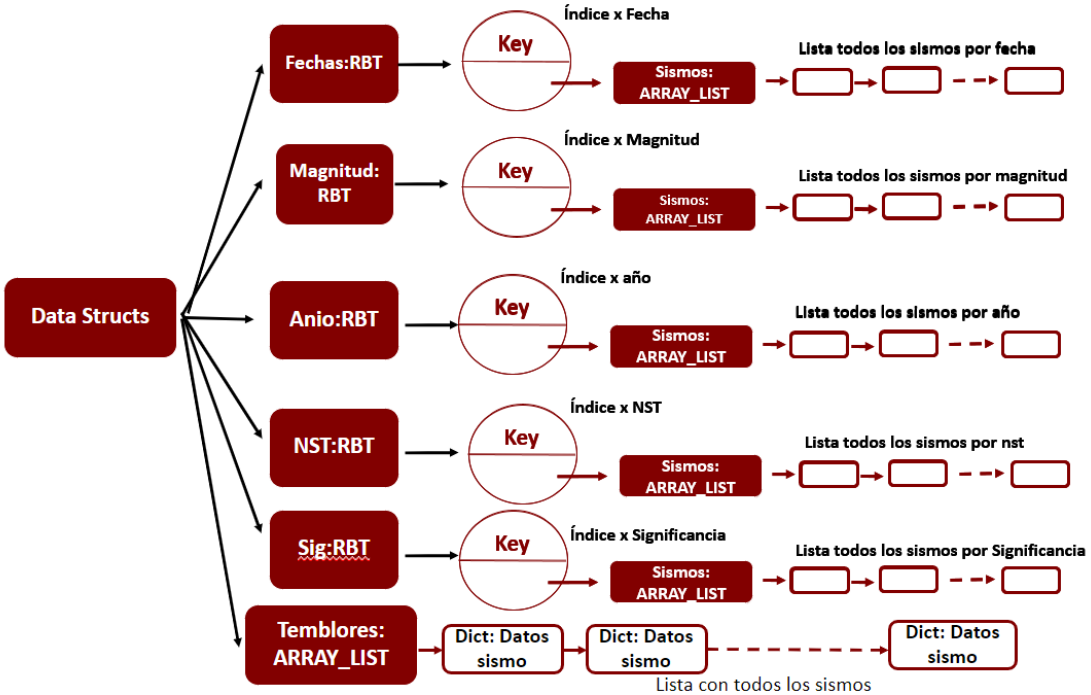
ANÁLISIS DEL RETO

Estudiante 1: Gabriela Zambrano, 202211712, g.zambranoz@uniandes.edu.co

Estudiante 2: María José Mantilla, 202121670, m.mantillav@uniandes.edu.co

Estudiante 3: Dayanna Rueda, 202015345, l.ruedap@uniandes.edu.co

Diagrama estructura de datos



Nota: se implementaron estas estructuras a lo largo del reto, es decir un mapa pudo ser implementado para resolver varios requerimientos.

Carga de Datos

Plantilla para el documentar y analizar cada uno de los requerimientos.

```python
def loadData(data_structs,archivo):
    """
    Carga los datos de los archivos CSV en el modelo
    """
    archivo = cf.data_dir + "earthquakes/temblores-utf8-"+ archivo
    archivo_leido = csv.DictReader(open(archivo, encoding="utf-8"))
    for cada_linea in archivo_leido:
        if cada_linea["felt"] == "":
            cada_linea["felt"] = "UnKnown"
        else:
            cada_linea["felt"] = round(float(cada_linea["felt"]),3)

        if cada_linea["cdi"] == "":
            cada_linea["cdi"] = "UnKnown"
        else:
            cada_linea["cdi"] = round(float(cada_linea["cdi"]),3)

        if cada_linea["mmi"] == "":
            cada_linea["mmi"] = "UnKnown"
        else:
            cada_linea["mmi"] = round(float(cada_linea["mmi"]),3)

        if cada_linea["tsunami"] == "0":
            cada_linea["tsunami"] = "False"
        if cada_linea["nst"] == "":
            cada_linea["nst"]=1
        else:
            cada_linea["nst"]=float(cada_linea["nst"])


        if cada_linea["gap"] == "":
            cada_linea["gap"]=1
        else:
            cada_linea["gap"]=float(cada_linea["gap"])

        cada_linea["depth"]=float(cada_linea["depth"])
        cada_linea["sig"]=float(cada_linea["sig"])
        cada_linea["mag"]=round(float(cada_linea["mag"]),3)


        tiempo,update=aproximar_tiempo(cada_linea["time"],cada_linea["updated"])
        cada_linea["time"] = tiempo
        cada_linea["updated"] = update


        dic_imp =  {"code": cada_linea["code"],
                    "time" : cada_linea["time"],
                    "lat" : round(float(cada_linea["lat"]),3),
                    "long": round(float(cada_linea["long"]),3),
                    "mag" : round(float(cada_linea["mag"]),3),
                    "title" : cada_linea["title"],
                    "depth": float(cada_linea["depth"]),
                    "felt" : cada_linea["felt"],
                    "cdi" : cada_linea["cdi"],
                    "mmi": cada_linea["mmi"],
                    "tsunami" : cada_linea["tsunami"]
                    }
        add_data_lista(data_structs,dic_imp)
        add_data_arbol_list(data_structs["fechas"],cada_linea,"time")
        add_data_arbol_list(data_structs["magnitud"],cada_linea,"mag")
        add_data_arbol_list(data_structs["nst"],cada_linea,"nst")
        add_data_arbol_list_anio(data_structs["anio"],cada_linea)
        add_data_arbol_list(data_structs["sig"],cada_linea,"sig")
    return lt.size(data_structs["temblores"])
```

Descripción

Con el fin de realizar la carga de datos, del archivo de sismos se creó una función llamada loadData, en esta, lo primero que realiza es acceder al nombre del archivo, a partir de su tamaño, luego, se realiza la lectura del documento con csv.DictReader (esta herramienta va leyendo línea a línea el documento). Luego, se realizan varios procedimientos con los valores extraídos de cada línea del documento, algunos de estos procesos son completar los espacios vacíos ya sea con UnKown o con un “1”, o aproximar los valores en formato “float” a tres decimales. Luego, se crea un diccionario con valores específicos que se van agregando poco a poco a una lista, la cual queda en el orden que van entrando los datos. Adicionalmente, se agregan los valores a los diferentes mapas, estos fueron creados para resolver de forma más eficiente los requerimientos. Finalmente, se retorna el tamaño de la lista creada.

Entrada	Entran como parámetros: • control (control de datos para el modelo, que posteriormente será el parámetro control[“model”]. • archivo (archivo que contiene los datos para cargar en las estructuras).
Salidas	Retorna:

	- 1 lista con todos los sismos que se encuentran en el archivo - Árbol “fechas”, datos ordenados en listas teniendo como llave la fecha del sismo. - Árbol “magnitud”, datos ordenados en listas teniendo como llave la magnitud del sismo. - Árbol “nst”, datos ordenados en listas teniendo como llave el nst del sismo. - Árbol “anio”, datos ordenados en listas teniendo como llave del año en el que ocurrió el sismo.
Implementado (Sí/No)	Si, implementado por Maria Jose Mantilla y Gabriela Zambrano

Análisis de complejidad

La carga de todos los archivos es la misma, lo único que cambia los tamaños de los documentos:

Pasos	**Complejidad**
Formar nombre de archivo	O(1)
Leer archivo	O(n)
Iteración en archivo	O(n)*O(log(n))=O(nlog(n))
Condicionales y cambios de valores	O(1)
Creación de diccionario para la lista	O(1)
Añadir datos a lista	O(1)
Añadir datos a los mapas	O(2log(n))
TOTAL	***O(nlog(n))***

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	62303,5059 ms
Condición, herramientas y recursos utilizados:	• csv.DictReader • Un ciclo for en cada función de carga de cada uno de los tres archivos
Computador donde se ejecuta:	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz (8 GB Windows 11 Home)

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

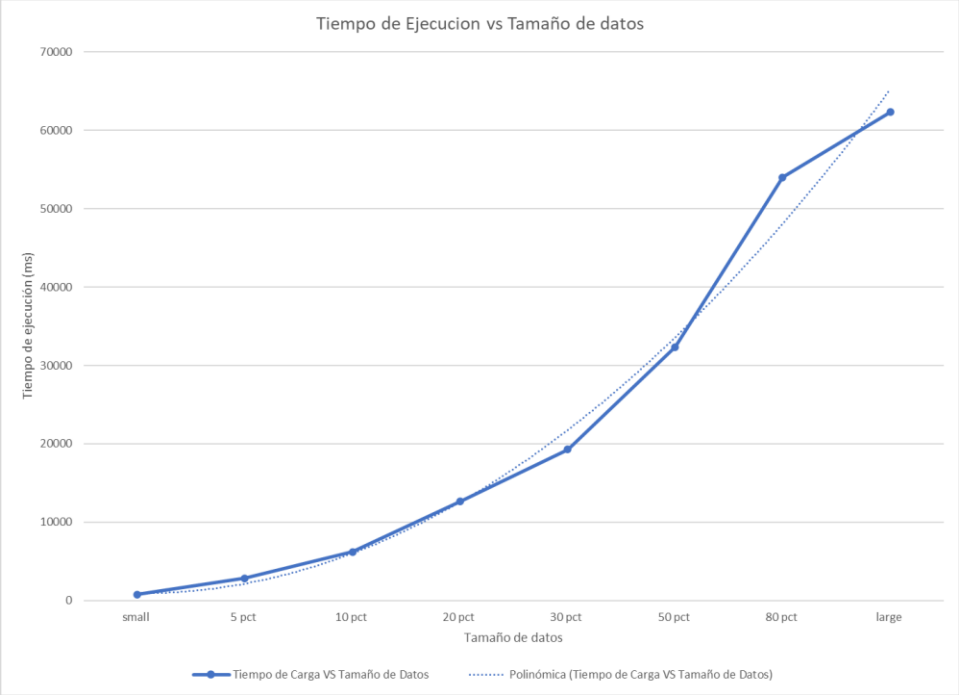
Tiempo

Entrada	Tiempo de Ejecución Real [ms]
Small	770,7397
5 pct	2862,7665
10 pct	6196,6088
20 pct	12631,9262
30 pct	19291,9037
50 pct	32350,6236
80 pct	54000,514
Large	62303,5059

Graficas

Las gráficas con la representación de las pruebas realizadas.

Tiempo de Ejecución



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad

Con las pruebas realizadas, se puede observar que se obtuvo una línea de tendencia polinómica de grado 2, este resultado se encuentra de acorde a la complejidad encontrada, la cual fue de O(nlog(n)). Por lo tanto, se puede decir que los resultados obtenidos fueron acertados y que la complejidad encontrada fue la correcta. Adicionalmente, mientras más aumenten los datos, mayor será el tiempo de ejecución.

Requerimiento 1

```python
def req_1(data_structs,inicialdate,finaldate):
    """
    Función que soluciona el requerimiento 1
    """
    date_map = data_structs["fechas"]
    rango_llaves = om.keys(date_map,inicialdate,finaldate)
    total_fechas = lt.size(rango_llaves)
    total_sismos = 0
    altura = om.height(date_map)
    nodos = om.size(date_map)
    elementos = lt.size(data_structs["temblores"])
    lista_final = lt.newList("ARRAY_LIST")
    for llave in lt.iterator(rango_llaves):
        pareja = om.get(date_map,llave)
        sismos = me.getValue(pareja)
        sismo_x_fecha= lt.size(sismos)
        total_sismos += sismo_x_fecha
        lista_dic = lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sismos):
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_dic,pequeno)
        details=creartabla(lista_dic,lista_dic["elements"][0])

        dict_grande = big_dict(llave,"time",sismo_x_fecha,details)
        lt.addLast(lista_final,dict_grande)
    return lista_final,total_sismos,total_fechas,altura,nodos,elementos
```

Descripción

Para resolver el requerimiento utilizamos el mapa creado anteriormente, cuyas llaves son el tiempo en el que ocurrieron los sismos y mediante la función keys obtuvimos las llaves que se encontraban entre el rango indicado. Luego iteramos sobre esta lista de llaves con el fin de extraer el interior y adicionar el tamaño a un contador para ver el total de sismos en el rango de fechas. Finalmente, iteramos sobre la lista que contiene cada llave para crear la tabla respectiva de cada fecha y luego, retornamos la lista final después de todas las iteraciones.

Entrada	Entra por parámetro: • data_structs (Datos almacenados en los archivos ingresados) • Fecha inicial • Fecha final
Salidas	Retorna: • El número total de ventos entre las fechas indicada • Todos los eventos ocurridos en el intervalo de más reciente a más antiguo
Implementado (Sí/No)	Si, implementado por María José Mantilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

f = tamaño mapa de fecha

s= número de sismos por fecha

f > s

Pasos	Complejidad
Extraer llaves en un rango de fechas	O(logf)
Size,contadores y listas	O(1)
Iterar sobre el rango de llaves	O(f)
Extraer valor	O(1)
Size y contadores	O(1)
Iterar sobre la lista valor de cada llave	O(s)
Crear dic y addlast	O(1)
addLast	O(1)
TOTAL	***O(f*s)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	15885.41 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, get, getValue, iterator, sublist). • Un ciclo for para la iteración de los sismos.
Computador donde se ejecuta:	11th Gen Intel(R) Core i7 – 11800H @ 2.30 GHz 2.30GHz 16GB

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Fecha inicial: 1999-03-21T05:00

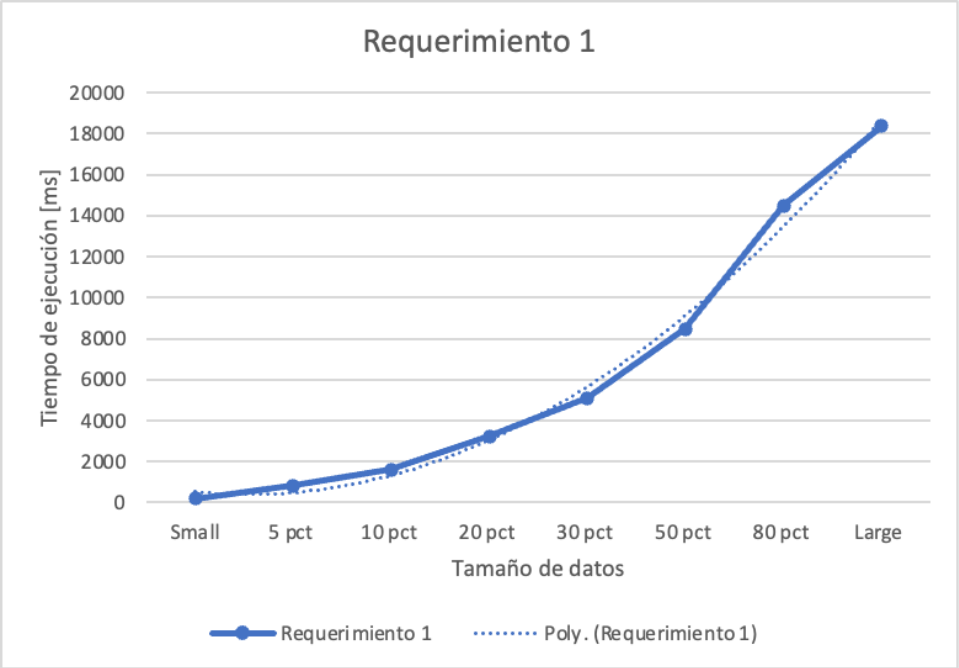
Fecha final: 2004-10-23T17:30

Entrada	Tiempo de Ejecución Real [ms]
Small	201,36
5 pct	821,84
10 pct	1602,82

20 pct	3234,29
30 pct	5103,9
50 pct	8475,13
80 pct	14475,21
Large	15885.41

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Con base en los resultados obtenidos se puede decir que el tiempo que toma este requerimiento al llevarse a cabo se debe más que todo a la iteración que hace sobre todas las llaves del rango y luego la iteración sobre los sismos dentro de cada llave lo que en el peor caso es O(f). Sin embargo, gracias a la función .keys() de los árboles se puede reducir la complejidad al no tener que iterar sobre todo el árbol sino sobre las llaves que ya ofrece mediante búsqueda binaria.

Requerimiento 2

```python
def req_2(data_structs,mag_inf,mag_sup):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    #Se extraen las llaves en el rango de magnitud
    mag_map = data_structs["magnitud"]
    rango_llaves = om.keys(mag_map,mag_inf,mag_sup)
    total_llaves = lt.size(rango_llaves)

    #Se crea un contador vacio
    total_sismos = 0

    #Se halla la cantidad total de eventos
    elementos = lt.size(data_structs["temblores"])

    #Se crea una lista vacia
    lista_final = lt.newList("ARRAY_LIST")

    #Se itera sobre las llaves del rango de magnitudes y se obtiene su valor
    for llave in lt.iterator(rango_llaves):
        pareja = om.get(mag_map,llave)
        sismos = me.getValue(pareja)

    #Se halla la cantidad de sismos dobre este rango y se va sumando al contador de total de sismos
        sismo_entre_magnitudes= lt.size(sismos)
        total_sismos += sismo_entre_magnitudes

    #Se crea una lista vacía
        lista_dic = lt.newList("ARRAY_LIST")

    #Se iteran sobre los eventos dentro del rango de magnitudes y se agrega el valor "time" a la lista vacía anterior
        for sismo in lt.iterator(sismos):
            pequeno =  min_dict(sismo,"time")
            lt.addLast(lista_dic,pequeno)

    #Se ordenan los valores del más reciente al más antiguo
        quk.sort(lista_dic, cmpFechaReciente)

    #Se halla la cantidad de valores dentro de la lista
        size_lista_dic = lt.size(lista_dic)

    #Se crea una sublista en caso de que hayan más de 6 valores dentro de la lista
        if size_lista_dic > 6:
            sublista = resumir_lista(lista_dic,3,1)
        else:
            sublista = lista_dic

    #Se crea la tabla de datails
        details=creartabla(sublista,sublista["elements"][0])

    #Se crean los diccionarios para la tabla grande
        dict_grande  = big_dict(llave, "mag", sismo_entre_magnitudes, details)

    #Se agragan a la lista vacia inicial
        lt.addLast(lista_final, dict_grande)

    return lista_final, total_sismos, total_llaves, elementos
```

Descripción

En el requerimiento 2, en primer lugar, se extraen las llaves en el rango de magnitud. Asimismo, se crea un contador, una lista vacía, y se halla el total de lista de temblores. Posteriormente, se itera sobre las llaves del rango de magnitudes y se obtiene su valor. En base a esto, se halla la cantidad de sismos sobre este rango y se va sumando al contador. Después, se crea una lista vacía, y se itera sobre los eventos dentro del rango de magnitudes y se agrega el valor “time” a la lista vacía anterior. Seguidamente, se ordenan los eventos del más reciente al más antiguo, para así a partir de la cantidad obtenida del total de estos eventos se crea una sublista si el total es mayor a 6. Finalmente se crea la tabla de details, que se añade a los diccionarios agregados a la lista inicial para la tabla total.

Entrada	Entra por parámetro: • data_structs (Datos almacenados en los archivos ingresados). • mag_inf (El límite inferior de la magnitud). • mag_sup (El límite inferior de la magnitud).
Salidas	Retorna: • El número total de eventos entre las magnitudes indicadas.

	• Todos los eventos ocurridos en el intervalo de más fuerte al más débil.
Implementado (Sí/No)	Si, implementado por Dayanna Rueda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

m = tamaño mapa de magnitud

s= número de sismos por magnitud

m > s

Pasos	**Complejidad**
Extraer llaves en un rango de magnitud	O(log(m))
Size,contadores y listas	O(1)
Iterar sobre el rango de llaves de magnitud	O(m)* (O(slog(s))+O(s)) = O(m*slog(s))
Extraer valor	O(1)
Size y contadores	O(1)
Iterar sobre la lista valor de cada llave	O(s)
Ordenamiento quick sort	O(slog(s))
Crear dic y addlast	O(1)
TOTAL	*O(m*s*log(s))*

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	12191,38 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, keys, newList, iterator, get, getValue, addLast). • Un ciclo for para la iteración de los sismos. • Función de ordenamiento (Del más reciente al más antiguo).
Computador donde se ejecuta:	Apple M1 8GB

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Límite inferior de la magnitud: 3.500

Límite superior de la magnitud: 6.500

Entrada	Tiempo de Ejecución Real [ms]
Small	109,49
5 pct	391,48
10 pct	786,99
20 pct	1698,74
30 pct	2559,73
50 pct	4711,02
80 pct	9115,43
Large	12191,38

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

A partir del desglose de análisis de complejidad, encontramos que aquella complejidad más alta consiste en la iteración sobre el rango de llaves de magnitud denotada con O(m). Dentro de esta iteración hay una iteración sobre la lista valor de cada llave de magnitud denotada con O(s). Y, también un ordenamiento Quick Sort que tiene una complejidad O(slog(s)). Por lo tanto, realizando los cálculos, estas dos últimas operaciones se suman, y se deja la mayor siendo esta O(s*log(s)) y se multiplica con la primera iteración, para un total de O(m*s*log(s)). La cual podría llegar a ser considerada mayor a una curva exponencial si el valor de s es un número muy cercano a n. Así que, esta complejidad coincide con la línea de tendencia exponencial plasmada en la gráfica.

Requerimiento 3

```python
def req_3(data_structs, mag_min, depth_max):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    #Sacar llaves en rango de magnitud
    mapa_mag=data_structs["magnitud"]
    mag_max=om.maxKey(mapa_mag)
    llaves_estaciones=om.keys(mapa_mag,mag_min,mag_max)

    #Creacion de contadores y mapas
    total_eventos=0
    filtrado_eventos=om.newMap(omaptype="RBT")

    #Comprobacion de profundidad máxima
    for estacion in lt.iterator(llaves_estaciones):
        sismos=me.getValue(om.get(mapa_mag,estacion))
        for sis in lt.iterator(sismos):
            prof=int(sis["depth"])
            if prof<=depth_max:
                total_eventos+=1
                add_data_arbol_list(filtrado_eventos,sis,"time")

    #Sacar los 10 sismos mas recientes
    fechas_diferentes=om.size(filtrado_eventos)
    keys=om.keySet(filtrado_eventos)
    llaves_finales=lt.subList(keys,lt.size(keys)-9,10)
    lista_final=lt.newList("ARRAY_LIST")

    #Hacer diccionarios para la impresión de sismos finales
    for ll in lt.iterator(llaves_finales):
        sismo_x_fecha=me.getValue(om.get(filtrado_eventos,ll))
        lista_final_sis=lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sismo_x_fecha):
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_final_sis,pequeno)
        details=creartabla(lista_final_sis,lista_final_sis["elements"][0])

        dict_grande = big_dict(ll,"time",lt.size(sismo_x_fecha),details)
        lt.addLast(lista_final,dict_grande)

    return fechas_diferentes,total_eventos,lista_final
```

Descripción

Para el requerimiento 3, en primer lugar, extraemos las llaves en el rango de magnitud. Luego, creamos un contador y un mapa vacío. Posteriormente, filtros a partir del rango de profundidad y vamos agregando elementos al mapa. Después, en una sublista ubicamos los 10 eventos más recientes, y creamos una lista vacía. Finalmente, creamos los diccionarios que van a ser agregados a esta lista vacía para su posterior impresión.

Entrada	Entra por parámetro:

	• data_structs (Datos almacenados en los archivos ingresados) • mag_min (La magnitud mínima del evento) • depth_max (La profundidad máxima del evento)
Salidas	Retorna: • El número total de ventos entre los límites de magnitud y profundidad indicados. • Los diez (10) eventos cronológicamente más recientes que cumplan con las condiciones de profundidad y magnitud indicados.
Implementado (Sí/No)	Si, implementado por Dayanna Rueda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

n=total de datos

m= sismos por mag

n>m

Pasos	**Complejidad**
Extracción de llaves mayores al mag mínimo	O(log(n))
Creación mapas y contadores	O(1)
Iteración sobre sismos	O(n)*O(m)*O(log(n))=O(n*m*log(n))
Iteración sobre sismos de cada mag	O(m)
Comprobación de que cumple con la profundidad máxima e inserción al mapa	O(log(n))
Extracción de llaves de sismos que cumplen las condiciones	O(n)
Sublista con 10 más recientes	O(10)
Iteración sobre llaves	O(10)=O(1)
Iteración sobre sismos de cada llave	O(1)
Creación diccionarios	O(1)
TOTAL	*O(n*m*log(n))*

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	2264,78 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, keys, newList, iterator, get, getValue, addLast). • Un ciclo for para la iteración de los sismos. • Función de ordenamiento (Del más reciente al más antiguo).
Computador donde se ejecuta:	Apple M1 8GB

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

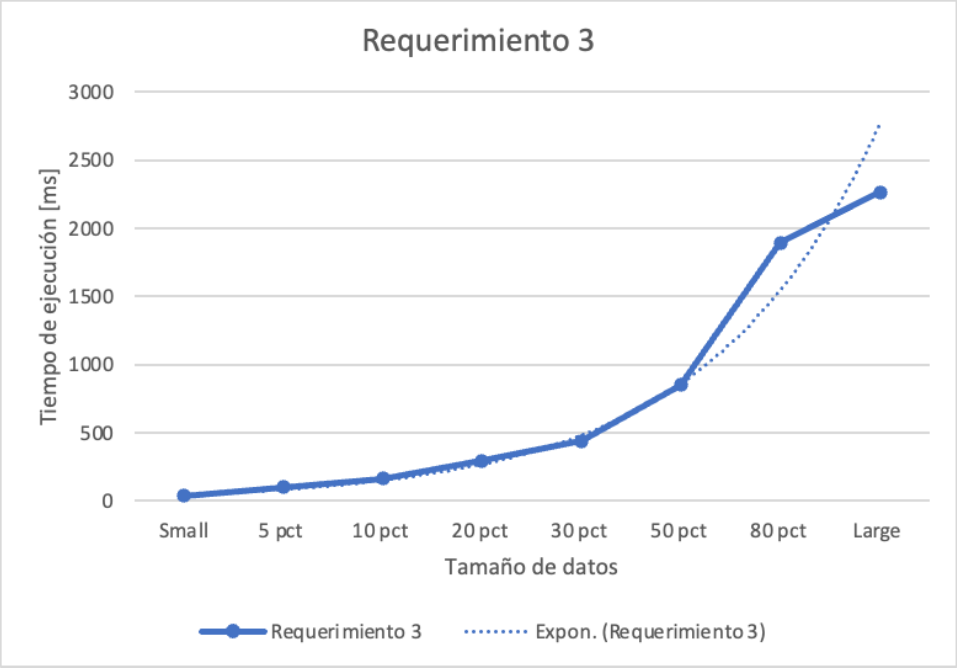
Magnitud mínima del evento: 4.700

Profundidad máxima del evento: 10.000

Entrada	Tiempo de Ejecución Real [ms]
Small	36,05
5 pct	102,97
10 pct	163,00
20 pct	297,14
30 pct	439,71
50 pct	850,60
80 pct	1892,16
Large	2264,78

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Teniendo en cuenta la tabla con la recopilación de datos de las pruebas, su respectiva gráfica, y el análisis de complejidad, podemos deducir que con coherentes el uno con el otro. Lo anterior, puesto que el total de complejidad correspondiente a O(n*m*log(n)) es resultado de la iteración sobre sismos de cada magnitud, dentro de la cual hay una Iteración sobre sismos de cada magnitud, y dentro de esta una comprobación de que cumple con la profundidad máxima e inserción al mapa. Por lo tanto, se podría decir que la complejidad es exponencial si el valor de m es un número muy cercano a n, ya que es la complejidad que prima por ser aquella de mayor valor. Así que, la línea de tendencia exponencial de la gráfica es acorde a su complejidad.

Requerimiento 4

Plantilla para el documentar y analizar cada uno de los requerimientos.

```python
def req_4(data_structs,sig_min,dis_max):
    """
    Función que soluciona el requerimiento 4
    """
    #sacar llaves en mayores a sig_min
    sig_map = data_structs["sig"]
    sig_max=om.maxKey(sig_map)
    llaves_sig=om.keys(sig_map,sig_min,sig_max)
    new_time_map = om.newMap(omaptype="RBT")
    total_eventos = 0

    #filtrar por dis_max
    for valor in lt.iterator(llaves_sig):
        pareja = om.get(sig_map,valor)
        sismos=me.getValue(pareja)
        for sismo in lt.iterator(sismos):
            dis = int(sismo["gap"])
            if dis_max>=dis:
                total_eventos+=1
                add_data_arbol_list(new_time_map,sismo,"time")

    #Sacar los 15 sismos mas recientes
    total_fechas=om.size(new_time_map)
    total=om.keySet(new_time_map)
    fechas=lt.subList(total,lt.size(total)-14,15)
    lista_final=lt.newList("ARRAY_LIST")

    #Hacer diccionarios para imprimir de sismos finales
    for fecha in lt.iterator(fechas):
        pareja = om.get(new_time_map,fecha)
        sismo_x_fecha=me.getValue(pareja)
        lista_pequeno=lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sismo_x_fecha):
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_pequeno,pequeno)
        details=creartabla(lista_pequeno,lista_pequeno["elements"][0])

        dict_grande = big_dict(fecha,"time",lt.size(sismo_x_fecha),details)
        lt.addLast(lista_final,dict_grande)

    return lista_final,total_eventos,total_fechas
```

Descripción

Para este requerimiento primero extraemos todas las llaves del mapa fechas, luego para cada fecha extraemos la lista de sismos y verificamos para cada uno que su significancia sea mayor a la mínima y su distancia azimutal menor a la máxima. Luego, para cada sismo que pase este filtro se agrega a una lista y después de crear la tabla con todos los sismos, se agrega a una lista final que va a tener todas las fechas.

Entrada	Entran por parámetro: • data_structs (Datos almacenados en los archivos ingresados) • Significancia mínima • Distancia azimutal máxima
Salidas	Retorna: • El total de eventos registrados que cumplan las condiciones • Los 15 eventos más recientes por fecha y que cumplan las condiciones.
Implementado (Sí/No)	Sí, implementado por María José Mantilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

s=tamaño del mapa sig

i= sismos por fecha

f>s

Pasos	**Complejidad**
Extracción de llaves mayores al sig mínimo	O(log(s))
Creación mapas y contadores	O(1)
Iteración sobre llaves del mapa sig	O(s)*O(i)=O(s*i*log(i))
Iteración sobre sismos de cada sig	O(i)
Filtrar si dis del sismo es menos que el dis_max	O(1)
Insertar sismo al mapa	O(log(i))
Extracción de llaves del nuevo mapa	O(s)
Sublista con 15 más recientes	O(15)???
Iteración sobre llaves extraídas	O(15)
Iteración sobre sismos de cada fecha	O(1)
Creación diccionarios	O(1)
TOTAL	***O(s*i*log(i))***

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	4533,2 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, get, getValue, iterator, sublist). • Un ciclo for para la iteración de los sismos. • Condicionales dentro del ciclo para filtrar aquellos datos que cumplen las condiciones especificadas.
Computador donde se ejecuta:	11th Gen Intel(R) Core i7 – 11800H @ 2.30 GHz 2.30GHz 16GB

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Significancia mínima = 300

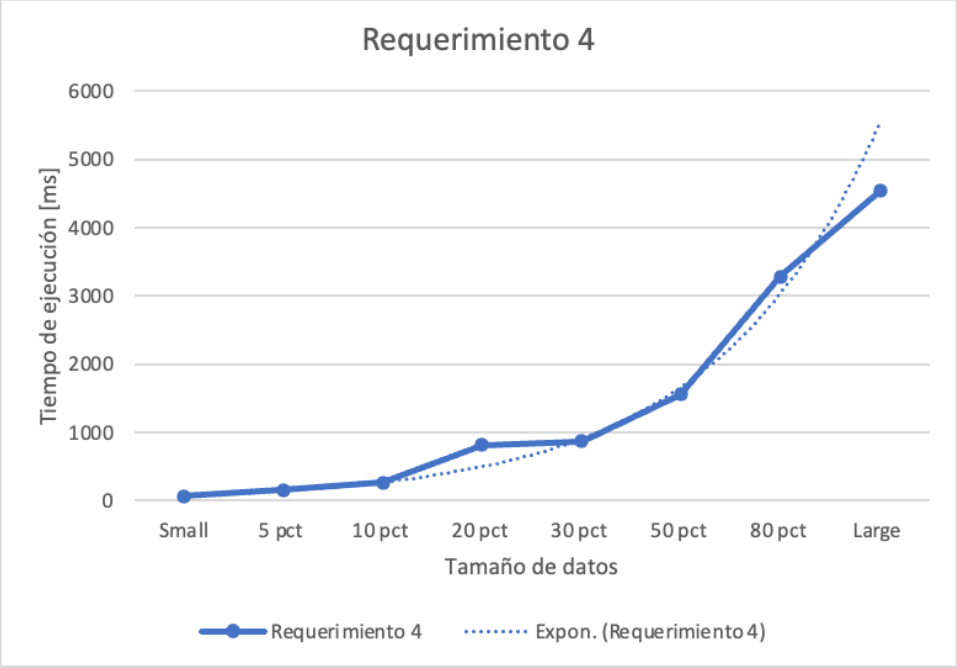
Distancia azimutal máxima= 45

Tiempo

Entrada	Tiempo de Ejecución Real [ms]
Small	62,77
5 pct	157,36
10 pct	261,47
20 pct	820,09
30 pct	869,08
50 pct	1557,49
80 pct	3284,12
Large	4533,2

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al observar los resultados obtenidos en las pruebas pudimos ver que el tiempo en el que corria el requerimiento no era tan eelevado esto se debe a que la complejidad temporal tampoco es tan elevada en el peor caso. Debido a que en el requerimiento se crean varios mapas con el fin de reducir esta complejidad, además como se realiza la sublista antes de hacer la otra iteración, estas van a tener complejidad constante. Por esta razón, lo que va a presentar mayor complejidad va a ser la primera iteración sobre las llaves del mapa filtrado de significancia y la iteración sobre los sismos en cada llave siendo esta de O(s*i).

Requerimiento 5

Plantilla para el documentar y analizar cada uno de los requerimientos.

```
def req_5(data_structs, depth_min, nst_min):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    mapa_estaciones=data_structs["nst"]
    est_max=om.maxKey(mapa_estaciones)

    llaves_estaciones=om.keys(mapa_estaciones,nst_min,est_max)

    total_eventos=0
    filtrado_eventos=om.newMap(omaptype="RBT")
    for estacion in lt.iterator(llaves_estaciones):
        sismos=me.getValue(om.get(mapa_estaciones,estacion))
        for sis in lt.iterator(sismos):
            prof=int(sis["depth"])
            if prof>=depth_min:
                total_eventos+=1
                add_data_arbol_list(filtrado_eventos,sis,"time")

    fechas_diferentes=om.size(filtrado_eventos)

    keys=om.keySet(filtrado_eventos)
    llaves_finales=lt.subList(keys,lt.size(keys)-19,20)
    lista_final=lt.newList("ARRAY_LIST")

    for ll in lt.iterator(llaves_finales):
        sismo_x_fecha=me.getValue(om.get(filtrado_eventos,ll))
        lista_final_sis=lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sismo_x_fecha):
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_final_sis,pequeno)
        details=creartabla(lista_final_sis,lista_final_sis["elements"][0])

        dict_grande = big_dict(ll,"time",lt.size(sismo_x_fecha),details)
        lt.addLast(lista_final,dict_grande)

    return fechas_diferentes,total_eventos,lista_final
```

Descripción

En este requerimiento para encontrar los 20 eventos más antiguos para una profundidad dada y con registros de un número mínimo de estaciones, se implementó un mapa en donde los datos están organizados por nst(número de estaciones). Luego se extraen todos los sismos que son mayores al nst minimo y sobre todos los sismos que cumplen esto, se comprueba que adicionalmente el sismo cumpla con la profundidad mínima y aquellos que si se agregan a un mapa final ordenado por fecha. Finalmente, se extraen los 20 sismos más recientes y se genera el diccionario a imprimir de cada uno. Por último, se retornan los contadores necesarios y la lista final.

Entrada	Entran por parámetro: • data_structs (Datos almacenados en los archivos ingresados) • Profundidad mínima • Cantidad mínima de estaciones
Salidas	Retorna: • El total de sismos registrados con la información disponible. • Los 20 eventos más recientes por fecha y que cumplan las condiciones.
Implementado (Sí/No)	Sí, implementado por Gabriela Zambrano

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

n=total de datos

s= sismos por nst

n>s

Pasos	**Complejidad**
Extracción de llaves mayores al nst mínimo	O(log(n))
Creación mapas y contadores	O(1)
Iteración sobre sismos	O(n)*O(s)=O(n*s*log(n))
Iteración sobre sismos de cada nst	O(s)
Comprobación de que cumple con la profundidad mínima	O(log(n))
Extracción de llaves de sismos que cumplen las condiciones	O(n)
Sublista con 20 más recientes	O(20)
Iteración sobre llaves	O(20)=O(1)
Iteración sobre sismos de cada llave	O(1-3)
Creación diccionarios	O(1)
TOTAL	***O(n*s*log(n))***

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	2904,1208 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, get, getValue, iterator, sublist). • Un ciclo for para la iteración de los sismos. • Condicionales dentro del ciclo para filtrar aquellos datos que cumplen las condiciones especificadas.
Computador donde se ejecuta:	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz (8 GB Windows 11 Home)

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Profundidad mínima = 13

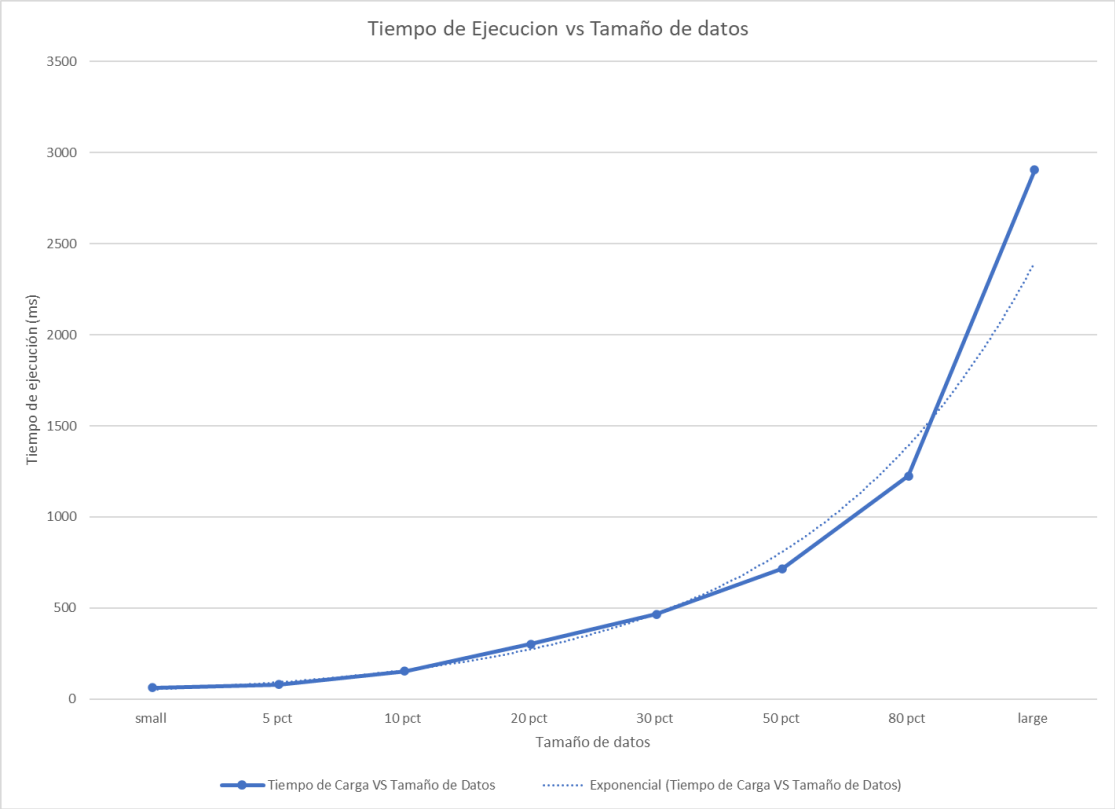
Nst mínimas = 50

Tiempo

Entrada	Tiempo de Ejecución Real [ms]
Small	62,6932
5 pct	79,8097
10 pct	153,6657
20 pct	301,9699
30 pct	466,5254
50 pct	716,1968
80 pct	1225,33913
Large	2904,1208

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al realizar las pruebas se observó que la línea de tendencia obtenida, al comparar el tiempo de ejecución con la cantidad de datos que se estaban analizando, fue exponencial. Esto no se encuentra completamente de acuerdo con la complejidad encontrada del algoritmo, debido a que esta fue de O(n*s*log(n)) la cual podría llegar a ser considerada mayor a una curva exponencial si el valor de s es un número muy cercano a n. Por lo tanto, los resultados y la complejidad no están de acuerdo entre sí.

Requerimiento 6

Plantilla para el documentar y analizar cada uno de los requerimientos.

```python
def req_6(data_structs,anio, latitud, longitud, radio, n):
    """
    Función que soluciona el requerimiento 6
    """


    mapa_anios=data_structs["anio"]
    an = om.get(mapa_anios,anio)
    if an is None:
        return(0,0,0,0,0,0)
    else:
        lista_sismos = me.getValue(an)


    total_eventos_en_area=0
    #Comprobar si estan en el radio necesario
    sis_en_espacio=om.newMap(omaptype="RBT")
    sis_x_fechas=om.newMap(omaptype="RBT")
    for sismo in lt.iterator(lista_sismos):
        lat1,lon1,lat2,lon2 = map(math.radians, [latitud, longitud, float(sismo["lat"]), float(sismo["long"])])
        dif_long=lon2-lon1
        dif_lat=lat2-lat1
        raiz=((math.sin(dif_lat/2))**2)+((math.cos(lat1))*(math.cos(lat2)*((math.sin(dif_long/2))**2)))
        ars=math.asin(math.sqrt(raiz))
        d=2*(ars)*6371
        if d <=radio:
            total_eventos_en_area+=1
            sismo["distance"]=d
            add_data_arbol_list(sis_en_espacio,sismo,"sig")
            add_data_arbol_list(sis_x_fechas,sismo,"time")


    #Encontrar el evento mas significativo
    sismo_m=om.maxKey(sis_en_espacio)
    s_m=me.getValue(om.get(sis_en_espacio,sismo_m))
    if lt.size(s_m)!=1:
        mas_dyfi=0
        for sismo in lt.iterator(s_m):
            dfyi=sismo["felt"]
            if dfyi!="UnKnown" and dfyi>mas_dyfi:
                mas_dyfi=dfyi
                sismo_mas=sismo
        if mas_dyfi ==0:
            sismo_mas=s_m["elements"][0]
    else:
        sismo_mas=s_m["elements"][0]

    codigo=sismo_mas["code"]
```

```
    #Armar diccionario evento mas significativo para imprimir
    evento_mas_s={"time":sismo_mas["time"],
                  "mag":sismo_mas["mag"],
                  "lat":sismo_mas["lat"],
                  "long":sismo_mas["long"],
                  "depth":sismo_mas["depth"],
                  "sig":sismo_mas["sig"],
                  "gap":sismo_mas["gap"],
                  "distance":sismo_mas["distance"],
                  "nst":sismo_mas["nst"],
                  "title":sismo_mas["title"],
                  "cdi":sismo_mas["cdi"],
                  "mmi":sismo_mas["mmi"],
                  "magType":sismo_mas["magType"],
                  "type":sismo_mas["type"],
                  "code":sismo_mas["code"]}
    evento_mas_sig=lt.newList("ARRAY_LIST")
    lt.addLast(evento_mas_sig,evento_mas_s)
    om.deleteMax(sis_en_espacio)


    #Encontrar distancias al mas significativo

    min_fecha=om.minKey(sis_x_fechas)
    llaves_sis_menores=om.keys(sis_x_fechas,min_fecha,sismo_mas["time"])
    lt.removeLast(llaves_sis_menores)
    if lt.size(llaves_sis_menores)>n:
        llaves_men=lt.subList(llaves_sis_menores,lt.size(llaves_sis_menores)-n,n)
    else:
        llaves_men=llaves_sis_menores
    pre_events=lt.size(llaves_men)


    max_fecha=om.maxKey(sis_x_fechas)
    llaves_sis_mayores=om.keys(sis_x_fechas,sismo_mas["time"],max_fecha)
    lt.removeFirst(llaves_sis_mayores)
    if lt.size(llaves_sis_mayores)>n:
        llaves_mayo=lt.subList(llaves_sis_mayores,1,n)
    else:
        llaves_mayo=llaves_sis_mayores
    pos_events=lt.size(llaves_mayo)

    fechas2= lt.newList("ARRAY_LIST")
    cant_eventos=0
    lista_final=lt.newList("ARRAY_LIST")
    for llave in lt.iterator(llaves_men):
        sis=me.getValue(om.get(sis_x_fechas,llave))
        lista_final_sis=lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sis):
            if lt.isPresent(fechas2,sismo["time"])==0:
                lt.addLast(fechas2,sismo["time"])
            cant_eventos+=1
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_final_sis,pequeno)
            details=creartabla(lista_final_sis,lista_final_sis["elements"][0])
```

```
            cant_eventos+=1
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_final_sis,pequeno)
            details=creartabla(lista_final_sis,lista_final_sis["elements"][0])
            fecha=sismo["time"]

        dict_grande = big_dict(fecha,"time",1,details)
        lt.addLast(lista_final,dict_grande)

    for llave in lt.iterator(llaves_mayo):
        sis=me.getValue(om.get(sis_x_fechas,llave))
        lista_final_sis=lt.newList("ARRAY_LIST")
        for sismo in lt.iterator(sis):
            if lt.isPresent(fechas2,sismo["time"])==0:
                lt.addLast(fechas2,sismo["time"])
            cant_eventos+=1
            pequeno =  min_dict(sismo,"mag")
            lt.addLast(lista_final_sis,pequeno)
            details=creartabla(lista_final_sis,lista_final_sis["elements"][0])
            fecha=sismo["time"]

        dict_grande = big_dict(fecha,"time",1,details)
        lt.addLast(lista_final,dict_grande)

    dif_fechas=lt.size(fechas2)

    return total_eventos_en_area,evento_mas_sig, pre_events,pos_events,lista_final,codigo,cant_eventos,dif_fechas
```

Descripción

En este requerimiento para encontrar el evento más significativo que se encuentra dentro de una región dada en un año especifico, se implementa un mapa por años. De este se extraen todos los sismos que ocurrieron en este año y luego implementando la fórmula de Haversine, con la latitud y longitud del sismo y los datos ingresados, se comprueba si el sismo ocurrió dentro del rango indicado. Luego de realizar esta filtración, se comprueba el sismo de mayor significancia y luego se encuentran los n sismos más cercanos por fecha antes del sismo más significativo y los n sismos más cercanos por fecha después del sismo. Finalmente, se generan los diccionarios de cada sismo a imprimir, se retornan los contadores necesarios y la lista final.

Entrada	Entran por parámetro: • data_structs (Datos almacenados en los archivos ingresados) • Año • Latitud del punto • Longitud del punto • Radio de la zona de los sismos • Numero de sismos antes y después por fecha del sismo más significativo
Salidas	Retorna: • El evento más significativo en el área y en el año indicado • El total de sismos registrados dentro del área con la información disponible. • Los N eventos más cercanos por fecha y que cumplan las condiciones, al sismo más significativo.
Implementado (Sí/No)	Sí, implementado por Gabriela Zambrano

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

n=datos totales

m=número de sismos que pide el usuario

N>=m

Pasos	**Complejidad**
Extraer valor del año esperado del mapa anios	O(2log(n))
Creación mapas y contadores	O(1)
Iteración sobre sismos	O(n)*O(log(n))=O(nlog(n))
Calculo de formula	O(1)
Comprobación de si está en el área esperada	O(1)
Añadir datos	O(log(n))
Iteración sobre los sismos que cumplen	O(n)

Condicionales para encontrar el sismo más significativo	O(1)
Creación del diccionario del sismo más significativo	O(1)
Creación listas y contadores finales	O(1)
Hacer sublista con fechas antes del sismo más significativo	O(m)
Hacer sublista con fechas despues del sismo más significativo	O(m)
Formar diccionario sismos antes	O(m*m)=O(m^2)
Iteración sobre sismos antes	O(m)
Iteración sobre sismos en cada fecha	O(1-3)
Comprobación de fecha, isPresent	O(m)
Formar diccionario sismos después	O(m*m)=O(m^2)
Iteración sobre sismos después	O(m)
Iteración sobre sismos en cada fecha	O(1-3)
Comprobación de fecha, isPresent	O(m)
TOTAL	***O(m^2)***

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos. Además, se asumió que n=m.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	271,8985 ms
Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, get, getValue, iterator, sublist). • Un ciclo for para la iteración de los sismos en el año ingresado. • Condicionales dentro del ciclo para filtrar aquellos datos que cumplen las condiciones especificadas.
Computador donde se ejecuta:	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz (8 GB Windows 11 Home)

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Año= 2018

N=5

Latitud= 4.674

Longitud= -74.068

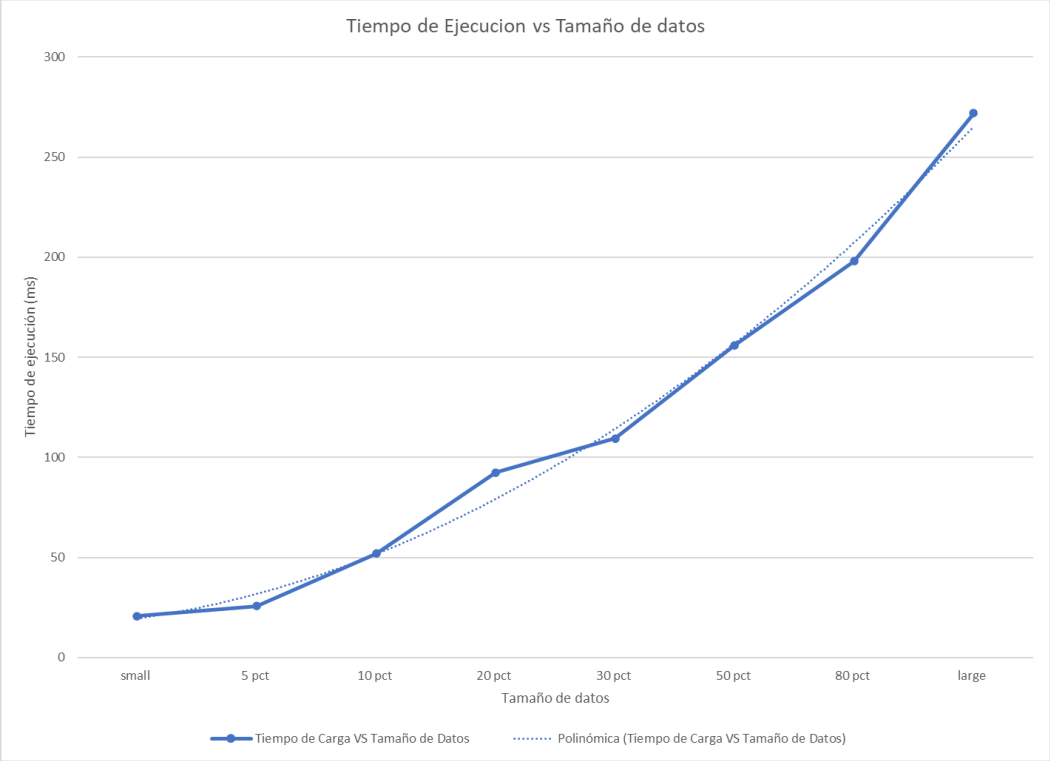
Radio=4500

Tiempo

Entrada	Tiempo de Ejecución Real [ms]
Small	20,6725
5 pct	25,8031
10 pct	51,9356
20 pct	92,4939
30 pct	109,4159
50 pct	156,0822
80 pct	198,0178
Large	271,8985

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al realizar las pruebas se observó que se obtuvo una línea de tendencia polinómica de grado 2, al relacionar el tiempo de ejecución con la cantidad de datos. Esto se podría ver relacionado con que el n escogido fue 5, siendo este mucho menor que la cantidad de datos presentados, por lo tanto, no se obtiene una curva exponencial. Sin embargo, si se obtiene una curva polinómica de grado 2 y esto si se ve relacionado con la complejidad encontrada que fue de O(m^2). Lo que se puede concluir de estos resultados, es que el valor de n que sea ingresado por el usuario afectara de gran forma el rendimiento y el tiempo que se puede demorar el algoritmo en dar un resultado.

Requerimiento 7

Plantilla para el documentar y analizar cada uno de los requerimientos.

```python
def req_7(data_structs,anio,title,propiedad,casillas):
    """
    Función que soluciona el requerimiento 7
    """
    anios = data_structs["anio"]
    pareja = om.get(anios,anio)
    fechas = me.getValue(pareja)
    total_sis_anio = lt.size(fechas)
    mapa_title = om.newMap(omaptype="RBT")
    mapa_propiedad = om.newMap(omaptype="RBT")
    lista_val_hist = lt.newList("ARRAY_LIST")
    for fecha in lt.iterator(fechas):
        if title in fecha["title"] and fecha[propiedad]!= "" and fecha[propiedad]!= 1:
            add_data_arbol_list(mapa_title,fecha,"time")
            add_data_arbol_list(mapa_propiedad,fecha,propiedad)
            lt.addLast(lista_val_hist,fecha[propiedad])
    total_sis_his = om.size(mapa_title)
    prop_min = om.minKey(mapa_propiedad)
    prop_max =om.maxKey(mapa_propiedad)
    sismos_anio = om.keySet(mapa_title)
    lista_final_fechas = lt.newList("ARRAY_LIST")
    for cada_llave in lt.iterator(sismos_anio):
        pareja_2 = om.get(mapa_title,cada_llave)
        fechas = me.getValue(pareja_2)
        for fecha in lt.iterator(fechas):
            dic_imp = {"time": fecha["time"],
                       "lat" : fecha["lat"],
                       "long": fecha["long"],
                       "title": fecha["title"],
                       "code" : fecha["code"],
                      propiedad : fecha[propiedad]}
            lt.addLast(lista_final_fechas, dic_imp)

    ancho_intervalo = round((prop_max-prop_min)/casillas,2)
    intervalos = lt.newList("ARRAY_LIST")
    lt.addLast(intervalos,prop_min)
```

```python
for i in range(1,casillas):
    primer_elemento = round(prop_min + i*ancho_intervalo,2)
    x = lt.isPresent(intervalos,primer_elemento)
    if x == 0:
        lt.addLast(intervalos,primer_elemento)
    siguiente = round(primer_elemento + ancho_intervalo,2)
    lt.addLast(intervalos,siguiente)
marcas = lt.newList("ARRAY_LIST")
nombres = lt.newList("ARRAY_LIST")
mapa_count= om.newMap(omaptype="RBT")
for i in range(1,lt.size(intervalos)):
    primero = lt.getElement(intervalos,i)
    segundo = lt.getElement(intervalos,i+1)
    A = round(primero+segundo,2)
    B = round(A/2,2)
    lt.addLast(marcas,B)
    cada_nombre = f"({primero} - {segundo}]"
    lt.addLast(nombres,cada_nombre)
    count=0
    for val in lt.iterator(lista_val_hist):
        if i == 1:
            if val >= primero and val <= segundo:
                count+=1
        else:
            if val > primero and val <= segundo:
                count+=1
    om.put(mapa_count,cada_nombre,count)
conteos = om.valueSet(mapa_count)
return total_sis_anio,total_sis_his,prop_min,prop_max,lista_final_fechas,lista_val_hist,intervalos,marcas,nombres,conteos
```

Descripción

Para resolver este requerimiento primero sacamos del árbol de años el valor de la llave que corresponde al año que entra como parámetro, luego iteramos sobre esta lista extraída. Luego, mediante un condicional filtramos los sismos que pertenecían al área ingresada por parámetro y descartamos lo sismos en los cuales la propiedad no tenía ningún valor. Con estos sismos creamos 3 estructuras un árbol cuyas llaves son las fechas filtradas, otro árbol cuyas llaves son los valores de la propiedad ingresada y una lista con todos los valores de dicha propiedad en cada sismo. Luego iteramos sobre las llaves del primer árbol para crear una lista de diccionario la cual se iba a imprimir y así tener ordenada de más antiguo a más reciente los sismos. Ahora bien, para construir el histograma calculamos el año de los intervalos con la diferencia entre el valor mínimo y máximo, que extrajimos del árbol, divido por la cantidad de casillas que entran por parámetro. Luego, creamos 3 listas con valores que le pasamos a la función hist() en matplotlib, la primera contenía los límites de cada barra es decir cada número que delimita los intervalos para personalizar las casillas, otra con el valor intermedio de los intervalos para poner la etiqueta de cada barra y, por último, la lista con las etiquetas que en este caso son los intervalos. Para finalizar, realizamos un mapa cuyas llaves eran los intervalos y los valores la cantidad de valores que había dentro de cada uno.

Entrada	Entran por parámetro: • data_structs (Datos almacenados en los archivos ingresados) • Año relevante • Título de la región asociada • Propiedad de conteo • El número de segmentos o casillas(bins) en los que se divide el histograma
Salidas	Retorna: • El número de eventos sísmicos del año • El número de eventos sísmicos utilizados en el histograma • Valor mínimo y máximo de la propiedad • Histograma • Lista de eventos que cumplen las condiciones
Implementado (Sí/No)	Sí, implementado por María José Mantilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

c = número de casillas (parámetro)

i = número de intervalos = c+1

f = sismos ocurridos en un año especifico

a = cantidad de fechas en las que ocurrió un sismo durante el año

s= sismos ocurridos en una fecha específica del año

f > a >s >i >c

Pasos	**Complejidad**
Extraer valor del año esperado del mapa anios	O(1)
Creación mapas y contadores	O(1)
Iteración sobre sismos del año	O(f) = O(f*log(f))
Filtrado por área y propiedad	O(1)
Agregar datos a un mapa y lista	O(3*log(f)) = O(log(f))
Sacar max y min /size	O(1)
Extraer llaves del mapa fecha nuevo	O(a)
Iterar sobre cada llave	O(a) = O(a*s)
Iterar sobre sismos de cada llave	O(s)
Crear diccionario y addLast	O(1)
Calcular ancho de intervalo	O(1)
Iterar sobre el número de casillas	O(c) = O(c*i)
Condicionales y addLast	O(3)
Comprobación de intervalo ispresent	O(i)
Iterar sobre el largo de la lista intervalos	O(i) = O(i*f)
GetElement para primer y segundo numero	O(2)
Cálculo de valor medio y etiqueta	O(2)
addLast	O(2)
Iteracion sobre lista con valores de propiedad	O(a)
Condicionales, contadores y put en el mapa	O(9)
TOTAL	***O(f*log(f))***

Nota: Al trabajar con mapas tomamos la complejidad de los peores casos. Además, se asumió que n=m.

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Tiempo de ejecución con todos los datos (large):	403,48 ms

Condición, herramientas y recursos utilizados:	• DISClib.ADT import list e import map (size, get, getValue, iterator, sublist). • Un ciclo for para la iteración de los sismos en el año ingresado. • Condicionales dentro del ciclo para filtrar aquellos datos que cumplen las condiciones especificadas.
Computador donde se ejecuta:	11th Gen Intel(R) Core i7 – 11800H @ 2.30 GHz 2.30GHz 16GB

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Valores de prueba:

Año= 2020

Área = Alaska

Propiedad = mag

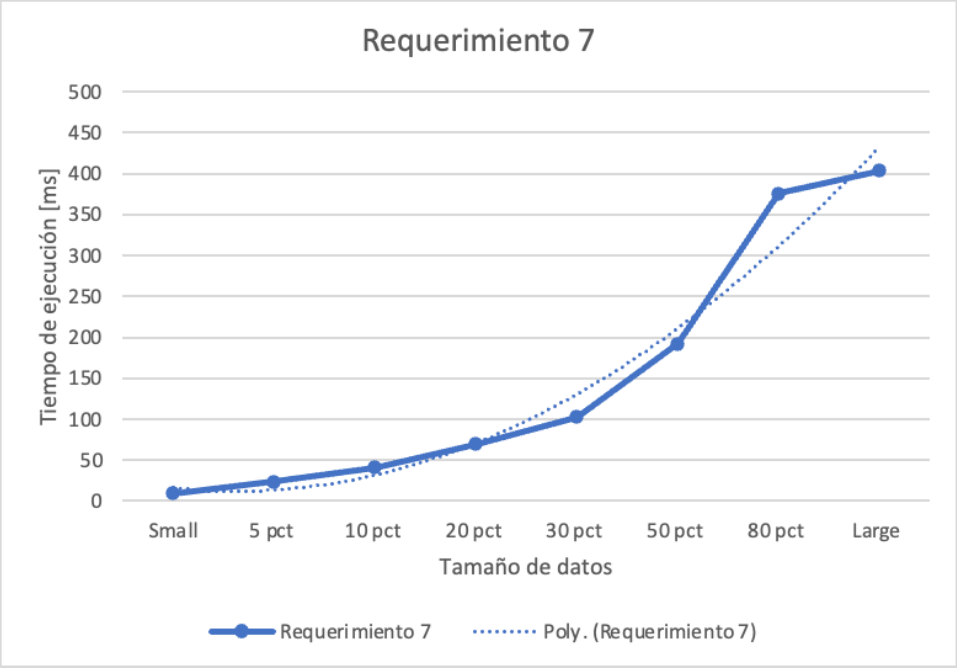
Numero de casilla = 10

Tiempo

Entrada	Tiempo de Ejecución Real [ms]
Small	9,54
5 pct	23,92
10 pct	40,83
20 pct	69,22
30 pct	102,3
50 pct	192,54
80 pct	375,45
Large	403,48

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al ver las pruebas realizadas y el análisis de complejidad pudimos ver que el requerimiento no tiene una complejidad temporal demasiado alta, lo que se puede ver evidenciado en los tiempos que comparados a otros requerimientos tiene mucho mejor rendimiento. En cuanto a la complejidad, aunque es este se llevan a cabo varias iteraciones, en cuyo caso 2 de ellas son una dentro de otra, debido a que se va disminuyendo el tamaño de la lista estas complejidades no son tan malas. En primer lugar, tomamos el peor caso donde todos los sismos están dentro de un año siendo este de largo f y luego la cantidad de fechas tiene que ser menor siendo este a y luego cada sismo dentro de cada fecha en el peor caso sería s. Por lo que al iterar sobre las llaves que representan cada fecha y luego los sismos en su interior es lo que ofrece mayor complejidad temporal siendo (a*s), sin embargo, como estamos realizando pruebas con muchos datos la complejidad más alta seria de f*log(f) ya que esta es mayor para datos mayores.