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

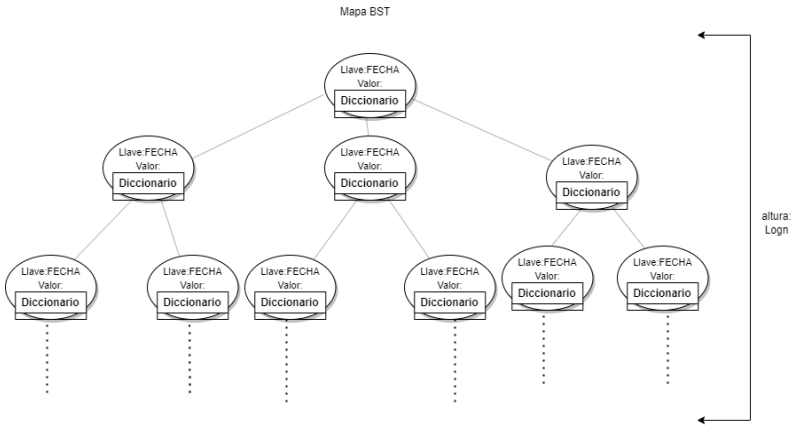
ANÁLISIS DEL RETO 3

Mateo Alejandro Quiroga, 202224583, m.quirogac@uniandes.edu.co

Hernando José Díaz Beltrán, 202220213, h.diazb@uniandes.edu.co

Felipe González, 202111623, jf.gonzalezp1@uniandes.edu.co

Carga de datos



Descripción

Para la carga de datos se implementó un código el cual creará un mapa BTS el cual cada llave son las fechas en el formato %Y-%m-%dT%H:%M:%S.%fz, donde cada valor es el diccionario que contiene la información del sismo en un diccionario.

Entrada	Archivo csv
Salidas	Estructura de datos con la información del csv tabulada
Implementado (Sí/No)	Si, implementado por Hernando José Díaz y Felipe González

```
def new_data_structs():
    """
    Inicializa las estructuras de datos del modelo. Las crea de
    manera vacía para posteriormente almacenar la información.
    """
    sismo = { "seg" : None

    }

    sismo["seg"] = om.newMap(omaptype="RBT",cmpfunction=compareDates
                             )
    return sismo
```

```
def add_seg(sismo,data):
    fechasis = data["time"]

    if data.get("cdi")== "":
        cdi = "Unavailable"
    else:
        cdi = data.get('cdi')


    if data.get("mmi")== "":
        mmi = "Unavailable"
    else:
        mmi = data.get('mmi')

    if data.get("nst")== "":
        nst = 1
    else:
        nst = data.get('nst')

    if data.get("gap")== "":
        gap = 0.00
    else:
        gap = data.get('gap')

    dicc = {}
    dicc["time"] = data.get('time')
    dicc["mag"] = data.get('mag')
    dicc["lat"] = data.get('lat')
    dicc["long"] = data.get('long')
    dicc["depth"] = data.get('depth')
    dicc["sig"] = data.get('sig')
    dicc["gap"] = gap
    dicc["nst"] = nst
    dicc["title"] = data.get('title')
```

```
dicc["cdi"] = cdi
dicc["mmi"] = mmi
dicc["magType"] = data.get('magType')

dicc["type"] = data.get('type')

dicc["code"] = data.get('code')


fecha = datetime.datetime.strptime(fechasis,"%Y-%m-%dT%H:%M:%S.%fz")

om.put(sismo,fecha,dicc)
return sismo
```

```python
def load_data(control, tamaño):
    """
    Carga los datos del reto
    """
    tamaño = int(tamaño)
    tm = 'small'
    if tamaño == 2:
        tm = "5pct"
    if tamaño == 3:
        tm = "10pct"
    if tamaño == 4:
        tm = "20pct"
    if tamaño == 5:
        tm = "30pct"
    if tamaño == 6:
        tm = "50pct"
    if tamaño == 7:
        tm = "80pct"
    if tamaño == 8:
        tm = "large"

    control["model"] = model.new_data_structs()
    sismo = control["model"]
    load_date(sismo,tm)
    primero_ultimos,size = model.primeros_ultimos_5(sismo["seg"])

    return primero_ultimos,size
```

```python
def load_date(sismo,tm):

    temblorefile = cf.data_dir + 'earthquakes/temblores-utf8-' + tm + '.csv'
    input_file = csv.DictReader(open(temblorefile, encoding='utf-8'))

    for data in input_file:

        model.add_seg(sismo["seg"],data)


    return sismo
```

Análisis de complejidad

Pasos	Complejidad
Comparar tamaño elegido por el usuario de la carga de datos (if)	O(1)
crear nueva estructura (newmap)	O(n)
recorrer el archivo csv	O(n)
crear el diccionario con los datos necesarios para el sismo	O(1)
datetime de la fecha	O(1)
añadir el dato al mapa bts (put)	O(log(n))
TOTAL	***O(nlog(n))***

Pruebas Realizadas

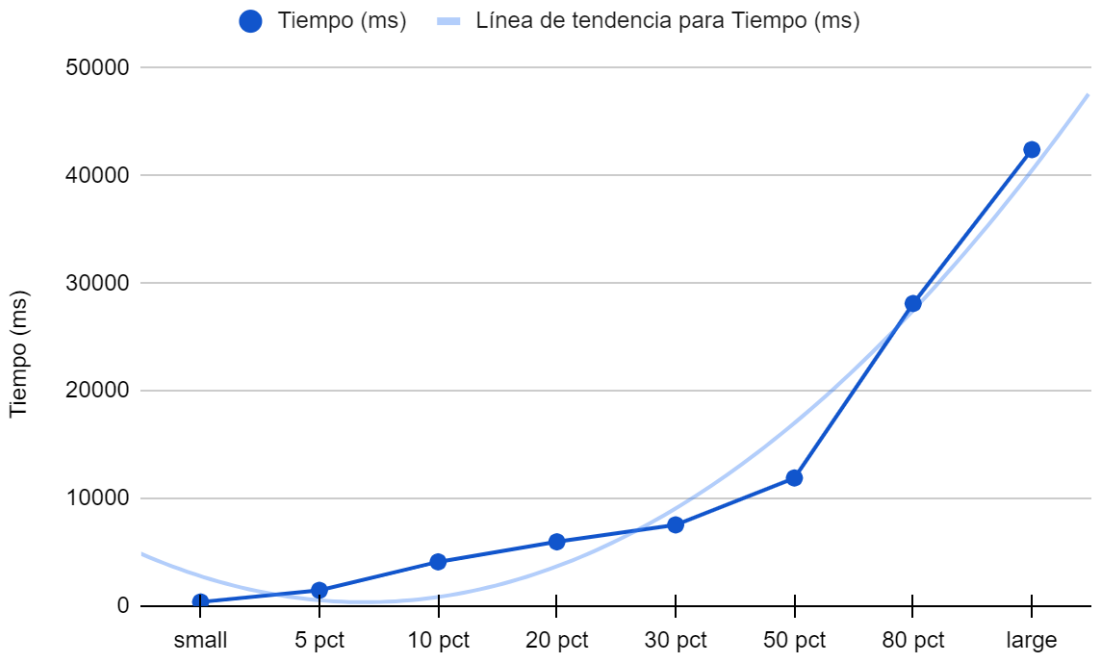
Procesadores	Intel(R) Core(TM) i5-10300H @2.50 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	Tiempo (s)
small	383.11
5 pct	1466.26
10 pct	4111.7
20 pct	5975.5
30 pct	7548.36
50 pct	11908.14
80 pct	28125.23
large	42425.7

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Dato 1	383.11
5 ptc	Dato 2	1466.26
10 ptc	Dato 3	4111.7
20 ptc	Dato 4	5975.5
30 ptc	Dato 5	7548.36
50 ptc	Dato 6	11908.14
80 ptc	Dato 7	28125.23
large	Dato 8	42425.7

Gráficas



Análisis

Como se puede observar el comportamiento se comporta como una gráfica n log n, a medida que va aumentando la carga se demora más, aun así, esto es una mejor con respecto cargas de datos, ya que esta carga está organizada a medida que los datos se ingresan en el árbol. Asimismo, para implementar este código fue más sencillo y menos complejo de planificar.

Requerimiento <<1>>

Descripción

```python
def req_1(sismos,inicial,final):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    inicial = datetime.datetime.strptime(inicial,"%Y-%m-%dT%H:%M")
    final = datetime.datetime.strptime(final,"%Y-%m-%dT%H:%M")

    valores = om.valueSet(sismos)
    bst = om.newMap(omaptype="RBT",cmpfunction=compareDates)
    for valor in lt.iterator(valores):
        copia = valor.copy()
        lista = lt.newList("ARRAY_LIST")

        fecha = datetime.datetime.strptime(valor["time"][:-11],"%Y-%m-%dT%H:%M")
        entry = om.get(bst,fecha)
        if entry:
            lista = me.getValue(entry)
            lt.removeLast(lista)
        if valor != None:
            lt.addLast(lista,copia)
        copia.pop("time")
        lt.addLast(lista,fecha)
        om.put(bst,fecha,lista)

    return om.values(bst,inicial,final)
```

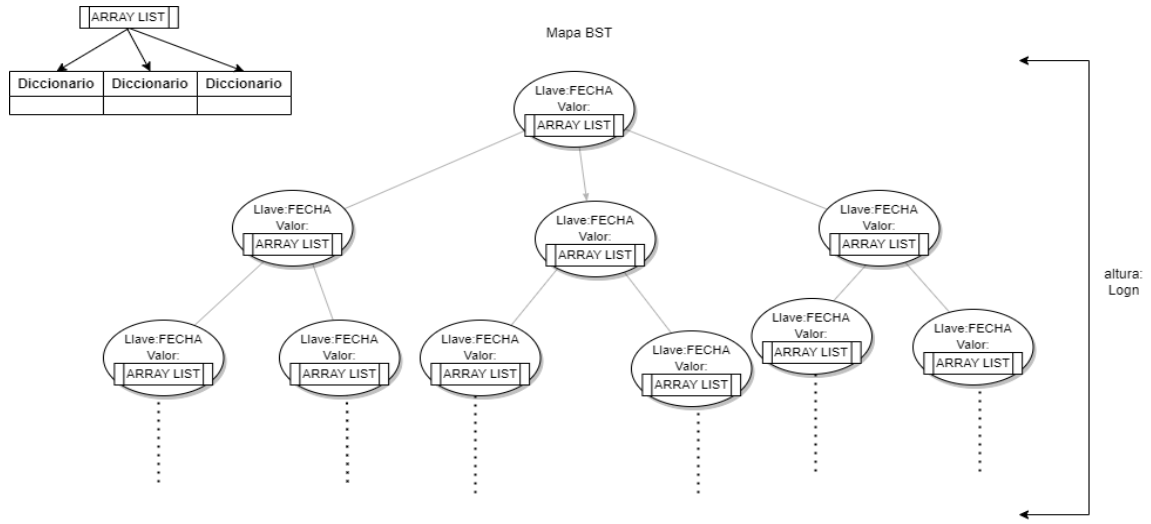
```python
def tabulate_req_1(bst):

    lista = lt.newList("ARRAY_LIST")

    contador = 0
    for cada_lista in lt.iterator(bst):
        lista2 = lt.newList("ARRAY_LIST")
        fecha = lt.lastElement(cada_lista)
        lt.removeLast(cada_lista)
        size = lt.size(cada_lista)
        contador = contador + size
        lt.addLast(lista2,fecha)
        lt.addLast(lista2,size)

        tabla = tabulate(lt.iterator(cada_lista),headers = "keys",tablefmt="grid")

        lt.addLast(lista2, tabla)
        lt.addLast(lista,lista2)

    return lt.size(bst),contador, lista
```

Este primer requerimiento retorna los eventos ocurridos entre un rango de fechas entregado por el usuario. Comienza obteniendo los datos de la estructura de datos principal e itera en

ellos para ir agregando los eventos sísmicos a una lista, modificando su *time* para que sea mostrado según los minutos, como lo solicita el requerimiento. Finalmente, aprovechando la característica de un árbol RBT (ya están organizados de manera ascendente), seleccionamos únicamente los eventos en las fechas solicitadas.

Entradas	Estructura de datos del modelo, fecha inicial y final a consultar.
Salidas	Tabla organizada con los eventos sísmicos en el intervalo de fechas entregado.
Implementado (Sí/No)	Si, implementado por Mateo Quiroga y Hernando Díaz.

Diagrama



Análisis de complejidad

Pasos	Complejidad
Convertir una fecha al formato necesitado (strptime)	O(1)
Obtener los valores de un mapa (valueSet)	O(n)
Crear un árbol RBT *bst* (newMap)	O(n)
Recorrer los eventos sísmicos de *sismo* (iterator)	O(n)
Crear un arreglo *lista* (newList)	O(1)
Convertir una fecha al formato necesitado (strptime)	O(1)

Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un elemento a la última posición de un arreglo (addLast)	O(1)
Agregar un elemento a un árbol (put)	O(log (n))
Agrega a una lista los valores de un rango de llaves entregado (values)	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

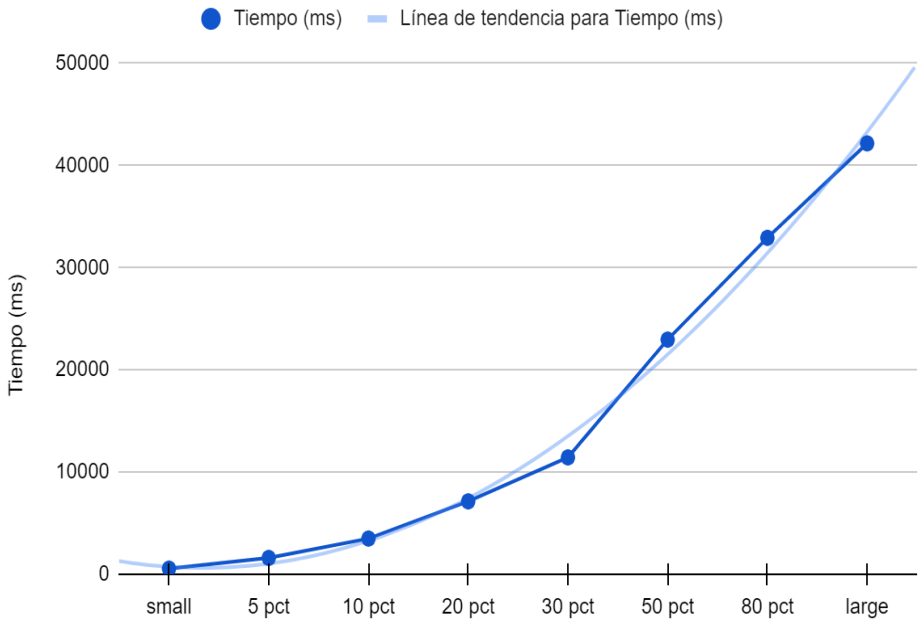
Procesadores	Intel(R) Core(TM) i5-10300H @2.50 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
small	550.24
5 pct	1596.53
10 pct	3492.6
20 pct	7130.72
30 pct	11435.16
50 pct	22963.82
80 pct	32925.32
large	42170.28

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Dato 1	550.24
5 ptc	Dato 2	1596.53
10 ptc	Dato 3	3492.6
20 ptc	Dato 4	7130.72
30 ptc	Dato 5	11435.16
50 ptc	Dato 6	22963.82
80 ptc	Dato 7	32925.32
large	Dato 8	42170.28

Gráficas



Análisis

En este primer requerimiento vemos una notoria demora en la impresión de datos, esto se debe a que en un inicio traspasamos toda la información de nuestra estructura general de datos a otro mapa, esto con el fin de pasarla con el formato de fecha en minutos como lo solicita el requerimiento, posteriormente obtenemos el filtrado de los datos en el rango de fechas pertinente.

El requerimiento queda de complejidad n gracias a la organización anteriormente mencionada, donde tenemos que pasar por cada evento sísmico.

Requerimiento <<2>>

Descripción

```
def req_2(sismo):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    valores = om.valueSet(sismo)
    mag = om.newMap(omaptype="RBT",cmpfunction=compareMag)
    contador = 0
    for valor in lt.iterator(valores):
        magnitud = float(valor["mag"])
        fecha = valor["time"]

        llavevalor = om.get(mag,magnitud)
        if llavevalor is None:
            mapa = om.newMap(omaptype="RBT",cmpfunction=compareDates2)

        else:
            mapa = me.getValue(llavevalor)
            contador = contador + 1

        om.put(mapa,fecha,valor)
        om.put(mag,magnitud,mapa)

    return contador,mag
```

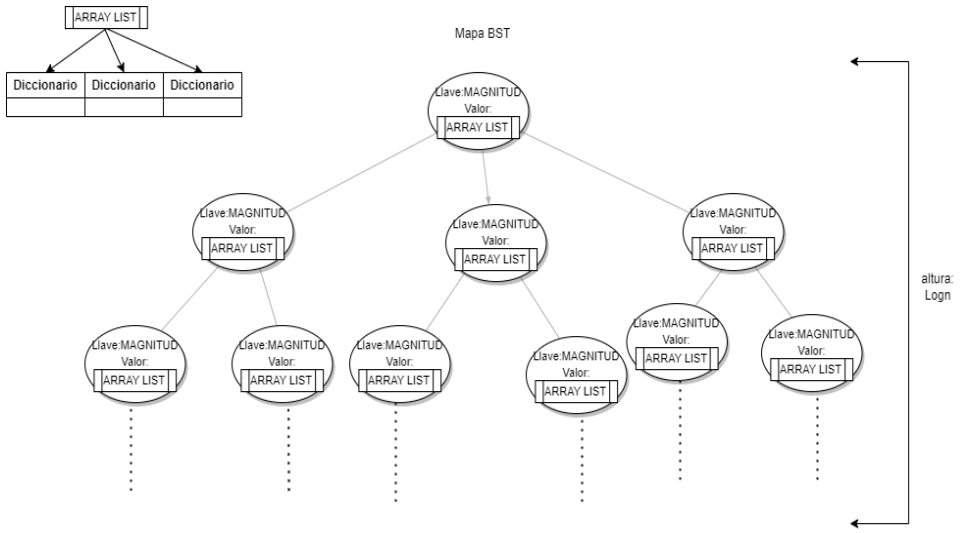
```python
def tabulate_req_2(bts,inf,sup):
    inf = om.ceiling(bts,inf)
    sup = om.floor(bts,sup)

    filtrado = om.values(bts,inf,sup)
    contador = 0
    listas = lt.newList("ARRAY_LIST")
    for cada_mapa in lt.iterator(filtrado):
        lista = lt.newList("ARRAY_LIST")
        vals = om.valueSet(cada_mapa)
        events = lt.size(vals)
        vals = ultimos_primeros(vals)
        contador = lt.size(vals) + contador

        tabla = tabulate(lt.iterator(vals),headers = "keys", tablefmt="grid")

        magnitud = lt.getElement(vals,1)["mag"]


        lt.addLast(lista,magnitud)
        lt.addLast(lista,events)
        lt.addLast(lista,tabla)
        lt.addLast(listas,lista)
    return lt.size(listas),filtrado,listas
```

```python
def req_2(control,inf,sup):
    """
    Retorna el resultado del requerimiento 2
    """
    # TODO: Modificar el requerimiento 2
    sismo = control["model"]
    consult_size,datos = model.req_2(sismo["seg"])
    total_mag,total_events, lista_final = model.tabulate_req_2(datos,inf,sup)
    lista_final = model.ultimos_primeros(lista_final)

    print(total_mag, total_events,consult_size)
    input()
    return total_mag, total_events,consult_size,lista_final
```

Para este requerimiento se recorrió todos los sismos para encontrar los sismos con magnitud dentro del rango dado, se creó un mapa donde las llaves eran la magnitud y luego estas se organizaban los sismos (diccionarios) por orden cronológico dentro de una lista. En este requerimiento se usó las funciones put, ceiling, floor, values de la libreria orderedmaps.

Entrada	los límites inferior y superior de la magnitud
Salidas	El número total de eventos sísmicos ocurridos entre las magnitudes indicadas
Implementado (Sí/No)	Si, implementado por Mateo Quiroga y Hernando Díaz.

Diagrama



Análisis de complejidad

Pasos	Complejidad
Obtener los valores del mapa de sismos (valueSet)	O(n)
Crear un árbol RBT "*bst*" (newMap)	O(n)
Recorrer los eventos sísmicos de *sismo* (for)	O(n)
encontrar el tiempo y la magnitud del diccionario	O(1)
Buscar si la llave valor de la magnitud existe	O(1)
crear base de datos para guardar	O(1)
comparar si existe la llave	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
contar que un elemento fue añadido	O(1)
Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
TOTAL	***O(n log(n))***

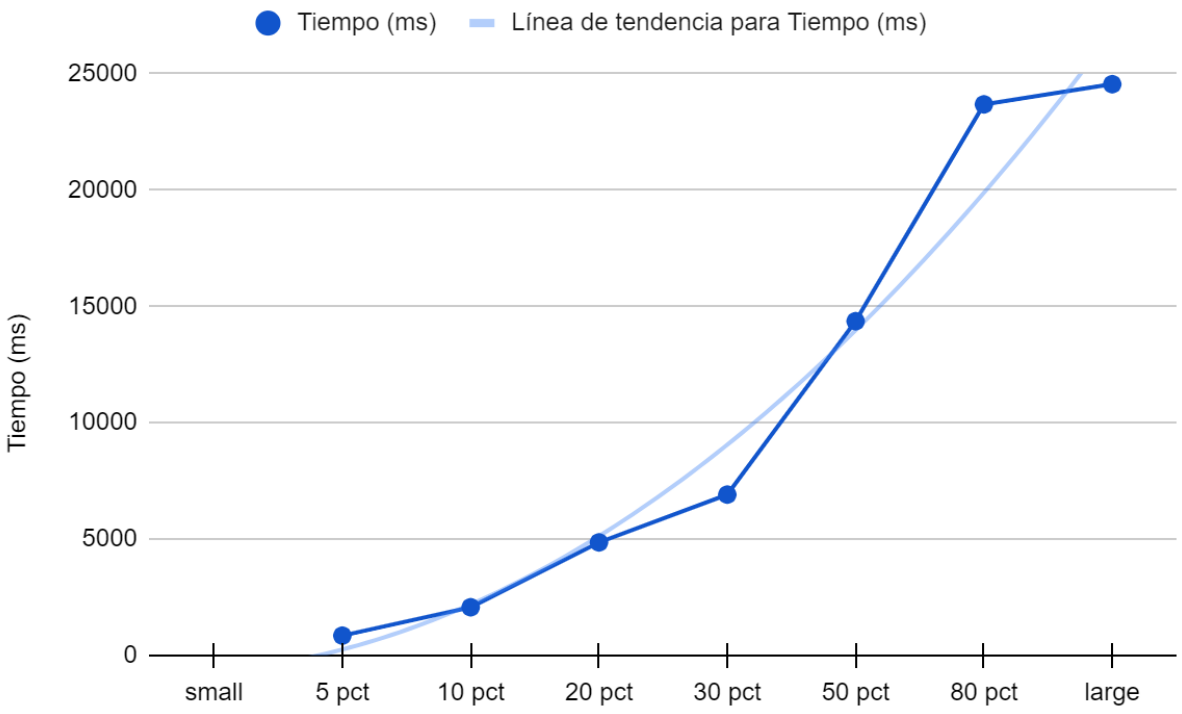
Pruebas Realizadas

Entrada	Tiempo (s)
small	175.44
5 ptc	851.15
10 ptc	2071.35
20 ptc	4858.91
30 ptc	6913.58
50 ptc	14365.33
80 ptc	23685.38
large	24552.22

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Dato 1	175,44
5 ptc	Dato 2	851.15
10 ptc	Dato 3	2071.35
20 ptc	Dato 4	4858.91
30 ptc	Dato 5	6913.58
50 ptc	Dato 6	14365.33
80 ptc	Dato 7	24552.22
large	Dato 8	23685.38

Graficas



Análisis

Como podemos observar, ya que estamos usando un gran espacio para guardar los datos, esto se compensa en el tiempo que se tarda encargar el requerimiento, esta complejidad cumple ser O(n log(n)) siendo una complejidad menor a buscar elementos dentro de listas en retos pasados, considerando a cantidad de datos relacionada a esta búsqueda es satisfactorio pensar que el tiempo a sido reducido considerablemente.

Requerimiento <<3>>

Descripción

```python
def req_3(sismo, mag, depth):
    """
    Función que soluciona el requerimiento 3
    """
    values = om.valueSet(sismo)
    bst = om.newMap(omaptype = "RBT", cmpfunction = compareDates)
    cont = 0

    for value in lt.iterator(values):
        if float(value["mag"]) > mag and value != None and float(value["depth"]) >= depth:
            cont += 1
            copy = value.copy()

            lists = lt.newList("ARRAY_LIST")
            date = datetime.datetime.strptime(value["time"][:-11],"%Y-%m-%dT%H:%M")
            entry = om.get(bst, date)

            if entry:
                lists = me.getValue(entry)
                lt.removeLast(lists)

            if value != None:
                lt.addLast(lists, copy)

            copy.pop("time")
            lt.addLast(lists, date)
            om.put(bst, date, lists)

    return(om.size(bst), cont, tabulate_req_3(bst, om.size(bst)))
```

```python
def tabulate_req_3(bst, num):
    ini = om.select(bst, num - 19)
    ult = om.select(bst, num - 1)
    lista_de_listas = om.values(bst, ini, ult)

    lista = lt.newList("ARRAY_LIST")

    for cada_lista in lt.iterator(lista_de_listas):
        lista2 = lt.newList("ARRAY_LIST")
        fecha = lt.lastElement(cada_lista)
        lt.removeLast(cada_lista)
        size = lt.size(cada_lista)

        lt.addLast(lista2,fecha)
        lt.addLast(lista2,size)

        tabla = tabulate(lt.iterator(cada_lista), headers = "keys", tablefmt = "grid")

        lt.addLast(lista2, tabla)
        lt.addLast(lista, lista2)

    return lista
```

Entrada	La magnitud mínima del evento y la profundidad máxima del evento

```python
def req_3(control, mag, depth):
    """
    Retorna el resultado del requerimiento 3
    """
    start= get_time()
    sismo = control["model"]
    mag = float(mag)
    depth = float(depth)

    total_dates, total_events,datos = model.req_4(sismo["seg"], mag, depth)
    datos = model.ultimos_primeros(datos)

    end = get_time()
    delta = delta_time(start, end)

    return total_dates, total_events, delta
```

Salidas	El número total de eventos sísmicos registrados dentro de los límites de magnitud y profundidad indicados.
Implementado (Sí/No)	Si, implementado por Felipe González

Análisis de complejidad

Pasos	**Complejidad**
Obtener los valores del mapa (valueSet)	O(n)
Crear el árbol RBT "*bst*" (newMap)	O(n)
Recorrer los eventos en *sismo* (for)	O(n)
Crear un array *lista* (newList)	O(1)
Convertir una fecha al formato %Y-%m-%dT%H:%M (strptime)	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Comparar si existe	
Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un diccionario a la última posición de un arreglo (addLast)	O(1)
Eliminar un elemento del diccionario (pop)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
TOTAL	***O(n log(n))***

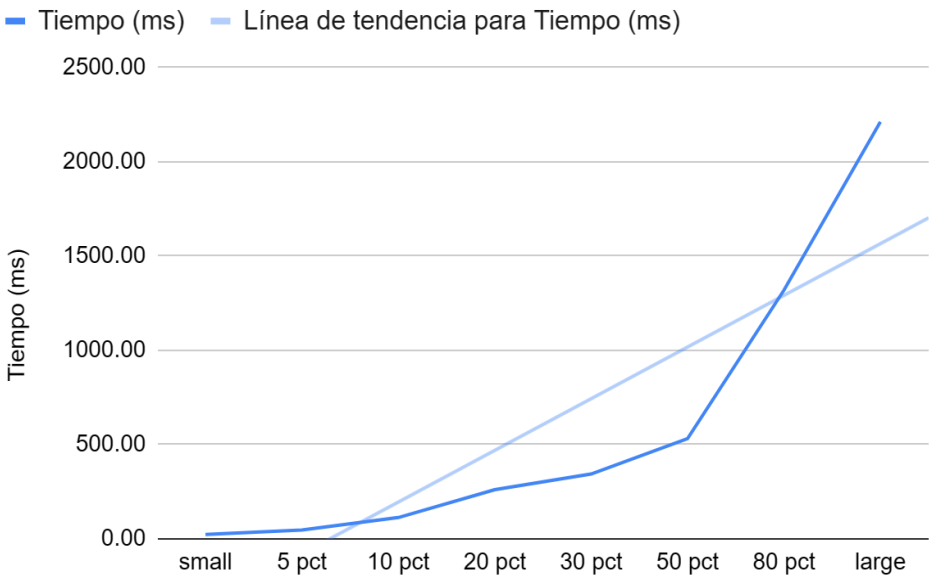
Pruebas Realizadas

Procesador	AMD Ryzen 5
Memoria Ram	8GB
sistema operativo	Windows 11

Tablas de datos

Entrada	Tiempo (ms)
small	23.03
5 pct	46.53
10 pct	112.94
20 pct	260.44
30 pct	343.91
50 pct	531.46
80 pct	1319.25
large	2212.25

Graficas



Análisis

Requerimiento <<4>>

```python
def req_4(sismo,sig,gap):
    # TODO: Realizar el requerimiento 4
    valores = om.valueSet(sismo)
    bst = om.newMap(omaptype="RBT",cmpfunction=compareDates)
    contador = 0
    for valor in lt.iterator(valores):
        if int(valor["sig"])>=sig and float(valor["gap"])<=gap and valor != None:
            copia = valor.copy()
            contador =contador + 1
            lista = lt.newList("ARRAY_LIST")

            fecha = datetime.datetime.strptime(valor["time"][:-11],"%Y-%m-%dT%H:%M")
            entry = om.get(bst,fecha)
            if entry:
                lista = me.getValue(entry)
                lt.removeLast(lista)
            if valor != None:
                lt.addLast(lista,copia)
            copia.pop("time")
            lt.addLast(lista,fecha)
            om.put(bst,fecha,lista)
    num = om.size(bst)
    ini = om.select(bst,num-15)
    ult = om.select(bst,num-1)
    lista_de_listas = om.values(bst,ini,ult)
    total_dates = om.size(bst)
    total_events = contador
    return total_dates, total_events,lista_de_listas
```

```python
def tabulate_req_4(lista_de_listas):
    lista = lt.newList("ARRAY_LIST")
    for cada_lista in lt.iterator(lista_de_listas):
        lista2 = lt.newList("ARRAY_LIST")
        fecha = lt.lastElement(cada_lista)
        lt.removeLast(cada_lista)
        size = lt.size(cada_lista)

        lt.addLast(lista2,fecha)
        lt.addLast(lista2,size)

        tabla = tabulate(lt.iterator(cada_lista),headers = "keys",tablefmt="grid")

        lt.addLast(lista2, tabla)
        lt.addLast(lista,lista2)

    return lista
```

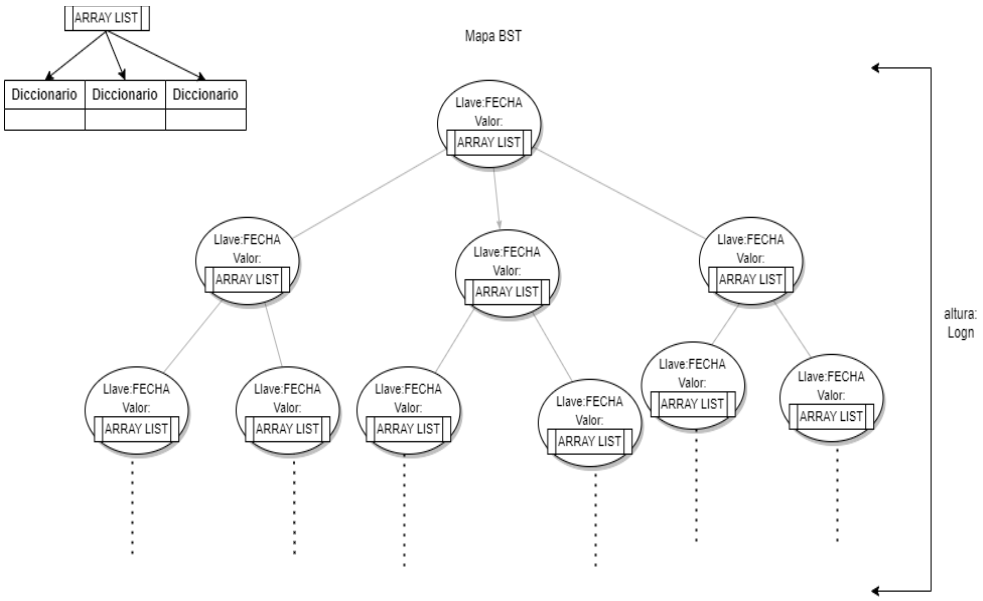
```python
def req_4(control,sig,gap):
    """
    Retorna el resultado del requerimiento 4
    """
    # TODO: Modificar el requerimiento 4
    inicial = get_time()

    sismo = control["model"]
    sig = int(sig)
    gap = float(gap)
    total_dates, total_events,datos = model.req_4(sismo["seg"],sig,gap)

    datos = model.tabulate_req_4(datos)
    datos = model.ultimos_primeros(datos)
    final = get_time()
    delta = delta_time(inicial,final)
    return total_dates, total_events,datos,delta
```

Descripción

Aquí en el requerimiento se recorrió todos los sismos que cumplieran con la significancia mínima y la distancia azimutal máxima. asi contar el número de llaves que contiene el mapa para saber el total de fechas, y contar cuantos elementos hay en cada llave para conocer el número de eventos. En este requerimiento se uso las funciones put,values set y values de la libreria orderedmaps

Entrada	distancia azimutal máxima y significancia mínima
Salidas	15 eventos sísmicos más recientes con los parámetros dados
Implementado (Sí/No)	Si, Hernando Diaz

Diagrama



Análisis de complejidad

Pasos	Complejidad
Obtener los valores del mapa de sismos (valueSet)	O(n)
Crear un árbol RBT "*bst*" (newMap)	O(n)
Recorrer los eventos sísmicos de *sismo* (for)	O(n)
Crear un arreglo *lista* (newList)	O(1)
Comparar si el sismo tiene una significancia mayor y una distancia azimutal menor a los parámetros.	O(1)
copiar el diccionario de sismo	O(1)
Convertir una fecha al formato %Y-%m-%dT%H:%M (strptime)	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Comparar si la llave existe.	O(1)

Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un diccionario a la última posición de un arreglo (addLast)	O(1)
Eliminar un elemento del diccionario (pop)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
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).

Muestra	Salida	Tiempo (ms)
small	Dato1	76.48
5 pct	Dato2	556.61
10 pct	Dato 3	785.47
20 pct	Dato4	1862.86
30 pct	Dato 5	2794.18
50 pct	Dato 6	4874.68
80 pct	Dato7	9070.53
large	Dato 8	10742.45

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

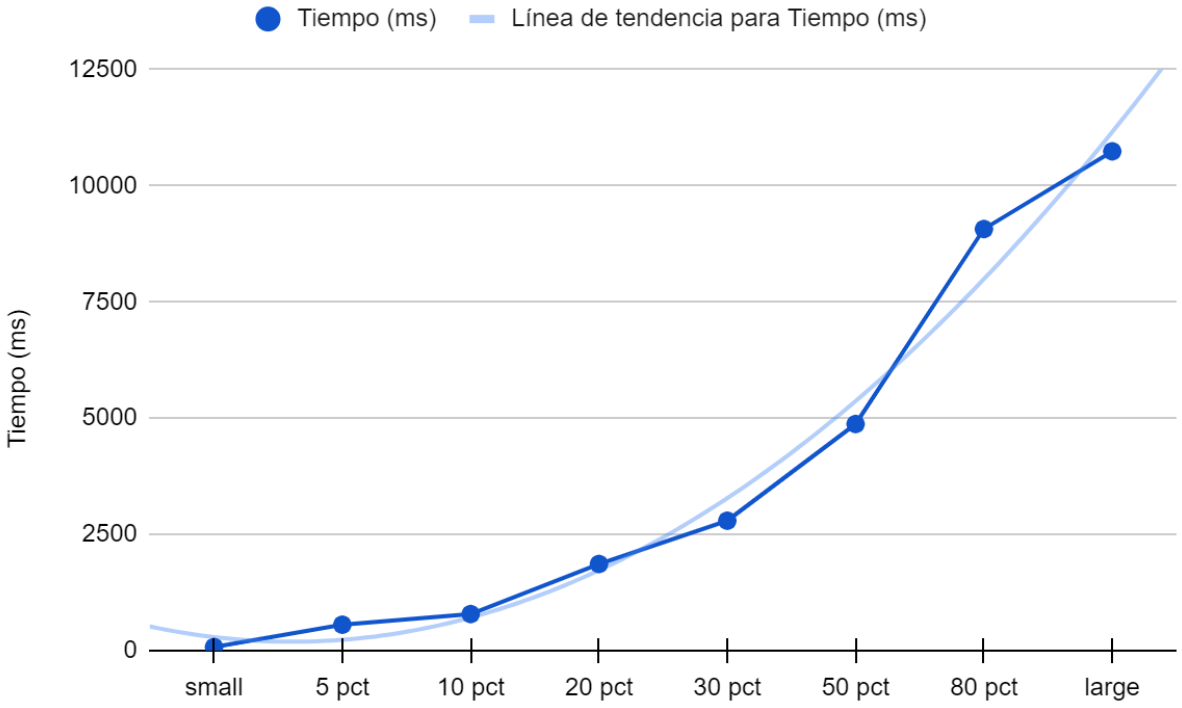
Procesadores	**Intel(R) HD Graphics 620**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	Tiempo (ms)
small	76.48
5 pct	556.61

10 pct	785.47
20 pct	1862.86
30 pct	2794.18
50 pct	4874.68
80 pct	9070.53
large	10742.45

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al ejecutar las pruebas del requerimiento, a medida que aumentaba el tamaño de la carga de datos, las pruebas tendían a tardar más tiempo, este crecimiento no es lineal, sino Nlogn, lo cual concuerda con el análisis de complejidad. Para este requerimiento se utilizó el mapa de la carga de datos para conseguir un mapa donde se guardaba cada diccionario por fecha y cada sismo (representado con un diccionario) se guardaba en una lista, en caso de haber más de un sismo en un periodo específico de tiempo.

Requerimiento <<5>>

Descripción

```python
def req_5(sismo, depth, nst):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    valores = om.valueSet(sismo)
    bst = om.newMap(omaptype="RBT",cmpfunction=compareDates)
    contador = 0
    for valor in lt.iterator(valores):

        if float(valor["depth"])>=depth and float(valor["nst"])>=nst and valor != None:
            copia = valor.copy()
            contador =contador + 1

            lista = lt.newList("ARRAY_LIST")

            fecha = datetime.datetime.strptime(valor["time"][:-11],"%Y-%m-%dT%H:%M")
            entry = om.get(bst,fecha)
            if entry:
                lista = me.getValue(entry)
                lt.removeLast(lista)
            if copia != None:
                lt.addLast(lista,copia)
            copia.pop("time")
            lt.addLast(lista,fecha)
            om.put(bst,fecha,lista)
    num = om.size(bst)
    data = tabulate_req_5(bst,num)
    total_dates = om.size(bst)
    total_events = contador
    return total_dates, total_events,data
```

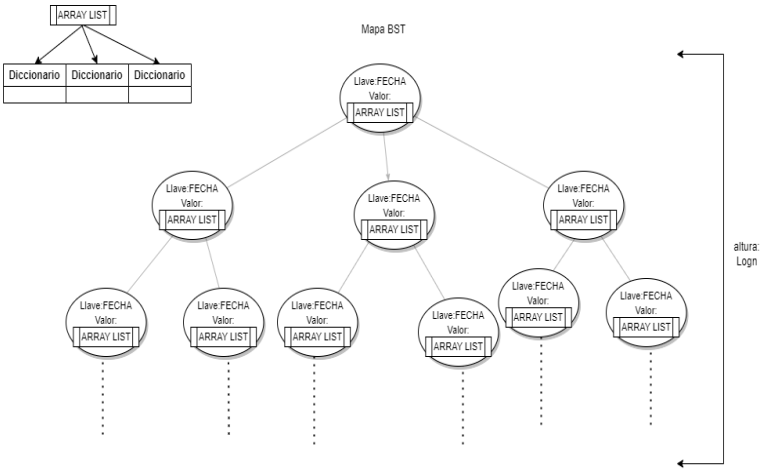
```python
def tabulate_req_5(bst,num):
    ini = om.select(bst,num-19)
    ult = om.select(bst,num-1)
    lista_de_listas = om.values(bst,ini,ult)

    lista = lt.newList("ARRAY_LIST")


    for cada_lista in lt.iterator(lista_de_listas):
        lista2 = lt.newList("ARRAY_LIST")
        fecha = lt.lastElement(cada_lista)
        lt.removeLast(cada_lista)
        size = lt.size(cada_lista)

        lt.addLast(lista2,fecha)
        lt.addLast(lista2,size)

        tabla = tabulate(lt.iterator(cada_lista),headers = "keys",tablefmt="grid")

        lt.addLast(lista2, tabla)
        lt.addLast(lista,lista2)

    return lista
```

El requerimiento 5 se encarga de retornar los 20 eventos sísmicos más recientes despreciando aquellos que no superen la profundidad dada y que no estén identificados por el mínimo de estaciones dado.

Inicialmente itera en la estructura de datos principal para filtrar la respectiva información, si la fecha con el evento sísmico no se encuentra en nuestro árbol de entrega, se agrega a este y se elimina la llave time para organizar la tabla *details*. Seguidamente, iteramos en nuestro árbol de datos y seleccionamos los 20 eventos más recientes para tomar y organizar la información *time, events* y *details* (este último anteriormente ya formado) que finalmente será retornada en nuestra tabla final.

Entrada	Estructura de datos del modelo (sismo), profundidad mínima del evento (deth) y el número mínimo de estaciones que detectan el evento (nst).
Salidas	Tabla organizada con los datos solicitados del jugador. Si no existen los datos retorna "0" en cada ítem solicitado.
Implementado (Sí/No)	Si, implementado por Mateo Quiroga

Diagrama



Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Obtener los valores de un mapa (valueSet)	O(n)
Crear un árbol RBT *bst* (newMap)	O(n)

Recorrer los eventos sísmicos de *sismo* (iterator)	O(n)
Crear un arreglo *lista* (newList)	O(1)
Convertir una fecha al formato necesitado (strptime)	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un elemento a la última posición de un arreglo (addLast)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
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).

Procesadores	Intel(R) Core(TM) i5-10300H @2.50 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Entrada	**Tiempo (s)**
small	49.46
5 pct	196.51
10 pct	427.47
20 pct	888.09
30 pct	1518.66
50 pct	2770.13
80 pct	4769.59
large	5814.15

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato 1	49.46
5 ptc	Dato 2	196.51
10 ptc	Dato 3	427.47
20 ptc	Dato 4	888.09
30 ptc	Dato 5	1518.66
50 ptc	Dato 6	2770.13
80 ptc	Dato 7	4769.59
large	Dato 8	5814.15

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Podemos ver que, aunque no existen ciclos dentro de ciclos ni funciones de ordenamiento complejas, la complejidad temporal del requerimiento escala a O(n log(n)) gracias a la función de orden característica de los árboles rojo-negro *RBT,* que se repite n veces. Dentro del requerimiento usamos funciones instantáneas o de complejidad O(1) dadas por la implementación del tipo de dato abstracto Maps, con *getValue* y *get* saber las llaves facilita la navegación por la estructura de datos reduciendo tiempos de búsqueda y acortando la

iteración de datos. Algunas funciones de módulos anteriores, como aquellas relacionadas con Array Lists (*newList, addLast, removeLast*), siguen estando presentes para la entrega final y tabulación de los datos solicitados. Para obtener el tamaño de nuestros mapas (size) y los valores de un árbol (valueSet) usamos una complejidad n.

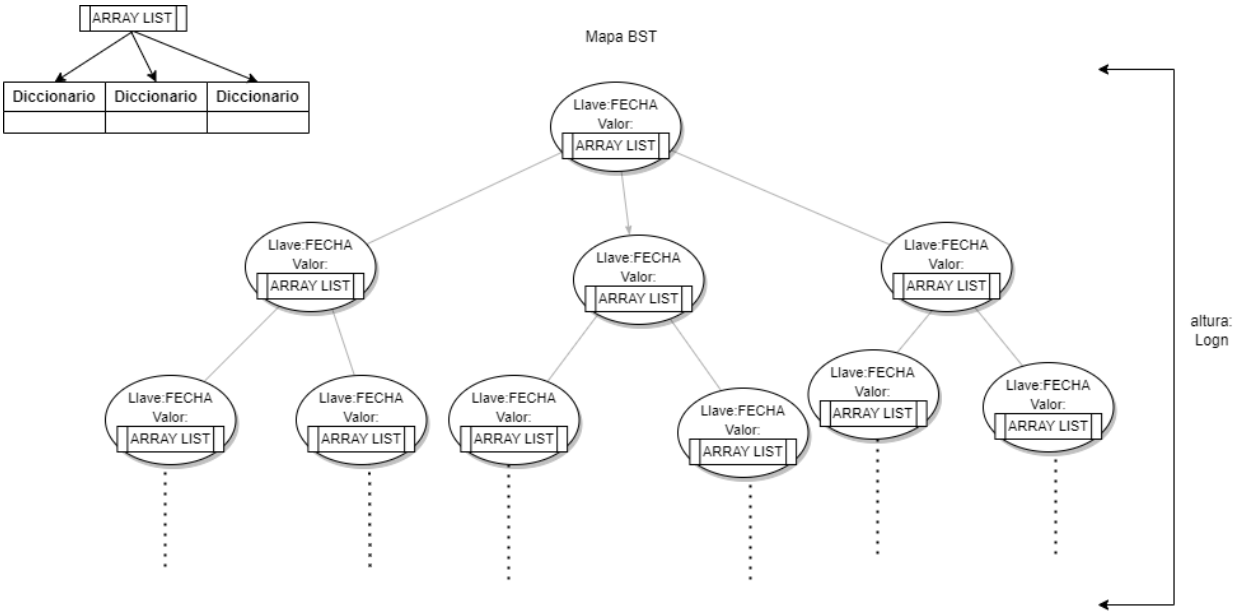
Requerimiento <<6>>

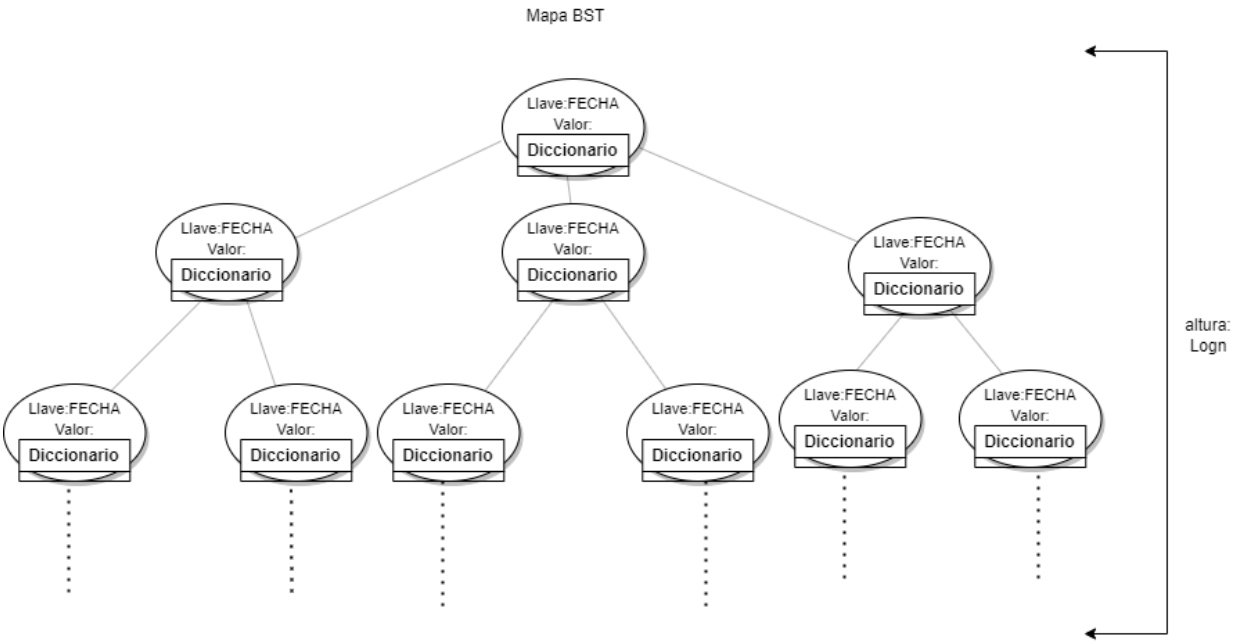
Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Diagrama





Entrada	El año relevante, latitud, longitud de referencia, radio del área, número de N eventos más cercanos
Salidas	El evento más significativo y los N eventos más próximos cronológicamente alrededor de un punto.
Implementado (Sí/No)	Si, implementado por Mateo Quiroga y Hernando Díaz

Análisis de complejidad

Pasos	Complejidad
Crear un árbol RBT "*bst*" (newMap)	O(n)
Encontrar un rango para el año escogido en formato %Y-%m-%dT%H:%M:%S.%fz	O(1)
Obtener el min y max llave (ceiling y floor)	O(n)
Obtener los valores del mapa de sismos (value)	O(n)
Recorrer los eventos sísmicos de los valores de el arbol *filtrado*(for)	O(n)
Calcular la distancia con la formula de haversine	O(1)
Comparar si la distancia es menor a el radio dado	O(1)
Crear un arreglo *lista* (newList)	O(1)
copiar el diccionario de filtrado	O(1)
Convertir una fecha al formato %Y-%m-%dT%H:%M (strptime)	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Comparar si la llave existe.	O(1)
Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un diccionario a la última posición de un arreglo (addLast)	O(1)
Eliminar un elemento del diccionario (pop)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
comparar el valor de significancia si es mayor al anterior dato iterado	O(1)
Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
TOTAL	***O(n log(n))***

Pruebas Realizadas

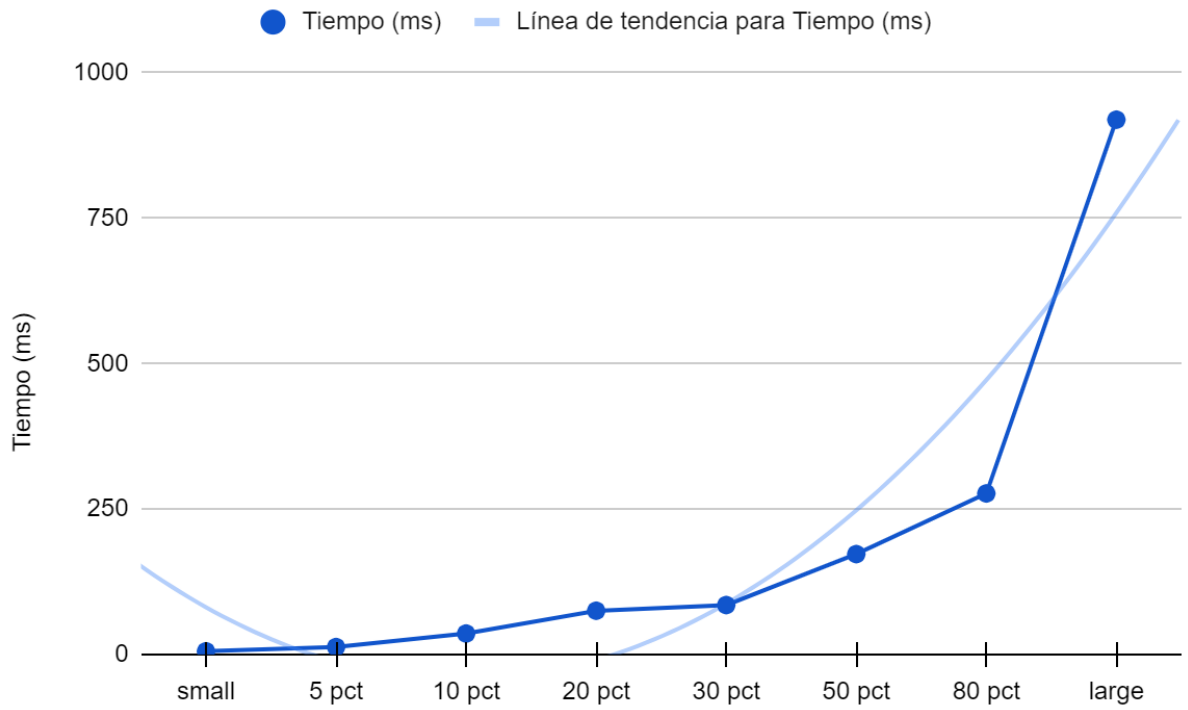
Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	Tiempo (s)
small	5.42
5 ptc	12.7
10 ptc	35.95
20 ptc	75.16
30 ptc	84.9
50 ptc	172.51
80 ptc	276.81
large	919.38

Tablas de datos

Muestra	Salida	Tiempo (ms)
small	Dato 1	5.42
5 ptc	Dato 2	12.7
10 ptc	Dato 3	35.95
20 ptc	Dato 4	75.16
30 ptc	Dato 5	84.9
50 ptc	Dato 6	172.51
80 ptc	Dato 7	276.81
large	Dato 8	919.38

Graficas



Análisis

En este requerimiento se contuvo una complejidad de nlogn el cual coincide con la toma de datos con diferentes tamaños descarga. Esta complejidad era consecuencia de recorrer una lista y poner cada elemento de esa lista en un mapa BTS, haciendo que en cada iterada se haga una complejidad Ologn. Asimismo, se puede observar que se utilizó un mapa que

contenía las fechas y diccionarios de cada sismo, para crear una ya filtrada y contando cuantos sismos pasaban con un mismo tiempo

Requerimiento <<7>>

.

Descripción

Entrada	los eventos sísmicos ocurridos en una región y un año especifico según alguna propiedad de interés
Salidas	El número de eventos sísmicos dentro del periodo anual relevante y de su histograma en un diagrama de barras
Implementado (Sí/No)	Si

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Crear un árbol RBT "*bst*" (newMap)	O(n)
Encontrar un rango para el año escogido en formato %Y-%m-%dT%H:%M:%S.%fz	O(1)
Obtener el min y max llave (ceiling y floor)	O(n)
Obtener los valores del mapa de sismos (value)	O(n)
Recorrer los eventos sísmicos de los valores de el arbol *filtrado*(for)	O(n)
Calcular la distancia con la formula de haversine	O(1)
Comparar si la distancia es menor a el radio dado	O(1)
Crear un arreglo *lista* (newList)	O(1)
copiar el diccionario de filtrado	O(1)
Convertir una fecha al formato %Y-%m-%dT%H:%M (strptime)	O(1)
Obtener los valores de un mapa dada una llave específica (get)	O(1)
Obtener el valor de una pareja llave-valor (getValue)	O(1)
Comparar si la llave existe.	O(1)
Eliminar el último elemento de una arreglo (removeLast)	O(1)
Añadir un diccionario a la última posición de un arreglo (addLast)	O(1)
Eliminar un elemento del diccionario (pop)	O(1)
Agregar un elemento a un árbol (put)	O(log(n))
comparar el valor de significancia si es mayor al anterior dato iterado	O(1)

Determinar el tamaño de nuestro árbol *bst* (size)	O(n)
TOTAL	***O(n log(n))***

Pruebas Realizadas

Tablas de datos

Entrada	Tiempo (ms)
small	412.15
5 pct	704.11
10 pct	1973.81
20 pct	5193.76
30 pct	10521.34
50 pct	25317.62
80 pct	39679.98
large	60494.52

Graficas

Análisis