Estructuras de Datos y Algoritmos – Reto 3 – Grupo 13 Sección 2
Análisis de complejidad de los requerimientos

Req 2 - Juan Sebastián Núñez Cortes, 202021672, j.nunezc@uniandes.edu.co
Req 3 - Santiago Rodríguez Bernal, 202011182, s.rodriguez63@uniandes.edu.co

A continuación, se presenta un análisis de complejidad para cada requerimiento del Reto 3, considerando el código utilizado y las gráficas que se obtuvieron a partir de las pruebas de tiempos. Las gráficas junto con los tiempos se encuentran en el archivo 'Tablas Reto 3'.

Carga de datos: La carga de datos se ocupa de añadir cada avistamiento al catálogo de avistamientos, en una lista de tipo Array List. Adicionalmente, se crean cinco índices (1 tabla de Hash y 4 árboles de tipo RBTs) para poder ser consultados al momento de probar los requerimientos. En general esta carga de datos es bastante rápida.

Porcentaje de la muestra [pct]	Tamaño de la muestra (ARRAYLIST)	Requerimiento 1 [ms]	Requerimiento 2 [ms]	Requerimiento 3 [ms]	Requerimiento 4 [ms]	Requerimiento 5 [ms]	Requerimiento 6 [ms]	Carga de datos (1) [m
0,5% (small)	803,00	31,25	0,00	15,62	0,00	0,00	31,25	109,38
5,00%	4016,00	46,88	15,62	78,12	31,25	15,62	46,88	359,38
10,00%	8033,00	62,50	46,88	140,62	46,88	0,00	62,50	703,12
20,00%	16066,00	125,00	93,75	281,25	109,38	15,62	46,88	1234,38
30,00%	24099,00	171,88	156,25	468,75	171,88	15,62	93,75	1750,00
50,00%	40166,00	203,12	281,25	890,62	296,88	31,25	62,50	2843,75
80,00%	64265,00	281,25	453,12	1406,25	500,00	31,25	62,50	4390,62
100% (large)	80332,00	343,75	578,12	1796,88	656,25	31,25	93,75	5250,00

Tabla 1. Resumen de la toma de tiempos del Reto 3.

Requerimiento 1: El requerimiento 1 utiliza principalmente una única función para obtener toda la información guardada en la Tabla de Hash. Esta función se denomina 'getCityInfo' y obtiene el total de ciudades que han reportado avistamientos de ovnis (cantidad de llaves de la tabla), los avistamientos de una ciudad dada por parámetro (valor de una llave de la tabla), el total de dichos avistamientos (tamaño de la lista que corresponde al valor de la llave), y la ciudad con más avistamientos. Al analizar el código utilizado, gran parte de las operaciones se basa en complejidades constantes. Sin embargo, dado que para cada ciudad en la Tabla de Hash hay una lista con sus avistamientos, hay una sola lista que se tiene que ordenar por fecha con la función 'sortDateUfos'. Esta función se emplea para ordenar los avistamientos con mergesort, por lo tanto, su complejidad es de O(nLog(n)). Por otro lado, dado que se debe obtener la ciudad con más avistamientos, se revisan todas las llaves de la Tabla de Hash en un ciclo, de manera que la complejidad de este algoritmo es de O(n). Dado que O(nLog(n)) tiene una mayor complejidad, esta corresponde a la complejidad del requerimiento 1.

Requerimiento 2: Al igual que en el caso anterior, el requerimiento 2 utiliza una única función para acceder a la información guardada. Sin embargo, esta vez se consulta un árbol RBT que se construyó en la carga de datos. Dicho árbol utiliza como llaves la duración en segundos de un avistamiento, y como valor una lista de tipo arreglo con los avistamientos. En cuanto a la función empleada, esta se denomina 'getSecondInfo', y al acceder a los valores, las operaciones realizadas en general son constantes. No obstante, debido a que se utiliza las funciones om.maxKey y om.getValue, se debe iterar sobre el árbol para obtener la llave más grande y el valor que se busca. Ahora bien, dado que se trata de un árbol RBT, en este se garantiza el balanceo, por lo cual la complejidad es proporcional a la longitud de la rama de la raíz hasta una hoja, es decir O(Log(n)). Luego, se obtiene un rango de valores con la función om.values, por lo cual en el peor caso se consultan todos los valores, es decir la complejidad es de O(n). Por otro lado, se hace un ciclo sobre la lista de rangos de tamaño (n), y dado que cada valor corresponde a una lista de distinto tamaño (m), para agregar todos los valores a una única lista se considera una complejidad de O(n*m). Finalmente, se ordenan estos valores por su duración en segundos con mergesort, por lo cual la complejidad es de O(nLog(n)). Así pues, la complejidad del requerimiento 2 corresponde a la mayor complejidad de las que fueron analizadas, es decir es O(n*m).

Requerimiento 3: En este caso empleamos una única función para acceder a la información del RBT que ha sido creado. En este las llaves son las horas en formatos HH:MM:SS y los valores son una lista de todos los avistamientos en cada hora del día. La función que obtiene los datos requeridos es 'getTimeInfo', la cual en resumen tiene la misma complejidad que el requerimiento 2. Esta función realiza operaciones de tiempos constantes, pero para acceder a la mayor llave, a un valor en específico y a una lista de valores dentro del rango se examina el árbol, por lo cual la complejidad de las operaciones om.maxKey y om.getValue, al ser un RBT, es de O(Log(n)). Asimismo, dado que se utiliza la función om.values, hay otra complejidad adicional que corresponde a O(n). Por otro lado, de manera similar al requerimiento 2, se crea una única lista debido a que la lista de rangos es una lista de listas. Así pues, debido a que las iteraciones son realizadas sobre cada lista (m) dentro del rango dado (n), la complejidad del algoritmo es de O(n*m). Finalmente, la lista creada en las operaciones anteriores se debe ordenar por fecha, pues hay varios elementos repetidos por lo cual se hace necesario organizarlos cronológicamente usando las fechas de cada uno. Considerando que se utiliza el algoritmo de ordenamiento mergesort, su complejidad es O(nLog(n)). En cuanto a la complejidad del requerimiento 3, esta seria O(n*m).

Requerimiento 4: Nuevamente, se emplea una única función para acceder a la información de un RBT. En este árbol las llaves son las fechas en formato YYYY-MM-DD y los valores son una lista de todos los avistamientos de cada fecha. La función que obtiene los datos requeridos es 'getDateInfo', la cual en resumen tiene la misma complejidad que el

requerimiento 2. Esta función realiza operaciones de tiempos constantes, pero para acceder a la menor llave, a un valor en específico y a una lista de valores dentro del rango se examina el árbol, por lo cual la complejidad de las operaciones om.minKey y om.getValue, al ser un RBT, es de O(Log(n)). Asimismo, dado que se utiliza la función om.values, hay otra complejidad adicional que corresponde a O(n). Por otro lado, de manera similar al requerimiento 2, se crea una única lista debido a que la lista de rangos es una lista de listas. Así pues, debido a que las iteraciones son realizadas sobre cada lista (m) dentro del rango dado (n), la complejidad del algoritmo es de O(n*m). Finalmente, la lista creada en las operaciones anteriores se debe ordenar por fecha con otra función. Considerando que nuevamente se utiliza el algoritmo de ordenamiento mergesort, su complejidad es O(nLog(n)). En cuanto a la complejidad del requerimiento 4, esta seria O(n*m).

Requerimiento 5: Para este requerimiento 5 se utiliza una única función para acceder a la información guardada en un RBT. En este caso, la estructura utilizada es más compleja, ya que se trata de un árbol de árboles. El primer RBT ocupa llaves con las longitudes de los avistamientos, y cada valor de estas llaves es otro árbol RBT construido con llaves a partir de las latitudes de los avistamientos. De esta manera, los valores de este árbol corresponden a una lista con todos los avistamientos de una longitud y una latitud en específico. En cuanto a la función utilizada 'getGeographicInfo', esta se encarga primero de obtener una lista de valores del mapa de longitudes con la función om.values, por lo cual su complejidad es de O(n). Luego se itera n veces sobre esta lista, y se vuelve a llamar a la función om.values, pero esta vez con el rango de latitudes y se itera m veces. Por último, se debe iterar k veces sobre cada lista del rango para tener una única lista con todos los avistamientos dentro de ambos rangos. Dado que esto se realiza en un solo ciclo, y debido a que las iteraciones son sobre valores distintos, la complejidad de este pedazo del código es de O(n) para las iteraciones realizadas sobre la lista de longitudes, O(m) para obtener el rango de valores del mapa de latitudes, y O(k) para las iteraciones sobre la lista de latitudes. Así pues, la complejidad del ciclo es de O(n*m*k). Como último paso, se ordena por ubicación la lista de avistamientos obtenida, y dado que se utiliza mergesort, la complejidad de esta función es de O(nLog(n)). Al comparar las 3 grandes complejidades obtenidas, O(n), O(n*m*k), y O(nLog(n)), se puede concluir que la complejidad del requerimiento 5 es de O(n*m*k).

Requerimiento 6: El requerimiento 6 al ser una representación gráfica del requerimiento 5, comparte su misma complejidad de O(n*m*k). Sin embargo, se es necesario agregar la complejidad que implica todo el proceso de creación del mapa y markers con la información. Para el proceso de creación se empleo la función de sublist para extraer las primeras 5 y últimas 5 ubicaciones lo que significo una complejidad constante, adicionalmente se iteraron estos 10 elementos lo que implicó un complejidad de O(n), y a medida que se iteraban se iba creando cada marker, donde no solo se marcaba el mapa, sino que adicionalmente cada punto

en el mapa tiene la información sobre esta, por lo tanto su complejidad según lo encontrado sobre folium es O(n ^2). Por lo tanto, la complejidad del requerimiento 6 equivale a O(n ^2).