ANÁLISIS DEL RETO

Ximena Lopez, 202312848, ax.lopez@uniandes.edu.co

Juan David Torres, 202317608, jd.torresa1@uniandes.edu.co

Sofia Losada, 20221008, s.losadam@uniandes.edu.co

Requerimiento <<1>>

```
def req_1(control, fecha_inicio, fecha_final):
"""
Función que soluciona el requerimiento 1
"""
# TODO: Realizar el requerimiento 1
fecha_in = datetime.datetime.strptime(fecha_inicio, '%Y-%m-%dT%H:%M')
fecha_fin = datetime.datetime.strptime(fecha_final, '%Y-%m-%dT%H:%M')
lst_rango_fechas = om.keys(control["temblores"],fecha_in, fecha_fin)
total = 0
lista_final = lt.newList("SINGLE_LINKED")
for lst_fecha in lt.iterator(lst_rango_fechas):
respuesta = me.getValue(om.get(control["temblores"],lst_fecha))
tamanio = lt.size(respuesta["By_time"])
total += tamanio
dic = {"time" : lst_fecha, "mag": lst_fecha, "Adicional": lt.newList("ARRAY_LIST")}
info_adicional = dic["Adicional"]
orden = merg.sort(respuesta["By_time"], compare_results_list)
if tamanio < 6:
for ele in lt.iterator(orden):
d = nuevo(ele)
lt.addLast(info_adicional,d)
else:
```

```
for date in range(1,4):
info = lt.getElement(orden,date)
info = nuevo(info)
lt.addLast(info_adicional,info)
for date in range(0,3):
info = lt.getElement(orden,date,(tamanio-2+date))
info = lt.addLast(info_adicional,info)
lt.addLast(info_adicional,info)
lt.addFirst(lista_final,info_adicional)
return lista_final, total
```

Descripción

Se realizó un programa que utiliza un árbol cuyas llaves son la columna “time” (fechas) del archivo. De esta manera se filtran las fechas de acuerdo al rango dado por parámetro, se obtienen los datos respectivos a las fechas, se almacenan en una lista y se retornan.

Entrada	- Fecha inicial del intervalo - Fecha final del intervalo
Salidas	• El número total de eventos sísmicos ocurridos durante las fechas indicadas. • Todos los eventos ocurridos en el intervalo ordenados cronológicamente desde el más reciente al más antiguo. ○ La fecha y hora del evento (time). ○ La magnitud del evento (mag) ○ La Latitud donde ocurrió el evento (lat). ○ La longitud donde ocurrió el evento (long). ○ La profundidad del evento (depth). ○ La significancia del evento (sig). ○ La distancia azimutal del evento (gap). ○ El número de estaciones utilizadas para medir el evento (nst). ○ El título del evento sísmico (title). ○ La intensidad máxima del evento reportada por el sistema DYFI (cdi). ○ La Intensidad máxima instrumental estimada para el evento (mmi). ○ El algoritmo de cálculo de magnitud del evento (magType). ○ El tipo del evento sísmico (type). ○ El código del evento (code).
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Por medio del comando "*om.keys*" se obtienen las llaves del árbol que están dentro del rango de fechas.	O(n)
Paso 2 Mediante un *for* se itera sobre la lista que contiene las llaves que están dentro del rango, se obtienen los respectivos valores y se almacenan en un diccionario con la información adicional.	O(m)
Paso 3 Dentro del *for* se realiza un condicional que hace que se retornen tres resultados y la cantidad total son mas de 6	O(m)
TOTAL	***O(n*m)***

Pruebas Realizadas

Procesadores	1,8 GHz Intel Core i5 de dos núcleos
Memoria RAM	8 GB 1600 MHz DDR3
Sistema Operativo	MacOS Big Sur 11.7.10

Para esta prueba se ingresó de fecha inicial 1999-03-21T05:00 y como fecha final 2004-10-23T17:30.

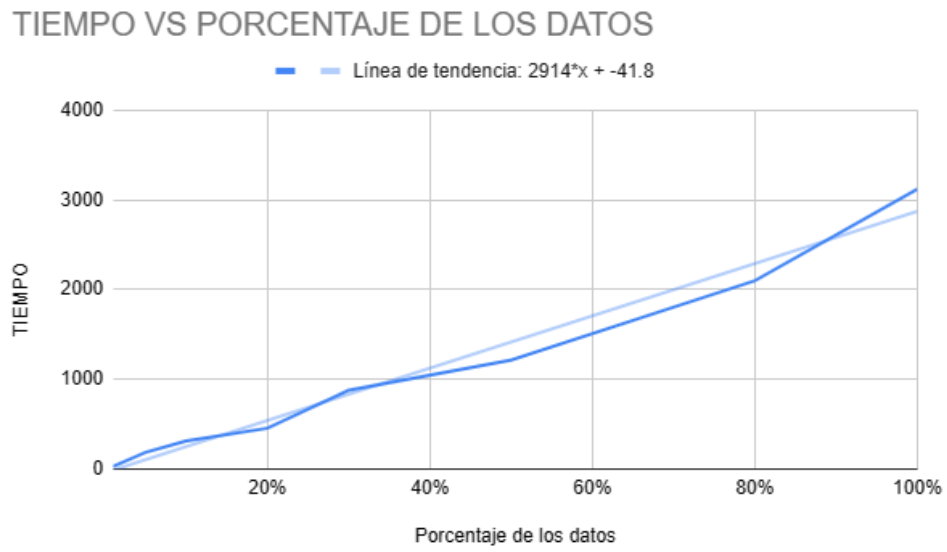
Tablas de datos

Las tablas con la recopilación de datos de las pruebas. Tiempo en ms

	PRUEBA1	PRUEBA2	PRUEBA3	PROMEDIO
1%	26.14499998	28.63800001	33.3915	29.3915
5%	287.4010001	130.7214	137.4106001	185.1776667
10%	282.7085999	216.4484999	443.7916999	314.3162666
20%	439.4751999	439.4634	481.6006	453.5130666
30%	1272.1358	670.3358001	691.2828999	877.9181666
50%	1209.2351	1203.6789	1232.2683	1215.060767
80%	2156.7897	2063.981299	2072.207	2097.659333
100%	2345.92859	2259.1966	4753.3077	3119.47763

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.

Se puede observar que la gráfica sigue un comportamiento relativamente lineal. Esto va en línea con el análisis de complejidad, donde se estimó como O(n*m), donde n es el número de fechas y m es un número de datos filtrado para una fecha específica. A medida que aumenta la cantidad de datos, aumenta el número de fechas, con lo que aumenta el tiempo de ejecución. Sin embargo, debido a que las fechas están reducidas hasta el minuto, es poco probable que haya más de un evento para una misma fecha. Esto quiere decir que el aporte de m a la complejidad es relativamente pequeño.

Requerimiento <<2>>

```
def req_2(analyzer,initialmag, finalmag ):

"""

Función que soluciona el requerimiento 2

"""

total=0

lista= lt.newList("SINGLE_LINKED")

lst = om.keys(analyzer['temblores_mag'], initialmag, finalmag)

for lstmag in lt.iterator(lst):

result= me.getValue(om.get(analyzer['temblores_mag'],lstmag))
```

```
tamano=lt.size(result["By_mag"])
total+=tamano
diccionario={"mag":lstmag,"Events":tamano,"Details":lt.newList("ARRAY_LIST")}
detalles=diccionario["Details"]
ordenada= merg.sort(result["By_mag"],compare_results_list)
if tamano<6:
for cada in lt.iterator(ordenada):
dato=nuevo(cada)
lt.addLast(detalles,dato)
else:
for b in range(1,4):
dato = lt.getElement(ordenada, b)
dato=nuevo(dato)
lt.addLast(detalles,dato)
for b in range (0,3):
dato = lt.getElement(ordenada, (tamano-2+b))
dato=nuevo(dato)
lt.addLast(detalles,dato)
lt.addFirst(lista,diccionario)
return lista,total
```

Descripción

Se implementa un algoritmo que utiliza el árbol temblores_mag cuyas llaves son las magnitudes *"mag"* del archivo. Se filtran las maginitudes del arbol dado los parámetros de entrada y se retorna una lista con la información de la actividad sismica correspondiente al rango de las magnitudes.

Entrada	- El límite inferior de la magnitud - El límite superior de la magnitud
Salidas	• Todos los eventos ocurridos en el intervalo ordenados por su magnitud desde más fuerte al más débil (mag). ○ La fecha y hora del evento (time). ○ La magnitud del evento (mag) ○ La Latitud donde ocurrió el evento (lat).

	○ La longitud donde ocurrió el evento (long). ○ La profundidad del evento (depth). ○ La significancia del evento (sig). ○ La distancia azimutal del evento (gap). ○ El número de estaciones utilizadas para medir el evento (nst). ○ El título del evento sísmico (title). ○ La intensidad máxima del evento reportada por el sistema DYFI (cdi). ○ La Intensidad máxima instrumental estimada para el evento (mmi). ○ El algoritmo de cálculo de magnitud del evento (magType). ○ El tipo del evento sísmico (type). ○ El código del evento (code).
Implementado (Sí/No)	Si, implementado por Ximena Lopez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Se obtienen las llaves del árbol “temblores_mag” que están dentro de un rango dado.	O(n)
Paso 2 Mediante un *for* se itera sobre la lista que contiene las llaves que están dentro del rango, se obtienen los respectivos valores y se almacenan en un diccionario con la información adicional.	O(m)
Paso 3 Se realiza un merge para ordenar la lista de resultados	O(nlogn)
Paso 3 se realiza un condicional que hace que se retornen tres resultados y la cantidad total interna sean 6. Lo anterior haciendo uso de un For anidado	O(m)
TOTAL	***O(n*m*m*nlogn)***

Pruebas Realizadas

Para estas pruebas se tomó la magnitud mínima de 3.5 y máxima de 6.5

Tablas de datos

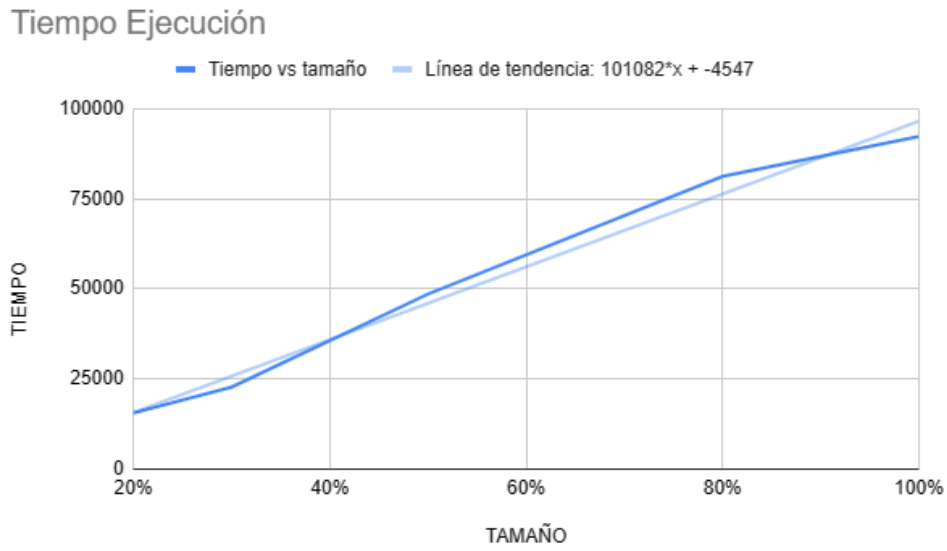
Las tablas con la recopilación de datos de las pruebas.

	PRUEBA1	PRUEBA2	PRUEBA3	PROMEDIO
1%	366	369.1353002	664.7556001	466.5187001

5%	4145.9969	2711.3613	2410.7837	3089.380633
10%	10634.4125	6186.1138	5444.253	7421.5931
20%	21536.6303	12630.5957	12446.6459	15537.9573
30%	31195.714	18607.8497	18396.1408	22733.23483
50%	68033.7739	38493.5481	39036.8107	48521.37757
80%	113306.5605	65373.0546	64856.032	81178.54903
100%	127480.8736	74205.748	75280.44289	92322.35483

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Como se ve dentro de la gráfica, el comportamiento de la función es aproximadamente lineal y se acerca a la línea de tendencia. Si se interpreta n como las posibles magnitudes, y m los datos para una magnitud dada, es posible afirmar que el crecimiento de complejidad se debe principalmente a m. En un rango de magnitudes, las posibles magnitudes que se pueden registrar son realmente una cantidad pequeña, mientras que, por el otro lado, es posible que existan varios eventos con una misma magnitud. Esto implica que en medida que crece el volumen de datos, llega un punto en el que “no existirán nuevas magnitudes”, por lo que realmente tan solo m estaría creciendo. Esto da un comportamiento relativamente lineal.

Requerimiento <<3>>

```
def req_3(data_structs,magnitud,profundidad):

"""
```

```
Función que soluciona el requerimiento 3
"""
#cambie a float la profundidad carga de datos
mapa = data_structs["temblores_mag"]
lista_final= lt.newList("ARRAY_LIST")
diccionario=om.newMap("RBT")
maxima_mg = om.maxKey(mapa)
llaves = om.values(mapa,magnitud,maxima_mg)
for cada in lt.iterator(llaves):
mapita=cada["By_depth"]
minimo_prof= om.minKey(mapita)
resultado= om.values(mapita,minimo_prof,profundidad)
for ca in lt.iterator(resultado):
for temblor in lt.iterator(ca):
lt.addLast(lista_final,temblor)
fecha = datetime.datetime.strptime(temblor["time"], "%Y-%m-%dT%H:%M:%S.%fZ")
dates = fecha.strftime('%Y-%m-%dT%H:%M')
entry= om.get(diccionario,dates)
if entry:
valor= me.getValue(entry)
else:
valor={"time":dates,"events":0,"details":lt.newList("ARRAY_LIST")}
om.put(diccionario,dates,valor)
lista= valor["details"]
h= nuevo(temblor)
lt.addLast(lista,h)
valor["events"]+=1
pri= primeros(diccionario)
```

```
return pri, lt.size(lista_final)
```

Descripción

Se desea consultar los 10 eventos sísmicos más recientes que superen una magnitud mínima y no superen una profundidad máxima indicada. Para esto, se utiliza el mapa "temblores_mag", se crea un mapa nuevo, de tipo RBT llamado diccionario. Se recorren las llaves e temblores_mag dentro de un rango. Después se filtran por fecha y se añaden junto con su respetiva información a una lista que se ordena y se retorna.

Entrada	- La magnitud mínima del evento (mag). - La profundidad máxima del evento (depth).
Salidas	• Los diez (10) eventos cronológicamente más recientes que cumplan con las condiciones de profundidad y magnitud indicados. o La fecha y hora del evento (time). o La magnitud del evento (mag) o La Latitud donde ocurrió el evento (lat). o La longitud donde ocurrió el evento (long). o La profundidad del evento (depth). o La significancia del evento (sig). o La distancia azimutal del evento (gap). o El número de estaciones utilizadas para medir el evento (nst). o El título del evento sísmico (title). o La intensidad máxima del evento reportada por el sistema DYFI (cdi). o La Intensidad máxima instrumental estimada para el evento (mmi). o El algoritmo de cálculo de magnitud del evento (magType). o El tipo del evento sísmico (type). o El código del evento (code).
Implementado (Sí/No)	Si, implementado por Ximena Lopez Cruz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Se asiga a la variable mapa data_strucst["temblores_mag"] y con om.values se da la lista de los valores del mapa.	O(n)
Paso 2	O(M)

Mediante un for se itera sobre la lista de los valores obtenidos para conseguir los valores del mapa “By_depth”	
Paso 3 Con un for anidado se itera sobre los valores del mapa “By_depth”	O(m)
Paso 4 Se itera sobre las listas de cada resultado utilizando un for anidado para acceder a cada dato de los sismos.	O(m)
Paso 5 A los valores filtrados se obtiene su respectiva fecha y se obtiene en diccionario los valores asociados a esta.	O(1)
Paso 6 Se llama a la función primeros, para conseguir los 10 primeros datos requeridos. En ella hay un for por 10 llaves.	O(10)
TOTAL	***O(n*M*m*m) Donde cada m es diferente porque ya tiene un prefiltrado***

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).

Para las pruebas se ingresó una magnitud mínima de 4.7 y una profundidad máxima de 10.0.

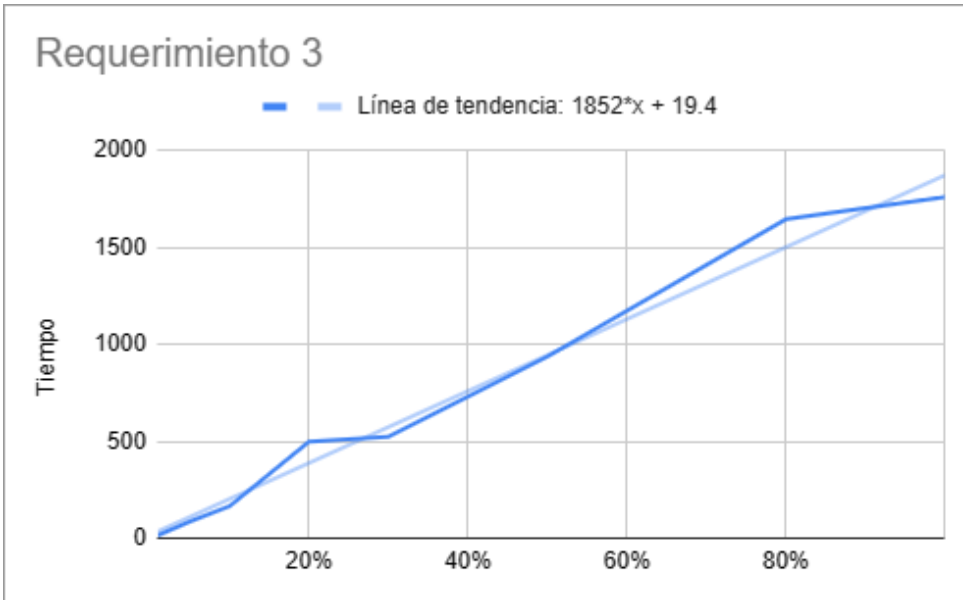
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

1%	18.01040006	18.1825	15.64539993	17.27943333
5%	82.95360005	82.5309	99.04269993	88.17573333
10%	154.5928999	189.2153001	156.7412001	166.8498
20%	394.7268	321.6056999	780.2008998	498.8444666
30%	496.0188	531.2594999	546.8390001	524.7057667
50%	910.2800001	939.4844999	962.7169	937.4938
80%	1897.7252	1496.629999	1538.626999	1644.327399
100%	1762.5114	1750.6166	1762.6467	1758.591567

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.

Los resultados de complejidad teóricos dan como resultado una complejidad exponencial, a pesar de que los recorridos completos que va haciendo están previamente pre_filtrados. Sin embargo, la complejidad temporal registrada con los tiempos de ejecución obtenidos da como resultado una gráfica de complejidad lineal, es decir que a medida que la cantidad de datos va aumentando, la complejidad lo hace con una proporción similar.

Requerimiento <<4>>

Descripción

Entrada	Data structures, minimum significiance, maximum gap
Salidas	Array list, size, # of dates.
Implementado (Sí/No)	Si – Juan David Torres Albarracín.

```
def req_4(data_structs, min_sig, max_gap):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    sig_om = data_structs['temblores_sig']
    gap_om = data_structs['temblores_gap']
```

```
results = lt.newList("ARRAY_LIST", cmpfunction=cmp_quakes)


max_sig = om.maxKey(sig_om)
sig_keys = om.keys(sig_om, min_sig, max_sig)
min_gap = om.minKey(gap_om)
gap_keys = om.keys(gap_om, min_gap, max_gap)


entries_map = mp.newMap(lt.size(sig_keys)*2.1,
                        maptype="PROBING",
                        loadfactor=0.5,
                        cmpfunction=compare_elements)


dates_map = mp.newMap(lt.size(sig_keys),
                        maptype="PROBING",
                        loadfactor=0.5,
                        cmpfunction=compare_elements)
for key in lt.iterator(sig_keys):
    quakes_list = me.getValue(om.get(sig_om,key))
    for quake in lt.iterator(quakes_list):
        mp.put(entries_map, quake['code'], quake)

for key in lt.iterator(gap_keys):
    quakes_list = me.getValue(om.get(gap_om, key))
    for quake in lt.iterator(quakes_list):
        if mp.contains(entries_map, quake['code']):
```

```
            lt.addLast(results, quake)
            if not mp.contains(dates_map, quake['time']):
                mp.put(dates_map, quake['time'],1)


    merg.sort(results, req4_sort_criteria)
    size = lt.size(results)
    dates = mp.size(dates_map)


    if size>15:
        return_list = lt.subList(results,1,15)

    else:
        return_list = results

    return return_list, size, dates
```

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar significancia máxima en el árbol de significancias.	O(lgN) (Caso promedio árbol rojo-negro)
Crear lista con las llaves entre el rango de la significancia mínima como parámetro y la llave superior.	O(N)
Buscar gap mínimo en el árbol de gaps.	O(lgN) (Caso promedio en árbol rojo negro)
Crear lista con las llaves entre el gap mínimo y el máximo pasado como parámetro.	O(N)
Iterar a través de las llaves de significancia.	O(n)
Obtener un elemento dado su llave.	O(lgN) (Caso promedio en árbol rojo negro)
Iterar a través de los terremotos dada una significancia. (anidado)	O(m), donde m es un grupo reducido de datos previamente filtrado por una significancia dada.

Insertar un elemento a un mapa.	O(3) (Caso promedio en un mapa con lineal probing)
Iterar a través de una lista con llaves.	O(n)
Obtener un valor dada una llave.	O(lgN) (Caso promedio en árbol rojo negro)
Iterar a través de unos datos dado un gap.	O(m), donde m es un grupo reducido de dados previamente filtrado por un gap dado.
Mirar si un mapa contiene una llave dada.	O(1)
Añadir un elemento al final de una lista (ARRAY_LIST).	O(1)
Merge sort.	O(nlogn)
Mirar el tamaño de un array_list y un mapa.	O(1)
TOTAL	***O(n*m) ó O(nlogn), dependerá del rango de llaves.***

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).

Para las pruebas realizadas se ingresó una significancia mínima de 300 y una distancia azimutal máxima de 45.0.

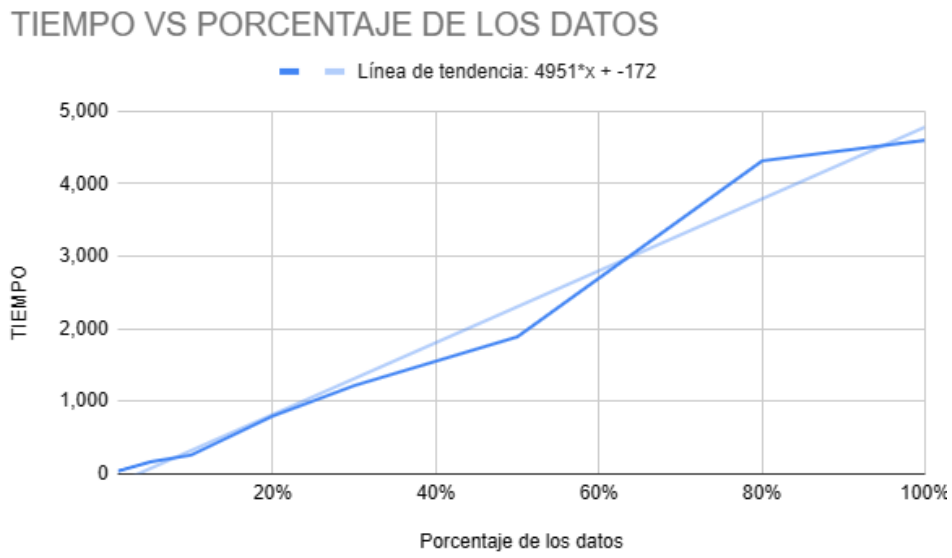
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

	PRUEBA1	PRUEBA2	PRUEBA3	PROMEDIO
1%	43	32.44309998	34	37
5%	183.4503	158.1914999	157	166.1642667
10%	413.6458001	331.2818	33.50830007	259.4786334
20%	822.2697001	794.5515	777.6476998	798.1562999
30%	1215.1573	1191.1156	1246.561	1217.6113
50%	2,010	1798.1626	1858.2843	1,889
80%	3920.4016	5549.4082	3475.5048	4315.104867
100%	5,961	3892.6404	3944.77979	4,600

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

De acuerdo a los resultados obtenidos, la función de la gráfica sigue una trayectoria parecida a una función linearítmica. Esto obedece al análisis hecho, donde dentro de un ciclo que itera a través de determinadas llaves se hace búsqueda de un elemento dentro de un árbol rojo-negro, para una complejidad estimada de O(nlogn). Es posible explicar la disminución de la complejidad en la parte final de la gráfica debido a la naturaleza de los árboles, donde a mayor altura, los nuevos niveles podrán almacenar más datos, lo que implica un crecimiento logarítmico en árboles balanceados. A mayor número de datos, los mayores niveles podrán soportar una mayor cantidad de datos sin comprometer el rendimiento.

Requerimiento <<5>>

```
def req_5(control, depth_min, min_estaciones_mon ):

"""

Función que soluciona el requerimiento 5

"""

# TODO: Realizar el requerimiento 5

key_max = om.maxKey(control["temblores_depth"])

lst_rango_depth = om.keys(control['temblores_depth'], depth_min, key_max)

total = 0

lst_final = lt.newList("SINGLE_LINKED")

for lst_depth in lt.iterator(lst_rango_depth):
```

```
valores_rango_depth = me.getValue(om.get(control['temblores_depth'],lst_depth))
for cada in lt.iterator(valores_rango_depth['By_depth_lst']):
estaciones_mon = cada['nst']
if estaciones_mon == "":
estaciones_mon = 0
else:
estaciones_mon = float(cada['nst'])
if estaciones_mon >= min_estaciones_mon:
total += 1
lt.addLast(lst_final,cada)
merg.sort(lst_final, compare_results_list)
lista_final_1 =lt.newList("ARRAY_LIST")
if lt.size(lst_final) <= 20:
top_20 = lst_final
else:
top_20 = lt.subList(lst_final,1,20)
if lt.size(top_20) < 6:
for ele in lt.iterator(top_20):
d = nuevo(ele)
lt.addLast(lista_final_1,d)
else:
for dato in range(1,4):
info = lt.getElement(top_20,dato)
lt.addLast(lista_final_1,info)
for i in range(lt.size(top_20) - 2, lt.size(top_20) + 1):
info = lt.getElement(top_20,i)
lt.addLast(lista_final_1,info)
return lista_final_1, total
```

Descripción

Se realizó un algoritmo que utiliza el mapa *'temblores_depth'* cuyas llaves son la columna depth del csv. De esta manera, se obtienen los elementos del árbol cuyas profundidades están en el rango dado. De esta cantidad de datos ya filtrada se obtienen los datos que cumplen con ser mayor al valor mínimo de *nst* y los valores se añaden a una lista, la cual se organiza por fechas y después se obtienen las primeras 20 posiciones, para retornarlas respondiendo al top 20 del requerimiento.

Entrada	- La profundidad mínima del evento - El número mínimo de estaciones que detectan el evento
Salidas	• Los veinte (20) eventos cronológicamente más recientes que cumplan con las condiciones de profundidad y número de estaciones especificados. o La fecha y hora del evento (time). o La magnitud del evento (mag) o La Latitud donde ocurrió el evento (lat). o La longitud donde ocurrió el evento (long). o La profundidad del evento (depth). o La significancia del evento (sig). o La distancia azimutal del evento (gap). o El número de estaciones utilizadas para medir el evento (nst). o El título del evento sísmico (title). o La intensidad máxima del evento reportada por el sistema DYFI (cdi). o La Intensidad máxima instrumental estimada para el evento (mmi). o El algoritmo de cálculo de magnitud del evento (magType). o El tipo del evento sísmico (type). o El código del evento (code).
Implementado (Sí/No)	Si - Sofia

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Obtener la llave mayor del árbol 'temblores_depth' por medio del comando "om.maxKey"	O(lgN)
Paso 2 Por medio del comando "om.keys" se obtiene una lista "lst_rango_depth" de tamaño (t) de las llaves del árbol que pertenecen a un rango de valores dado.	O(N)
Paso 3	O(n)

Se itera sobre esta lista “lst_rango_depth” y se obtienen los valores respectivos de cada llave y se almacena en el diccionario “valores_rango_depth”.	
Paso 4 En un *for* anidado se recorre el diccionario “valores_rango_depth” en la llave “By_depth_lst” Se accede a los valores en “nst”.	O(1)
Paso 5 Se recorren los valores de “nst” para filtrar los valores que son mayores al parámetro dado y así añadir los respectivos valores y su información adicional a la lista “lst_final”.	O(m)
Paso 6 Se ordena la lista por fecha, utilizando un merge.	O(nlogn)
Paso 7 Si la cantidad de elementos de lst_final es menor o igual a 20 se renombra la lista a “top_20” si no se crea una sublista de lst_final, con las primeras 20 posiciones y se asigna a “top_20”	O(n)
Paso 8 Si el tamaño de la lista top_20 es mayor a 6 se obtienen los tres primeros y los tres últimos y se retornan.	O(n)
TOTAL	***O(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).

Para las pruebas realizadas se usó una profundidad mínima de 23.0 km y un mínimo número de estaciones de 38.

Tablas de datos

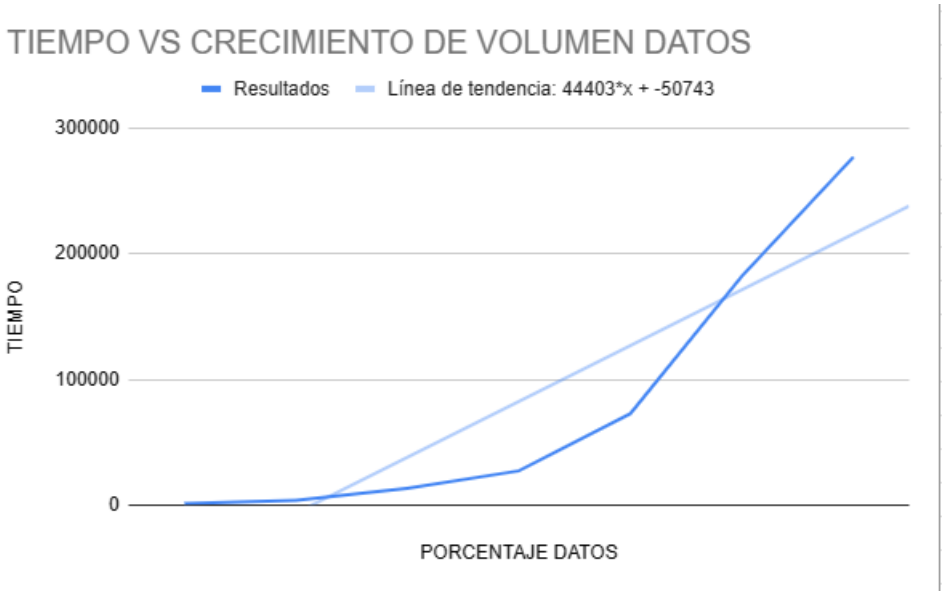
Las tablas con la recopilación de datos de las pruebas.

	PRUEBA1	PRUEBA2	PRUEBA3	PROMEDIO
1%	124.1166	161.2117	121.1701001	135.4994667
5%	1216.1588	1169.9849	1131.8675	1172.6704
10%	3630.249	3526.464	4101.3195	3752.6775
20%	13241.8615	13271.9366	13612.997	13375.59837
30%	29916.6423	26284.8216	25385.2989	27195.5876
50%	72,560	74037.8293	71899.5418	72832.3201

80%	195534.896	185417.7764	164863.4027	181938.6917
100%	297051.7821	256921.6	276986.6911	276986.6911

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Teniendo en cuenta las pruebas realizadas y el análisis de complejidad. La gráfica tiene un comportamiento variable. Pues el algoritmo procesa los datos y ejecuta el programa de manera correcta, pero a medida que aumenta la cantidad de datos, el tiempo de ejecución también aumenta. Después de analizar el programa detenidamente, esto puede deberse a utilizar Single Linked list en lugar de Array list, pues el programa dentro de una iteración añade al final los datos que va filtrando. Otra posible causa, fue haber usado un algoritmo de ordenamiento, en lugar de un árbol rbt cuyos datos se añaden en orden. Por otro lado, en el volumen de datos medio la gráfica se comporta por debajo de la lineal, acercándose a nlogn.

Requerimiento <<6>>

```
def req_6(data_structs,lat, long, radius, n_events, f_year):
    """
    Función que soluciona el requerimiento 6
    """
```

```
# TODO: Realizar el requerimiento 6
all_q = data_structs['quakes_req6']
area_q = lt.newList("ARRAY_LIST")

most_sig= 0
sig_event = None
post_events = 0
pre_events = 0

delta_ti = mp.newMap(lt.size(area_q)*2.1,
maptype="PROBING",
loadfactor=0.5,
cmpfunction=compare_elements)

return_list = lt.newList("ARRAY_LIST")

for quake in lt.iterator(all_q):
if quake['time'].year == f_year:
dist = harvesine_formula(lat,long,quake)
if dist<=radius:
if not quake['sig']:
quake['sig']=0
lt.addLast(area_q, quake)
if float(quake['sig'])>float(most_sig):
most_sig = quake['sig']
sig_event = quake
sig_code = sig_event['code']
```

```
for quake in lt.iterator(area_q):
diff_t = (sig_event['time']-quake['time']).total_seconds()
mp.put(delta_ti, diff_t,quake)
times_list = lt.newList('ARRAY_LIST')
for t in lt.iterator(mp.keySet(delta_ti)):
lt.addLast(times_list,t)
merg.sort(times_list, req6_sort_criteria)


dates_map = mp.newMap(n_events*4.1,
maptype="PROBING",
loadfactor=0.5,
cmpfunction=compare_elements)


for key in lt.iterator(times_list):
event = me.getValue(mp.get(delta_ti, key))
if key<0 and pre_events<n_events:
lt.addLast(return_list, event)
pre_events+=1
if not mp.contains(dates_map, event['time']):
mp.put(dates_map, event['time'],1)
elif key>0 and post_events<n_events:
lt.addLast(return_list, event)
post_events+=1
if not mp.contains(dates_map, event['time']):
mp.put(dates_map, event['time'],1)
lt.addLast(return_list, sig_event)
if not mp.contains(dates_map, sig_event['time']):
```

```
mp.put(dates_map, sig_event['time'],1)

merg.sort(return_list, req6_sort_criteria2)

total_events = lt.size(return_list)

total_dates = mp.size(dates_map)

radius_events = lt.size(area_q)


return return_list, post_events, pre_events, total_events, total_dates, sig_code, sig_event, radius_events
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos, latitud, longitud, radio, número de eventos, año.
Salidas	Eventos, evento con mayor significancia, número de eventos previos, número de eventos posteriores, número total de fechas, número total de eventos.
Implementado (Sí/No)	Implementado por Juan David Torres Albarracín.

Función auxiliar:

Fórmula del semiverseno para calcular la distancia entre dos puntos en el globo.

```
def harvesine_formula(lat1, long1, data):
    #Defines the distance between a given point and a seismic event as defined by
the Harvesine formula.

    # Radius of the Earth in kilometers
    R = 6371.0

    # Convert latitude and longitude from degrees to radians
    lat1 = math.radians(lat1)
    long1 = math.radians(long1)
    lat2 = math.radians(float(data['lat']))
    long2 = math.radians(float(data['long']))

    # Haversine formula
    dlat = lat2 - lat1
    dlong = long2 - long1

    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlong
/ 2)**2
```

```
    c = 2 * math.asin(math.sqrt(a))

    # Calculate the distance
    distance = R * c

    return distance
```

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Iterar a través de los terremotos.	O(n)
Mirar si el elemento corresponde al año deseado.	O(1)
Calcular la distancia del evento con respecto al punto dado por parámetro.	O(1) (todos los pasos en la función auxiliar son cálculos aritméticos)
Mirar si la distancia es menor al radio dado.	O(1)
Añadir el terremoto al final de un array_list.	O(1)
Verificar si el evento en cuestión supera los datos temporales del terremoto con mayor significancia.	O(1)
Iterar a través de la lista con los elementos ocurridos dentro del radio.	O(1)
Calcular la diferencia de tiempo entre un evento dado y el evento de mayor significancia.	O(1)
Añadir un elemento a un mapa.	O(3) (tiempo promedio para un mapa con lineal probing)
Merge sort con los elementos dentro del radio, prioriza los elementos más cercanos en tiempo al evento con mayor significancia.	O(nlogn)
Iterar a través de los elementos en la lista con los deltas de tiempo.	O(n)
Obtener un elemento del mapa con un delta de tiempo dado como llave.	O (1)
Verificar si el elemento es posterior o anterior al evento de mayor significancia, y que no se hayan añadido más elementos de los ingresados por parámetro.	O(1)
Añadir el elemento al final de un array_list.	O(1)
Mirar si un mapa contiene la fecha del elemento	O(1)
Añadir la fecha del evento a un mapa.	O(1)
Añadir el evento más significativo de último a la lista de retorno (array_list).	O(1)
Hacer merge sort de los elementos seleccionados, priorizando los eventos más recientes.	O(nlogn)

Sacar el tamaño de la lista con el número total de eventos, el mapa con las fechas totales, y los eventos ocurridos dentro del radio.	O(1)
TOTAL	***O(nlogn).***

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).

Para las pruebas realizadas se usó el año 2022, radio de 3000 km, 4.674 de latitud, -74.068 de longitud, 5 eventos.

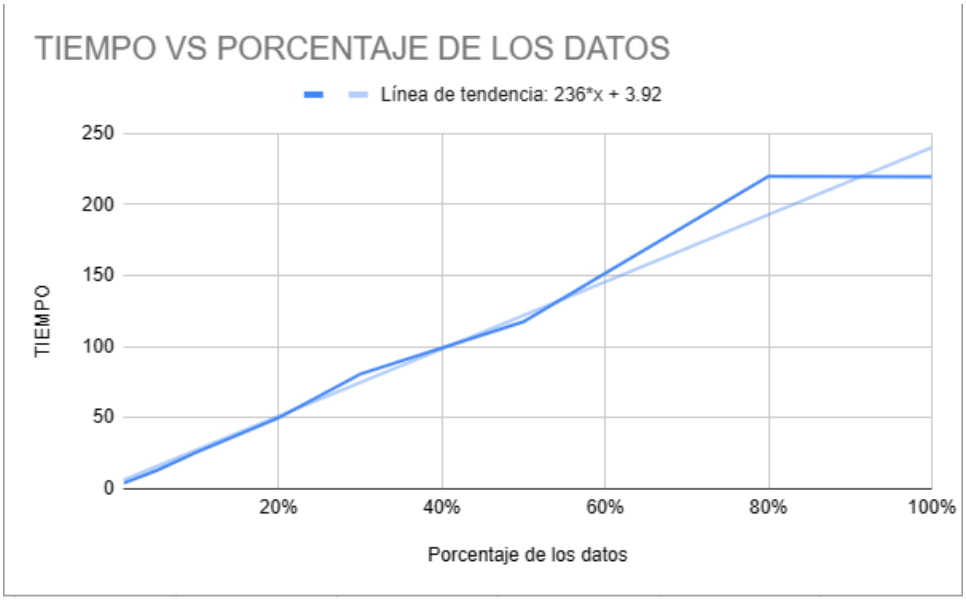
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

	PRUEBA1	PRUEBA2	PRUEBA3	PROMEDIO
1%	3.919199944	4.721500039	3.353700042	3.998133341
5%	13.69599998	12.57830012	12.26950002	12.84793337
10%	26.96269989	25.38199985	25	25.94256655
20%	54.24660003	45.82309997	50	49.99693334
30%	78.70169997	75.39789987	88.0561	80.71856661
50%	123.4617	107.4313	121.7957	117.5629
80%	214.2287	220.1423	225	219.8706
100%	223.9018	213.290199	221.2631	219.485033

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

La gráfica sigue un comportamiento lineal, dejando de crecer en el intervalo entre el 80% y el 100% de los datos. Si bien la complejidad se estimó como nlogn, es posible afirmar que el merge sort no influye realmente en el crecimiento de la complejidad si se siguen usando los mismos parámetros de entrada. Esto se debe a que, al aumentar el volumen de datos, el merge sort se hace sobre un número constante máximo de datos ingresado como parámetro (en este caso 5). Esto implicaría que el crecimiento realmente se debe al recorrido lineal O(n) que se hace sobre todos los datos cargados. Esto explica el crecimiento lineal del tiempo de ejecución, más no linearítmico. La parte linearítmica, realmente, alcanza un valor máximo, mientras que el aumento se da por el recorrido a todos los datos. Esto explica el crecimiento lineal.

Requerimiento <<7>>

```
def req_7(data_structs,año, titulo, condicion, bins):

"""

Función que soluciona el requerimiento 7

"""

#crear nuevo mapa

#funcion de agregar

#añadir data
```

#llamar la funcion en carga de datos

```
mapa= data_structs["By_year"]
result= me.getValue(om.get(mapa,año))
mapa_de_lacondicon= om.newMap(omaptype='RBT',
cmpfunction=compareDates)
mapa_data= om.newMap(omaptype='RBT',
cmpfunction=compareDates)
diccionario ={}
titulos = om.keySet(result["arbol"])
cantidad_año= lt.size(result["lista"])
totales=0
for cada in lt.iterator(titulos):
if titulo in cada:
lisat_condicion= me.getValue(om.get(result["arbol"],cada))
for data in lt.iterator(lisat_condicion):
totales+=1
if not data[condicion]=="" and not data[condicion]==0:
mapa_data=uptime(mapa_data,data)
mapa_de_lacondicon= add_data_req_condicion(mapa_de_lacondicon, float(data[condicion]))
if float(data[condicion]) in diccionario:
diccionario[float(data[condicion])]+=1
else:
diccionario[float(data[condicion])]=1
minimo = om.minKey(mapa_de_lacondicon)
maximo= om.maxKey(mapa_de_lacondicon)
usados = 0
for cada in lt.iterator(om.valueSet(mapa_de_lacondicon)):
usados+= int(cada)
```

```
posible_lo(diccionario, bins, condicion, minimo,maximo)

lista=sacas(mapa_data, condicion)

return totales,cantidad_año, usados,minimo, maximo,lista
```

Descripción

Entrada	- El año relevante - El título de la región asociada - La propiedad de conteo (magnitud, profundidad o significancia). - El número de casillas en los que se divide el histograma.
Salidas	El número de eventos sísmicos dentro del periodo anual relevante. • El número de eventos sísmicos utilizados para crear el histograma de la propiedad. • Valor mínimo y valor máximo de la propiedad consultada en el histograma. • El histograma con la distribución de los eventos sísmicos según la propiedad. • Listado de los eventos que cumplen las condiciones de conteo para el histograma. o La fecha y hora del evento (time). o La Latitud donde ocurrió el evento (lat). o La longitud donde ocurrió el evento (long). o El título del evento sísmico (title). o El código del evento (code). o Propiedad del conteo en el histograma (mag, Depth, sig).
Implementado (Sí/No)	Si por Ximena Lopez Cruz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1: Se hace el llamado a la funcion keyset para obtener todas las llaves	O(1)
Paso 2 : Se itera con un For por cada una de las llaves del mapa.	O(n)
Paso 3: Se realiza un if in para saber si la región está dentro del título que es la llave.	O(1)
Paso 4: Si se cumple la condición anterior se obtiene una lista de los valores de esa llave dentro del mapa	O(1)

Paso 5: Se realiza la búsqueda de un máximo y un mínimo en árbol rojo-negro.	O(nlogn + nlogn)
Paso 5: Se realiza un For iterativo por cada valor dentro de la lista, a su vez si cumple con la condición se va a agregando a dos mapas y un diccionario.	O(M)
Paso 6: Se realiza un For por cada uno de los vales de ocurrencia para obtener la cantidad total de valores usados para la gráfica.	O(N)
Paso 7: Se realiza la gráfica con los datos de ocurrencia del diccionario	O()
TOTAL	***O(M*n*nlogn)***

Pruebas Realizadas

Para todas las pruebas se utilizó el año 2020, 'Alaska' como área de interés, 'mag' como la propiedad de interés, y un número de bins de 10.

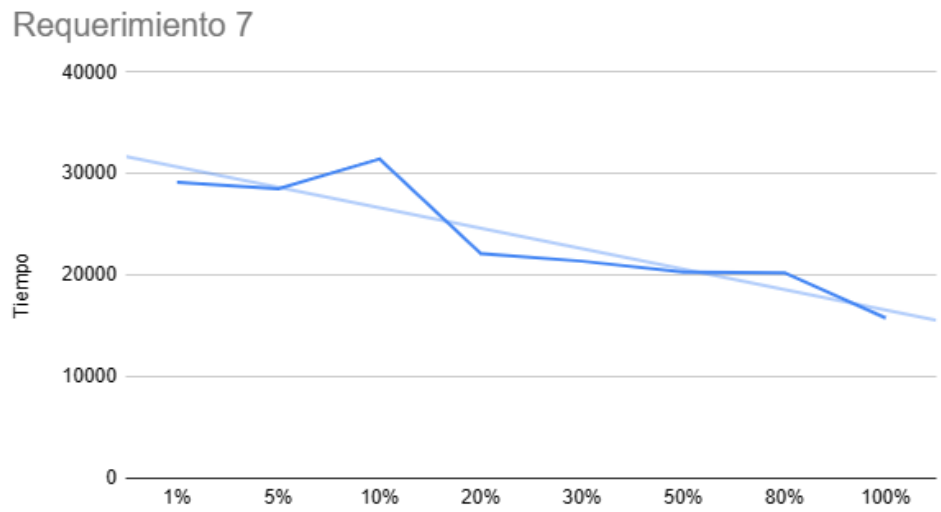
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

	PROMEDIO
1%	29178.7079
5%	28522.42818
10%	31453.38853
20%	22114.41693
30%	21382.42949
50%	20316.55412
80%	20247.56328
100%	15785.02466

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.

La complejidad teórica ***O(M*n* nlogn).*** Sin embargo, en la gráfica de los tiempos ejecutados, se tiene una complejidad con tendencia lineal con pendiente negativa, es decir que a medida que los datos aumentan la complejidad va disminuyendo proporcionalmente. Siendo su mayor punto en el 10 porciento de los datos y luego sigue disminuyendo. Respecto a la complejidad de la gráfica no fue posible encontrarla. Sin embargo, con cada una de las cantidades de datos tiene tiempos de ejecución similares.

Requerimiento <<8>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

El requerimiento 8 es particular en medida que se apoya de los demás requerimientos. Es decir, su tiempo de ejecución viene dado por el tiempo de ejecución de los demás requisitos, sumado al tiempo de ejecución del requerimiento 8. Se usará una estructura fundamental que puede cambiar levemente dependiendo de cada requerimiento, pero que está presente para la creación de todos los mapas interactivos. Cabe resaltar que se agregó un límite máximo de 100000 marcadores por mapa. Esto quiere decir que, en caso de que una respuesta tenga más de esta cantidad, solo se mostrarán los cien mil primeros resultados. Este límite se estableció debido a que resulta demasiado impráctico realizar un

mapa interactivo con demasiados datos: el tiempo de creación excedería las horas, y el peso de este superaría los gigabytes.

```
m= folium.Map(tiles=MAP_TILE,
                      attr=MAP_ATTRIBUTES)
            mCluster = MarkerCluster(name="Cluster").add_to(m)
            path = '.\\Data\\maps\\req0.html'
            for result in lt.iterator(data_structs['lista_temblores']):
                mssg=''
                for key in result:
                    mssg += f'{key}: {result[key]}\n'

folium.Marker(location=[float(result['lat']),float(result['long'])],
                           tooltip=
html.escape(result['title']).replace('`','&#96;'),
                           popup=Popup(mssg,parse_html=True)).add_to(mCluster)
                props+=1
                if props>MAX_MAP_PROPS:
                    break
            folium.LayerControl().add_to(m)
            m.save(path)
            os.system(f'start {path}')
```

Pasos	**Complejidad**
Iterar a través de la lista con los temblores ingresada como parámetro.	O(n)
Iterar a través de las llaves que componen el dato en específico. Esto creará una cadena con la información del evento para que se muestre en el mapa.	O(m), donde m es el número de columnas o llaves con las que cuenta un dato. Su valor máximo teórico es 27, que es el número de columnas en el csv. Sin embargo, algunos resultados podrán tener un menor número de llaves. Es decir, en el peor caso, este componente tendrá una complejidad O(27).
TOTAL	***O(n) + O(tiempo de ejecución del requerimiento específico)***

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).

Para todas las pruebas se usó la opción de crear un mapa interactivo para la opción 1, con las entradas de fechas entre 1999-03-21T05:00 y '2004-10-23T17:30.

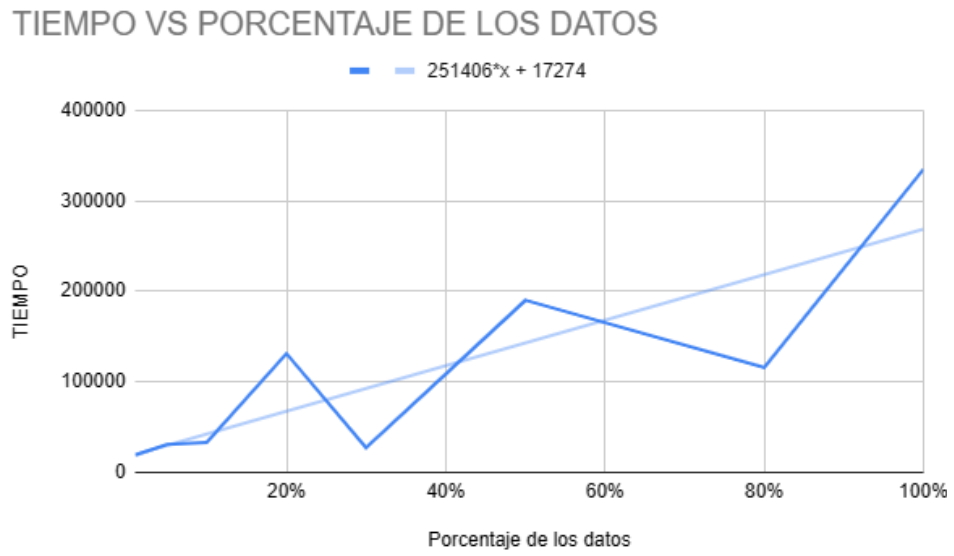
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

	PROMEDIO
1%	19278.41602
5%	30827.10867
10%	33083.83018
20%	131023.344
30%	27,192
50%	190090.7826
80%	115849.6342
100%	335009.8571

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

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

De acuerdo al análisis de complejidad realizado y los resultados obtenidos en la gráfica, la complejidad del requerimiento aumenta de forma lineal O(n). Esto se debe a que, como se analizó en el requerimiento 1 (requerimiento en base al cual se realizaron las pruebas), este sigue una complejidad de O(n). Igualmente, la estructura base en la creación del mapa interactivo sigue una complejidad de O(n). O(n)+O(n) implica una complejidad lineal de O(n), que es lo que se observa en la gráfica. Las fluctuaciones en el tiempo de ejecución pueden deberse a detalles específicos de la forma en que funciona internamente la librería folium usada en la elaboración de los mapas. Sin embargo, la linea de tetndencia es clara en cuanto al crecimiento lineal del tiempo de ejecución.