# ANÁLISIS DEL RETO

*Isabella Vizcaíno López, 202213064, i.vizcaino@uniandes.edu.co*

*Adriana Velásquez, 202026792, am.velasquezm1@uniandes.edu.co*

*Daniel Daza, 202215047, d.dazao@uniandes.edu.co*

## Carga de Datos

### Descripción

(imagen)

(imagen-esquema de la estructura de datos)

La carga de datos que implementamos para este reto es mucho más compleja que para los retos anteriores. Decidimos hacer un data structure dividido en 4 partes: *siniestros*, *fechas, años y clases*. Siniestros es una ARRAY_LIST con todos los accidentes. Fechas es un árbol RBT donde cada llave es una fecha (año, mes, día, hora, minuto) y su respectivo nodo es una lista con los accidentes que ocurrieron en esa fecha. Años es una taba de Hash por año donde sus valores son tablase Hash por mes, y los valores de estos son arboles RBT donde las llaves son la hora (hora, minuto, segundo) y el valor es una lista con los respectivos accidentes del año y el mes respectivo. Y clases es un hash por clases de accidentes donde cada clase tiene un árbol RBT donde la llave es la hora (hora, minuto, segundo) y su respectivo valor es una lista con los accidentes que son de la respectiva clase y ocurrieron en una respectiva hora.

## Requerimiento <<1>>

### Descripción

```
153  def req_1(data_structs, initialDate, finalDate):
154      """
155      gets accidents by range
156      """
157
158      lst = om.values(data_structs["fechas"], initialDate, finalDate)
159      num_siniestros = 0
160
161      for lstdate in lt.iterator(lst):
162          num_siniestros += lt.size(lstdate["lstaccidentes"])
163
164      return num_siniestros, lst
165
```

Este requerimiento se encarga de retornar todos los accidentes en un rango de fechas dado. Lo primero que hace es obtener todos los accidentes en el rango de fechas. Para después contar cada elemento de la lista y retornar el número de accidentes junto a la lista con todos los accidentes.

| | |
|---|---|
| **Entradas** | Diccionario con las estructuras de datos, fecha inicial y fecha final. |
| **Salidas** | Lista con los accidentes y la cantidad de accidentes de la lista. |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener lista por rango de fechas. (om.values) | O(n) |
| Iterar por cada elemento de la lista obtenida. | O(n) |
| ***TOTAL*** | ***O(n)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

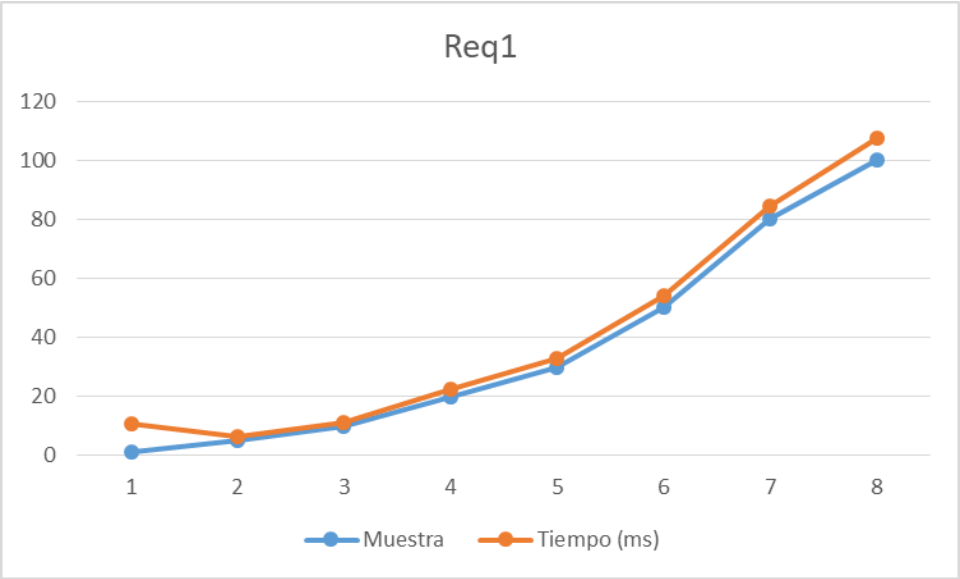
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Tiempo (ms) |
|---|---|
| small | 9,66 |
| 5 pct | 1,613 |
| 10 pct | 0,998 |
| 20 pct | 2,379 |
| 30 pct | 2,669 |
| 50 pct | 4,159 |
| 80 pct | 4,323 |
| large | 7,359 |

## Graficas




## Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.

# Requerimiento <<2>>

## Descripción

```python
def req_2(data_structs, año, mes, hora_i, hora_f):
    """
    Función que soluciona el requerimiento 2
    """
    num_siniestros = 0

    mes_kv = mp.get(data_structs["años"], año)
    mes_hash = me.getValue(mes_kv)
    hora_kv = mp.get(mes_hash["meses"], mes)
    hora_tree = me.getValue(hora_kv)
    lst = om.values(hora_tree["horas"], hora_i, hora_f)
    for lstdate in lt.iterator(lst):
        num_siniestros += lt.size(lstdate["lstaccidentes"])

    return num_siniestros, lst
```

Este requerimiento se encarga de reportar todos los accidentes en un intervalo de horas del día para un mes y año dados. Para resolver este requerimiento se busca el año y el mes por las tablas de Hash de la carga de datos, para usar el árbol con los accidentes del año y del mes. Con este árbol fácilmente obtenemos el rango de accidentes por fecha inicial y fecha final.

| | |
|---|---|
| **Entrada** | La estructura de datos, el año, el mes, la hora inicial para el rango y la hora final para el rango. |
| **Salidas** | Lista con los accidentes que ocurrieron en el año, mes, y en el rango de fechas ingresadas. Además, devuelve la cantidad de accidentes. |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Paso 1 | O(...) |
| Paso 2 | O(...) |
| Paso …. | O(...) |
| ***TOTAL*** | ***O(...)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |

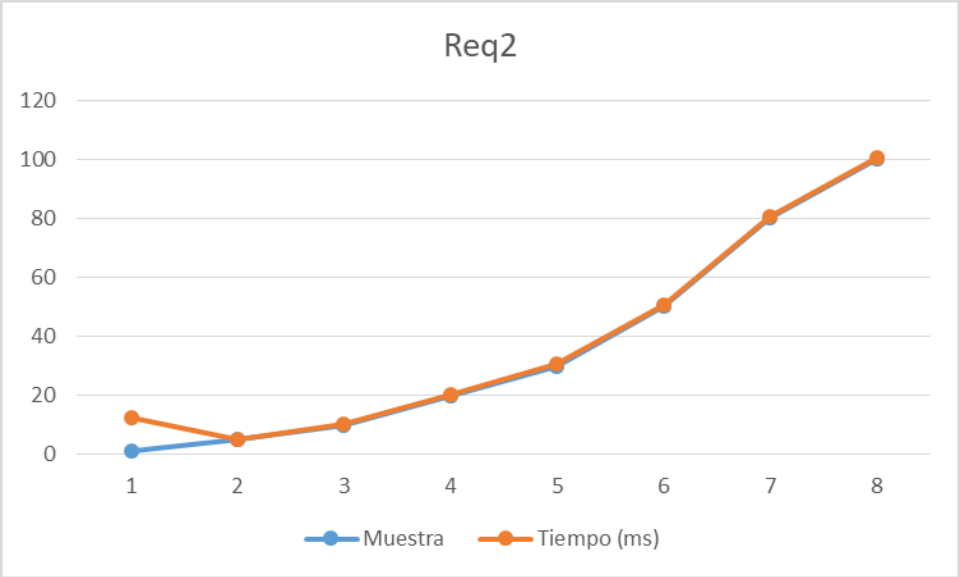
| | |
|---|---|
| **Sistema Operativo** | Windows 11 |

## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Tiempo (ms) |
|---|---|
| small | 11,402 |
| 5 pct | 0,222 |
| 10 pct | 0,429 |
| 20 pct | 0,291 |
| 30 pct | 0,708 |
| 50 pct | 0,632 |
| 80 pct | 0,674 |
| large | 0,69 |

## Graficas




## Análisis

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

# Requerimiento <<3>> - Adriana Velásquez

El requerimiento 3 busca reportar los 3 accidentes más recientes de una clase particular ocurridos a lo largo de una vía. Para este fin tiene como entrada una estructura de datos, una clase y una vía. Esta estructura de datos tiene un mapa con llave clases" y como valor tiene un mapa ordenado. Este mapa tiene como llave las fechas y como valor una lista de accidentes ocurridos en dichas fechas. Posteriormente, para cada lista en las listas y para cada accidente dentro de la lista, si la vía requerida está contenida dentro del nombre de la vía del accidente entonces se adiciona el accidente a una lista de accidentes. Se suma este accidente al número de accidentes. Por último, se crea una sublista para encontrar los 3 más recientes. Se retorna el número de accidentes y la lista con los accidentes

```
def req_3(data_structs, clase, via):
    """
    Reportar los 3 accidentes más recientes de una clase
    particular ocurridos a lo largo de una vía

    tiene una esctructura de datos como entrada, una clase y una vía

    retorna el numero de accidentes y una lista con el ranking
    """

    # TODO: Realizar el requerimiento 3
    num_siniestros = 0
    data = lt.newList("ARRAY_LIST")

    mes_kv = mp.get(data_structs["clases"], clase)
    tree = me.getValue(mes_kv)
    lst = om.valueSet(tree["dates"])
    for lstdate in lt.iterator(lst):
        for accidente in lt.iterator(lstdate["lstaccidentes"]):

            if via in accidente["DIRECCION"]:
                num_siniestros += lt.size(lstdate["lstaccidentes"])

                lt.addLast(data, accidente)

    rank = lt.subList(data, -2, 3)

    return num_siniestros, rank
```

| | |
|---|---|
| **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.<br><br>Los 3 accidentes más recientes |
| **Implementado (Sí/No)** | Sí, Adriana Velásquez |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|

| | |
|---|---|
| Encontrar la llave valor del hash por clases para una clase en especifico | O(1) |
| Obtener el valor a partir de la llave valor | O(1) |
| Obtener los valores del Árbol de fechas | O(N) |
| Iterar para cada accidente para encontrar los accidentes que estén en la vía | O(N) |
| Contar los accidentes | O(N) |
| Agregar los accidentes al final de la lista | O(N) |
| Encontrar la sublista con los 3 accidentes más recientes | O(N) |
| ***TOTAL*** | ***O(N)*** |

# Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |
| **Procesador** | Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz |

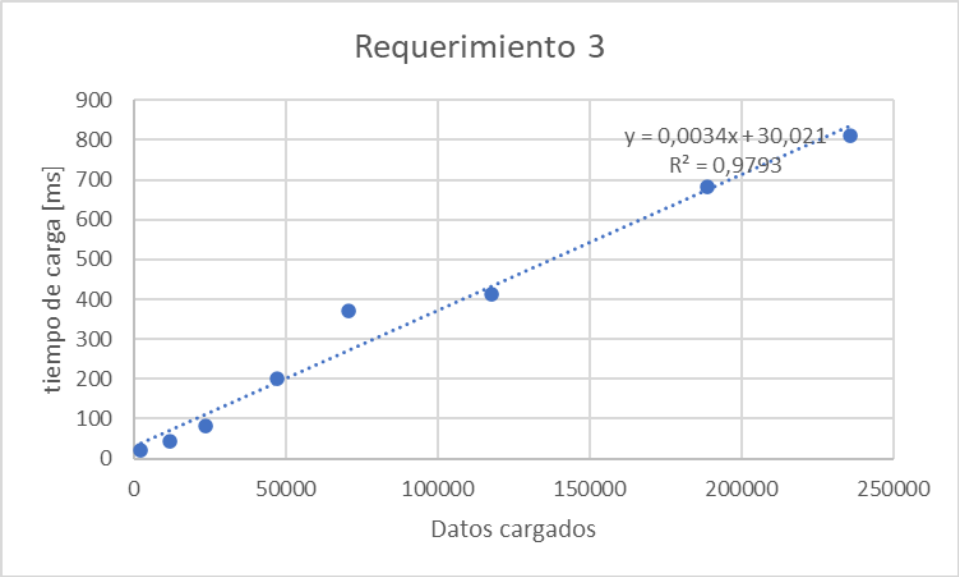
## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Datos | Tiempo (ms) |
|---|---|---|
| 1 | 2358 | 19,76 |
| 2 | 11790 | 42,8 |
| 3 | 23580 | 83,131 |
| 4 | 47160 | 201,192 |
| 5 | 70740 | 370,86 |
| 6 | 117900 | 412,455 |
| 7 | 188640 | 684 |
| 8 | 235801 | 811,205 |

## Graficas

Las gráficas con la representación de las pruebas realizadas.




## Análisis

Se realizo una regresión lineal teniendo en cuenta los datos de tiempo de carga para los 8 distintos tamaños de archivos.  Como era de esperarse, la función es aproximadamente lineal. Esto se observa ya que los datos se acoplan al ajuste lineal realizado a través de la ecuación y = 0,0034x + 30,021 además se encontró que el coeficiente de correlación lineal r^2 = 0.9793. Dado que es cercano a 1 los datos tienen una alta relación con la relación de 0(N) teórica encontrada anteriormente. Esto se debe a que en la estructura de datos se utilizaron tablas de hash y mapas ordenados reduciendo el tiempo de ejecución a diferencia de los laboratorios pasados.

# Requerimiento <<4>> - Daniel Daza

## Descripción

El requerimiento solicita los 5 accidentes más recientes para un intervalo de fechas y una gravedad dadas. En primer lugar, se saca una lista con los accidentes ocurridos en el intervalo, luego se guarda en una lista los accidentes cuya gravedad sea la solicitada y por último se saca una sublista con los 5 más recientes si existieron más de 5 accidentes, si existieron 5 o menos simplemente se retorna la lista.

```python
def req_4(data_base, fi, ff, grav):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    acc_range = om.values(data_base, fi, ff)
    grav_acc = lt.newList()
    for accident in lt.iterator(acc_range):
        for i in lt.iterator(accident["lstaccidentes"]):
            if i["GRAVEDAD"] == grav:
                lt.addFirst(grav_acc, i)
    if lt.size(grav_acc) <= 5:
        return grav_acc
    else:
        return lt.subList(grav_acc, 1, 5)
```

| | |
|---|---|
| **Entrada** | Árbol RBT, Fecha Inicial, Fecha Final, Gravedad |
| **Salidas** | Lista con los 5 accidentes más recientes |
| **Implementado (Sí/No)** | Si – Daniel Daza |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener una lista con el rango de fechas con om.values() | O(om.values()) |
| Iterar la lista del rango de fechas para encontrar los que tengan la gravedad solicitada | O(N) |
| Retornar la sublista | O(lt.sublist()) |
| ***TOTAL*** | ***O(N)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

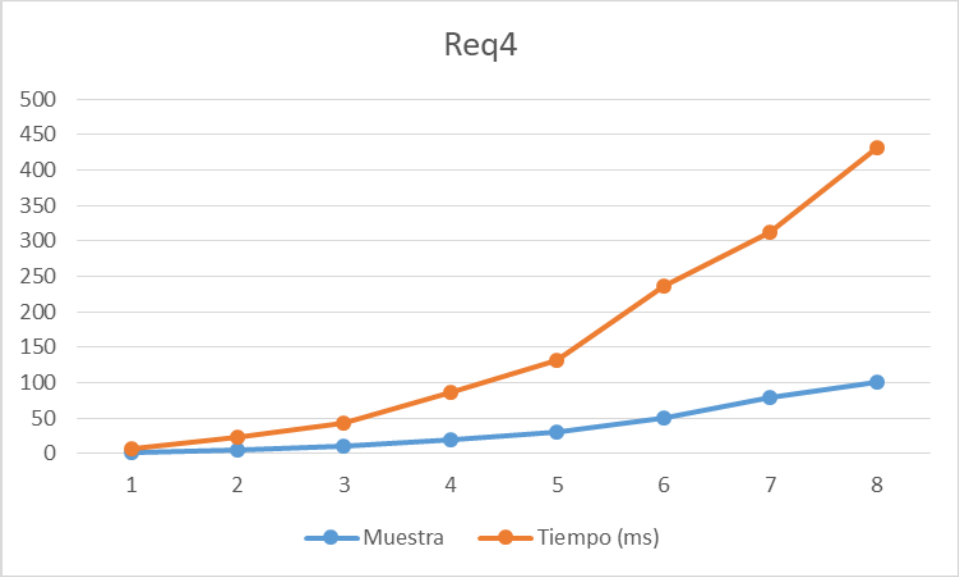
### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 5,398 |

| | |
|---|---|
| 5 pct | 17,778 |
| 10 pct | 32,458 |
| 20 pct | 66,855 |
| 30 pct | 101,923 |
| 50 pct | 186,009 |
| 80 pct | 232,431 |
| large | 330,584 |

### Graficas




### Análisis

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

## Requerimiento <<5>> - Isabella Vizcaíno

### Descripción

Este requerimiento se encarga de reportar los 10 accidentes menos recientes ocurridos en un mes y año para una localidad de la ciudad.

| | |
|---|---|
| **Entrada** | Parámetros necesarios para resolver el requerimiento. |
| **Salidas** | Respuesta esperada del algoritmo. |
| **Implementado (Sí/No)** | Si – Isabella Vizcaíno |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Paso 1 | O(...) |
| Paso 2 | O(...) |
| Paso …. | O(...) |
| ***TOTAL*** | ***O(...)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

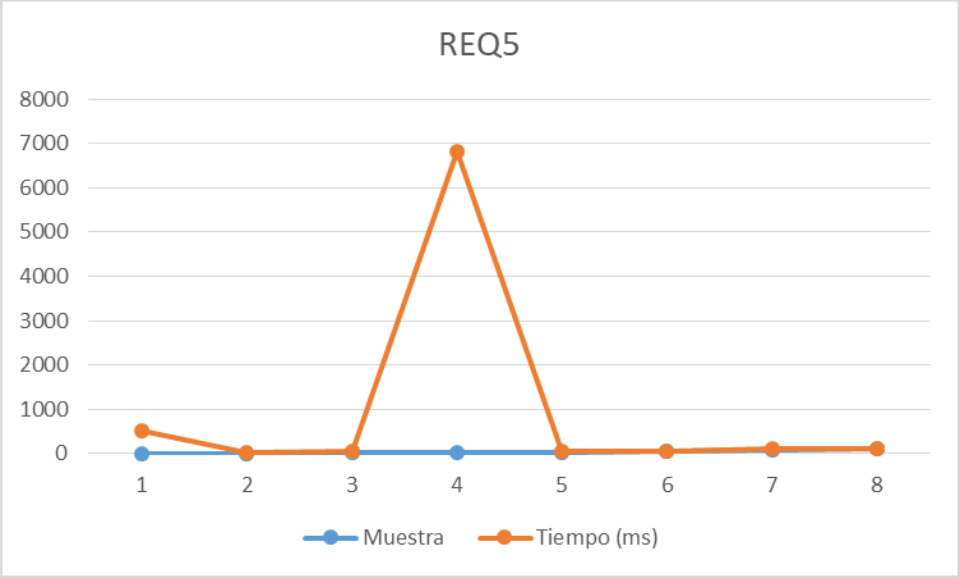
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 36,76 |
| 5 pct | 28,958 |
| 10 pct | 28,96 |
| 20 pct | 6794,61 |
| 30 pct | 33,11 |
| 50 pct | 8,057 |
| 80 pct | 15,351 |
| large | 22,392 |

## Graficas




## Análisis

# Requerimiento <<6>>

## Descripción

```python
def req_6(data_structs, año, mes, latitud, longitud, radio, N):

    # TODO: Realizar el requerimiento 6

    centro = (latitud, longitud)
    num_siniestros = 0
    radrank = 0
    data = lt.newList()

    mes_kv = mp.get(data_structs["años"], año)

    mes_hash = me.getValue(mes_kv)
    accidentes_kv = mp.get(mes_hash["meses"], mes)

    acc = me.getValue(accidentes_kv)
    accidentes_lst = (acc["lstaccidentes"])

    for accidente in lt.iterator(accidentes_lst):

        latitud_sinis = float(accidente["LATITUD"])
        longitud_sinis = float(accidente["LONGITUD"])
        coordenada = latitud_sinis, longitud_sinis
        rad = haversine(centro, coordenada)
        rad = float(rad)

        if (rad <= radio and radrank <= rad):
            lt.addLast(data, accidente)

    num_siniestros = lt.size(data)
    rank = lt.subList(data, 1, N)

    return num_siniestros, rank
```

Mostrar los N accidentes ocurridos dentro de una zona específica para un mes y un año.

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

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Encontrar las parejas llave valor  para años | O(k) |
| Encontrar las parejas llave valor para meses | O(k) |
| Iterar la lista de accidentes Y analizar por radios | O(N) |
| ***TOTAL*** | ***O(N)*** |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

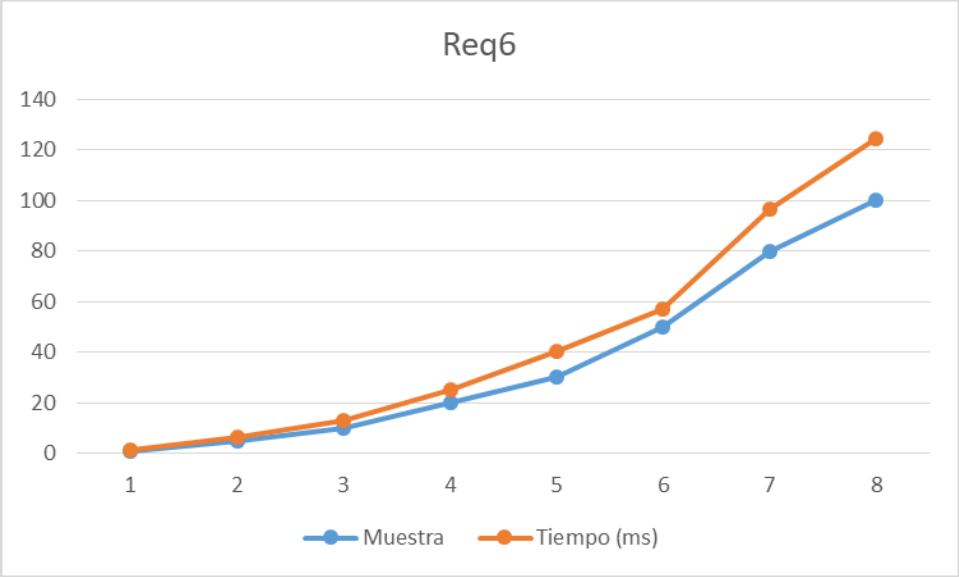
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 0,449 |
| 5 pct | 1,491 |
| 10 pct | 3,057 |
| 20 pct | 5,347 |
| 30 pct | 10,215 |
| 50 pct | 7,284 |
| 80 pct | 16,432 |
| large | 24,309 |

## Graficas




**Análisis** Se realizo una regresión lineal teniendo en cuenta los datos de tiempo de carga para los 8 distintos tamaños de archivos.  Como era de esperarse, la función es aproximadamente lineal. Esto se observa ya que los datos se acoplan al ajuste lineal realizado. los datos tienen una alta relación con la

relación de 0(N) teórica encontrada anteriormente. Esto se debe a que en la estructura de datos se utilizaron tablas de hash y mapas ordenados reduciendo el tiempo de ejecución a diferencia de los laboratorios pasados.

# Requerimiento <<7>>

## Descripción

El requerimiento solicita reportar los accidentes más temprano y más tarde para cada día de un mes y año dado, y graficar el histograma de frecuencias de accidentes por hora para ese mismo mes y año. Por lo tanto, primero se obtiene una lista en el intervalo de fechas dado, luego se itera por accidente para encontrar el primero de cada día; el último; y hacer un conteo por horas de cada accidente por día. Finalmente se retorna una lista con los accidentes más temprano y tarde; y un diccionario con el número de accidentes por hora.

```
def req_7(database, f1, f2):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    acc_range = om.values(database, f1, f2)
    horas = {"0":0, "1":0, "2":0, "3":0, "4":0, "5":0, "6":0,
             "7":0, "8":0, "9":0, "10":0, "11":0, "12":0, "13":0,
             "14":0, "15":0, "16":0, "17":0, "18":0, "19":0, "20":0,
             "21":0, "22":0, "23":0}
    first_last = {}
    dia = "0"
    for accident in lt.iterator(acc_range):
        for i in lt.iterator(accident["lstaccidentes"]):
            horas[(i["HORA_OCURRENCIA_ACC"].split(":")[0])] +=1
            dia_i = (i["FECHA_OCURRENCIA_ACC"].split("/"))[2]
            if dia_i != dia and dia == "0":
                first_last[dia_i] = []
                first_last[dia_i].append(i)
            elif dia_i != dia and dia != "0":
                anterior = me.getValue(om.get(database, om.floor(database, datetime.strptime(i["FECHA_OCURRENCIA_ACC"], "%Y/%m/%d"))))
                first_last[(lt.firstElement(anterior["lstaccidentes"])["FECHA_OCURRENCIA_ACC"].split("/"))[2]].append(lt.firstElement(anterior["lstaccidentes"]))
                first_last[dia_i] = []
                first_last[dia_i].append(i)
            dia = dia_i
    first_last[max(first_last.keys())].append(lt.lastElement(lt.lastElement(acc_range)["lstaccidentes"]))
    return first_last, horas

```

| | |
|---|---|
| **Entrada** | Árbol RBT, fecha inicial, fecha final |
| **Salidas** | Diccionario cuyas llaves son el día y los valores los accidentes más temprano y tarde; Diccionario con el número de accidentes por hora del día. |
| **Implementado (Sí/No)** | Si - Grupal |

## Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Obtener la lista con los accidentes entre las fechas | O(k) |
| Iterar por accidente: (tiene 2 condiciones que se nombran a continuación) | O(n) |

| | |
|---|---|
| Sumar al diccionario el accidente de acuerdo a la hora | O(N) |
| Si es un primer accidente del día agregarlo a una lista, si es el último de un día también se agrega a una lista | O(k) |
| Agregar el último a la lista ya que este no se agrega con el recorrido | O(1) |
| ***TOTAL*** | ***O(n)*** |

# Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

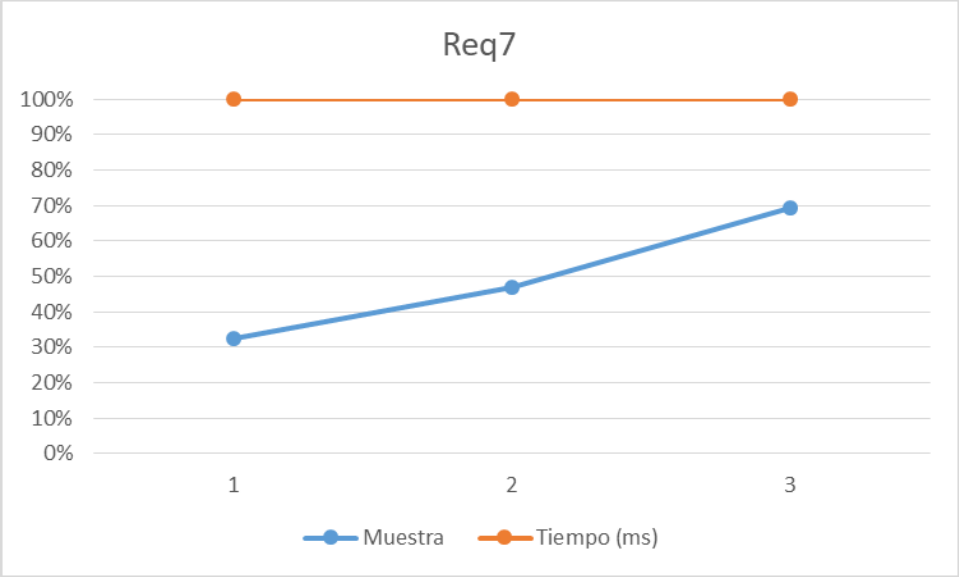
| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| **Muestra** | **Tiempo (ms)** |
|---|---|
| small | 2,08 |
| 5 pct | 5,685 |
| 10 pct | 4,45 |
| 20 pct | ... |
| 30 pct | |
| 50 pct | |
| 80 pct | |
| large | |

## Graficas




## Análisis

Se realizo una regresión lineal teniendo en cuenta los datos de tiempo de carga para los 8 distintos tamaños de archivos.  Como era de esperarse, la función es aproximadamente lineal. Esto se observa ya que los datos se acoplan al ajuste lineal realizado. los datos tienen una alta relación con la relación de 0(N) teórica encontrada anteriormente. Esto se debe a que en la estructura de datos se utilizaron tablas de hash y mapas ordenados reduciendo el tiempo de ejecución a diferencia de los laboratorios pasados.

# Requerimiento <<8>> - (BONO)

## Descripción

```python
def req_8(data_structs, fecha_inicial, fecha_final, tipo):
    """
    Función que soluciona el requerimiento 8
    """
    #sacar los accidentes por fechas
    #sacar los tipos de accidente
    #esto en una lista
    keylo=fecha_inicial
    keyhi= fecha_final
    lt1=lt.newList("ARRAY_LIST", compare)
    lista=om.values(data_structs["fechas"], keylo, keyhi)
    for j in lt.iterator(lista):
        for i in lt.iterator(j["lstaccidentes"]):
            if str(i["CLASE_ACC"])==str(tipo).upper():
                lt.addLast(lt1, i)
    m=folium.Map(location=[4.6097, -74.0817], zoom_start=13)
    colors = {"SOLO DANOS":"orange", "CON HERIDOS":"red", "CON MUERTOS":"black"}
    pops = {"SOLO DANOS":"Sólo daños", "CON HERIDOS":"Con heridos", "CON MUERTOS":"Con muerto
    #iterar por elemento en la lista
    for i in lt.iterator(lt1):
        folium.Marker(location=[i["LATITUD"], i["LONGITUD"]],
                      popup=pops[i["GRAVEDAD"]],
                      icon=folium.Icon(color=colors[i["GRAVEDAD"]], icon="info-sign")).add_to
    m.save("map.html")
    webbrowser.open_new_tab("map.html")

# Funciones utilizadas para comparar elementos dentro de una lista
```

| | |
|---|---|
| **Entrada** | La estructura de datos, año, mes y tipo de accidente. |
| **Salidas** | Mapa con las señalaciones de los accidentes. |

## Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

| **Procesadores** | **AMD Ryzen 7 4800HS with Radeon Graphics** |
|---|---|
| **Memoria RAM** | 12,0 GB (11,8 GB usable) |
| **Sistema Operativo** | Windows 11 |

## Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

| Muestra | Tiempo (ms) |
|---|---|
| small | 1299,6 |
| 5 pct | 4364,28 |
| 10 pct | 11925,2 |
| 20 pct | 22232,7 |
| 30 pct | 42867,6 |
| 50 pct | 97137 |
| 80 pct | 196842 |

| | |
|---|---|
| large | 299662 |

## Graficas