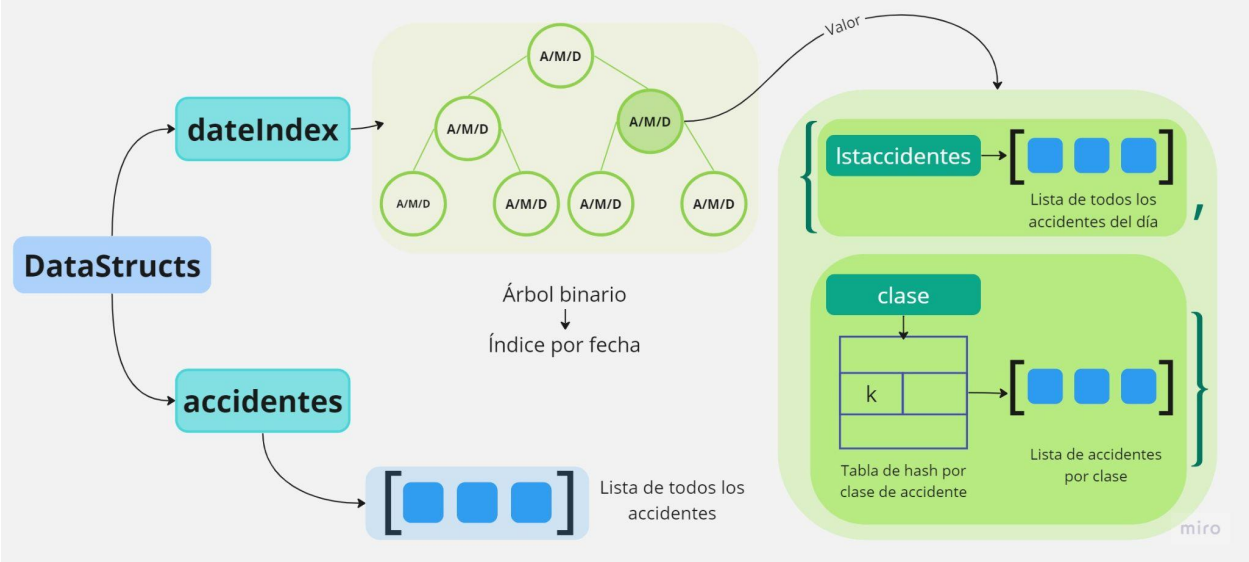
Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ANÁLISIS DEL RETO

Jacobo Zarruk, 202223913, j.zarruk@uniandes.edu.co

María José Amorocho, 202220179, m.amorocho@uniandes.edu.co

Diagrama carga de datos



Requerimiento 1

```
def req_1(data_structs,fecha_inicio,fecha_fin):
    """
    Función que soluciona el requerimiento 1
    """
    # TO DO: Realizar el requerimiento 1
    fecha_i = datetime.datetime.strptime(fecha_inicio, "%Y/%m/%d")
    fecha_in = fecha_i.date()
    fecha_f = datetime.datetime.strptime(fecha_fin, "%Y/%m/%d")
    fecha_fi = fecha_f.date()
    entry = om.values(data_structs["dateIndex"], fecha_in, fecha_fi)
    total_accidentes = 0
    lst_accidentes = lt.newList(datastructure='SINGLE_LINKED')
    for date in lt.iterator(entry):
        total_accidentes += lt.size(date["lstaccidentes"])
        merg.sort(date["lstaccidentes"],compareHour2)
        lt.addFirst(lst_accidentes, date["lstaccidentes"])

    return total_accidentes,lst_accidentes
```

Descripción

El requerimiento reporta todos los accidentes dado un rango de fechas. Para esto,

Entrada	Fecha inicial, fecha final
Salidas	Número de accidentes registrados entre el rango de fechas dado y su respectiva información.
Implementado (Sí/No)	Si. María José y Jacobo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Primero se toman las fechas brindadas por parámetro y se pasan en formato datetime para que puedan ser comparadas con las llaves del árbol organizado por días.	O(1)
Se usa la función om.values para recuperar, del árbol organizado por días, todas las llaves que se encuentran entre el rango de fechas solicitado.	O(1)
Se establece el total de accidentes como cero, y se crea una lista vacía tipo linked list que almacenará todas los accidentes ocurridos dentro del rango de fechas solicitado.	O(1)
Por cada llave del árbol organizado por días, se toma su valor y el índice 'lstaccidentes', una lista que contiene todos los accidentes ocurridos en ese día específico. Esta lista se ordena de forma descendente de acuerdo a la hora en que ocurrieron los accidentes y se agrega -en primer lugar- a la lista 'lst_accidentes'.	O(n)
Se retorna la lista 'lst_accidentes' (que es una lista de listas organizada del accidente más reciente al más antiguo, también teniendo en cuenta la hora).	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. El rango de fechas dado por parámetro fueron: 2016/11/01 - 2016/11/08.

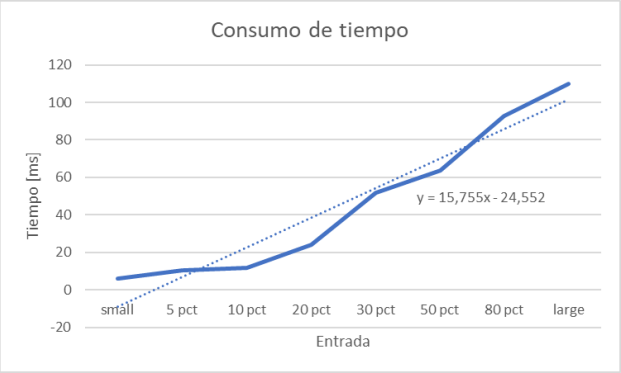
Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

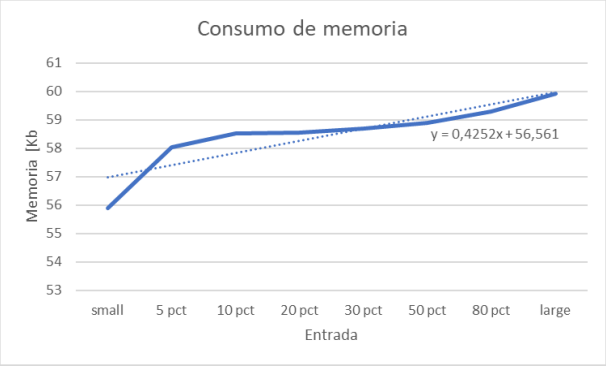
Entrada	Tiempo (s)	Memoria (Kb)
small	5.85	55.89
5 pct	10.64	58.04
10 pct	11.64	58.52
20 pct	24.24	58.55
30 pct	51.96	58.69
50 pct	63.66	58.90
80 pct	92.78	59.30
large	109.98	59.91

Tablas de datos

Muestra	Salida (Número de accidentes)	Tiempo (ms)	Memoria (Kb)
small	9	5.85	55.89
5 pct	46	10.64	58.04
10 pct	85	11.64	58.52
20 pct	181	24.24	58.55
30 pct	276	51.96	58.69
50 pct	435	63.66	58.90
80 pct	668	92.78	59.30
large	823	109.98	59.91

Gráficas





Análisis

Respecto al consumo de tiempo, se evidencia claramente una relación proporcional, con una tendencia lineal, entre el tamaño de la entrada y el crecimiento temporal. Se recalca que aunque el recorrido del árbol hecho al inicio del algoritmo tenga una complejidad de $c\ log(n)$, lo que agiliza las búsquedas, procedimientos como el recorrido interno de listas, aumentan la complejidad del código a O(n), un comportamiento teórico que se asimila a los valores obtenidos. En cuanto al consumo de memoria, la gráfica parece seguir un orden de crecimiento de carácter logarítmico, más que lineal.

Requerimiento 3

Descripción

```python
def req_2(data_structs, clase, via):
    """
    Función que soluciona el requerimiento 2
    """
    # TO DO: Realizar el requerimiento 2
    arbol = data_structs["dateIndex"]
    dias = om.keySet(arbol)
    lst_acc_via = lt.newList('ARRAY_LIST')
    for dia in lt.iterator(dias):
        key_value_dia = om.get(arbol,dia)
        mapa_clase = me.getValue(key_value_dia)["clase"]
        contains = mp.contains(mapa_clase,clase)
        if contains:
            acc_clase = mp.get(mapa_clase,clase)
            lst_acc_clase = me.getValue(acc_clase)
            for accidente in lt.iterator(lst_acc_clase):
                if via in accidente["DIRECCION"]:
                    lt.addLast(lst_acc_via,accidente)

    numero_acc = lt.size(lst_acc_via)
    lst_3_acc_recientes = lt.subList(lst_acc_via,numero_acc-2,3)
    merg.sort(lst_3_acc_recientes,sort_criteria_date)
    return lst_3_acc_recientes,numero_acc
```

En el requerimiento se reportan los 3 accidentes más recientes de una clase particular ocurridos a lo largo de una vía. Primero se buscan los accidentes de la clase particular por cada día, y se verifica si dentro de su dirección se encuentra la vía ingresada por parámetro; de ser así, el accidente se introduce en una lista, donde se toman los tres últimos datos.

Entrada	Clase del accidente, nombre de la vía de la ciudad
Salidas	El número total de accidentes ocurridos en la vía en cuestión que correspondan a la clase y los tres accidentes más recientes que cumplan con los anteriores requisitos
Implementado (Sí/No)	Implementado por María José

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Primero, se toma el árbol en donde se encuentran todos los accidentes (organizados por día) y se crea una lista con sus llaves (‘dias’), que corresponden a la fecha en la que se registraron accidentes.	O(1)
Se crea una lista vacía para almacenar todos los accidentes de una clase particular ocurridos a lo largo de una vía específica.	O(1)
Para cada día, se toma el índice donde están ordenados los accidentes de acuerdo a su clase. Este índice corresponde a un mapa donde cada llave es una clase de accidente y su valor es una lista de accidentes correspondientes a	O(n)

esa clase. Así pues, se busca la llave que corresponde a la clase de accidente entrada por parámetro y se extrae su valor ('lst_acc_clase').	
Para cada accidente dentro de la lista de accidentes por clase, se extrae su dirección para ver si el nombre de la vía entrada por parámetro se encuentra en la dirección. De ser así, el accidente se añade a la lista 'lst_acc_via', que guarda todos los accidentes de una clase particular ocurridos en una vía específica.	O(n)
De la lista 'lst_acc_via' sw obtiene su tamaño, que corresponde a la cantidad de accidentes. Además, se crea una sublista con los últimos tres elementos de 'lst_acc_via', que posteriormente se organiza de acuerdo a la fecha. Se recorta tanto el número de accidentes como los tres últimos accidentes organizados por fecha.	O(1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. La clase de accidente que se usó como dato de entrada fue 'choque', y como nombre de la vía 'Av Avenida de las Américas'.

Procesadores	**Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.10 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

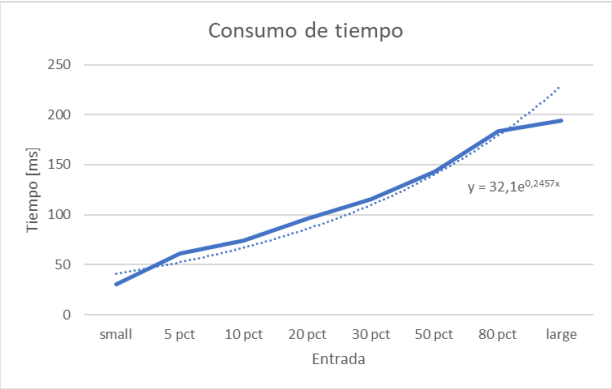
Entrada	Tiempo (ms)	Memoria (kb)
small	30.18	62.81
5 pct	61.54	63.04
10 pct	74.54	63.30
20 pct	95.69	63.77
30 pct	115.79	63.88
50 pct	143.12	64.02
80 pct	183.51	64.51
large	194.03	64.54

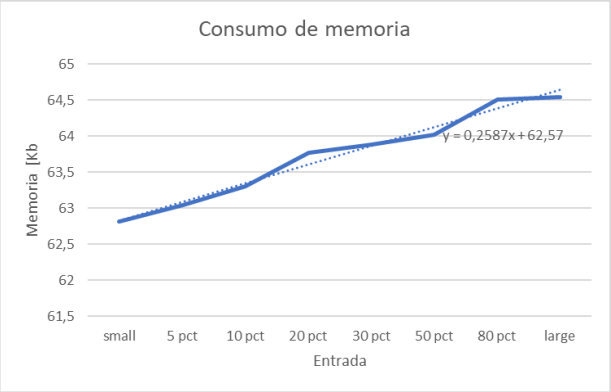
Tablas de datos

Muestra	Salida (Número de accidentes)	Tiempo (ms)	Memoria (Kb)
small	31	30.18	62.81
5 pct	166	61.54	63.04
10 pct	357	74.54	63.30
20 pct	694	95.69	63.77

30 pct	1052	143.12	63.88
50 pct	1739	115.79	64.02
80 pct	2811	183.51	64.51
large	3531	194.03	64.54

Gráficas





Análisis

Respecto al orden de crecimiento temporal, se puede apreciar una tendencia exponencial; esto se debe a los dos recorridos que se hacen en el algoritmo: el primero para obtener el índice de clase del accidente por cada día, y el segundo para verificar si dentro de los accidentes de esa clase en particular se encuentra la dirección introducida por parámetro. No obstante, se recalca que el uso de un subíndice en donde se clasifican los accidentes del día por su clase, es un factor que ayuda a que el orden de crecimiento disminuya un poco, pues no se debe iterar sobre todos los accidentes ocurridos en el día para verificar si son de una clase especial, si no sobre aquellos sobre los que ya se tiene certeza que pertenecen a una clase particular. Por otro lado, el consumo de memoria sigue una tendencia entre lineal y logarítmica. Se destaca que, aunque el tamaño de entrada cambie considerablemente, el consumo de memoria no aumenta de manera tan distintiva.

Requerimiento 4

```python
def req_4(data_structs, gravedad, fecha_in, fecha_fi):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    top = lt.newList(datastructure='ARRAY_LIST')
    fecha_i = datetime.datetime.strptime(fecha_in, "%Y/%m/%d")
    fecha_in = fecha_i.date()
    fecha_f = datetime.datetime.strptime(fecha_fi, "%Y/%m/%d")
    fecha_fi = fecha_f.date()
    accidentes = om.values(data_structs["dateIndex"], fecha_in,fecha_fi)
    for accident in lt.iterator(accidentes):
        for accidente in lt.iterator(accident["lstaccidentes"]):
            if accidente["GRAVEDAD"] == gravedad:
                lt.addFirst(top, accidente)
    top_5 = lt.newList(datastructure='ARRAY_LIST')
    info = ["CODIGO_ACCIDENTE","FECHA_HORA_ACC","DIA_OCURRENCIA_ACC","LOCALIDAD","DIRECCION","CLASE_ACC","LATITUD","LONGITUD"]
    for acci in lt.iterator(lt.subList(top,1,5)):
        lt.addLast(top_5,{})
        for i in info:
            top_5["elements"][lt.size(top_5)-1][i]=acci[i]
    return top_5, lt.size(top)
```

Descripción

En este requerimiento se reportan los 5 accidentes más dada una gravedad y un rango de fechas dados por el usuario. Primero uso la librería de datetime para poder tener los accidentes que suceden dentro del rango de fechas, después utilizó iterator dos veces, el primero para tomar todos los accidentes de una fecha y el segundo otro para tomar uno por uno de los accidentes de la fecha, a continuación comparo la gravedad del accidente para saber si es la misma a la dada por parámetro y teniendo en cuenta que se encuentran organizados en forma ascendente, en cuanto a la fecha, hago un addFirst para tenerlos en orden descendente, o sea teniendo primero el accidente más reciente. Finalmente utilizo un iterator con un subList para tomar los 5 primeros accidentes y de estos tomar, con otro for, los datos necesarios para la correcta impresión de la tabla.

Entrada	Gravedad de los accidentes, fecha de inicio y fecha de finalización.
Salidas	El top 5 de los accidentes más recientes ocurridos entre las fechas dadas y con la gravedad ingresada y la cantidad de accidentes ocurridos entre las fechas dadas.
Implementado (Sí/No)	Implementado por Jacobo.

Análisis de complejidad

Análisis de la complejidad de cada uno de los pasos del algoritmo.

Pasos	**Complejidad**
Se crea una nueva lista vacía.	O(1)
Se toman las fechas brindadas por parámetro y se pasan en formato datetime para que puedan ser comparadas con las llaves del árbol organizado por días.	O(1)

Se usa la función om.values para obtener, del árbol organizado por días, todas las llaves que se encuentran entre el rango de fechas solicitado.	O(1)
Se miran todos los accidentes en cada una de las fechas dentro del rango preestablecido con un lt.iterator().	O(N)
Se mira cada accidente en cada una de las fechas, dadas por el anterior lt.iterator(), con otro lt.iterator().	O(N)
Se añade el accidente a la lista primeramente creada, si este tiene la misma gravedad que la dada por parámetro.	O(N)
Se crea una nueva lista vacía.	O(1)
Se miran los primeros 5 accidentes, que son los 5 más recientes que cumplen con las características dadas por parámetro.	O(5)
Para cada uno de los accidentes se mira cada llave necesaria para la impresión.	O(9)
TOTAL	***O(N^2)***

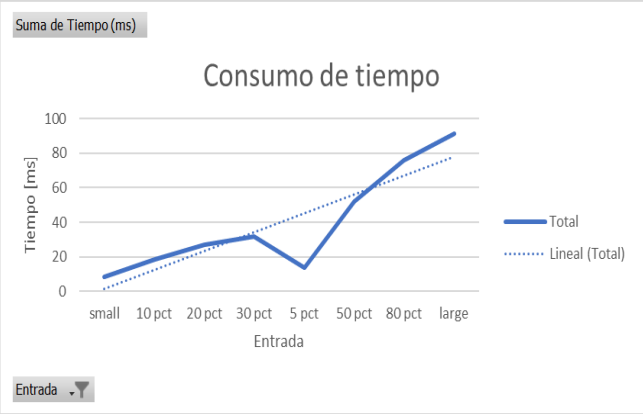
Pruebas Realizadas

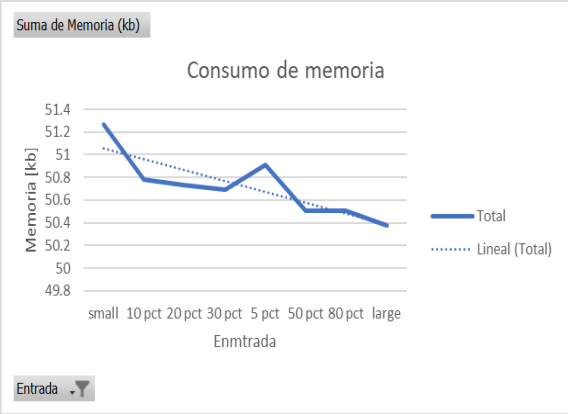
Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. La clase de accidente que se usó como dato de entrada fue 'CON MUERTOS', y el rango de fechas fueron '2016/10/01' y '2018/10/01'.

Procesadores	**AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11 Home Single Language

Entrada	Tiempo (ms)	Memoria (kb)
small	8.32	51.27
5 pct	13.56	50.91
10 pct	18.55	50.78
20 pct	26.69	50.73
30 pct	31.85	50.69
50 pct	51.84	50.51
80 pct	75.71	50.51
large	91.17	50.38

Gráficas





Requerimiento 6

```
def req_6(data_structs,mes,año,latitud,longitud,radio,cantidad):
    """
    Función que soluciona el requerimiento 6
    """
    # TO DO: Realizar el requerimiento 6
    arbol = data_structs["dateIndex"]
    meses = {"ENERO":1,"FEBRERO":2,"MARZO":3,"ABRIL":4,"MAYO":5,"JUNIO":6,"JULIO":7,"AGOSTO":8,"SEPTIEMBRE":9,"OCTUBRE":10,
             "NOVIEMBRE":11,"DICIEMBRE":12}
    mess = meses[mes]
    if mes == "ENERO" or mes == "MARZO" or mes == "MAYO" or mes =="JULIO" or mes == "AGOSTO" or mes == "OCTUBRE" or mes == "DICIEMBRE":
        dia_fin = 31
    elif mes == "FEBRERO":
        dia_fin = 28
    else:
        dia_fin = 30
    fecha_i = datetime.datetime(año,mess,1)
    fecha_f = datetime.datetime(año,mess,dia_fin)
```

```
dias = om.values(arbol,fecha_i.date(),fecha_f.date())
lst_acc = lt.newList(datastructure='ARRAY_LIST')
for entry in lt.iterator(dias):
    lst_accidents = entry["lstaccidentes"]
    for accidente in lt.iterator(lst_accidents):
        lat2 = math.radians(float(accidente["LATITUD"]))
        lon2 = math.radians(float(accidente["LONGITUD"]))
        lat1 = math.radians(latitud)
        lon1 = math.radians(longitud)
        sin2 = (math.sin((lat2-lat1)/2))**2
        part2 = math.cos(lat1)*math.cos(lat2)*(math.sin((lon2-lon1)/2))**2
        distancia = 2 * math.asin(math.sqrt(sin2 + part2)) * 6371
        if distancia <= radio:
            accidente["Distancia"] = distancia
            lt.addLast(lst_acc,accidente)
merg.sort(lst_acc,compareDistancia)
if cantidad <= lt.size(lst_acc):
    sub_lst = lt.subList(lst_acc,1,cantidad)
else:
    sub_lst = 0
return sub_lst
```

Descripción

La función muestra los N accidentes ocurridos dentro de una zona específica para un mes y un año. Primero se buscan los accidentes dentro del rango de fechas, para luego calcular la distancia de cada uno con la zona dada y verificar que se encuentren dentro del área solicitada. Los datos se ordenan de mayor a menor cercanía, y solo se retorna la cantidad de accidentes dada por parámetro.

Entrada	Mes, año, latitud y longitud del centro de área, radio del área, número de accidentes
Salidas	Número n de accidentes ocurridos dentro de una zona específica para un mes y un año
Implementado (Sí/No)	Implementado por María José y Jacobo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Se toma el árbol donde están organizados los accidentes por día.	O(1)
Se establece un diccionario con los meses del año y su valor numérico, así como la cantidad de días que tiene cada uno para poder buscar las llaves del árbol que contengan estos datos; es decir, para buscar todos los días del mes en el año dado por parámetro.	O(1)
Se crea una nueva lista 'lst_acc' que guardará todos los accidentes ocurridos dentro del área solicitada por parámetro.	O(1)
El árbol con los accidentes organizados por día se recorre desde el primer día del mes dado, hasta su último día (teniendo en cuenta el año al que corresponde). Para cada día en el árbol se toma el índice de 'lstaccidentes', que contiene una lista con todos los accidentes ocurridos en ese día.	O(n)
Para cada accidente en 'lstaccidentes' se toma su longitud y latitud, y se calcula la distancia de éste con respecto al centro de área introducido por parámetro. Si la distancia está dentro del radio dado, al accidente se le crea un nuevo índice nombrado 'Distancia', que guarda la distancia calculada entre las coordenadas dadas por parámetro y el punto donde ocurrió el accidente.	O(n)
Una vez añadido el índice de distancia, el accidente se añade en la lista 'lst_acc'.	O(n)
La lista 'lst_acc' se ordena de forma ascendente respecto al índice de distancia de cada accidente.	O(n)
Se compara la cantidad de elementos en la lista 'lst_acc' con la cantidad de accidentes pedidos por parámetro. Si es posible obtener la cantidad de elementos solicitados, se retorna una sublista con la cantidad de accidentes pedidos por parámetro (desde la primera posición hasta la posición n). De otra forma se retorna 0, indicando que no es posible obtener el top debido a que no existen suficientes elementos.	O(1)
TOTAL	***O(n^2)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron: mes: Enero, año: 2022, latitud: 4.674, longitud: -74.068, radio: 5, top: 3.

Procesadores	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.10 GHz
Memoria RAM	8 GB
Sistema Operativo	Windows 10

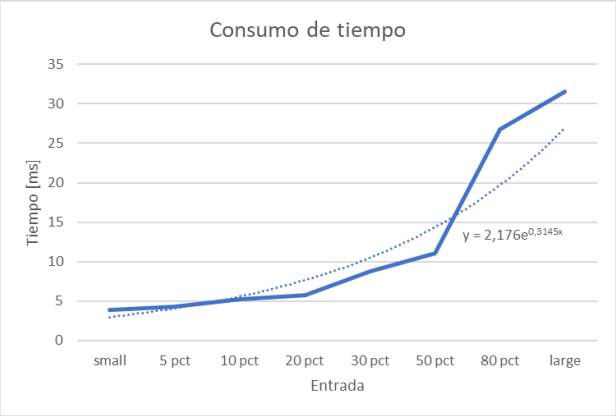
Entrada	Tiempo (s)	Memoria (Kb)
small	3.90	48.34
5 pct	4.33	57.27
10 pct	5.21	62.89
20 pct	5.76	64.04
30 pct	8.81	64.18
50 pct	11.02	64.29
80 pct	26.77	64.42
large	31.58	65.13

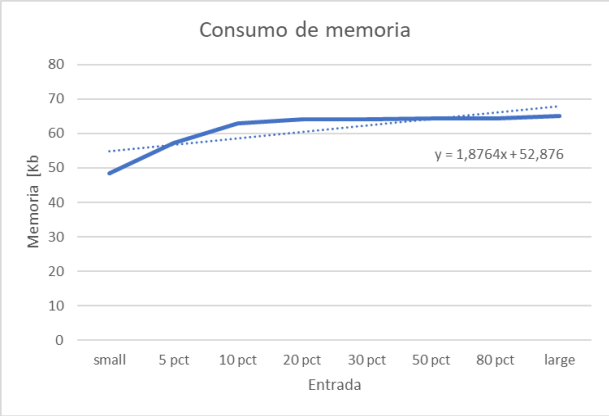
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida (Códigos de accidentes)	Tiempo (ms)	Memoria (Kb)
small	10562094 , 10561695, 10561926	3.90	48.34
5 pct	10560125, 10560392, 10560181	4.33	57.27
10 pct	10560607,10560125,10560936	5.21	62.89
20 pct	10561600, 10562382, 10560607	5.76	64.04
30 pct	10561600, 10562382, 10560362	8.81	64.18
50 pct	10561600, 10562382, 10560362	11.02	64.29
80 pct	10561600, 10562382, 10560362	26.77	64.42
large	10561600, 10562382, 10560362	31.58	65.13

Gráficas





Análisis

En relación con el consumo de tiempo, se ve claramente que el algoritmo tiene una tendencia a seguir un orden de crecimiento de $O(n^2)$, lo que concuerda con los valores teóricos esperados. Lo anterior puede deberse a que, en este caso en particular, se deben recorrer todos los accidentes ocurridos dentro de un mes, sin importar su clase o gravedad, lo que no permite hacer una filtración y disminuir el tamaño de datos a iterar. Respecto al consumo de memoria, se observa que esta mantiene un orden de crecimiento aproximadamente logarítmico. Se intuye que uno de los factores que contribuye a esto es cuando se compara la distancia obtenida con el rango de área solicitada, pues de no cumplir con este parámetro, el accidente no es añadido a la lista y es 'desechado'.

Requerimiento 7

```python
def req_7(data_structs,mes,ano):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    accidents = {}
    cantidad = 0
    horas = {"0:00:00": 0, "1:00:00": 0, "2:00:00": 0, "3:00:00": 0, "4:00:00": 0, "5:00:00": 0, "6:00:00": 0, "7:00:00": 0, "8:00:00": 0, "9:00:00": 0, "
    meses = {"ENERO": "01", "FEBRERO": "02", "MARZO": "03", "ABRIL": "04", "MAYO": "05", "JUNIO": "06", "JULIO": "07", "AGOSTO": "08", "SEPTIEMBRE": "09",
    fecha_i = datetime.datetime.strptime(ano+"/"+meses[mes]+"/01", "%Y/%m/%d")
    fecha_in = fecha_i.date()
    fecha_f = datetime.datetime.strptime(ano+"/"+meses[mes]+"/31", "%Y/%m/%d")
    fecha_fi = fecha_f.date()
    accidentes = om.values(data_structs["dateIndex"], fecha_in,fecha_fi)
    for accident in lt.iterator(accidentes):
        for accidente in lt.iterator(accident["lstaccidentes"]):
            if accidente["FECHA_OCURRENCIA_ACC"] not in accidents:
                accidents[accidente["FECHA_OCURRENCIA_ACC"]] = lt.newList(datastructure='ARRAY_LIST')
                lt.addLast(accidents[accidente["FECHA_OCURRENCIA_ACC"]], accidente)
            else:
                lt.addLast(accidents[accidente["FECHA_OCURRENCIA_ACC"]], accidente)
            cantidad += 1
            if ":" in accidente["HORA_OCURRENCIA_ACC"][:2]:
                horas[accidente["HORA_OCURRENCIA_ACC"][:1] + ":00:00"] += 1
            else:
                horas[accidente["HORA_OCURRENCIA_ACC"][:2] + ":00:00"] += 1
    primero_ultimo = {}
    info = ["CODIGO_ACCIDENTE","FECHA_HORA_ACC","DIA_OCURRENCIA_ACC","LOCALIDAD","DIRECCION","GRAVEDAD","CLASE_ACC","LATITUD","LONGITUD"]
    for fecha in accidents:
        primero_ultimo[fecha] = lt.newList(datastructure='ARRAY_LIST')
        sa.sort(accidents[fecha],sort_req7)
        lt.addLast(primero_ultimo[fecha],{})
        for inf in info:
            primero_ultimo[fecha]["elements"][0][inf] = lt.firstElement(accidents[fecha])[inf]
        lt.addLast(primero_ultimo[fecha],{})
        for inf in info:
            primero_ultimo[fecha]["elements"][1][inf] = lt.lastElement(accidents[fecha])[inf]
    return primero_ultimo, horas, cantidad
```

Descripción

La función del requerimiento 7 recibe una estructura de datos que contiene información sobre accidentes y procesa esta información para obtener los primeros y últimos accidentes por fecha, así como un conteo de accidentes por hora. Utiliza diccionarios para almacenar y organizar los datos. El código itera sobre la lista de accidentes, realiza operaciones de inserción y actualización en los diccionarios, y finalmente devuelve los resultados obtenidos. En general, la complejidad del código es lineal en función del número de accidentes y el número de fechas presentes en los accidentes.

Entrada	Mes a analizar y año a analizar.
Salidas	El primer y último accidente de cada día del mes y año ingresados, un diccionario con la cantidad de accidentes y la cantidad total de accidentes.
Implementado (Sí/No)	Implementado por María José y Jacobo.

Análisis de complejidad

Análisis de la complejidad de cada uno de los pasos del algoritmo.

Pasos	Complejidad
Inicialización de variables y diccionarios.	O(1)

Creación de objetos ‘datetime’ y conversiones de fecha.	O(1)
Bucle principal.	O(N)
Operaciones dentro del bucle principal.	O(1)
Bucle final y operaciones adicionales.	O(m)
Ordenamiento de listas.	O(m * log(m))
TOTAL	***O(n + m * log(m))***

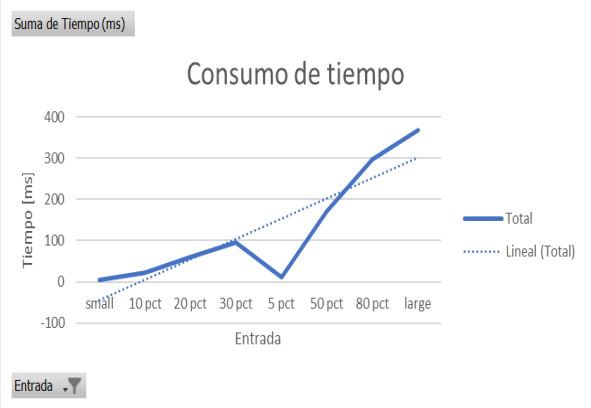
Pruebas Realizadas

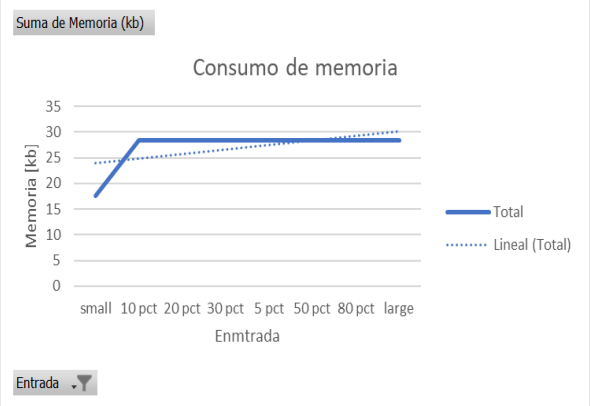
Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. La clase de accidente que se usó como dato de entrada fue el mes de ‘DICIEMBRE’, y año ‘2019’.

Procesadores	**AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11 Home Single Language

Entrada	Tiempo (ms)	Memoria (kb)
small	4.63	17.53
5 pct	10.43	28.32
10 pct	21.71	28.41
20 pct	59.44	28.41
30 pct	94.95	28.41
50 pct	171.23	28.41
80 pct	297.28	28.41
large	367.06	28.41

Gráficas





Requerimiento 8

```python
def req_8(data_structs,clase,fecha_in,fecha_fi):
    """
    Función que soluciona el requerimiento 8
    """
    # TODO: Realizar el requerimiento 8
    cantidad = 0
    accs = lt.newList(datastructure='ARRAY_LIST')
    fecha_i = datetime.datetime.strptime(fecha_in, "%Y/%m/%d")
    fecha_in = fecha_i.date()
    fecha_f = datetime.datetime.strptime(fecha_fi, "%Y/%m/%d")
    fecha_fi = fecha_f.date()
    accidentes = om.values(data_structs["dateIndex"], fecha_in,fecha_fi)
    for accident in lt.iterator(accidentes):
        for accidente in lt.iterator(accident["lstaccidentes"]):
            if accidente["CLASE_ACC"] == clase:
                lt.addLast(accs, accidente)
                cantidad += 1
    return cantidad,accs
```

Descripción

La función del requerimiento 8 recibe una estructura de datos que contiene información sobre accidentes, junto con una clase de accidente, una fecha de inicio y una fecha de fin. El código busca los accidentes que pertenecen a la clase proporcionada y están dentro del rango de fechas especificado. Luego, devuelve la cantidad total de accidentes encontrados y una lista con los detalles de cada accidente. En resumen, la función busca y recopila información específica de accidentes según la clase y las fechas proporcionadas. La complejidad del código depende del número de accidentes en el rango de fechas y puede considerarse lineal.

Entrada	Gravedad de los accidentes, fecha de inicio y fecha de finalización.
Salidas	La cantidad de accidentes en el rango de fecha y que tienen la misma clase a la dada.
Implementado (Sí/No)	Implementado por María José y Jacobo.

Análisis de complejidad

Análisis de la complejidad de cada uno de los pasos del algoritmo.

Pasos	**Complejidad**
Inicialización de variables y estructuras de datos.	O(1)
Conversiones de fecha	O(1)

Bucle principal.	O(N)
Operaciones dentro del bucle principal.	O(1)
TOTAL	***O(N)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. La clase de accidente que se usó como dato de entrada fue el mes de 'DICIEMBRE', y año '2019'.

Procesadores	**AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz**
Memoria RAM	16.0 GB
Sistema Operativo	Windows 11 Home Single Language

Entrada	Tiempo (ms)	Memoria (kb)
small	4.12	8.81
5 pct	10.35	26.72
10 pct	14.03	50.19
20 pct	23.93	97.75
30 pct	36.08	137.56
50 pct	55.79	244.84
80 pct	83.18	389.84
large	109.61	492.38

Gráficas