Análisis de Complejidad Temporal
Reto 3

Juan Camilo Colmenares Ortiz - 202011866 - j.colmenares@uniandes.edu.co
Juan Andrés Ospina Sabogal - 202021727 - ja.ospinas1@uniandes.edu.co

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]
samll%	71424.079	2646017.109

Carga de datos

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]
samll%	1081.717	109.911

Requerimiento 1:

- Se le pide al usuario 3 inputs los cuales son:

    1. La característica de contenido
    2. El valor mínimo de la característica de contenido
    3. El valor máximo de la característica de contenido

- Estos datos se le pasan a la función R_1

- La función R_1 toma una lista de un árbol en O(logn) con la función om.values() en donde el árbol binario al que se accede depende del input 1 del usuario, en base a este input se escoge un árbol binario de los 6 que hay con las diferentes características de contenido y el rango son los inputs 2 y 3 que nos dio el usuario

- Después recorremos la lista anteriormente descrita lo cual es O(n) y al recorrer esta lista hallamos a los artistas únicos y el total de reproducciones

- Posteriormente imprimimos la cantidad de artistas y el total de reproducciones

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]

samll%	774.662	32.812

Requerimiento 2: (Juan Camilo Colmenares Ortiz - 202011866)

- Se le pide al usuario 4 inputs los cuales son:

    1. Valor mínimo de la característica Energy
    2. Valor máximo de la característica Energy
    3. Valor mínimo de la característica Danceability
    4. Valor máximo de la característica Danceability

- Se recurre a la función R_2y3 y se le pasan como parámetros los inputs dados por el usuario

- Creamos una lista llamada randomness en donde se guardará la información de 5 pistas aleatorias que nos piden imprimir y creamos un mapa llamado unique_tracks en donde guardaremos las pistas como llaves para saber el total de pistas únicas.

- Esta función recurre a la función R_1() con el fin de obtener una lista con eventos pertenecientes al rango dado por el usuario para la característica de Energy, como se había descrito anteriormente, acceder a esta lista tiene una complejidad de O(logn).

- Después recorremos esta lista lo cuál es O(n) con el fin de ver cuáles eventos cumplen con la condición del rango de Danceability, chequeamos cada uno de los eventos con el fin de ver si se debe añadir al mapa de unique_tracks al igual que añadimos el evento a la lista randomness.

- Por último retornamos el tamaño del mapa y la lista randomness

- Se recurre a la función random_selector() con el fin de asegurarnos de que la selección de los eventos de la lista sea aleatoria, la complejidad de esta es O(1) ya que se genera un número aleatorio mayor a 0 y menor al tamaño de la lista, esto se hace 5 veces y el número que sale es la posición a la que se accede de la lista lo cual es O(1), por último se le imprime al usuario la cantidad de tracks únicos y los 5 eventos aleatorios.

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]
samll%	821.366	19.012

Requerimiento 3: (Juan Andrés Ospina Sabogal - 202021727)

- Se le pide al usuario 4 inputs los cuales son:

1. Valor mínimo de la característica Instrumentalness

2. Valor máximo de la característica Instrumentalness
3. Valor mínimo de la característica Tempo
4. Valor máximo de la característica Tempo

- Se recurre a la función R_2y3 y se le pasan como parámetros los inputs dados por el usuario.

- Creamos una lista llamada randomness en donde se guardará la información de 5 pistas aleatorias que nos piden imprimir y creamos un mapa llamado unique_tracks en donde guardaremos las pistas como llaves para saber el total de pistas únicas.

- Esta función recurre a la función R_1() con el fin de obtener una lista con eventos pertenecientes al rango dado por el usuario para la característica de Energy, como se había descrito anteriormente, acceder a esta lista tiene una complejidad de O(logn).

- Después recorremos esta lista lo cuál es O(n) con el fin de ver cuáles eventos cumplen con la condición del rango de Danceability, chequeamos cada uno de los eventos con el fin de ver si se debe añadir al mapa de unique_tracks al igual que añadimos el evento a la lista randomness.

- Por último retornamos el tamaño del mapa y la lista randomness

- Se recurre a la función random_selector() con el fin de asegurarnos de que la selección de los eventos de la lista sea aleatoria, la complejidad de esta es O(1) ya que se genera un número aleatorio mayor a 0 y menor al tamaño de la lista, esto se hace 5 veces y el número que sale es la posición a la que se accede de la lista lo cual es O(1), por último se le imprime al usuario la cantidad de tracks únicos y los 5 eventos aleatorios.

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]
samll%	1044.580	20.017

Requerimiento 4:

- Se le pide la usuario que escoja entre dos opciones:
    1. Escoger los géneros de los cuales desea saber las reproducciones y los artistas
    2. Añade un nuevo género

- Cuando se escoge la opción 1 se hace uso de la función R_4() por cada género que el usuario ingresó, esta función recibe como parametro un género, después en base al género que el usuario ingresó, el programa busca la llave en un mapa lo cual es O(1), esto con el fin de hallar los tempos de ese género, una vez se obtienen el tempo menor y el tempo mayor, se recurre a om.values() la cual es O(logn), esta nos retorna una lista con todos los valores del rango del género.

- Después recorremos esta lista lo cual es O(n) con el fin de obtener los artistas únicos y además de obtener los 10 primeros artistas los cuales vamos a imprimir, el modo de hallar artistas únicos es metiendo estos en un mapa llamado unique_artists en donde las llaves son las id de los aristas, de este modo no se repiten, también tenemos una lista llamada artist_10 en donde se van a guardar los primeros 10 artistas

- Una se tiene la lista de 10 artistas y la cantidad de artistas únicos que equivale a al mp.size(unique_artists) pasamos estos datos a los diferentes géneros y los vamos imprimiendo uno por uno

- Cuando se escoge la opción 2 al usuario se le pregunta:

    1. Genero
    2. Valor mínimo de tempo de ese género
    3. Valor máximo de tempo de ese género

- Posteriormente se actualizan los mapas que llevan control de los géneros que hay y añade el tempo que tiene cada género

Tamaño de archivo	Tiempo de ejecución[ms]	Uso de memoria [kb]
samll%	1186.551	9.698

Requerimiento 5:

- Se le pide al usuario tiempos de inicio y final. Estos tiempos marcan el intervalo deseado para el análisis.
- Para el requerimiento se hace uso de tres BST: analyzer['hash_generos'], analyzer['track_hashtags'], y analyzer['sentiment_values'].
- Primero, se obtiene una lista con los valores del BST analyzer['hash_generos'] en el rango de horas que ingresó el usuario. La complejidad de esta operación es O(log n). Posteriormente, se recorre la lista obtenida por la operación anterior. Cada elemento es una tabla de hash cuyas llaves son géneros y cuyos valores son listas con los videos de dicho género en una hora específica. Se obtiene el tamaño de la lista y se guarda la información en una tabla de hash auxiliar: sus llaves serán los géneros y sus valores serán la cantidad de reproducciones por género. Esta operación tiene complejidad O(n) en su peor caso, pues n sería la cantidad de datos solicitada por el usuario (la cantidad de datos que existan en el intervalo de horas elegido). Una vez completa la tabla de hash auxiliar, se procede a recorrer la misma en búsqueda del género con más repeticiones y la cantidad total de repeticiones. Esta operación es O(9), pues solo se tienen en cuenta 9 géneros (9 parejas llave-valor en la tabla de hash). Además de los datos ya obtenidos, se desea saber cuales son los videos con más hashtags. Para esto se crea un max heap. De nuevo se recorre la lista de datos obtenida con el método om.values, característico de las tablas de

símbolos ordenadas. En cada elemento del conjunto se recorre los 9 géneros (llaves) hasta encontrar el género mayor. Allí se insertan los videos al heap. La función de comparación del heap compara la cantidad de hashtags de cada video, por lo cual, al estar orientado a mayor, el primer elemento del heap será el video con más hashtags. Una vez lleno el heap, se procede a extraer su primer elemento 10 veces. Para cada hashtag se obtiene el vader_avg y se calcula el promedio de los valores para cada video con más de un hashtag. Para los videos cuyos hashtags no poseen vader_avg, el vader_avg se deja como cero. Este ciclo tiene una complejidad constante, pues no depende de la cantidad de datos. Se repetirá 10 veces independientemente de la cantidad de datos. El subciclo, por otro lado (el cálculo de vader_avg), se repite tantas veces como haya hashtags en cada video, porque se deben tener en cuenta todos los hashtags para el promedio. Es decir, su complejidad es O(n), siendo n la cantidad de hashtags de cada video.

En pocas palabras, la complejidad del requerimiento en el peor caso es O(n).