Reto 3 - Informe

Req. 2 - Carlos Arturo Holguín Cárdenas, 202012385, c.holguinc@uniandes.edu.co

Req. 3 - Daniel Hernández Pineda, 202013995, d.hernandez24@uniandes.edu.co

IMPORTANTE: Con la autorización del profesor, se hizo uso de listas nativas de Python en el archivo view.py para poder hacer uso de la librería "tabulate". Para que esta librería funcione adecuadamente, se recomienda usar la versión 3.7 de Python en el intérprete, además de instalarla con el comando "pip install tabulate" en caso de no tenerla instalada

Requerimiento 1

Preámbulo

Entiéndase: **m** = máximo número de avistamientos en una ciudad; **n** = número total de avistamientos. Si bien, teóricamente, **m** puede llegar a ser del mismo tamaño de **n**, se definen como variables separadas porque esto no es una suposición realista con nuestros datos.

Para este requerimiento, se cargó en el catálogo un mapa (tabla de hash) llamado MapReq1. Este mapa contiene como llaves las ciudades del archivo, y como valores árboles binarios de los avistamientos correspondientes a cada ciudad, ordenados según su fecha.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de las funciones REQ1(), addFirstREQ1() y addLastREQ1(). Primeramente, la función REQ1() recibe por parámetro la ciudad que se desea consultar, y accede a su árbol de avistamientos en el MapReq1 (con orden de crecimiento constante, dado que MapReq1() es una tabla de hash). Más adelante, se invocan las funciones addFirstREQ1() y addLastREQ1() para añadir los primeros y últimos 3 elementos del árbol a una lista que se da como respuesta. Estas funciones tienen implementaciones homólogas, invocando los métodos minKey(), maxKey(), select(), todos con complejidad **O(log(m))**. También se utiliza el método values(), el cual tiene complejidad **O(log(m + 3))** (como fue visto en clase). Los demás métodos o ciclos implementados dentro de estas funciones son de orden de crecimiento constante.

De esta manera, se puede resumir la complejidad del algoritmo como **O(log(m))**

Pruebas de tiempo

Se utilizó la siguiente entrada para recrear el peor caso:

- Ciudad: Phoenix

A continuación, los resultados obtenidos:

		Tiempo [ms]	
Archivo	# datos phoenix	Daniel	Carlos
small	8	0.00	0.00
5pct	29	0.00	0.00
10pct	56	0.00	0.00
20pct	99	0.00	0.00
30pct	154	0.00	0.00
50pct	224	0.00	0.00
80pct	353	0.00	0.00
large	437	0.00	0.00

Tabla 1. Pruebas de rendimiento del primer requerimiento

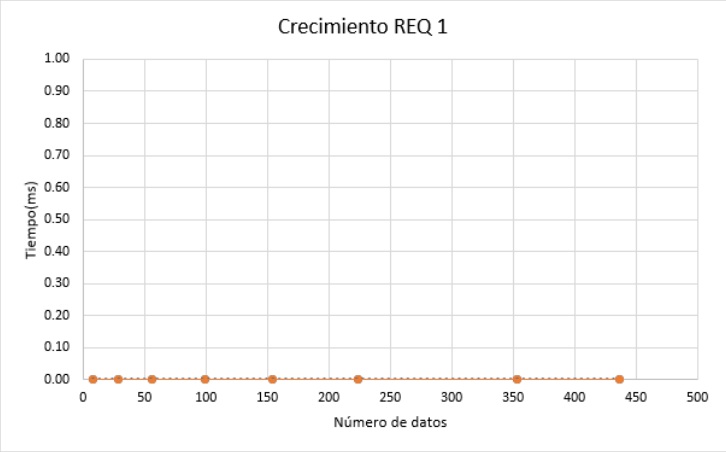

Figura 1. Gráfico del comportamiento del primer requerimiento

A pesar de la complejidad antes calculada, los resultados de las pruebas de rendimiento no son concluyentes. Esto se debe a que, puesto que el requerimiento se ejecuta en tiempos tan cortos por manejar siempre una cantidad significativamente pequeña de datos, los tiempos de ejecución no son lo suficientemente grandes para que el medidor de la librería “time” reconozca las diferencias.

De hecho, lo que sugieren estos resultados es que **m** es tan significativamente menor que **n**, que O(log(m)) termina asemejándose a un orden de crecimiento constante.

Requerimiento 2 (implementado por Carlos Holguín)

Preámbulo

Entiéndase: **m** = número de avistamientos en la duración final; **c** = número de avistamientos en la duración mínima; **n** = número total de avistamientos.

Análisis de complejidad

Para desarrollar este requerimiento solo haremos uso de la función REQ2(), primeramente, la función REQ2() recibe por parámetro la duración mínima y la duración máxima, posteriormente accede a su árbol de avistamientos en el MapReq2.1. Por otro lado, se hallan la llave mínima y máxima dentro del rango que estableció el usuario la complejidad en este caso es de **2log(m)**. Además, se hace uso 2 veces del método get() y 2 veces del método getvalue(), obteniendo una complejidad de **4log(m).** Mas adelante, se hacen dos recorridos en los subárboles que se crearon cuando se importaron los datos, esto representa una complejidad de **m** y **c.** Posteriormente se reorganizan los datos, priorizando el orden cronológico, por tanto, se hacen dos ciclos y se añaden en un nuevo árbol, esto representa una complejidad de **m, c, log(m) y log(c).** Finalmente, se recorren ambos arboles obteniendo una complejidad de **m y c.**

De esta manera, se puede resumir la complejidad del algoritmo como **O(log(m))**

Pruebas de tiempo

Se utilizaron las siguientes entradas para recrear el peor caso:

- Duración Inicial = 30 segundos
- duración Final = 600 segundos

Los resultados obtenidos fueron:

Archivo	# de avistamientos con duracion maxima	tiempo [ms]
small	63	0.00
5pct	296	15.62
10pct	558	35.50
20pct	1035	102.56
30pct	1438	177.87
50pct	2212	359.37
80pct	3136	921.30
large	3642	968.75

Tabla 2. Pruebas de rendimiento para el segundo requerimiento

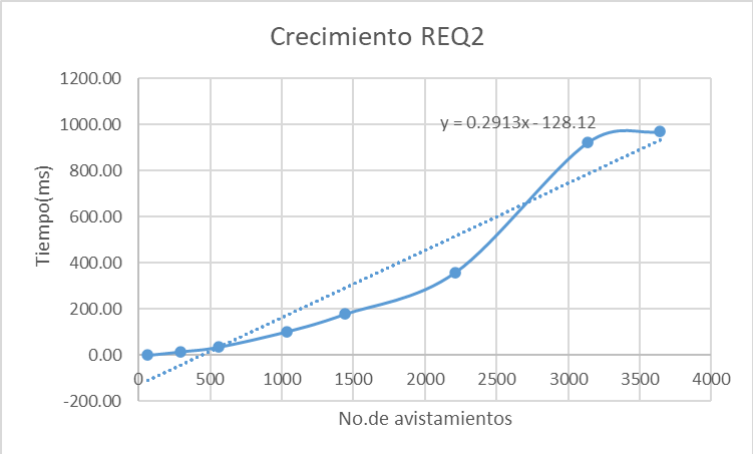

Figura 2. Gráfico del comportamiento del segundo requerimiento

Como se observó en las pruebas de rendimiento, el comportamiento del algoritmo es similar a un comportamiento lineal, además, el rendimiento del algoritmo es totalmente dependiente de las entradas, ya que, solo se operan los datos en este rango. En la prueba se busco el rango que tuviera una gran cantidad de datos para evaluar su rendimiento.

Requerimiento 3 (implementado por Daniel Hernández)

Preámbulo

Entiéndase: **n** = número total de avistamientos; **x** = número de parejas hora:minuto en el rango. Si bien, teóricamente, **x** puede llegar a ser del mismo tamaño de **n**, se definen como variables separadas porque esto no es una suposición realista con nuestros datos. En realidad, **z** es significativamente menor que **n.**

Para este requerimiento, se cargó en el catálogo un mapa ordenado (árbol) llamado MapReq3. Este mapa contiene como llaves las parejas hora:minuto del archivo, y como valores, tablas de hash que contienen listas ordenadas de los avistamientos correspondientes según fecha y demás información relevante para la carga de archivos.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de las funciones REQ3() y processInfoREQ3(). Primeramente, la función REQ3() hace uso del método values() (**O(log(n+x))**) para obtener una lista de las listas de avistamientos para cada hora:minuto dentro del rango dado. Luego, se recorre esta lista (**x** recorridos) y, para cada posición, se recorre su respectiva sublista de avistamientos ordenados según fecha. En conjunto, el recorrido de todas las sublistas acumula **n** ciclos en el peor caso, por lo que se realizan **x + n** ciclos en vez de **x*n**, que sería lo correspondiente si por cada iteración de **x** se tuviera una sublista de tamaño **n.** Posteriormente, se utiliza la función processInfoREQ3(), la cual tiene orden de crecimiento constante puesto que solo trabaja con métodos O(1) de listas.

De lo anteriormente analizado, se puede concluir, entonces, que la función tiene complejidad **O(n)** ya que **n** predomina entre los demás factores. En el peor caso, es inevitable tener que recorrer cada uno de los avistamientos para extraer su información correspondiente.

Pruebas de Tiempo

Se utilizó la siguiente entrada para recrear el peor caso:

- Hora inicial: 00:00
- Hora final: 23:59

Los resultados se muestran a continuación:

Archivo	# avistamientos (n)	Tiempo [ms]
small	803	9.38
5pct	4016	28.13
10pct	8033	50.78
20pct	16066	68.75
30pct	24099	97.63
50pct	40166	143.75
80pct	64265	225.00
large	80332	277.78

Tabla 3. Pruebas de rendimiento del tercer requerimiento

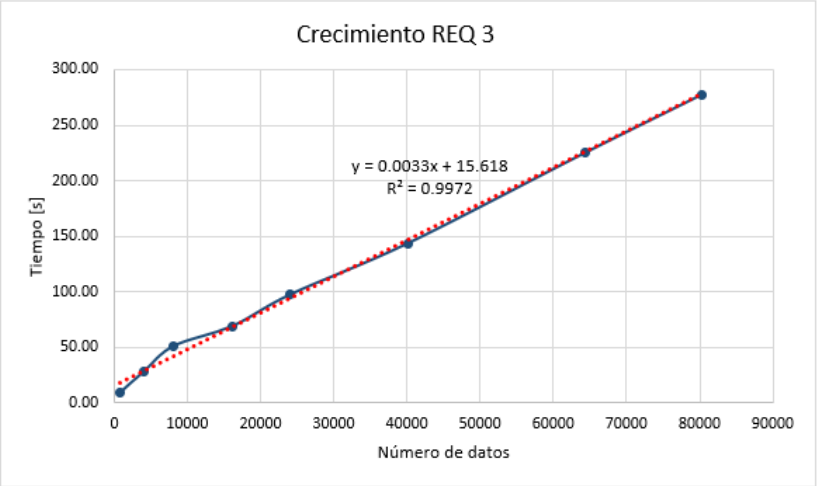

Figura 3. Gráfico del comportamiento del tercer requerimiento

Los resultados obtenidos confirman la complejidad calculada anteriormente, teniendo un comportamiento lineal que mediante regresión se aproxima satisfactoriamente. Dadas las entradas que se utilizaron, se sabe que se trata del peor caso porque la totalidad de los avistamientos entra en aquel rango de horas. Esto provoca que se recorran todos los avistamientos, uno por uno, realizando **n** recorridos como factor predominante.

Requerimiento 4

Preámbulo

Entiéndase: **n** = número total de avistamientos; **z** = número de fechas en el rango. Si bien, teóricamente, **z** puede llegar a ser del mismo tamaño de **n**, se definen como variables separadas porque esto no es una suposición realista con nuestros datos. En realidad, **z** es significativamente menor que **n.**

Para este requerimiento, se cargó en el catálogo un mapa ordenado (árbol) llamado MapReq4. Este mapa contiene como llaves las fechas del archivo, y como valores, listas de los avistamientos correspondientes.

Análisis de complejidad

Para desarrollar este requerimiento se hace uso de la función REQ4(). Para empezar, esta función hace uso del método values() (**O(log(n+z))**) para obtener una lista de las listas de avistamientos para cada fecha dentro del rango dado. Luego, se recorre esta lista (**z** recorridos) y, para cada posición, se recorre su respectiva sublista de avistamientos. En conjunto, el recorrido de todas las sublistas acumula **n** ciclos en el peor caso, por lo que se realizan **z + n** ciclos en vez de **z*n,** que sería lo correspondiente si por cada iteración de **z** se tuviera una sublista de tamaño **n.**

De lo anteriormente analizado, se puede concluir, entonces, que la función tiene complejidad **O(n)** ya que **n** predomina entre los demás factores. En el peor caso, es inevitable tener que recorrer cada uno de los avistamientos para extraer su información correspondiente.

Pruebas de tiempo

Se utilizaron las siguientes entradas para recrear el peor caso:

- Fecha inicial: 1906-11-11 (menor fecha existente en el archivo “large”)
- Fecha final: 2014-05-08 (mayor fecha existente en el archivo “large”)

Se obtuvieron los siguientes resultados:

		Tiempo [ms]	
Archivo	**# avistamientos (n)**	**Daniel**	**Carlos**
small	803	3.13	0.00
5pct	4016	28.13	15.62
10pct	8033	50.00	28.65
20pct	16066	68.75	40.15
30pct	24099	87.50	70.13
50pct	40166	143.75	96.75
80pct	64265	212.50	156.25
large	80332	259.38	190.20

Tabla 4. Pruebas de rendimiento para el cuarto requerimiento

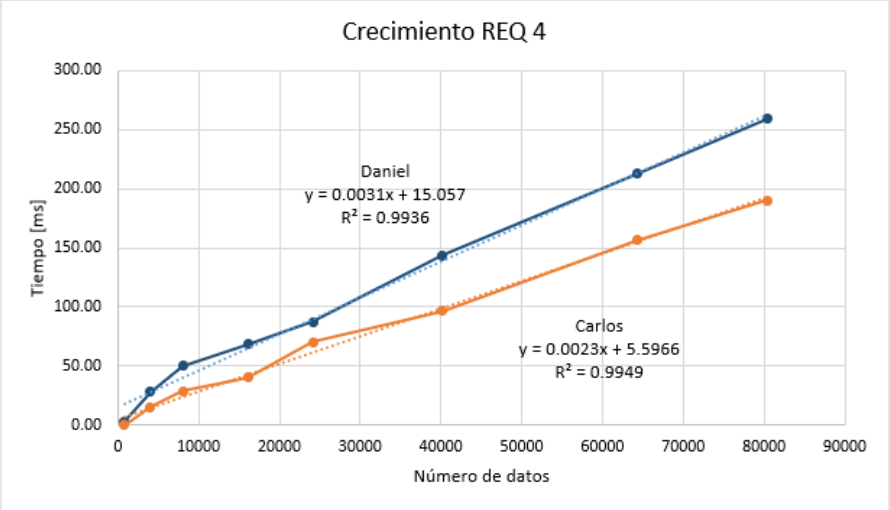

Figura 4. Gráfico del comportamiento del cuarto requerimiento

Como se predijo en el análisis de complejidad, el algoritmo creció linealmente a medida que crecía el volumen de los datos. Las regresiones lineales aplicadas se ajustan exitosamente a los valores de las pruebas de rendimiento realizadas. La distinción entre las pruebas realizadas por cada miembro del grupo tiene que ver con la máquina utilizada por cada uno, afectando principalmente la inclinación de la recta.

Requerimiento 5

Preámbulo

Entiéndase: **n** = número total de avistamientos; **x** = número total de longitudes en el rango; **m** = número total de latitudes en el rango; **z** = número de avistamientos en una coordenada específica. Teóricamente, **x,m,z** podrían aspirar a tener el mismo valor de **n**, aunque esto no sucede en la práctica. Por este motivo es que se definen como variables distintas.

Para este requerimiento, se cargó en el catálogo un mapa ordenado (árbol) llamado MapReq5. Este mapa contiene como llaves las longitudes del archivo, y como valores, árboles binarios que, a su vez, en sus llaves contienen las latitudes del archivo y en sus valores hay listas de los avistamientos correspondientes.

Análisis de complejidad

Para desarrollar este requerimiento solo se hace uso de la función REQ5(). En un primer momento, a partir del MapReq5 se obtienen los árboles de latitudes dentro del rango de longitudes. Esto se realiza mediente el método values() (**O(log(x+*k*x)), donde *k* es 1 en el peor caso**). Posteriormente, se recorre la lista obtenida anteriormente (**O(x)**). En cada uno de estos ciclos, se vuelve a aplicar el método values() (**O(log(m+ *km*)), donde *k* es 1 en el peor caso**), y se recorre la lista obtenida **O(m)** para obtener la lista correspondiente a los avistamientos de esa coordenada longitud-latitud. Finalmente, se recorre

esta última lista de avistamientos **O(z)** para agregar cada uno de sus elementos a una lista final donde se almacena la respuesta.

Nota: *k* es 1 cuando todos los elementos del árbol entran en el rango especificado

En general, los factores lineales deberían predominar sobre los logarítmicos. Por otro lado, vemos que, cuando se trata del peor caso, se recorren una vez todas las longitudes, una vez todas las latitudes, y, consecuentemente, una vez cada avistamiento del archivo. Por este motivo, lo que debió haber sido el producto **xmz** puede resumirse de tal manera que el algoritmo tiene una complejidad **O(n).**

Pruebas de tiempo

Se utilizaron las siguientes entradas para recrear el peor caso:

- Longitud inicial = -176.66 (menor longitud en el archivo "large")
- Longitud final = 178.44 (mayor longitud en el archivo "large")
- Latitud inicial = -82.86 (menor latitud en el archivo "large")
- Latitud final = 72.7 (mayor latitud en el archivo "large")

Los resultados obtenidos se muestran a continuación:

		Tiempo [ms]	
Archivo	**# avistamientos (n)**	**Daniel**	**Carlos**
small	803	12.50	15.625
5pct	4016	37.50	25.600
10pct	8033	65.63	31.250
20pct	16066	87.50	55.650
30pct	24099	112.50	78.400
50pct	40166	156.25	109.370
80pct	64265	234.38	145.250
large	80332	267.86	170.250

Tabla 5. Pruebas de rendimiento para el quinto requerimiento

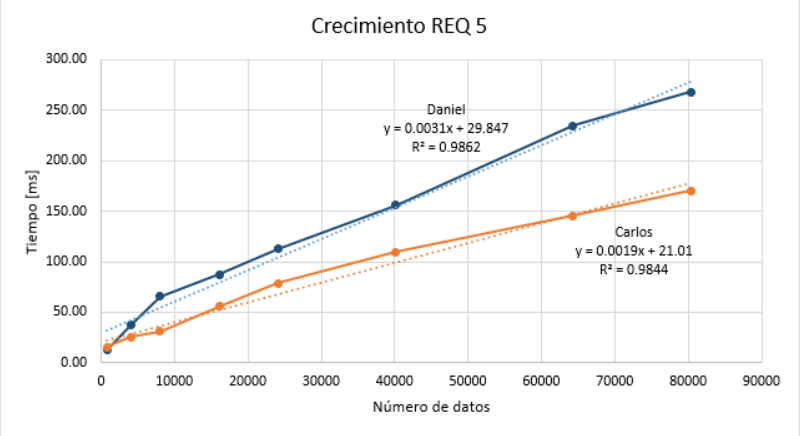

Figura 5. Gráfico del comportamiento del quinto requerimiento

Como era de esperarse, las pruebas de rendimiento mostraron un comportamiento lineal, soportando el análisis de complejidad realizado previamente para el algoritmo.