EDA, Reto 2 - Mapas

Samuel Mateo Montañez Gil*
Universidad de los Andes, Bogotá, Colombia.
(Dated: 8 de abril de 2022)

Se busca crear una aplicación que clasifique y busque datos relacionados con 3 archivos tipo *csv* (artistas, canciones y álbumes) de Spotify, a través del uso de mapas del DISClib proporcionado para el curso de Estructura de Datos y Algoritmos (EDA) de la universidad de los Andes.

I. ANÁLISIS DE COMPLEJIDAD

A. Carga de datos

Para la carga de datos se crearon 5 mapas y 3 listas. Las listas se usaron para encontrar cuáles fueron los primeros y últimos 3 datos cargados por cada uno de los archivos *.csv*. En el caso de los mapas, 2 se usaron para relacionar y referenciar los datos entre si, y los otros 3 para cumplir con los requerimientos de uso. Primero se cargo las canciones, luego se cargo los álbumes y por último los artistas.

Para todos los casos se usaron 3 mapas en sus funciones de carga. En el caso de las canciones la complejidad es de $O(NM)$ para la función *addSong*(), donde N son la cantidad de canciones y M la cantidad de artistas relacionados a estas canciones. Para estas se uso mapas según su popularidad, su *albumid* y su *artistid*. También, en los álbumes se obtuvo una complejidad de $O(KN)$ para su función *addAlbum*(), donde K son los álbumes y N las canciones obtenidas del mapa *idalbums*. En este caso se uso mapas según sus años de lanzamiento, su popularidad y su *artistid*. Finalmente, los artistas tuvieron una complejidad de $O(79K \cdot NM)$ para su función de carga *addArtist*(). Para estos se uso los mapas según popularidad, *artistid* y nombre de artista (*artistname*). Finalmente, al generar un mapa de países por cada uno de los artistas se iteró para el máximo de 79 países que existen en el campo de álbumes, *availablemarkets*.

B. Requerimiento 1, $O(Klog(K))$

Para el requerimiento 1 se uso la función *getAlbumsYear*(), la cual busca el año deseado y retorna la cantidad de álbumes con esta característica y una lista tipo $ARRAYLIST$ con los primeros y últimos 3 álbumes ordenados alfabéticamente. Para encontrar el año busca en el mapa *years* la existencia de la llave dada y de existir obtiene la pareja llave-valor, para luego obtener el valor con la función $me.getValue()$. Este proceso se desarrolla en $O(4)$. Posteriormente, obtiene el tamaño de la lista, organiza la lista con un $Mergesort$ y termina por enviar la lista a la función $get3FirstandLast()$ ($O(7)$) para conseguir los primeros y últimos 3 de la lista. Todo esto se logra en $O(8+Klog(K))$, donde K es la cantidad de álbumes encontrados para el año de entrada. De forma que se puede expresar, la notación de este requerimiento como $O(12 + Klog(K))$ o simplemente $O(Klog(K))$.

C. Requerimiento 2, $O(Mlog(M))$

Para el requerimiento 2 se uso la función *getPopularity*(), la cual recibe un valor de popularidad, entre 0 a 100, y retorna la cantidad de artistas con esta popularidad y una lista tipo $ARRAYLIST$ de los primeros y últimos 3 ordenados según sus seguidores y nombres. Para encontrar la lista busca en el mapa *popularity* y de existir consigue la pareja llave-valor y su valor. Con esto, obtiene la posición 1 de esta lista que a su vez es una sublista con estos artistas. Esto representa una operación 0(5). Luego, obtiene el tamaño de la lista, la ordena con $Mergesort$ y obtiene los primeros y últimos 3. Esta operación representa una complejidad de $O(8 + Mlog(M))$, donde M es la cantidad de artistas. De esta manera, se entiende que el proceso toma $O(13 + Mlog(M))$ o $O(Mlog(M))$.

D. Requerimiento 3, $O(Nlog(N))$

Para el requerimiento 3 se uso la función *getPopularity*(), la cual recibe un valor de popularidad, entre 0 a 100, y retorna la cantidad de canciones con esta popularidad y una lista tipo $ARRAYLIST$ de los primeros y últimos 3 ordenados según sus duración y nombres. Al usar la misma función que el requerimiento 2, el único cambio que se hace es al obtener el valor de la pareja llave-valor se solicita la posición 2 de la lista de listas. Esta al ser $ARRAYLIST$ da el resultado en la misma cantidad de tiempo que el anterior requerimiento. De esta forma se obtiene el resultado en $O(Nlog(N))$, donde N es la cantidad de canciones.

* s.montanez@uniandes.edu.co

E. Requerimiento 4, $O(Nlog(N))$

Para el requerimiento 4 se uso la función $getSGByArtistCountry()$, la cual busca el nombre y el país deseado y retorna la cantidad de albumes y canciones disponibles a la vez que una lista con la canción más popular. Esta función llama a $getSGMapofMaps()$ que busca en el mapa *artistname* y encuentra el valor del artista que es una lista de listas en $O(4)$. Luego pide la segunda posición de la lista que es el mapa *countryMap* del artista. En este busca el país deseado, obtiene su valor que es una lista de listas. En la primera posición obtiene un *int* con la cantidad de álbumes y en la segunda obtiene la lista de canciones, todo esto en $O(7)$. Después, obtiene el tamaño de la lista de canciones para por último ordenar las canciones. Estas dos últimas operaciones representan $O(1 + Nlog(N))$, donde N es la cantidad de canciones. Finalmente, la función retorna la cantidad de álbumes, la cantidad de canciones y la lista de canciones a la función principal $getSGByArtistCountry()$, por lo que esta función secundaria tiene una complejidad de $O(12 + Nlog(N))$. Esta función genera una sublista de un elemento para luego ser enviada al *view*. Todo este proceso se mide como $O(14 + Nlog(N))$ o simplemente $O(Nlog(N))$.

F. Requerimiento 5, $O(34)$

Para el requerimiento 5 se uso la función $getDiscography()$, la cual busca la discografía del artista deseado por el nombre del artista y retorna 5 valores, 3 de estos son la cantidad de albumes *single*, *compilation* y *album*. Los últimos 2 son listas de los primeros y últimos 3 albumes, y canciones. Para esto se busca en el mapa *artistname* el valor de la llave, el cual es una lista de listas. La función pide la primera posición de la lista que es la lista de artistas con este nombre, para la cual también pide la primera posición. Esto se realiza en $O(6)$. Después, se pide la cantidad de cada uno de los álbumes y se genera una lista de los primeros y últimos 3 albumes. Esto se realiza en $O(10)$. Por último, se intenta obtener las canciones más populares de cada album por un *try*, de no ser posible se agrega un *None*. Todo esto en $O(18)$. De esta manera se obtiene que para este requerimiento la complejidad es de $O(34)$.

G. Requerimiento 6, $O(Nlog(N))$

Para el requerimiento 6 se uso la función $getSGByArtistCountryTOP()$, la cual busca el nombre, el país y la cantidad de canciones deseada y retorna la cantidad de canciones encontradas y una lista con las primeras y últimas 3 canciones ordenadas por popularidad. Esta función llama a $getSGMapofMaps()$ la cual realiza el procedimiento descrito en el requerimiento 4 que da una complejidad de $O(12 + Nlog(N))$, donde N es la cantidad de canciones encontradas. Luego, esta verifica si la cantidad de canciones deseadas es mayor a las obtenidas, de ser así retorna todas las canciones encontradas. De lo contrario, crea una sublista con la cantidad de elementos deseados. Es decir que en el peor de los casos toma $O(1 + N)$. De esta manera se tiene que la operación final toma $O(13 + N + Nlog(N))$ o simplemente $O(Nlog(N))$.

II. PRUEBAS DE RENDIMIENTO

Para las pruebas de rendimiento se uso un Intel(R) Core (TM) $i7-8750H$ CPU @2,20GHz 2,21GHz con 12 GB de RAM y sistema operativo Windows 11-64 bits. Se realizaron dos pruebas principales una para encontrar la memoria y el tiempo del programa con el archivo *large* como muestra la tabla I, y otra para las diferencias de tiempo y memoria entre archivos como muestra la tabla II.

Cuadro I. Se clasifica los valores para la carga y los requerimiento con el archivo *large* en tiempo y memoria. No se pudo obtener la memoria de la carga debido a la gran cantidad de consumo hecha por esta, supera los límites de la librería usada, *tracemalloc*.

Proceso	Tiempo (ms)	Memoria (kB)
Carga	263400,71	
Requerimiento 1	20,90	7,32
Requerimiento 2	1,95	1,58
Requerimiento 3	47,69	3,82
Requerimiento 4	1,27	0,51
Requerimiento 5	0,31	1,44
Requerimiento 6	0,65	4,30

Cuadro II. Se clasifica los valores para la carga en tiempo y memoria según el archivo usado. No se pudo obtener la memoria de la carga *large*, debido a la gran cantidad de consumo hecha por esta, la cual supera los límites de la librería usada, *tracemalloc*. Las medias del archivo *large* son menores debido a que no se uso *tracemalloc*, el cual tiene un alto consumo de memoria.

Archivo	Tiempo (ms)	Memoria (kB)
small	5356,41	116191,84
5*pct*	44807,54	930860,10
10*pct*	80589,84	1617639,08
20*pct*	141374,91	2694758,01
30*pct*	186195,12	3544288,37
50*pct*	295424,16	4841837,08
80*pct*	434326,65	6237389,14
large	263400,71	

III. COMPARACIÓN RETO 1

Debido a que el reto 1, no se realizo análisis de complejidad se confrontará con respecto al Reto 1 realizado por Cristian Cortes, 202011908 y Natalia Villegas 202113370.

A. Requerimiento 1

Este requerimiento fue igual de eficiente que el realizado en las listas. Esto se debe a que hay que ordenar los datos encontrados por una categoría. Esto se puede realizar en la carga pero la extiende exponencialmente.

B. Requerimiento 2

Este es ligeramente más rápido, ya que solo se debe entrar a organizar las listas en comparación con la búsqueda del reto 1.

C. Requerimiento 3

De la misma manera que el requerimiento 2, este es menor ligeramente por no tener que detenerse a buscar.

D. Requerimiento 4

Este es identico en complejidad, por lo que ya se había mencionado anteriormente en el requerimiento 1. Se puede disminuir la complejidad en mapa si se desea aumentar el tiempo de carga.

E. Requerimiento 5

Este es menor debido a que se realizo el ordenamiento en el mapa. Como se observa no importa la entrada obtenida, siempre la respuesta se dará en un valor de tiempo específico.

F. Requerimiento 6

Este es idéntico debido a la necesidad de ordenar los datos obtenidos en la búsqueda. De lo contrario, puede disminuirse hasta $O(N)$ por el hecho de buscar la cantidad de canciones específicadas.