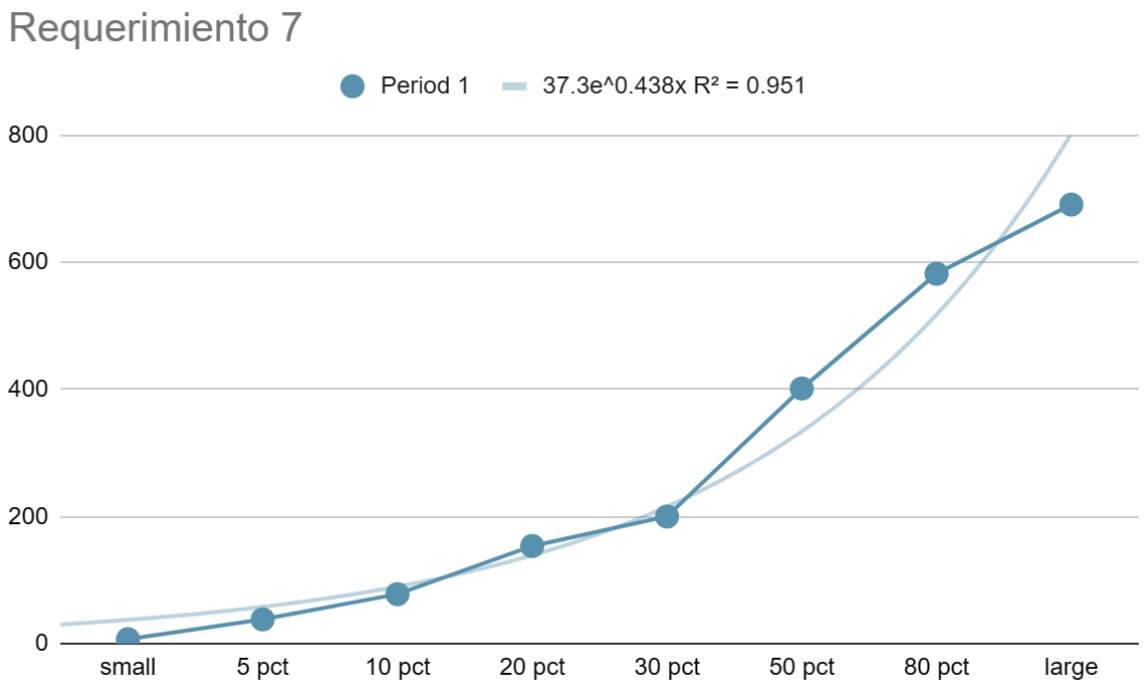
TIEMPOS:

Entrada	**Tiempo (ms)**
small	6.892
5 pct	38.105
10 pct	77.788
20 pct	153.587
30 pct	200.044
50 pct	401.382
80 pct	582.018
large	690.870

Tablas de datos:

Muestra	Entrada	Tiempo (ms)
small	2020, Alaska, mag, 10	6.892
5 pct	2020, Alaska, mag, 10	38.105
10 pct	2020, Alaska, mag, 10	77.788
20 pct	2020, Alaska, mag, 10	153.587
30 pct	2020, Alaska, mag, 10	200.044
50 pct	2020, Alaska, mag, 10	401.382
80 pct	2020, Alaska, mag, 10	582.018
large	2020, Alaska, mag, 10	690.870

Gráficas:



Análisis:

Ya que tocaba recorrer dos veces un árbol para sacar los valores limpios de los eventos sismicos se nos aumentó la complejidad a N al cuadrado pero igual la carga es bastante estable y sin requerir mucho tiempo ya que los filtrados según propiedades nos reducen el tamaño de las listas que iteramos para meterlos a los árboles