ANÁLISIS DEL RETO

Isabella Sarquis Buitrago, 202221542, i.sarquis@uniandes.edu.co

Sara Valentina Rozo, 202124144, s.rozoo@uniandes.edu.co

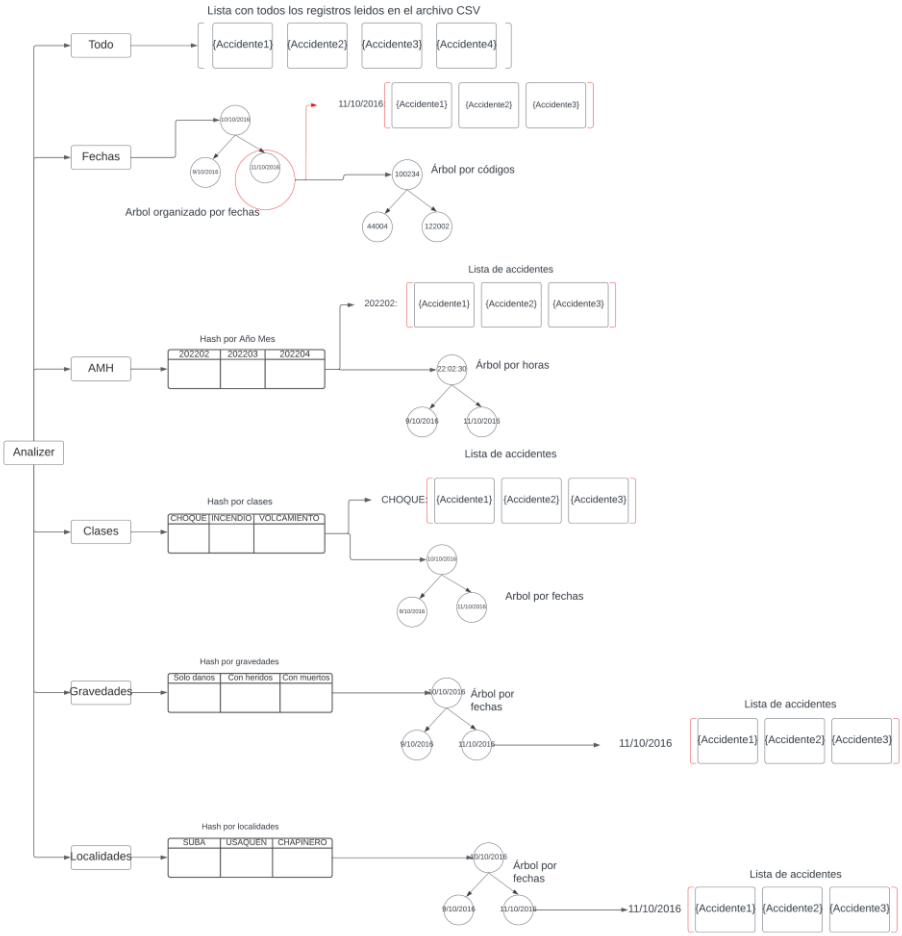
Tomás Osuna, 202212716, t.osuna@uniandes.edu.co

Carga de datos

Para el desarrollo de este reto se empleó un catálogo cuya estructura es un diccionario con 6 llaves, cada una clasifica los datos según una categoría.

- **Todo:** Es una lista que contiene todos los registros, el cual únicamente se utiliza para obtener el total de datos cargados.
- **Fechas:** Es un árbol ordenado por fechas (Year, Month, Day, Hour, Minute, Second). Cada llave tiene una lista asociada con todos los accidentes sucedidos en la fecha. Esta se utilizó en el requerimiento 1 y 7.
- **AMH:** Es un mapa donde cada llave es un código creado al unir el año y el mes, por ejemplo, febrero 2020 corresponde a 202202. Como valor asociado a las llaves, se tiene una lista con todos los accidentes sucedidos en ese mes y año, además se tiene un árbol ordenado por horas. Esta estructura se utilizó para los requerimientos 2 y 6.
- **Clases:** Es un mapa donde cada llave es una clase de accidente y como valores se tiene una lista con todos los accidentes de esta clase y un árbol ordenado por códigos. Esta estructura se utilizó para el requerimiento 2.
- **Gravedad:** Es un mapa donde cada llave representa una gravedad, los valores correspondientes son árboles organizados por las fechas. Cada valor dentro de cada nodo del árbol, son listas donde se guarda los accidentes repetidos la misma fecha y hora. Esta estructura se utilizó para el requerimiento 4.
- **Localidades:** Es un mapa donde cada llave representa una localidad, los valores correspondientes son árboles organizados por las fechas. Cada valor dentro de cada nodo del árbol, son listas donde se guarda los accidentes repetidos la misma fecha y hora. Esta estructura se utilizó para el requerimiento 5.

Esta estructura general se puede ver en el siguiente diagrama:



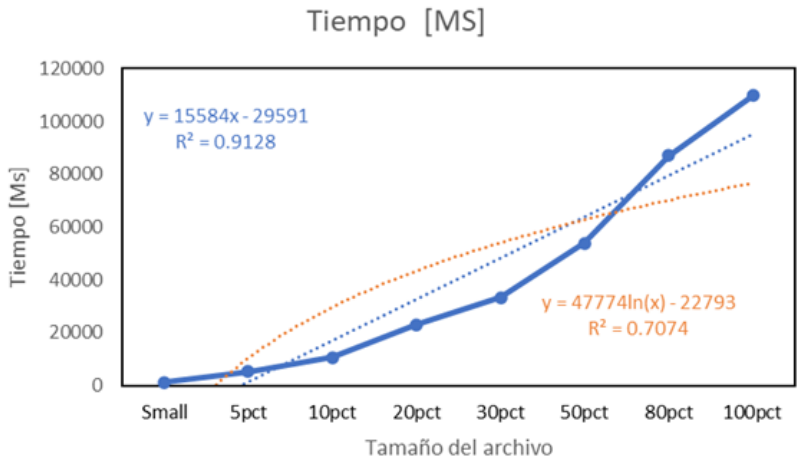
Complejidad: 8NlogN

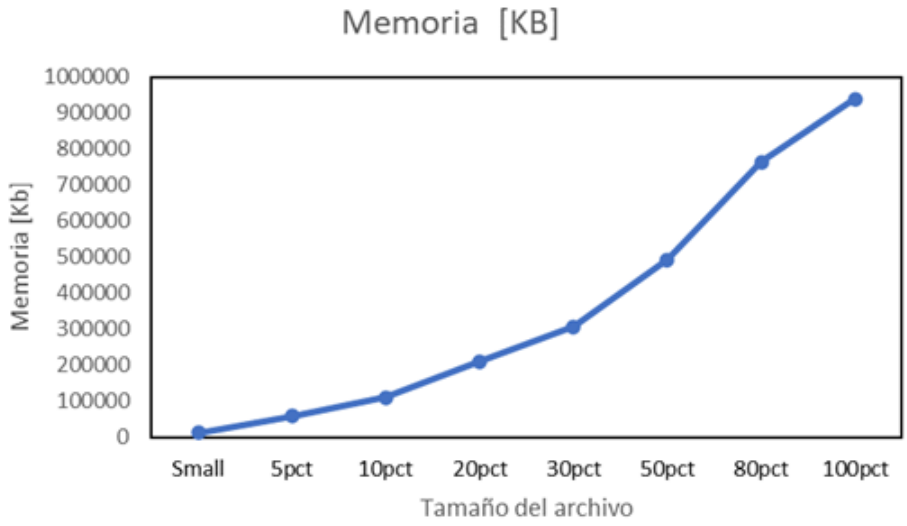
Pruebas Realizadas

Tamaño del archivo	Tiempo [MS]	Memoria [KB]
Small	1152.90	13212.01
5pct	5286.04	58818.56
10pct	10633.48	111179.92
20pct	23127.75	210650.17
30pct	33361.43	306961.83
50pct	53896.29	492888.56

80pct	87135.71	763784.65
100pct	109688.05	938496.16

GRÁFICAS





Requerimiento <<1>>

```
def req_1(Analyzer,fechainicial,fechafinal):
    """
    Función que soluciona el requerimiento 1
    """
    fechainicial1=fechainicial + " 00:00:00"
    fechainicial1_1=datetime.strptime(fechainicial1, "%Y/%m/%d %H:%M:%S")
    fechafinal1=fechafinal + " 23:59:59"
    fechafinal1_1=datetime.strptime(fechafinal1, "%Y/%m/%d %H:%M:%S")
    fechafinal2=fechafinal1_1+timedelta(seconds=1)
    lista=om.values(Analyzer["fechas"],fechainicial1_1,fechafinal2)
    elementos=0
    for sublistas in lt.iterator(lista):
        tamaño=lt.size(sublistas["lst"])
        elementos+=tamaño

    return lista,elementos
```

Descripción

Es importante tener en cuenta que ya tenemos dentro del diccionario una estructura que facilita el acceso para rangos de fechas, en este caso es un árbol de fechas donde están organizados los accidentes de acuerdo con este parámetro, en ese orden de ideas únicamente se procedió a extraer los valores dentro de un rango llaves, posteriormente se itera la lista que surge como resultado del paso anterior para obtener el número de elementos en el rango.

Entrada	Fecha inicial y final del intervalo
Salidas	El código devuelve una lista de lista con todos los accidentes en el rango de fechas, organizados del más reciente al menos reciente.
Implementado (Sí/No)	Si

Estructura utilizada

ARBOL -> “FECHAS”

Análisis de complejidad

Pasos	**Complejidad**
Asignaciónón de variables (límites del rango)	O(1)
Obtención de los valores dentro de los límites establecidos en el árbol.	O(2logN)
Iteración de la lista se retorna en el paso anterior. -Medición del tamaño, de cada una de las sub-listas que dentro de la lista	O(N)
TOTAL	***O(N +2LogN +1)***

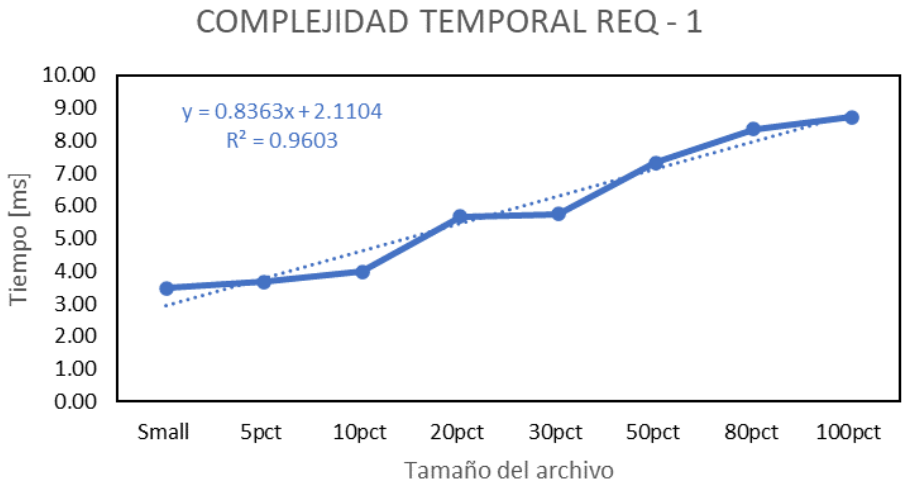
Pruebas Realizadas

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).

Tablas de datos

Tamaño del archivo	Tiempo [MS]
Small	3.48
5pct	3.68
10pct	3.99
20pct	5.67
30pct	5.76
50pct	7.33
80pct	8.35
100pct	8.73

Graficas



Análisis

El uso de árboles para la solución del requerimiento permitió alcanzar una complejidad máxima de O(N), pues no es necesario organizar los elementos utilizando algoritmos de ordenamiento para obtener el resultado deseado. Además, al tener un árbol balanceado, únicamente se requiere encontrar la llave superior e inferior del rango, aspecto que se realiza con búsqueda binaria y se garantiza una complejidad de O(Logn), y únicamente se extrae la lista de los datos entre los límites, pues estos ya se encuentran organizados.

En este caso es importante tener en cuenta que la complejidad O(N) no surge por las estructuras utilizadas sino por la naturaleza de los datos, pues las llaves tienden a repetirse y con el fin de que no se reemplacen los datos dentro del árbol se crean listas para guardar los accidentes que suceden en la misma fecha. El tamaño de cada lista varía, no obstante, este no supera los 4 elementos.

Los datos se ajustan a un modelo lineal, mostrando una correspondencia entre la complejidad Teórica y la experimental.

Requerimiento <<2>>

```
def req_2(Analyzer,año,mes,hora1,hora2):
    """
    Función que soluciona el requerimiento 2
    """
    horainicial=datetime.strptime(hora1,"%H:%M:%S")
    horafinal=datetime.strptime(hora2,"%H:%M:%S")
    horafinal1=horafinal+timedelta(seconds=1)
    mes1=datetime.strptime(mes,"%B").month
    año1=año
    codigoAMH= int(str(año1) + str(mes1))
    estructura=Analyzer["AMH"]
    arbol=me.getValue(mp.get(estructura,codigoAMH))
    siniestros=om.values(arbol["arbolhoras"],horainicial,horafinal1)

    total=0
    siniestros2=lt.newList(datastructure="ARRAY_LIST")
    for sublistas in lt.iterator(siniestros):
        total+=lt.size(sublistas)
        for siniestro in lt.iterator(sublistas):
            formato={"Código del accidente":siniestro["CODIGO_ACCIDENTE"],
                     "Hora del accidente":siniestro["HORA_OCURRENCIA_ACC"],
                     "Fecha y hora del accidente":siniestro["FECHA_HORA_ACC"],
                     "Dia del accidente":siniestro["DIA_OCURRENCIA_ACC"],
                     "Localidad":siniestro["LOCALIDAD"],
                     "Dirección":siniestro["DIRECCION"],
                     "Gravedad":siniestro["GRAVEDAD"],
                     "Clase de accidente":siniestro["CLASE_ACC"],
                     "Latitud":siniestro["LATITUD"],
                     "Longitud":siniestro["LONGITUD"]}
            lt.addLast(siniestros2,formato)
    merg.sort(siniestros2,comparefechas0)
    return siniestros2,total
```

Descripción

Este requerimiento se encarga de retornar una lista con todos los accidentes de un mes y año específicos ocurridos en un intervalo de horas. Primero busca el código del mes-año dentro del mapa, luego saca los valores en los límites especificados dentro del árbol de horas (valor correspondiente a la llave de los mapas), posteriormente itera la lista de listas resultante para obtener su tamaño y extraer los elementos. Finalmente se organizan los elementos de acuerdo a la fecha dejando la hora intacta.

Entrada	Horario inicial y final del intervalo deseado. Año y mes de consulta
Salidas	El algoritmo retorna una lista (y su tamaño) con todos los accidentes ocurridos en un mes y años determinados durante un intervalo de horas.
Implementado (Sí/No)	Si

Estructura utilizada

Mapa -> AMH

Análisis de complejidad

Pasos	**Complejidad**

Buscar la llave con el código Año-mes deseado en el mapa.	O(1)
Extraer la lista de valores dentro del rango de horas deseado en el árbol de horas correspondiente al Código año-mes.	O(2Logn)
Iterar la lista de listas que surge como resultado del paso anterior	O(N*K) Donde K es una constante, teniendo en cuenta que en las sub-listas únicamente se guardan los accidentes repetidos la misma hora y fecha.
TOTAL:	O(N*K) + 2Logn + 1

Pruebas Realizadas

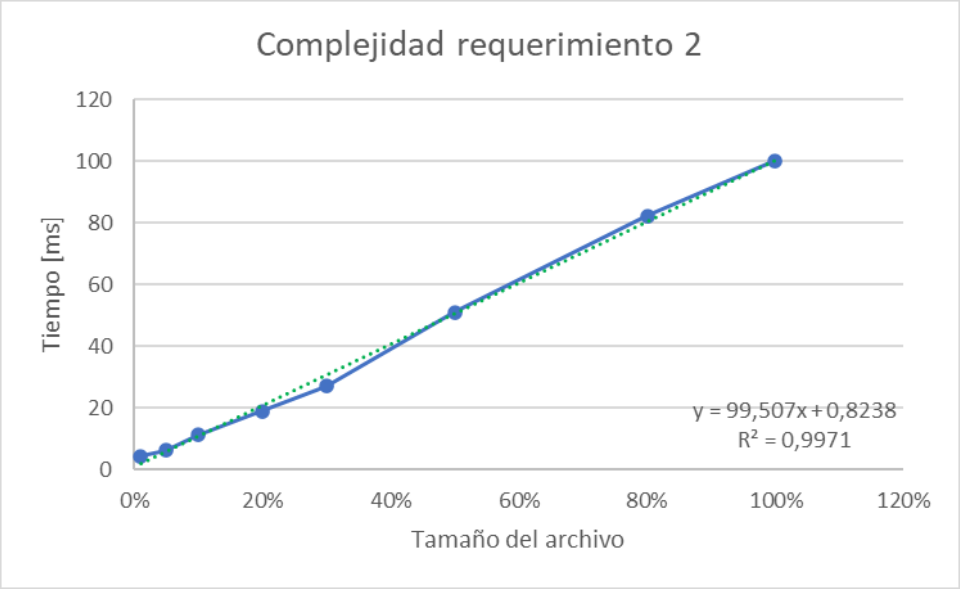
Las pruebas se realizaron para todos los archivos ingresando con los siguientes parámetros: el año 2020 y el mes diciembre desde las 00:00:00 hasta las 23:59:59. Se utilizó un computador con las siguientes especificaciones:

Procesadores	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 10

Tablas de datos

Tamaño	Tiempo [ms]
1%	4,4
5%	6,21
10%	11,22
20%	19,02
30%	27,01
50%	51,02
80%	82,24
100%	100,01

Graficas



Análisis

La complejidad temporal de este requerimiento es de O(N*K) + 2Logn + 1, tal como se puede ver en el análisis paso a paso. Gracias a la manera en la que se cargaron los datos, la complejidad de esta función proviene de la naturaleza de los árboles, no de iteraciones. Al extraer un rango de valores de un árbol balaceado, se utiliza una búsqueda binaria con complejidad O(logn) y se obtiene una lista, la cual es necesario iterar para darle el formato a los datos que se pide y el permitido en tabulate.

Por otro lado, en el código se implementó un doble for, sin embargo, en el segundo recorrido se iteran listas con menos de 4 elementos, por lo que se toma k como constante menor a 4, la cual no provoca un incremento significativo en la temporalidad de ejecución del programa.

Lo anterior se puede comprobar con la gráfica, ya que la complejidad del requerimiento se ajusta a una recta lineal con un coeficiente de relación (R^2) de 0,99, lo que significa que la función se comporta de la manera esperada teóricamente, es decir, O(N).

Requerimiento <<3>>

```
def req_3(analyzer,clase,via):
    MapClase=analyzer["clases"]
    FechasClase=me.getValue(mp.get(MapClase,clase))
    Min=datetime.strptime("2015/01/01 00:00:01", "%Y/%m/%d %H:%M:%S")
    Max=datetime.strptime("2022/12/31 23:59:59", "%Y/%m/%d %H:%M:%S")
    FechasRecientes=om.values(FechasClase,Min,Max)
    Fech= lt.subList(FechasRecientes, 0,lt.size(FechasRecientes))
    Resp=lt.newList(datastructure="ARRAY_LIST")
    for Requerimiento in lt.iterator(Fech):
        for Formato in lt.iterator(Requerimiento):
            Form={
                "CODIGO_ACCIDENTE":Formato["CODIGO_ACCIDENTE"],
               "FECHA_HORA_ACC": Formato["FECHA_HORA_ACC"],
               "DIA_OCURRENCIA_ACC":Formato["DIA_OCURRENCIA_ACC"],
               "LOCALIDAD":Formato["LOCALIDAD"],
               "DIRECCION": Formato["DIRECCION"],
               "GRAVEDAD":Formato["GRAVEDAD"],
               "CLASE_ACC": Formato["CLASE_ACC"],
               "LATITUD":Formato["LATITUD"],
               "LONGITUD":Formato["LONGITUD"]
               }
            if via in Form["DIRECCION"]:
                lt.addFirst(Resp,Form)
    Final=lt.newList(datastructure="ARRAY_LIST")
    for elements in range(1,4):
        lt.addLast(Final,lt.getElement(Resp,elements))
    return Final
```

Descripción

En este requerimiento se emplea para encontrar los 3 accidentes más recientes de una clase de accidente en específico a lo largo de una vía, de manera que recorre un mapa de las clases de accidentes para posteriormente pasar por un arbol con las fechas correspondientes de accidentes ocurridos al punto que se clasifica por si pasa en la vía indicada y acortando a solo los últimos 3 más recientes

Entrada	Clase de accidente y vía donde ocurrió
Salidas	El algoritmo retorna una lista con los 3 accidentes más recientes de una clase de accidente y vía especifica
Implementado (Sí/No)	Si (tomás Osuna)

Estructura utilizada

Mapa -> Clases

Árbol->fechas

Análisis de complejidad

Pasos	**Complejidad**
Extraer el mapa ordenado por gravedades del analyzer	O(1)

Darle el formato de datetime a lo minimo que podria ser y a lo mayor que podria ser	O(1)
Extraer el valor asociado a la llave del mapa con la clase ingresada por parámetro. Este valor es un árbol ordenado por fechas.	O(1)
Extraer los valores del árbol que se encuentran entre las fechas ingresadas por el usuario	O(Logn)
Crear una sublista con los últimos 5 valores	O(1)
Se recorre con doble for la lista para dejar los datos en el formato que se pide	O(k),k corresponde a una constante menor a n.
ese formato se pasa a una lista y se recorre los 3 elementos más recientes pasando a una lista final	O(3)
TOTAL	***O(Logn)+O(5k)+O(1)O(logn)+O(1)***

Pruebas Realizadas

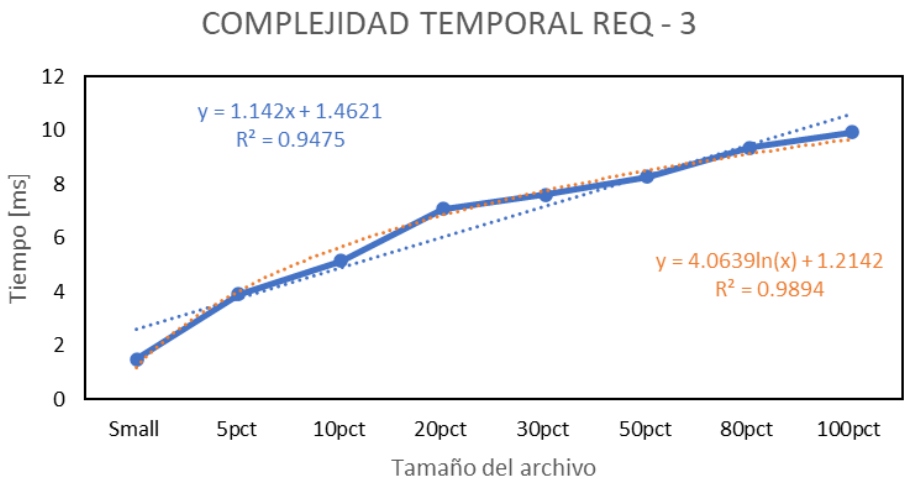
Las pruebas se realizaron para todos los archivos ingresando con los siguientes parámetros: choque como clase de accidente y avenida de las Américas como vía Se utilizó un computador con las siguientes especificaciones:

Procesadores	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 11

Tablas de datos

Tamaño	Tiempo [ms]
1%	1.51
5%	3.92
10%	5.16
20%	7.07
30%	7.6
50%	8.27
80%	9.36
100%	9.92

Graficas



Análisis

Al manejo de un árbol principalmente por sus fechas se puede denotar como al recorrerlo llega a ser más relacionada a la de log(n) que a n por mucho que se empleen listas y uso de elementos para demostrar los recientes además que uso la opción de emplear la posibilidad más baja y alta de fechas para poder determinar cómo extraer las fechas de la clase de accidente en vez de recorrer y buscar su mínimo y máximo generando más recorridos innecesarios, por ello no se implementó dicho procedimiento, además que se tomó las molestias de ordenarlo de una manera más simple en vez de usar otros métodos de ordenamientos que ya tienen complejidades altas y no constantes.

Requerimiento <<4>>

```
def req_4(analyzer, gravedad, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    fecha_final=datetime.strptime(fecha_final, "%Y/%m/%d %H:%M:%S")
    fecha_inicial=datetime.strptime(fecha_inicial, "%Y/%m/%d %H:%M:%S")
    mapa_gravedades=analyzer["gravedad"]
    arbol_fechas=me.getValue(mp.get(mapa_gravedades,gravedad))
    valores=om.values(arbol_fechas,fecha_inicial,fecha_final)
    total= lt.subList(valores, lt.size(valores)-4,5)
    final=lt.newList("ARRAY_LIST")
    for resultado in lt.iterator(total):
        for r in lt.iterator(resultado):
            r={"CODIGO_ACCIDENTE":r["CODIGO_ACCIDENTE"],
               "FECHA_HORA_ACC": r["FECHA_HORA_ACC"],
               "DIA_OCURRENCIA_ACC":r["DIA_OCURRENCIA_ACC"],
               "LOCALIDAD":r["LOCALIDAD"],
               "DIRECCION": r["DIRECCION"],
               "CLASE_ACC": r["CLASE_ACC"],
               "LATITUD":r["LATITUD"],
               "LONGITUD":r["LONGITUD"]}
            lt.addFirst(final,r)

    return final
```

Descripción

Se extraen todos los accidentes clasificados en la gravedad ingresada por parámetro; se obtiene un árbol ordenado por fechas; con la función values se sacan todos los valores entre las fechas dadas por parámetro y se extraen los 5 últimos con una sublista.

ESTRUCTURA UTILIZADA:

Entrada	Analyzer con toda la carga de datos, gravedad de los accidentes a buscar, fecha inicial y fecha inicial que se quiere buscar
Salidas	Lista con los 5 accidentes más recientes en el rango de fechas ingresadas con la gravedad dada por parámetro.
Implementado (Sí/No)	Si (Isabella Sarquis Buitrago)

Estructura utilizada

Análisis de complejidad

Pasos	**Complejidad**
Darle el formato de datetime a las fechas ingresadas como str por el usuario	O(1)
Extraer el mapa ordenado por gravedades del analyzer	O(1)
Extraer el valor asociado a la llave del mapa con la gravedad ingresada por parámetro. Este valor es un árbol ordenado por fechas.	O(1)
Extraer los valores del árbol que se encuentran entre las fechas ingresadas por el usuario	O(Logn)
Crear una sublista con los últimos 5 valores	O(1)
Se recorre con doble for la lista para dejar los datos en el formato que se pide,	O(5*k), donde se recorren únicamente 5 elementos en el primer for y k corresponde a una constante menor a n.
TOTAL	***O(Logn)+O(5k)+O(1)≈O(logn)***

Pruebas Realizadas

Se hicieron las pruebas en todos los archivos ingresando como parámetro la gravedad “CON MUERTOS”, la fecha inicial 2016/10/01 00:00:00 y la fecha final 2018/10/01 23:59:59. Se utilizó un computador con las siguientes especificaciones:

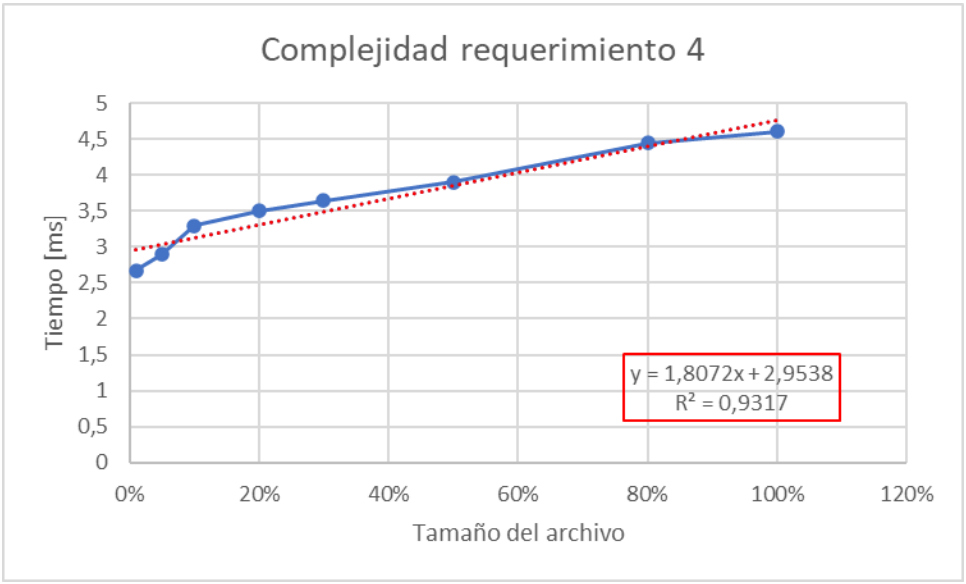
Procesadores	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 10

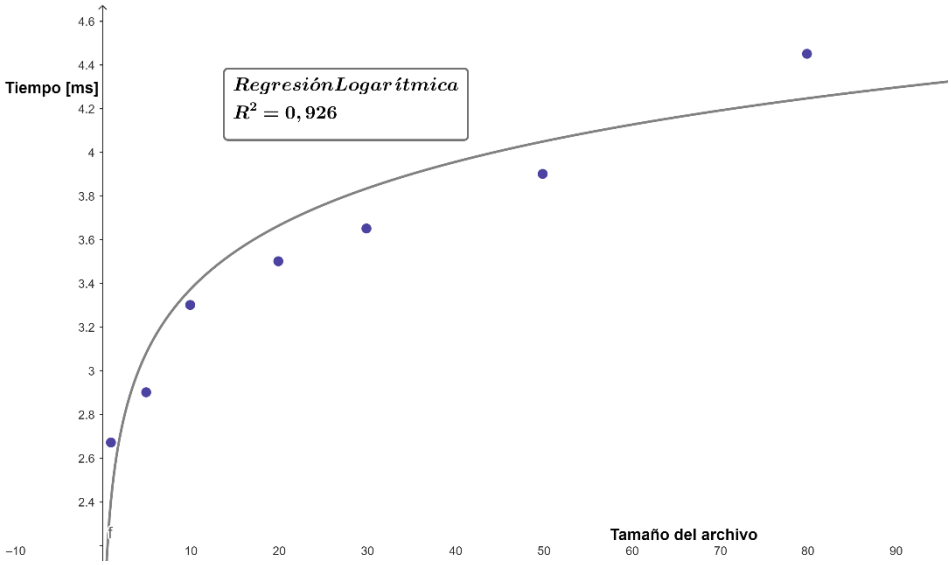
Tablas de datos

Tamaño	Tiempo [ms]
1%	2,67
5%	2,9

10%	3,3
20%	3,5
30%	3,65
50%	3,9
80%	4,45
100%	4,61

Graficas





Análisis

Como se evidencia en el análisis paso a paso de la complejidad, este requerimiento tiene una temporalidad aproximada de O(logn), la cual se produce por la búsqueda binaria al momento de extraer los valores de un árbol. Además, para dar el formato pedido a los datos se hace un doble ciclo, sin embargo, no se provoca una complejidad de O(n^2) porque se itera sobre sublista de 5 elementos.

En la gráfica se puede observar que tanto la regresión lineal como la logarítmica tienen un coeficiente de relación (R^2) de 0,93, lo que comprueba que la iteración no genera un cambio significativo en la complejidad de la función.

Además, gracias a los árboles se ahorra el implementar algoritmos de ordenamiento, los cuales tienen una complejidad mínima de O(nlogn), por lo que se corroboró que esta estructura de datos resulta óptima para ordenar datos.

Requerimiento <<5>>

```
def req_5(control,año,mes,localidad):
    """
    Función que                    o 5
    """
    estrucutra=control["localidades"]
    mes1=datetime.strptime(mes,"%B").month
    primerafecha=año+"/"+str(mes1)+"/"+"01" + " " + "00:00:00"
    ultimafecha=año+"/"+str(mes1+1)+"/"+"01" + " " + "00:00:00"
    fechainicial1=datetime.strptime(primerafecha, "%Y/%m/%d %H:%M:%S")
    fechafinal1=datetime.strptime(ultimafecha, "%Y/%m/%d %H:%M:%S")
    fechafinal2=fechafinal1-timedelta(days=1)
    arbolfechas=me.getValue(mp.get(estrucutra,localidad))
    lista=om.values(arbolfechas,fechainicial1,fechafinal2)

    tamaño=0
    listafinal=lt.newList(datastructure="ARRAY_LIST")
    for sublista in lt.iterator(lista):
        tamaño+=lt.size(sublista)
        for siniestro in lt.iterator(sublista):
                formato={"Código del accidente":siniestro["CODIGO_ACCIDENTE"],
                    "Hora del accidente":siniestro["HORA_OCURRENCIA_ACC"],
                    "Fecha y hora del accidente":siniestro["FECHA_HORA_ACC"],
                    "Dia del accidente":siniestro["DIA_OCURRENCIA_ACC"],
                    "Localidad":siniestro["LOCALIDAD"],
                    "Dirección":siniestro["DIRECCION"],
                    "Gravedad":siniestro["GRAVEDAD"],
                    "Clase de accidente":siniestro["CLASE_ACC"],
                    "Latitud":siniestro["LATITUD"],
                    "Longitud":siniestro["LONGITUD"]
                    }
                lt.addLast(listafinal,formato)
    listaaentregar=lt.subList(listafinal,1,10)
    return listaaentregar,tamaño
```

(parameter) control: Any

Descripción

Para la solución de este requerimiento, se procedió a buscar la localidad deseada en un mapa de localidades, donde cada llave tiene como valor un árbol organizado por fechas en el formato “A/M/D H:M:S". En este orden de ideas, se procedió a sacar los accidentes de todo un mes, entendiendo que este es un rango de fechas entre el 1 del mes deseado y 1 el primero del mes posterior. Finalmente, se extrajeron los valores de la lista de listas generada en el paso anterior y se creó una sublista con los primeros diez elementos, la cual se entrega al controlador.

Entrada	El algoritmo pide como parámetros de entrada la localidad en la que ocurrieron los accidentes, el mes y el año.
Salidas	En este caso se retorna: Una lista con los 10 accidentes menos recientes ocurridos en el mes y año seleccionado. El número de accidentes totales que ocurrieron en esa localidad
Implementado (Sí/No)	Si por Sara Valentina Rozo Oviedo

Estructura utilizada

MAPA -> LOCALIDADES

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Establecer el formato de los límites iniciales y finales del intervalo de tiempo deseado (en este caso los días de un mes en un año específico)	O (1)
Obtención del valor correspondiente a la llave de la localidad deseada, en el mapa de localidades.	O(1)
Obtención de la lista de valores en el rango de fechas del mes dentro del árbol de fechas de cada localidad.	O(2logN)
Iteración de la lista de listas para sumar el tamaño de cada sub-lista, darle el formato a los elementos e insertarlos en la lista recopiladora.	O(N*K) Donde K es una constante, teniendo en cuenta que en las sablistas únicamente se guardan los accidentes repetidos la misma hora y fecha.
Extracción de una sublista con 10 elementos desde la lista recopiladora.	O (1)
TOTAL	***O(N*k + 2LogN)***

Pruebas Realizadas

EQUIPO UTILIZADO:

Procesadores	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 11

Tablas de datos

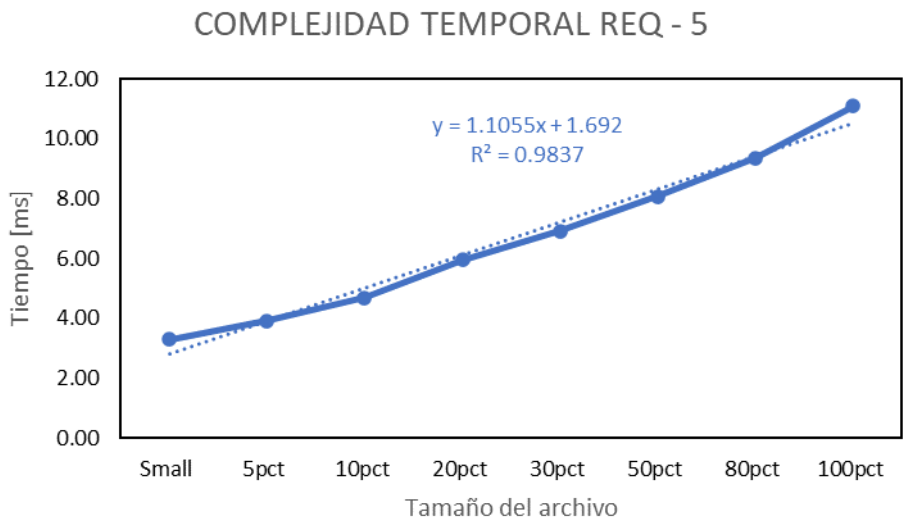
Para la toma de datos se utilizó los valores de referencia al mes de noviembre; el año 2016 y la localidad de Fontibón.

Tamaño del archivo	Tiempo [MS]
Small	3.30

5pct	3.92
10pct	4.70
20pct	5.96
30pct	6.92
50pct	8.08
80pct	9.36
100pct	11.10

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A partir de la información expuesta se puede concluir lo siguiente:

1. El valor experimental de la complejidad temporal corresponde con el valor teórico calculado, pues los datos y un modelo lineal tienen un coeficiente de correlación cercano 1. De esta forma se demuestra que el iterar las sublistas de la lista general obtenida con la función “Om.Values”, la complejidad es prácticamente constante, debido a los pocos elementos que las componen. Al tener un diccionario primero de localidades y luego un árbol con llaves correspondiente a la fecha y la hora del accidente, se filtran los datos de tal forma que se disminuye la probabilidad que dos accidentes hayan sido registrados al mismo tiempo.

2. El uso de árboles demostró ser eficiente para el cumplimiento del requerimiento, pues a diferencia de los mapas no ordenados, en este caso es posible primero identificar la información, por medio del uso de llaves, pero, sobre todo, extraer todos los accidentes dentro de un rango de fecha de forma organizada. Al estar balanceados (debido a que se emplearon arboles rojo negro) se garantiza una complejidad de O(LogN) para este proceso.

3. Al igual que en otros requerimientos la máxima complejidad surge de la naturaleza de los datos, existen algunos accidentes con misma fecha y hora de registro, razón por la cual fue necesario establecer como valor para cada llave del árbol una lista, con el fin de evitar el reemplazo de valores, aunque para algunas de las llaves eso significará guardar solo un elemento. En ese sentido, al obtener la lista de la función “Om.Values”, es necesario iterarla para obtener el número de accidentes totales ocurridos en el intervalo pedido, explicando la complejidad O(N*K).

Requerimiento <<6>>

```
def req_6(Analyzer,año,mes,latitud1,longitud1,radio,Topn):
    """
    Función que soluciona el requerimiento 6
    """
    arboldistancias=om.newMap(omaptype="RBT",comparefunction=distanciaalcentro)
    estructura=Analyzer["AMH"]
    mes1=datetime.strptime(mes,"%B").month
    codigoAMH= int(año + str(mes1))
    mapa=me.getValue(mp.get(estructura,codigoAMH))
    latitud1_1=float(latitud1)
    longitud1_1=float(longitud1)

    for siniestro in lt.iterator (mapa["lstsiniestros"]):
        distancia=0
        latitud2=float(siniestro["LATITUD"])
        longitud2=float(siniestro["LONGITUD"])
        diferencialatitudes=radians(latitud2-latitud1_1)
        diferencialongitudes=radians(longitud2-longitud1_1)
        raiz=sin((diferencialatitudes)/2)**2 + cos(radians(latitud1_1))*cos(radians(latitud2))*sin((diferencialongitudes)/2)**2
        distancia=2*asin(sqrt(raiz))
        radio=6370
        distanciafinal=distancia*radio
        formato = {"Código del accidente":siniestro["CODIGO_ACCIDENTE"],
                   "Fecha y hora del accidente":siniestro["FECHA_HORA_ACC"],
                   "Dia del accidente":siniestro["DIA_OCURRENCIA_ACC"],
                   "Localidad":siniestro["LOCALIDAD"],
                   "Dirección":siniestro["DIRECCION"],
                   "Gravedad":siniestro["GRAVEDAD"],
                   "Clase de accidente":siniestro["CLASE_ACC"],
                   "Latitud":siniestro["LATITUD"],
                   "Longitud":siniestro["LONGITUD"],
                   "Distancia":distanciafinal}
```

```
        if distancia<=float(radio):
            om.put(arboldistancias,distancia,formato)
    valores=mp.valueSet(arboldistancias)
    sublista=lt.subList(valores,1,int(Topn))
    return valores,sublista
```

Descripción

Primero, utilizando el mapa de código año-mes, se extrae el valor correspondiente al código y se obtiene una lista con todos los accidentes, la cual se itera para hacer los cálculos de distancia necesarios. Luego se crea un árbol ordenado por distancia, al cual solo ingresan los accidentes con una distancia menor o igual a la ingresada por el usuario. Finalmente, se extraen los valores del árbol ya organizados en una sublista con el número de elementos deseado.

Entrada	Se ingresa el analyzer con todos los datos cargados, el año y el mes de los datos a buscar, la latitud y la longitud inicial, el radio máximo del área y el topn de los crímenes que se quieren mostrar en pantalla.
Salidas	Se devuelve el top N, de accidentes más cercanos a la zona dentro del rango estipulado en los parámetros de entrada y el número total de los accidentes.
Implementado (Sí/No)	Si (Todos)

Estructura utilizada

MAPA -> “AMH”

Análisis de complejidad

Pasos	**Complejidad**
Creación del árbol donde se guardarán las distancias organizadamente.	O(1)
Buscar la llave con el código Año-mes deseado en el mapa.	O(1)
Iterar los elementos de la lista correspondiente y realizar los cálculos de distancia más el ajuste del formato.	O(N)
Insertar el nuevo accidente ya formateado en el árbol.	O(logN)
Extraer la lista de valores del árbol ya organizada	O(logN)
Extraer la sublista con el número de elementos estipulados.	O(1)
TOTAL	***O((NLogN) + LogN +3)***

Pruebas Realizadas

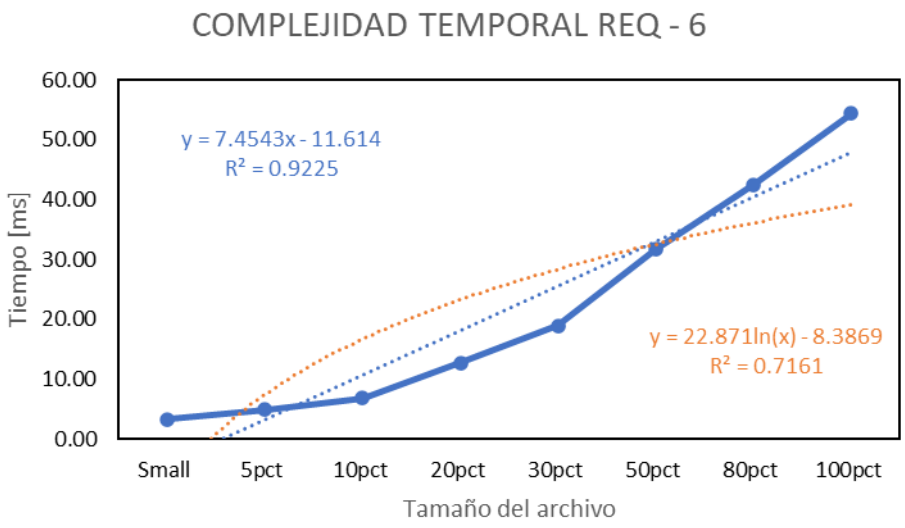
Se hicieron las pruebas del requerimiento para todos los archivos con los siguientes parámetros de entrada: Año 2022, mes Enero, latitud 4, longitud –74, radio 5km y top 3. Se utilizó un computador con las siguientes especificaciones:

Procesadores	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 10

Tablas de datos

Tamaño	Tiempo [ms]
1%	3,36
5%	4,91
10%	6,88
20%	12,7
30%	18,93
50%	31,72
80%	42,55
100%	54,39

Gráficas



Análisis

El uso de árboles facilita el acceso a los datos organizados de un rango. En este caso la complejidad surge de la construcción de la nueva estructura, pues es necesario realizar la inserción de cada uno de los accidentes de un mes (N), dentro del árbol, proceso que tiene una complejidad O(logN) en el peor caso. Para este requerimiento no se utilizó un algoritmo de ordenamiento como merge-sort, porque las distancias no quedan guardadas dentro del diccionario del accidente, por lo tanto, no sería posible su organización; mientras que, al implementar un árbol se permite almacenar las distancias correspondientes a cada uno de los accidentes sin procesos adicionales.

La complejidad experimental es coherente con la complejidad calculada. En la gráfica O(NLogN), se observa que, en los archivos pequeños, el crecimiento de la complejidad es menor a la esperada, sin embargo, a medida que crece el tamaño de los archivos, la pendiente de NlogN llega a ser mayor.

Requerimiento <<7>>

```
def req_7(Analyzer,año,mes):
    """
    Función que soluciona el requerimiento 7
    """
    estructura=Analyzer["fechas"]
    mes1=datetime.strptime(mes,"%B").month
    primerafecha=año+"/"+str(mes1)+"/"+"01" + " " + "00:00:00"
    fecha1formato=datetime.strptime(primerafecha, "%Y/%m/%d %H:%M:%S")
    listafinal=lt.newList(datastructure="ARRAY_LIST")
    centinela=True
    ultimafecha=año+"/"+str(mes1+1)+"/"+"01" + " " + "00:00:00"
    if mes1+1==13:
        ultimafecha=str(int(año)+1)+"/"+"1"+"/"+"01" + " " + "00:00:00"
    fecha2formato=datetime.strptime(ultimafecha, "%Y/%m/%d %H:%M:%S")

    while centinela:
        diareferenia=fecha1formato.day
        nodo1=om.ceiling(estructura,fecha1formato)
        nodo1_1=me.getValue(om.get(estructura,nodo1))
        arbolnodo=nodo1_1["codigo"]
        mincodigo=om.minKey(arbolnodo)
        dato=me.getValue(om.get(arbolnodo,mincodigo))
        diacodigo=datetime.strptime(dato["FECHA_OCURRENCIA_ACC"], "%Y/%m/%d").day
        formato={"Código del accidente":dato["CODIGO_ACCIDENTE"],
                 "Fecha y hora del accidente":dato["FECHA_HORA_ACC"],
                 "Dia del accidente":dato["DIA_OCURRENCIA_ACC"],
                 "Localidad":dato["LOCALIDAD"],
                 "Dirección":dato["DIRECCION"],
                 "Gravedad":dato["GRAVEDAD"],
                 "Clase de accidente":dato["CLASE_ACC"],
                 "Latitud":dato["LATITUD"],
                 "Longitud":dato["LONGITUD"]}
```

```
mayor=fecha1formato + timedelta(hours=23,minutes=59,seconds=59)
nodo2=om.floor(estructura,mayor)
nodo2_1=me.getValue(om.get(estructura,nodo2))
arbolnodo2=nodo2_1["codigo"]
mincodigo2=om.minKey(arbolnodo2)
dato2=me.getValue(om.get(arbolnodo2,mincodigo2))
diacodigo2=datetime.strptime(dato2["FECHA_OCURRENCIA_ACC"], "%Y/%m/%d").day

formato2={"Código del accidente":dato2["CODIGO_ACCIDENTE"],
          "Fecha y hora del accidente":dato2["FECHA_HORA_ACC"],
          "Dia del accidente":dato2["DIA_OCURRENCIA_ACC"],
          "Localidad":dato2["LOCALIDAD"],
          "Dirección":dato2["DIRECCION"],
          "Gravedad":dato2["GRAVEDAD"],
          "Clase de accidente":dato2["CLASE_ACC"],
          "Latitud":dato2["LATITUD"],
          "Longitud":dato2["LONGITUD"]}

if diacodigo == diareferenia and diacodigo2==diareferenia:
    lt.addLast(listafinal,formato)
    lt.addLast(listafinal,formato2)

elif diacodigo == diareferenia:
    lt.addLast(listafinal,formato)
    lt.addLast(listafinal,formato)

elif diacodigo2==diareferenia:
    lt.addLast(listafinal,formato2)
    lt.addLast(listafinal,formato2)
```

```
    fecha1formato=fecha1formato+timedelta(days=1)
    if fecha1formato>=fecha2formato:
        centinela=False

estructura2=Analyzer["AMH"]
codigoAMH= int(año + str(mes1))
arbol=me.getValue(mp.get(estructura2,codigoAMH))

listapython=[]
listafrecuencias=[]
i=0
p=0
while str(p)+str(i)!="24":
    hora1=str(p)+str(i)+":00:00"
    hora2=str(p)+str(i)+":59:00"
    listapython.append(hora1)
    lista=om.values(arbol["arbolhoras"],datetime.strptime(hora1, "%H:%M:%S"),datetime.strptime(hora2, "%H:%M:%S"))
    frecuencia=0
    for sublistas in lt.iterator(lista):
        frecuencia+=lt.size(sublistas)
    listafrecuencias.append(frecuencia)

    i+=1
    if i==10:
        p+=1
        i=0
return listafinal,listafrecuencias,listapython
```

Descripción

Este requerimiento se divide en dos grandes partes:

Primero: La obtención de los accidentes más temprano y más tardes para cada uno de los días.

Se establecen las fechas que delimitarán el rango sobre el cual se desea obtener información. Luego se realiza un ciclo While, en el cual se van sumando los días, se extrae el accidente más temprano usando la función cealing con el parámetro 00:00:00 y el accidente más tardío usando la función floor con el parámetro 23:59:59. Por último, se acude al árbol de códigos para extraer el accidente con el menor código perteneciente a los horarios escogidos. Este valor se agrega a una lista final.

Segundo: La obtención de la frecuencia por horas.

Se realiza un doble ciclo en donde, a través de la sumatoria de las variables “P” y “i”, se manejan las horarios límite del intervalo de horas de un día. Luego se extrae un rango de todos los valores dentro del intervalo obtenido, dejando una lista de listas que se itera para conocer la frecuencia de cada hora.

Entrada	Año y mes del cual se desea obtener resultados.
Salidas	El algoritmo devuelve tres listas, la primera contiene los accidentes más temprano y más tardes para cada uno de los días, las otras dos listas tienen las frecuencias y los títulos del eje x que deben ser graficados.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Estructura utilizada

ARBOL -> “fechas”

MAPA -> “AMH”

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
PARTE 1	
Establecer el formato de los limites iniciales y finales del intervalo de tiempo deseado.	O(1)
Creación del ciclo While (Necesario para iterar los días)	O(N)
Ceiling con base en un dia y la hora “00:00:00”	O(LogN)
Obtención del valor correspondiente al ceiling	O(LogN)
Búsqueda y obtención del accidente con el menor código en el árbol de códigos correspondiente a cada llave hora.	O(2logN)
Floor con base en un día y la hora “23:59:59”	O(logN)
Obtención del valor correspondiente al Floor	O(LogN)
Búsqueda y obtención del accidente con el menor código en el árbol de códigos correspondiente a cada llave hora.	O(2logN)
Inserción de los accidentes en la lista final	O(1)
PARTE 2	
Búsqueda del código Año-mes en el mapa AMH.	O(1)
Creación del ciclo While (que permitirá iterar las horas)	O(N)
Obtención del rango de valores entre una hora base y una hora final.	O(LogN)
Iteración de la lista de valores para obtener su tamaño y así la frecuencia por hora.	O(N)
Inserción de la frecuencia en lista python.	O(1)

TOTAL	***O(N^2+ 9NlogN)***

Pruebas Realizadas

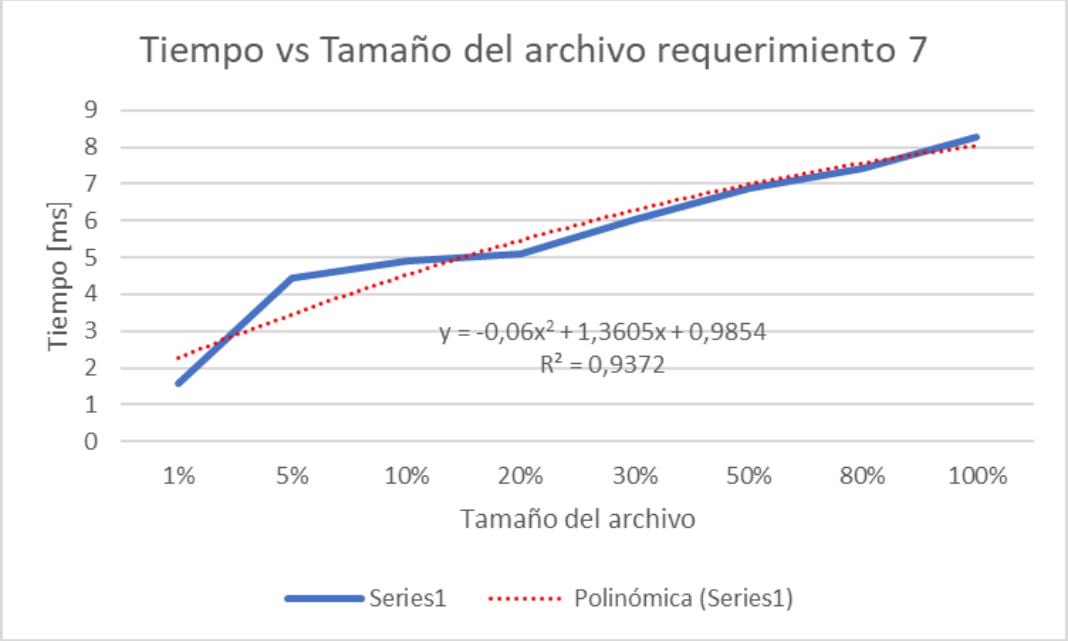
Se realizaron las pruebas en todos los archivos ingresando el año 2020 y el mes diciembre como parámetros. Se utilizó un computador con las siguientes especificaciones:

Procesadores	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 10

Tablas de datos

Tamaño	Tiempo [ms]
1%	7,07
5%	7,77
10%	8,49
20%	8,88
30%	9,12
50%	9,77
80%	10,35
100%	11,29

Gráficas



Análisis

Este requerimiento se trató principalmente con árboles y sus funciones. Se obtuvieron todos los accidentes que sucedieron en una sola hora utilizando el árbol ordenado por fechas creado en la carga

de datos. La implementación de este requerimiento se divide en dos partes; la primera se encarga de encontrar el primer y último accidente de cada día, lo cual tiene una complejidad de O(logn), pues se basa en búsquedas binarias dentro del árbol de fechas para obtener el dato más cercano a 00:00:00 y a 23:59:59. Esto se consiguió usando las funciones ceiling y floor que tienen los árboles ordenados.

La segunda parte se encarga de crear el histograma de frecuencias, por lo que cuenta con un doble ciclo para iterar cada hora y calcular la cantidad de elementos dentro de cada lista de horas, lo que justifica la complejidad de n^2.

Lo anterior concuerda con los datos graficados, ya que al hacer una regresión polinomial de grado 2, se obtiene un coeficiente de relación R^2 de 0,94, lo cual es bastante fuerte.

Requerimiento <<8>>

```
def req_8(Analyzer,Clase,FechIn,FechaFi):
    """
    Función que soluciona el requerimiento 8
    """
    FechaInicial=datetime.strptime(FechIn, "%Y/%m/%d")
    FechaFinal=datetime.strptime(FechaFi, "%Y/%m/%d")
    MapaClase=Analyzer["clases"]
    Fechas=me.getValue(mp.get(MapaClase,Clase))
    FechasRango=om.values(Fechas,FechaInicial,FechaFinal)
    ListaFechas= lt.subList(FechasRango, 0,lt.size(FechasRango)-1)
    final=lt.newList(datastructure="ARRAY_LIST")
    for Requerimiento in lt.iterator(ListaFechas):
        for Formato in lt.iterator(Requerimiento):
            Form={
                "CODIGO_ACCIDENTE":Formato["CODIGO_ACCIDENTE"],
               "FECHA_HORA_ACC": Formato["FECHA_HORA_ACC"],
               "DIA_OCURRENCIA_ACC":Formato["DIA_OCURRENCIA_ACC"],
               "LOCALIDAD":Formato["LOCALIDAD"],
               "DIRECCION": Formato["DIRECCION"],
               "GRAVEDAD":Formato["GRAVEDAD"],
               "CLASE_ACC": Formato["CLASE_ACC"],
               "LATITUD":Formato["LATITUD"],
               "LONGITUD":Formato["LONGITUD"]
               }
            lt.addFirst(final, Formato)
    return ListaFechas
```

Descripción

En este requerimiento se buscaba emplear un mapa con los puntos de donde ocurrieron los accidentes de una clase especifica en un rango de años suministrado, de manera que recorre el mapa de clases de accidentes y entrando al suministrado pasando al árbol de fechas de dichos accidentes y extrayéndolos todos en su debido orden para al final ser añadidos a una lista el cual se ara uso para determinarlos puntos de ubicación de dichos accidentes.

Entrada	Clase, fecha inicial y fecha final
Salidas	El algoritmo retorna una lista (y su tamaño) con todos los accidentes de una clase especifica en un rango de fechas
Implementado (Sí/No)	Si

Estructura utilizada

Mapa -> Clases

Arbol->fechas

Análisis de complejidad

Pasos	**Complejidad**
Darle el formato de datetime a las fechas ingresadas como str por el usuario	O(1)
Extraer el mapa ordenado por clase de accidente del analyzer	O(1)
Extraer el valor asociado a la llave del mapa con la gravedad ingresada por parámetro. Este valor es un árbol ordenado por fechas.	O(1)
Extraer los valores del árbol que se encuentran entre las fechas ingresadas por el usuario	O(Logn)
Crear una sublista	O(1)
Se recorre con doble for la lista para dejar los datos en el formato que se pide,	0(k) k =constante menos que n
TOTAL	***O(Logn)+O(N)+O(1)O(logn)***

Pruebas Realizadas

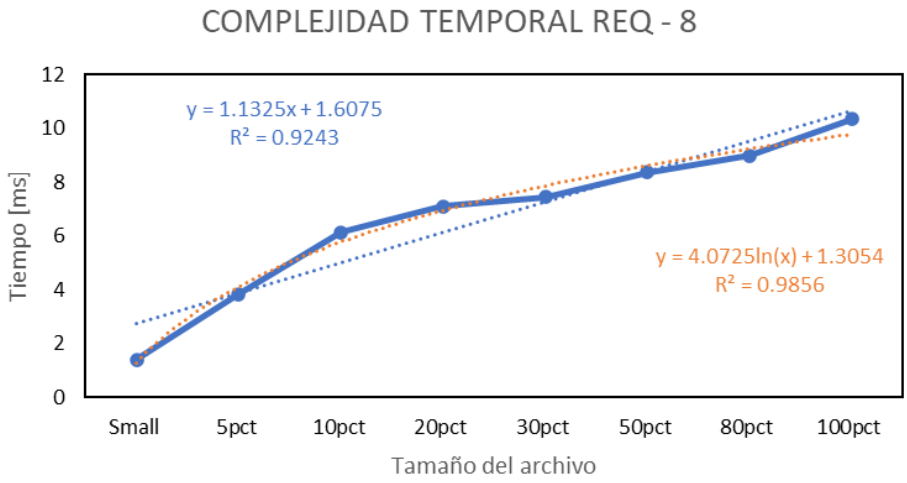
Las pruebas se realizaron para todos los archivos ingresando con los siguientes parámetros: ese uso choque como clase de accidente, el 20 diciembre del 2016 como fecha inicial y de final el 1 de diciembre del 2018. Se utilizó un computador con las siguientes especificaciones:

Procesadores	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
Memoria RAM	8 GB
Sistema operativo	Windows 11

Tablas de datos

Tamaño	Tiempo [ms]
1%	1.42
5%	3.84
10%	6.13
20%	7.11
30%	7.45
50%	8.37
80%	8.98
100%	10.33

Graficas



Análisis

Se puede analizar del procedimiento que hace bastante eficaz el recorrido del árbol de años dentro del mapa de clases de accidente debido a su rango de fechas determinadas haciendo que no tenga que obligatoriamente recorrer todo el árbol, además por mucho que se añada a otra lista para mostrar un formato más versátil para la extracción de información haciendo que la proyección del mapa sea más sencilla de producirse, cabe decir que el sistema fue complicado con la nueva librería de pythjon no antes utilizada, y más que todo con implementar los puntos de información de los lugares de accidentes. Pero aun así se hizo un recorrido de lugares dentro el view para proyectar cada punto en el mapa.