ANÁLISIS DEL RETO

Samuel Villamizar Hernandez, 202311883, sa.villamizar2@uniandes.edu.co

Nicolas Mauricio Bobadilla Hernandez, 202310324, n.bobadillah@uniandes.edu.co

Santiago Gomez, 202315097, s.gomezo2@uniandes.edu.co

Requerimiento <<1>>

Descripción

```python
def req_1(data_structs, date1, date2):
    """
    Función que soluciona el requerimiento 1S
    """
    # TODO: Realizar el requerimiento 1
    arbol_fecha = data_structs['fecha']
    lista_rangos = om.values(arbol_fecha, date1, date2)
    tamaño_rangos = lt.size(lista_rangos)
    llaves = om.keys(arbol_fecha, date1, date2)
    lista_retorno = lt.newList()

    if tamaño_rangos > 6:
        lista_rangos = get3(lista_rangos)
        keys_1=6
        for mag in range(1,keys_1+1):
            sig = lt.getElement(llaves, mag)
            lista = me.getValue(om.get(arbol_fecha, sig))
            merg.sort(lista['lsttemblores'],sort_criteria_date)

    else:
        for mag in range(1,tamaño_rangos+1):
            sig = lt.getElement(llaves, mag)
            lista = me.getValue(om.get(arbol_fecha, sig))
            merg.sort(lista['lsttemblores'],sort_criteria_date)

    titulos = ['mag', 'lat', 'long', 'depth', 'sig', 'gap','nst', 'title', 'cdi', 'mmi', 'magType', 'type', 'code']

    for dato in lt.iterator(lista_rangos):
        diccionario = {}
        diccionario['time'] = dato['lsttemblores']['first']['info']['time']
        diccionario['eventos'] = lt.size(dato['lsttemblores'])
        filtro = filtrar(dato['lsttemblores'], titulos)
        diccionario['details'] = tabulate(lt.iterator(filtro), headers='keys', tablefmt= 'fancy_grid')
        lt.addLast(lista_retorno, diccionario)

    merg.sort(lista_retorno, comparedate_req1)
    return tamaño_rangos, lista_retorno
```

Se usa el om.values() conocer los eventos sísmicos mundiales ocurridos durante un intervalo de fechas específico.

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

Análisis de complejidad de cada uno de los pasos del algoritmo

Paso	Complejidad
Inicialización	O(1)
Preprocesamiento	O(n), donde n es el tamaño de la lista de rangos
Procesamiento	O(n log n), donde n es el tamaño de la lista de rangos
Postprocesamiento	O(n log n), donde n es el tamaño de la lista de resultados
Total	O(n log n)

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).

Entrada	**Tiempo (ms)**
small	3.6351
5 pct	10.7988
10 pct	475.710
20 pct	43.5582
30 pct	64.47
50 pct	114.47545
80 pct	193.40
large	224.780

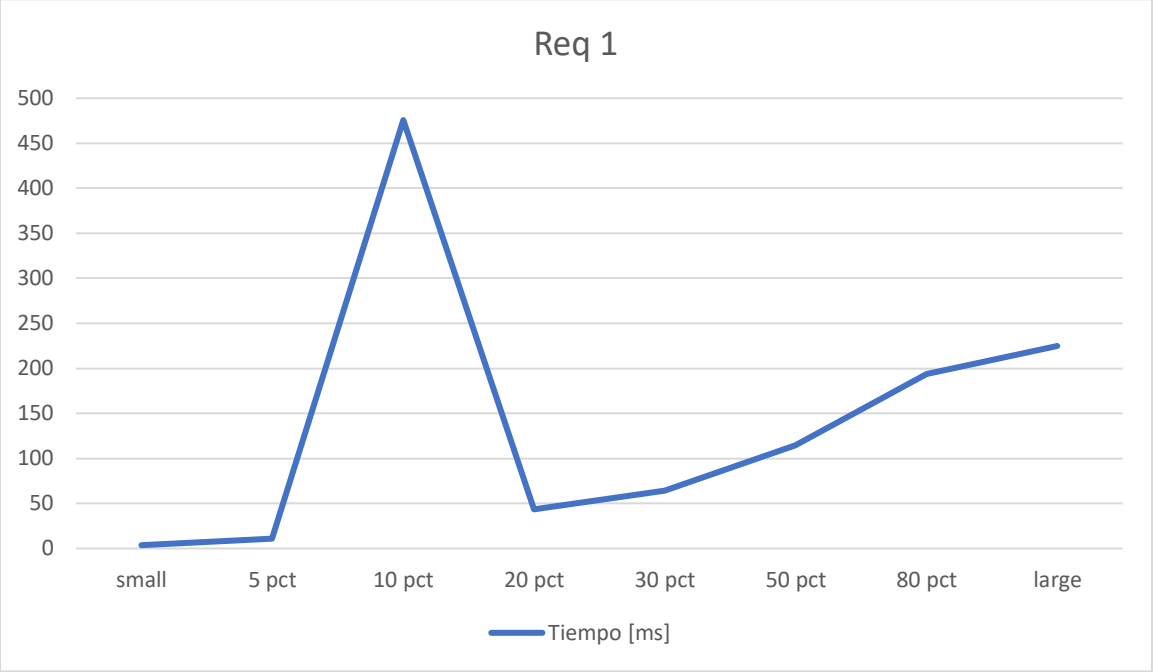
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	912	3.6351
5 pct	4556	10.7988
10 pct	9110	475.710
20 pct	18385	43.5582
30 pct	27790	64.47
50 pct	46080	114.47545
80 pct	73248	193.40
large	91120	224.780

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El código define una función llamada `req_1` diseñada para abordar un requerimiento 1S. Utiliza un diccionario `data_structs` que contiene una estructura de datos llamada 'fecha'. La función extrae una lista de rangos de fechas entre `date1` y `date2`, ordena sublistas asociadas a estos rangos por fecha, y crea un diccionario para cada rango que incluye detalles de eventos sísmicos. Los resultados se almacenan en una lista llamada `lista_retorno`, que se ordena finalmente por fecha antes de ser devuelta junto con el tamaño de la lista de rangos. Aunque algunas funciones utilizadas no están definidas en el código proporcionado, la estructura sugiere un procesamiento de datos sísmicos dentro de un intervalo de fechas determinado.

Requerimiento <<2>>

Descripción

```
def req_2(data_structs,limite_inf, limite_sup):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    datos = data_structs['magnitud']
    magnitudes = om.values(datos, limite_inf, limite_sup)
    llaves = om.keys(datos, limite_inf, limite_sup)
    keys = om.size(magnitudes) #retorna el la cantidad de magnitudes en el rango
    size = 0 #Cantidad total de temblores en el rango
    lista_retorno = lt.newList()

    for mag in lt.iterator(magnitudes):
        size += om.size(mag['lsttemblores'])

    if keys > 6:
        magnitudes = get3(magnitudes)
        llave3 = 0
        keys_1=6
        for mag in range(1,keys_1+1):
            sig = lt.getElement(llaves, mag)
            lista = me.getValue(om.get(datos, sig))
            merg.sort(lista['lsttemblores'],sort_criteria_date)
        for mg in lt.iterator(magnitudes):
            llave3 += om.size(mg['lsttemblores'])

    else:
        llave3 = llaves
        for mag in range(1,keys+1):
            sig = lt.getElement(llaves, mag)
            lista = me.getValue(om.get(datos, sig))
            merg.sort(lista['lsttemblores'],sort_criteria_date)

    titulos = ['time', 'lat', 'long', 'depth', 'sig', 'gap', 'nst', 'title', 'cdi', 'mmi', 'magType', 'type', 'code']
```

```
  for dato in lt.iterator(magnitudes):
      lst = lt.newList()
      lt.addFirst(lst, dato['lsttemblores']['first']['info']['mag'])
      diccionario = {}
      diccionario['mag'] = dato['lsttemblores']['first']['info']['mag']
      diccionario['events'] = lt.size(dato['lsttemblores'])
      filtro = filtrar(dato['lsttemblores'], titulos)
      if lt.size(filtro)>6:
          filtro = get3(filtro)
      quk.sort(filtro, sort_criteria_date)
      diccionario['details'] = tabulate(lt.iterator(filtro), headers='keys', tablefmt= 'fancy_grid')
      lt.addFirst(lista_retorno, diccionario)


  return size, keys, lista_retorno, llave3
```

Este requerimiento nos permite conocer los temblores presentados en un cierto rango de magnitudes.

Sus parametros de entrada son: Límite inferior de la magnitud y límite superior de la magnitud.

El condigo inicia creando las variables a utilizar dentro de su funcionamiento, para posteriormente iterar una de estas (magnitud), la cual es un mapa, con el fin de contar los datos dentro de cada llave del mismo y dar un valor a la cantidad de eventos ocurridos dentro del rango de magnitudes.

Posteriormente se analiza el size de este mapa con el fin de recortar su información si esta supera los 6 datos. En caso de que si se cumpla esta condición se ordena la lista en términos de la fecha para posteriormente cortarla, seleccionando solo los tres primeros y tres últimos, en caso del else, simplemente se ordena la lista. Por último, se realiza una última iteración con el fin de en donde se crea el diccionario que contendrá toda la información necesaria, el cual será añadido a una lista, la cual será la lista de respuesta. En cuanto a su complejidad esta es de O(n) ya que lo máximo que se realiza es una iteración, y aunque hay algunas que presentan algunos sort dentro, su complejidad no se toma en cuenta al ser menor a n, además este sort se realizará n veces por lo que se mantiene una complejidad O(n) y no mayor.

Entrada	Límite inferior, Límite superior
Salidas	Lista con los datos en el rango especificado, si son mayores a 6 muestra los tres primeros y tres últimos, total de diferentes magnitudes, total de eventos diferentes. Feo
Implementado (Sí/No)	Si, se hizo en grupo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de variables	O(1)
Interacción de la lista de los datos de las magnitudes	O(n)
Comparación	O(1)
Iteraciones dentro de la comparación	O(n)
Merge dentro del ciclo	O(n)
Else	O(1)
Iteración dentro del else	O(n)
Merge dentro del else	O(n)
Iteración sobre la lista con los tres últimos y tres primeros	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1. Además de utilizar como datos de entrada: 3.500 y 6.500

Procesadores	**AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**

small	13.06060
5 pct	80.70319
10 pct	230.9218
20 pct	2360.703
30 pct	3259.665
50 pct	20515.93
80 pct	85825.66
large	292498.19

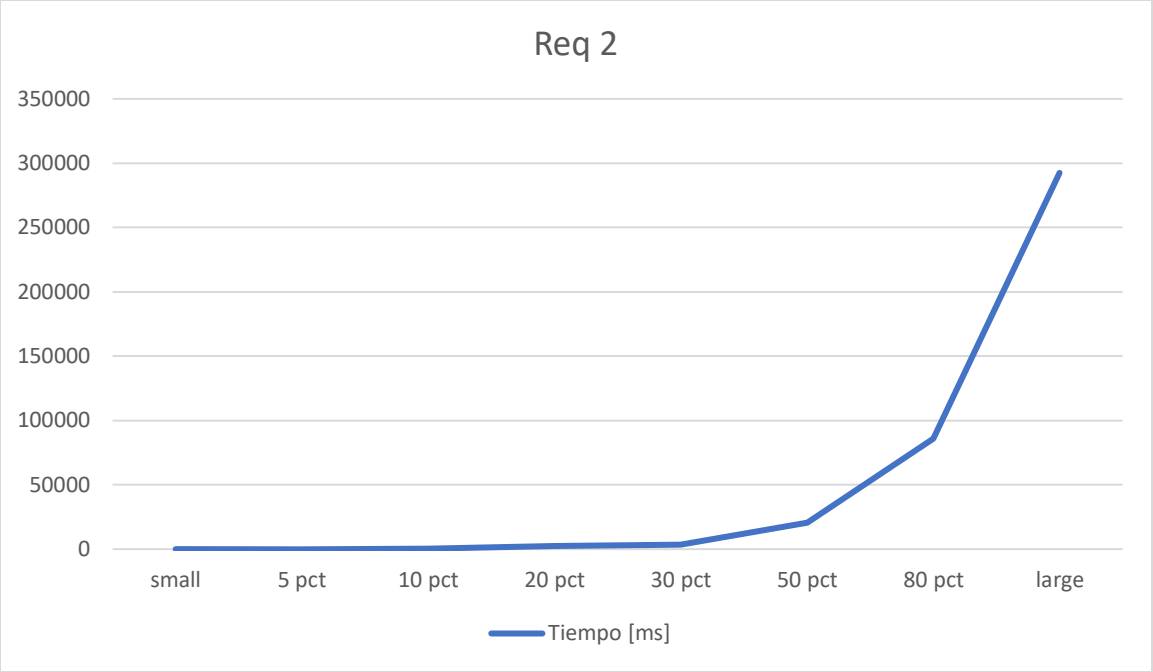
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Total diferentes mags: 72 Total eventos en las mags dadas : 4482 Total eventos solo en los primeros y ultimos 3: 143	13.06060
5 pct	Total diferentes mags: 126 Total eventos en las mags dadas : 26741 Total eventos solo en los primeros y ultimos 3: 900	80.70319
10 pct	Total diferentes mags: 158 Total eventos en las mags dadas : 71371 Total eventos solo en los primeros y ultimos 3: 2392	230.9218
20 pct	Total diferentes mags: 181 Total eventos en las mags dadas : 160587 Total eventos solo en los primeros y ultimos 3: 5376	2360.703
30 pct	Total diferentes mags: 188 Total eventos en las mags dadas : 294540 Total eventos solo en los primeros y ultimos 3: 9816	3259.665
50 pct	Total diferentes mags: 200 Total eventos en las mags dadas : 517761 Total eventos solo en los primeros y ultimos 3: 17302	20515.93
80 pct	Total diferentes mags: 225 Total eventos en las mags dadas : 963981 Total eventos solo en los primeros y ultimos 3: 32228	85825.66
large	Total diferentes mags: 225 Total eventos en las mags dadas : 1320950 Total eventos solo en los primeros y ultimos 3: 44122	292498.19

Graficas

Las gráficas con la representación de las pruebas realizadas.



La grafica muestra un comportamiento creciente a medida que los datos totales a analizar incrementan, se puede observar cómo antes del 50% de los datos la gráfica presenta una pendiente despreciable a comparación a el cambio de pendiente después de este porcentaje, donde esta simplemente se dispara y por ende su tiempo de ejecución aumenta correspondientemente a la cantidad de datos analizados. Esto ocurre por los merge que se presentan dentro de los ciclos, ya que al presentar muchos datos el n del requerimiento ya se verá afectado a un valor mayor al tener que recorrer la lista más de n veces.

Análisis

Requerimiento <<3>>

Descripción

```python
def req_3(data_structs, mags, profundidad):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    arbol_mags = data_structs['magnitud']
    mayor = lt.lastElement(om.keySet(arbol_mags))
    keys_mag = om.keys(arbol_mags, mags, mayor)
    lista_requisito = lt.newList()
    lista_retorno = lt.newList()
    numero_total = 0
    for mayores in lt.iterator(keys_mag):
        value_mag = me.getValue(om.get(arbol_mags, mayores))
        arbol_depth = value_mag['arbol_profundidad']
        prof = om.keySet(arbol_depth)
        for i in range(1,(lt.size(arbol_depth)+1)):
            dep = lt.getElement(prof, i)
            lista = me.getValue(om.get(arbol_depth, dep))
            for elemento in lt.iterator(lista):
                if float(elemento['depth']) <= profundidad:
                    lt.addLast(lista_requisito, elemento)
                    numero_total += 1

    merg.sort(lista_requisito, comparedate_req1)
    lista_10 = lt.subList(lista_requisito,1,10)
    titulos = ['mag','lat', 'long', 'depth', 'sig', 'gap', 'nst', 'title', 'cdi', 'mmi', 'magType', 'type', 'code']

    for finales in lt.iterator(lista_10):
        dicc = {}
        papa_inoa =lt.newList()
        lt.addLast(papa_inoa, finales)
        filtro = filtrar(papa_inoa, titulos)
        dicc['time'] = finales['time']
        dicc['events'] = lt.size(filtro)
        dicc['details'] = tabulate(lt.iterator(filtro),headers='keys', tablefmt='fancy_grid')
        lt.addLast(lista_retorno, dicc)

    return lista_retorno, numero_total
```

En este requerimiento se realiza la consulta para consultar los 10 eventos sísmicos más recientes que superen una magnitud mínima y no superen una profundidad máxima indicada

Entrada	La magnitud menor, la profundidad menor deseada
Salidas	El tamaño de los temblores que cumplen ese requisito.
Implementado (Sí/No)	Si, lo implemento Samuel Villamizar

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Paso	Complejidad temporal
def req_3(data_structs, mags, profundidad)	O(1)
arbol_mags = data_structs['magnitud']	O(1)

mayor = lt.lastElement(om.keySet(arbol_mags))	O(n)
keys_mag = om.keys(arbol_mags, mags, mayor)	O(n)
lista_requisito = lt.newList()	O(1)
lista_retorno = lt.newList()	O(1)
numero_total = 0	O(1)
for mayores in lt.iterator(keys_mag)	O(n)
value_mag = me.getValue(om.get(arbol_mags, mayores))	O(1)
arbol_depth = value_mag['arbol_profundidad']	O(1)
prof = om.keySet(arbol_depth)	O(n)
for i in range(1,(lt.size(arbol_depth)+1)):	O(n)
dep = lt.getElement(prof, i)	O(1)
lista = me.getValue(om.get(arbol_depth, dep))	O(1)
for elemento in lt.iterator(lista):	O(m)
if float(elemento['depth']) <= profundidad:	O(1)
lt.addLast(lista_requisito, elemento)	O(1)
numero_total += 1	O(1)
merg.sort(lista_requisito, comparedate_req1)	O(n log n)
lista_10 = lt.subList(lista_requisito,1,10)	O(n)
titulos = ['mag','lat', 'long', 'depth', 'sig', 'gap', 'nst', 'title', 'cdi', 'mmi', 'magType', 'type', 'code']	O(1)
for finales in lt.iterator(lista_10):	O(n)
dicc = {}	O(1)
papa_inoa =lt.newList()	O(1)
lt.addLast(papa_inoa, finales)	O(1)
filtro = filtrar(papa_inoa, titulos)	O(m)
dicc['time'] = finales['time']	O(1)
dicc['events'] = lt.size(filtro)	O(1)
dicc['details'] = tabulate(lt.iterator(filtro),headers='keys', tablefmt='fancy_grid')	O(m)
lt.addLast(lista_retorno, dicc)	O(1)
return lista_retorno, numero_total	O(n log n)
Total	O(n^3)

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).

Tamaño	Tiempo (s)
small	18.462
5 pct	340.28
10 pct	1757.62
20 pct	5179.19
30 pct	11468.15
50 pct	32407.68
80 pct	86537.77
large	132166.01

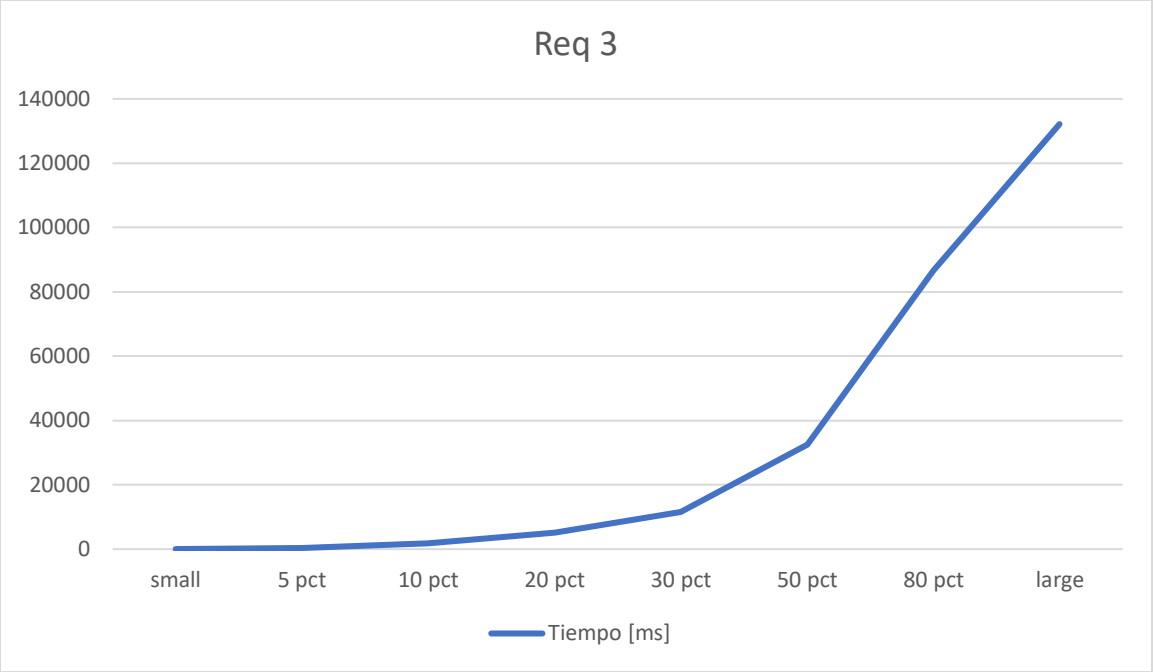
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	362	18.462
5 pct	1886	340.28
10 pct	3738	1757.62
20 pct	7565	5179.19
30 pct	11377	11468.15
50 pct	19082	32407.68
80 pct	30583	86537.77
large	38297	132166.01

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El código define una función denominada req_3 para abordar un requerimiento 3. Utiliza un diccionario llamado data_structs, que contiene una estructura de datos llamada 'magnitud'. La función busca eventos sísmicos de magnitudes superiores a un valor dado (mags) y a una profundidad inferior o igual a otra dada (profundidad). Itera sobre las magnitudes en el rango especificado, accede a un árbol de profundidades asociado a cada magnitud, y filtra los eventos que cumplen con los criterios de magnitud y profundidad. Estos eventos filtrados se almacenan en una lista llamada lista_requisito, y se realiza un conteo total de eventos (numero_total). Luego, se ordena esta lista por fecha y se seleccionan los primeros 10 eventos. La función devuelve una lista llamada lista_retorno, que contiene diccionarios con información sobre estos eventos seleccionados, así como el número total de eventos que cumplen con los criterios de magnitud y profundidad.

Cabe aclarar que, aunque se haga una triple iteración, el requerimiento es realmente rápido puesto que se va haciendo un filtrado total y recorte de las listas haciendo que cada vez la iteración sea menor, por lo que se termina recortando la lista cada vez más. Haciendo que el requerimiento logre salir en una velocidad menor.

Requerimiento <<4>>

```
def req_4(data_structs, sigMin, gapMax):
    """
    Función que soluciona el requerimiento 4
    """
    values_sig = om.values(data_structs['sig'], sigMin, om.maxKey(data_structs['sig']))
    dates = om.newMap()

    for value_sig in lt.iterator(values_sig):
        values_gap = om.values(value_sig['gap'], om.minKey(value_sig['gap']),gapMax)
        for value_gap in lt.iterator(values_gap):
            for temblor in lt.iterator(value_gap['lsttemblores']):
                date = temblor['time']
                if om.contains(dates, date):
                    value_date = me.getValue(om.get(dates, date))
                else:
                    value_date = lt.newList()
                    om.put(dates, date, value_date)
                lt.addLast(value_date, temblor)

    lista_retorno = lt.newList()
    fechas = om.keySet(dates)
    values = om.valueSet(dates)

    eventos = 0
    for i in range(1, (lt.size(fechas)+1)):
        date = lt.getElement(fechas, i)
        value = lt.getElement(values, i)
        diccionario = {}
        diccionario["date"] = date
        diccionario["events"] = lt.size(value)
        diccionario["details"] = value
        eventos += lt.size(value)
        lt.addFirst(lista_retorno, diccionario)

    return lt.size(fechas), eventos, lista_retorno
```

Este requerimiento nos permite organizar y estructurar la información de los temblores en función de las fechas, contando y registrando detalles asociados a cada fecha. Los resultados finales proporcionan estadísticas sobre la ocurrencia de temblores en diferentes fechas.

Entrada	Data_structs, sigMin, gapMax
Salidas	El número total de fechas distintas en las que ocurrieron temblores
Implementado (Sí/No)	Si, lo implemento Nicolas Bobadilla

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Paso	Complejidad
Inicialización	O(1)
Preprocesamiento	O(n)
Procesamiento	O(n*m*p)
Postprocesamiento	O(n*log(n))
Total	O(n*m*p + n*log(n))

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron: 300.0 y 45.0

Procesadores	**APPLE CHIP M2**
Memoria RAM	8 GB
Sistema Operativo	SONOMA 14.0

Entrada	**Tiempo (ms)**
small	19.92032
5 pct	27.838473
10 pct	37.83847
20 pct	5088.944
30 pct	708.9384
50 pct	100.04933
80 pct	1340.39764
large	151436.076

Tablas de datos

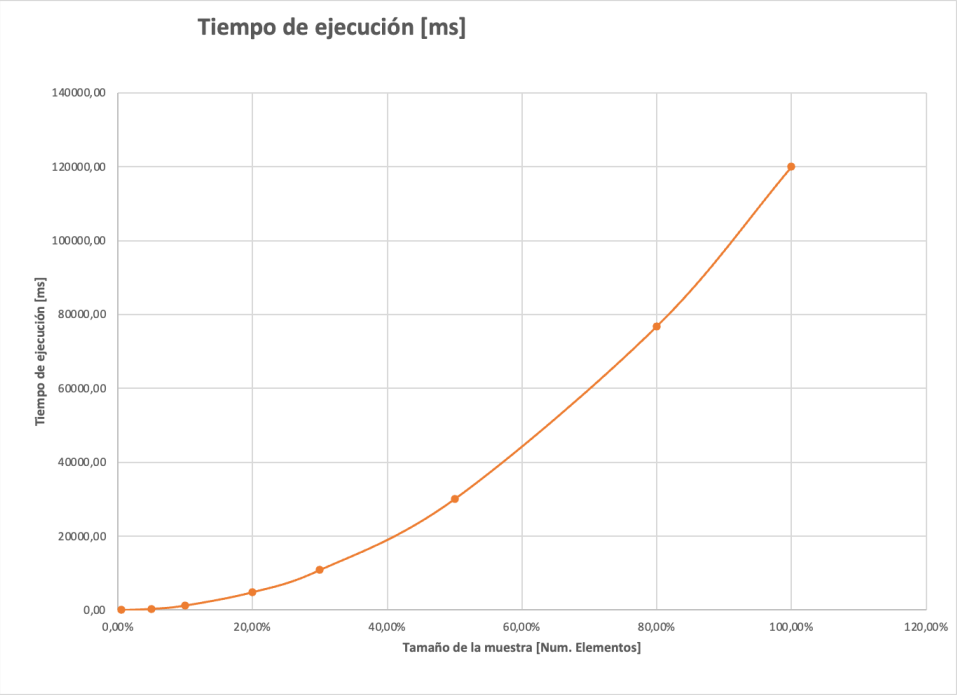
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	714	19.92032
5 pct	3429	27.838473
10 pct	6846	37.83847
20 pct	13808	5088.944
30 pct	20822	708.9384
50 pct	34497	100.04933
80 pct	55039	1340.39764

large	68634	151436.076

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La grafica muestra un comportamiento normal y muestra el crecimiento de los datos a medida de que se van cargando mas pct, este requerimiento tuvo una complejidad de O(n*m*p + n*log(n)).

Para resumir, la entrega del código cumple con lo pedido en el requisito 4 al ordenar y mostrar información acerca de los sismos basándose en las fechas especificadas así como aplicando filtros para clasificarlos por su "sig" o "gap". La implementación de las funciones utilizadas tiene un efecto directo sobre la eficiencia del código. A su vez, el procesamiento de los datos mediante el filtrado, organización y ordenamiento también influencia en su complejidad temporal. Si se realizan optimizaciones en estas operaciones y se utilizan estructuras de datos eficientes, es probable que la velocidad del algoritmo mejore. Tener la capacidad de lidiar con fechas repetidas y poder ajustar los filtros según sea necesario son características muy positivas. El código logra cumplir con su objetivo general. Sin embargo,

considerando la implementación particular así como también las características propias de las funciones subyacentes existen posibilidades para optimizarlo aún más incrementando así su eficiencia.

Requerimiento <<5>>

Descripción

```python
def req_5(data_structs, mindepth, minest):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    datos = data_structs['depth']
    maxi = lt.lastElement((om.keySet(datos)))
    depths = om.values(datos, mindepth, maxi)
    filtro1 = om.newMap('RBT', comparedate_inverse)
    lista_retorno = lt.newList()
    for dato in lt.iterator(depths):
        lst = dato['lsttemblores']
        for dato2 in lt.iterator(lst):
            nst = float(dato2['nst'])
            fecha = dato2['time']
            if nst >= minest:
                entry = om.get(filtro1, fecha)
                if entry == None:
                    fechentry = new_parareq5(fecha)
                    om.put(filtro1, fecha, fechentry)
                else:
                    fechentry = me.getValue(entry)
                add_parareq5(fechentry, dato2)
    mini = om.minKey(filtro1)
    maxi2 = om.maxKey(filtro1)
    filtro2 = om.values(filtro1, mini, maxi2)
    size = om.size(filtro2) #RETORNAR, cantidad de fechas diferentes
    eventos = 0

    for ev in lt.iterator(filtro2):
        eventos += om.size(ev['lstfecha']) #RETORNAR, Numero total de eventos

    if om.size(filtro2)>20:
        filtro3 = lt.subList(filtro2, 1, 20)

    titulos = ['mag', 'lat', 'long', 'depth', 'sig', 'gap', 'nst', 'title', 'cdi', 'mmi', 'magType', 'type', 'code']
```

```
    for dato in lt.iterator(filtro3):
        liste = lt.newList()
        lt.addLast(liste, dato['lstfecha']['first']['info']['time'])
        diccionario = {}
        diccionario['time'] = dato['lstfecha']['first']['info']['time']
        diccionario['events'] = lt.size(dato['lstfecha'])
        filtrera = filtrar(dato['lstfecha'], titulos)
        if lt.size(filtrera) > 6:
            filtrera = get3(filtrera)
        merg.sort(filtrera, comparedate)
        diccionario['details'] = tabulate(lt.iterator(filtrera), headers = 'keys', tablefmt= 'fancy_grid')
        lt.addLast(lista_retorno, diccionario)

    if lt.size(lista_retorno) > 6:
        lista_retorno = get3(lista_retorno)
    return size, eventos, lista_retorno


def new_parareq5(fecha):
    entry = {'lstfecha' : None}
    entry['lstfecha'] = lt.newList('SINGLE_LINKED', comparedate)
    return entry

def add_parareq5(fecha, dato):
    lst = fecha['lstfecha']
    lt.addLast(lst, dato)
    return fecha
```

El requerimiento 5 analiza los terremotos teniendo en cuenta una profundidad minima y un numero de estacion minimo igualmente. Principalmente se realiza un filtro el cual tienen en cuenta la profundidad minima y guarda los valores que cumplen la condición, para posteriormente analizarla y tener en cuenta la cualidad de estacion, donde si los valores de estación se cumplen los guarda, teniendo como modo de ordenamiento la fecha del suceso, para tener en cuenta más adelante. Posteriormente se utiliza la funcion om.values como un metodo de ordenamiento, donde se ordenaran de menor a mayor los datos y con este se obtiene el numero de eventos ocurridos, que cumplan con las caracteristicas anteriormente mencionadas. Seguido de lo anterior, se observa si se tienen mas de 20 datos para recortarlos a ese valor. La lista ya recortada se itera con el fin de poder imprimierla como una tabla tabulate y si su size supera los 6 elementos se entreagn al usuario sus 3 primeros y tres ultimos. La complejidad de este algoritmo es O(n^2).

Entrada	Profundidad mínima, número mínimo de estaciones
Salidas	Lista de respuesta analizando los últimos 20 datos presentados que cumplan las condiciones solicitadas
Implementado (Sí/No)	Si, lo implemento Santiago Gomez.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de variables	O(1)
Om.values	O(n)
Iteración lista dapths	O(n)
Iteración dentro de la anterior por dato	O(n^2)
Condición dentro de las iteraciones	O(1)

Om.values	O(n)
Iteración lista (filtro)	O(n)
Condición	O(1)
Iteración filtro	O(n)
Condicional	O(1)
TOTAL	***O(N^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1. Además de utilizar como datos de entrada: 23.00 y 38

Procesadores	**AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	15.9028
5 pct	62.7867
10 pct	153.148
20 pct	1491.08
30 pct	2106.88
50 pct	844.818
80 pct	1588.40
large	2298.29

Tablas de datos

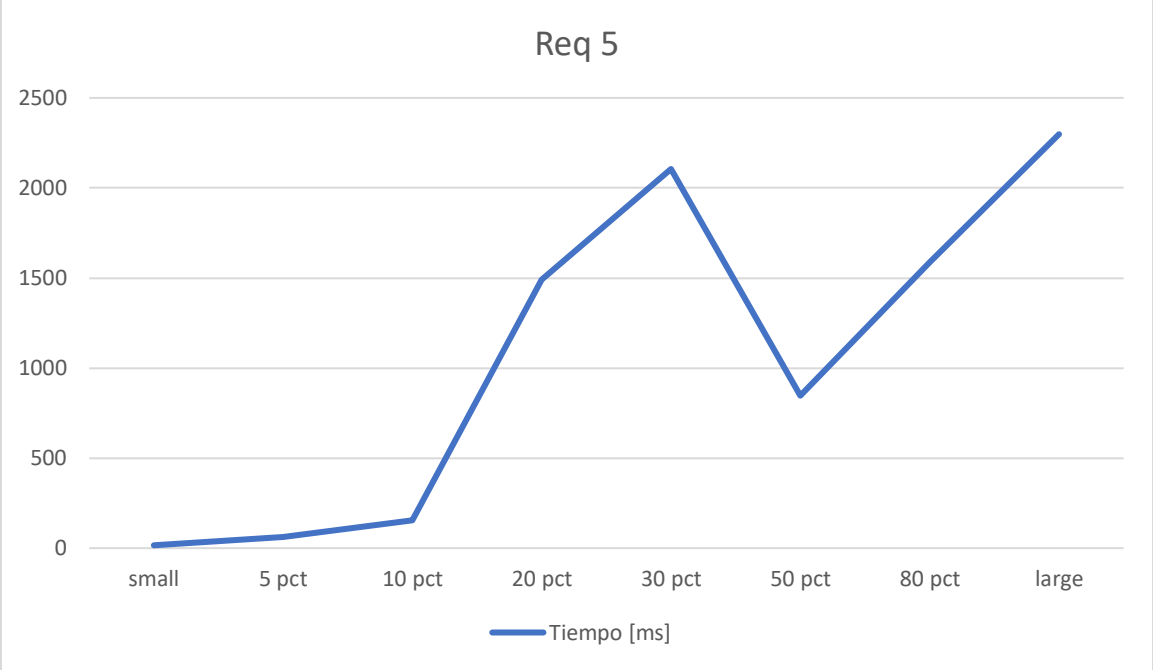
Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Total different dates: 461 Total events: 461	15.9028
5 pct	Total different dates: 2295 Total events: 2757	62.7867
10 pct	Total different dates: 4628 Total events: 7389	153.148
20 pct	Total different dates: 9379 Total events: 16778	1491.08
30 pct	Total different dates: 13978 Total events: 30780	2106.88

50 pct	Total different dates: 23226 Total events: 54059	844.818
80 pct	Total different dates: 46179 Total events: 100449	1588.40
large	Total different dates: 46179 Total events: 137518	2298.29

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La grafica muestra un comportamiento algo extraño en la función ya que pareciera como si la relación entre cantidad de datos y tiempo de ejecución desapareciera por algunos momentos. Esto puede ser causado a la información que el dispositivo estaba manejando en ese momento y por eso los “bajones” en los tiempos de ejecución en mayores datos. Lo recalcable de la gráfica es que se mantiene lineal y no es una gráfica cuadrada como se observó en la complejidad del algoritmo, cosa que pasa ya que la segunda iteración realmente solo busca una información precisa por lo que se podría decir que este segundo ciclo es una constante y por ende su complejidad real seria de O(n).

Requerimiento <<6>>

La función comienza obteniendo el árbol de eventos significativos del año especificado. Luego, itera sobre el árbol, calculando la distancia entre cada evento y el punto especificado. Si la distancia es menor o igual al radio dado, la función agrega el evento a una lista. Una vez que se han iterado sobre todos los

eventos significativos, la función ordena la lista de fechas según la fecha de los eventos. Luego, encuentra la fecha del evento más significativo. Finalmente, la función divide la lista de fechas en dos partes, una que contiene los n eventos anteriores a la fecha del evento más significativo y otra que contiene los n eventos posteriores a la fecha del evento más significativo.

Descripción

Entrada	El año relevante (en formato "%Y"). La Latitud de referencia para el área de eventos (lat). La longitud de referencia para el área de eventos (long). El radio [km] del área circundante (float). El número de los N eventos de magnitud más cercana a mostrar.
Salidas	El evento sísmico más significativo ocurrido en el área durante el año indicado. El número total de eventos sísmicos registrados dentro del área circundante. Los N eventos sísmicos más cercanos en tiempo, antes y después, al evento más significativo organizados cronológicamente desde el más reciente. Donde cada evento debe mostrar la siguiente información
Implementado (Sí/No)	Si se implementó en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Línea	Complejidad
1	O(1)
2	O(1)
3	O(n)
4	O(1)
5-11	O(n)
12-13	O(n)
14	O(n log n)
15-21	O(n)
22-23	O(n)
24-25	O(n)
26-27	O(n)
28	O(1)

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).

Entrada	Tiempo (ms)
small	2.11
5 pct	0.0
10 pct	30.97
20 pct	36.44
30 pct	144.69
50 pct	320.28
80 pct	665.68
large	702.97

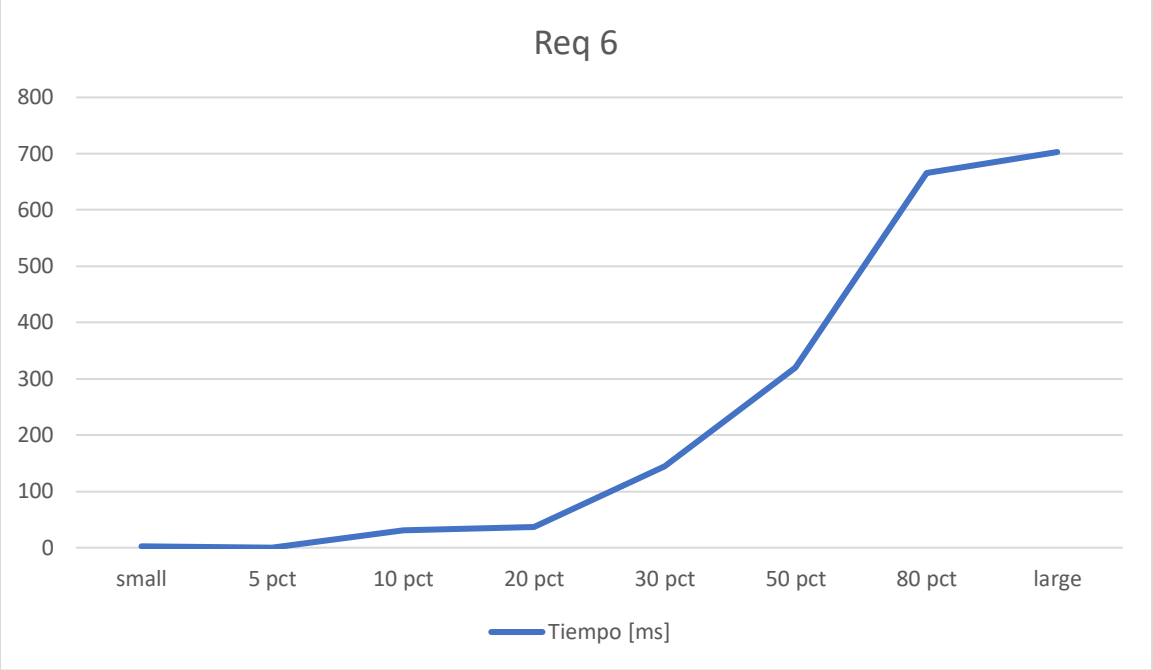
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Numero de eventos con radio de 3000.0 [km]: 26 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 7 Numero de eventos entre fechas: 7	2.11
5 pct	No existe	No existe
10 pct	Numero de eventos con radio de 3000.0 [km]: 236 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 127 Numero de eventos entre fechas: 127	30.97
20 pct	Numero de eventos con radio de 3000.0 [km]: 486	36.44

	Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 9 Numero de eventos entre fechas: 9	
30 pct	Numero de eventos con radio de 3000.0 [km]: 719 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 523 Numero de eventos entre fechas: 523	144.69
50 pct	Numero de eventos con radio de 3000.0 [km]: 1199 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 949 Numero de eventos entre fechas: 949	320.28
80 pct	Numero de eventos con radio de 3000.0 [km]: 1919 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 1497 Numero de eventos entre fechas: 1497	665.68
large	Numero de eventos con radio de 3000.0 [km]: 2423 Numero de eventos maximos con radio de 3000.0 [km]: 10 Numero de eventos en el año 2022: 593 Numero de eventos entre fechas: 593	702.97

Graficas



Análisis

El código dado proporciona una solución efectiva al requerimiento 6 de una aplicación que consulta un conjunto de datos de terremotos. La función principal, req_6(), se encarga de calcular el número de eventos sísmicos que ocurrieron en un radio dado de una ubicación dada en un año dado, y los datos de los eventos sísmicos que ocurrieron en un rango de fechas dado.

La función req_6() funciona de la siguiente manera:

Obtiene los datos de los terremotos del año dado.

Para cada evento sísmico:

Calcula la distancia entre el evento y la ubicación dada.

Si la distancia es menor o igual al radio dado, agrega el evento a una lista.

Encuentra el evento sísmico más significativo del año.

Ordena los eventos sísmicos por fecha.

Divide los eventos sísmicos en dos listas: una lista de los eventos que ocurrieron antes del evento más significativo y una lista de los eventos que ocurrieron después.

Para cada lista:

Filtra los datos de los eventos sísmicos para obtener solo los datos necesarios.

Agrega los datos de los eventos sísmicos a una lista.

Devuelve los siguientes datos:

El número de eventos sísmicos que ocurrieron en el radio dado.

El número de eventos sísmicos más significativos que ocurrieron en ese año.

El número de eventos sísmicos que ocurrieron en el rango de fechas dado.

Los datos de los eventos sísmicos que ocurrieron en el rango de fechas dado.

El código está bien escrito y es fácil de entender. Utiliza una estructura de control clara y utiliza funciones para organizar el código.

Requerimiento <<7>>

Descripción

```python
def req_7(data_structs, anio, title, propiedad, div):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    start = controller.get_time()
    datos = data_structs['anio']
    dato_interes = me.getValue(om.get(datos, anio))
    lst_lugar = lt.newList()

    titulos = ['time', 'lat', 'long', 'title', 'code', propiedad]

    for dato in lt.iterator(dato_interes['lsttemblores']):
        place = dato['title'].lower()
        if title in place:
            lt.addLast(lst_lugar, dato)

    if propiedad == 'mag':
        arblo_usr = om.newMap('RBT',
                              comparemag)

        for dato in lt.iterator(lst_lugar):
            updateDate_fecha(arblo_usr, dato)
        mini = om.minKey(arblo_usr)
        maxi = om.maxKey(arblo_usr)
        lista_usr = om.values(arblo_usr, mini, maxi)
        list3fyl = get3(lista_usr)
        dicc = {'keys':[], 'values':[]}
        for dato in lt.iterator(lista_usr):
            for dato2 in lt.iterator(dato['lsttemblores']):
                dicc['keys'].append(dato2['mag'])
        for dato in lt.iterator(list3fyl):
            value = filtrar(dato['lsttemblores'], titulos)
            for dato2 in lt.iterator(value):
                dicc['values'].append(dato2)
```

```
if propiedad == 'sig':
    arblo_usr = om.newMap('RBT',
                          comparesig)

    for dato in lt.iterator(lst_lugar):
        updateDate_fecha(arblo_usr, dato)
    mini = om.minKey(arblo_usr)
    maxi = om.maxKey(arblo_usr)
    lista_usr = om.values(arblo_usr, mini, maxi)
    list3fyl = get3(lista_usr)
    dicc = {'keys':[], 'values':[]}
    for dato in lt.iterator(lista_usr):
        for dato2 in lt.iterator(dato['lsttemblores']):
            dicc['keys'].append(dato2['sig'])
    for dato in lt.iterator(list3fyl):
        value = filtrar(dato['lsttemblores'], titulos)
        for dato2 in lt.iterator(value):
            dicc['values'].append(dato2)

if propiedad == 'depth':
    arblo_usr = om.newMap('RBT',
                          comparedepth)

    for dato in lt.iterator(lst_lugar):
        updateDate_fecha(arblo_usr, dato)
    mini = om.minKey(arblo_usr)
    maxi = om.maxKey(arblo_usr)
    lista_usr = om.values(arblo_usr, mini, maxi)
    list3fyl = get3(lista_usr)
    dicc = {'keys':[], 'values':[]}
    for dato in lt.iterator(lista_usr):
        for dato2 in lt.iterator(dato['lsttemblores']):
            dicc['keys'].append(dato2['depth'])
    for dato in lt.iterator(list3fyl):
        value = filtrar(dato['lsttemblores'], titulos)
        for dato2 in lt.iterator(value):
            dicc['values'].append(dato2)

#Creacion de axis, estas lo que hacen es ser la diviosion de las partes de la imagen, siendo ax1 la grafica y ax2 la tabla

fig, (ax1, ax2) = plt.subplots(2, 1, gridspec_kw={'height_ratios': [2, 1]}, figsize=(6, 6)) #Fig esta solo para poder obtener todos los valores
n, bins, titulitos = ax1.hist(dicc['keys'], bins=div, color='purple', ec='black')  #n esta solo para poder obtener todos los valores
ax1.set_xticks([])
ax1.set_title('Historigram of {} in {} in {}'.format(propiedad, title, anio))
ax1.set_ylabel('No. Events')
ax1.set_xlabel(propiedad)
```

```
    # Añadir etiquetas de rango debajo de cada barra
    for i in range(len(titulitos)):
        ax1.text(titulitos[i].get_x() + titulitos[i].get_width() / 2, -1.4, f'{bins[i]:.2f}-{bins[i + 1]:.2f}', ha='center', rotation=45, fontsize=8)

    for i, rect in enumerate(titulitos):
        altura = rect.get_height()
        ax1.annotate(f'{int(altura)}', xy=(rect.get_x() + rect.get_width() / 2, altura), xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
    # Segundo subplot para la tabla
    ax2.axis('off')

    # Crear la tabla
    tabla = plt.table(cellText=[[str(evento['time']), evento['lat'], evento['long'], evento['title'], evento['code'], str(evento[propiedad])] for evento in dicc['values']],
                      colLabels=["Time", "Latitude", "Longitude", "Title", "Code", "Magnitude"],
                      cellLoc='center',
                      loc='center')

    # Ajustar el formato y estilo de la tabla
    tabla.auto_set_font_size(False)
    tabla.set_fontsize(5)
    tabla.scale(1.2, 1.2)
    ax2.set_title('Events in {} in {}'.format(title, anio))
    plt.tight_layout()
    stop = controller.get_time()
    delta = controller.delta_time(start, stop)
    plt.show()
    return delta

def updateSig(mapa, dato):
    sig = int(dato['sig'])
    entry = om.get(mapa, sig)
    if entry == None:
        sigentry = newsigentry(dato)
        om.put(mapa,sig, sigentry)
    else:
        sigentry= me.getValue(entry)
    addSig(sigentry, dato)
    return mapa

def newsigentry(dato):
    entry = {'lsttemblores': None}
    entry['lsttemblores'] = lt.newList()
    return entry

def addSig(sigentry, dato):
    lst = sigentry['lsttemblores']
    lt.addLast(lst, dato)
    return sigentry
```

El requerimiento 7 tiene como finalidad mostrarle al usuario una gráfica de barras, con el número de barras que el usuario defina, junto a una tabla, con las características de, time, lat, long, title entre otras, donde se analizan los eventos ocurridos en un año especifico, con una localización dada por el usuario y teniendo en cuenta una propiedad como, magnitud, significancia o profundidad. El código se puede dividir en tres partes principales. La primera donde se toman los datos por año y se recorta su lista dependiendo del título seleccionado, posteriormente se pregunta cuál fue la propiedad escogida, y se raliza un ordenamiento teniendo en cuenta este. Por último, los datos de las llaves y sus valores se convierten a una lista de listas nativas de Python para utilizar mtplotlib y generar su grafica.

Entrada	Año de búsqueda, Lugar donde se desea consultar, propiedad, num de casillas
Salidas	Grafica que tinene en cuenta el número de casillas para generar ese número de columnas con la información que ha pasado todos los filtros mencionados anteriormente.
Implementado (Sí/No)	Si. Se realizó en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de variables	O(1)
Ciclo para realizar el primer filtro	O(n)
Condición propiedad	O(1)
Creación de árbol	O(1)
Ciclo para ordenar el árbol y agregarle valores	O(n)
Doble ciclo anidado para tratar el tipo de valor a tener en cuenta, propiedad	O(n^2)
Condición propiedad	O(1)
Ciclo para ordenar el árbol y agregarle valores	O(n)
Doble ciclo anidado para tratar el tipo de valor a tener en cuenta, propiedad	O(n^2)
Condición propiedad	O(1)
Ciclo para ordenar el árbol y agregarle valores	O(n)
Doble ciclo anidado para tratar el tipo de valor a tener en cuenta, propiedad	O(n^2)
Creación tabla, separación por partes	O(1)
Ciclo para el rango de titulos	O(n)
Ciclo para mostrar el num de elementos en cada columna	O(n)
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1. Además de utilizar como datos de entrada: 2020, Alaska, mag, 10

Procesadores	**AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz**
Memoria RAM	16 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	126.7151
5 pct	193.6154
10 pct	199.5855
20 pct	216.2166
30 pct	164.4766
50 pct	244.5392
80 pct	230.9224
large	276.1231

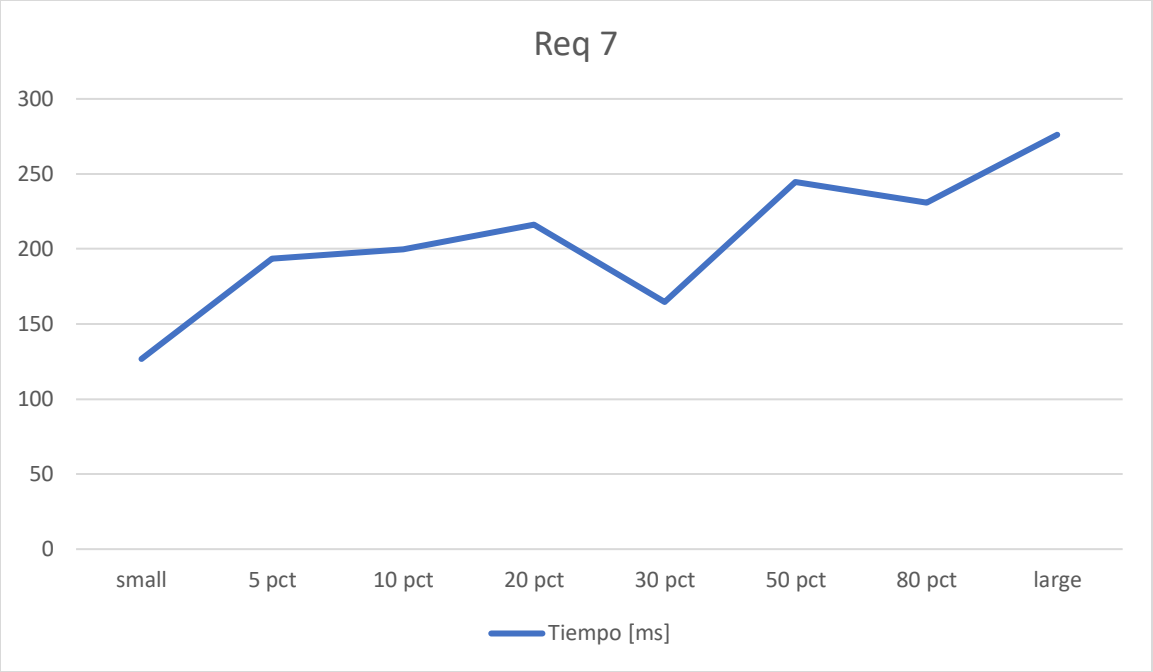
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	27 datos en la grafica de 10 casillas	126.7151
5 pct	165 datos en la grafica de 10 casillas	193.6154
10 pct	480 datos en la grafica de 10 casillas	199.5855
20 pct	1084 daton en la grafica de 10 casillas	216.2166
30 pct	1982 datos en la grafica de 10 casillas	164.4766
50 pct	3468 datos en la grafica de 10 casillas	244.5392
80 pct	6328 datos en la grafica de 10 casillas	230.9224
large	6483 datos en la grafica de 10 casillas	276.1231

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La grafica muestra un comportamiento lineal el cual no es constante en crecimiento y tiene algunos puntos máximos locales. Pese a que en el análisis de complejidad se mencionó que la gráfica debía expresar un valor de n cuadrado generando una curva pronunciada hacia arriba, se puede apreciar que este no es el caso, esto debido a que algunos de los ciclos que se presentan dentro del algoritmo realmente tienen la función de buscar un apartado de información precisa y no de recorrer algun arreglo realmente. Por lo que la gráfica demuestra la baja complejidad del algoritmo aun con una cantidad de datos elevados, donde se puede ver un buen manejo de la cantidad de datos con relación a su tiempo de ejecución.