Análisis de Complejidad Temporal
Reto 2

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

Para la implementación del reto 2 tomamos en cuenta múltiples consejos que se nos dieron en la sustentación del reto:

- Mejoramos nuestra organización en el view.py ya que se nos dijo que realizamos muchas cosas en este archivo cuando en realidad debieron haber estado en el model.py.

- Mejoramos la organización del controller.py ya que en este se encontraban procesos que pertenecian al model.py

- Otro consejo que nos dieron fue que no nos enfocaremos en hacer un pre-procesamiento de los datos tan robusto ya que el nuestro tardaba 7.25 minutos, esta recomendación tiene sentido ya que muchas funciones que se hacen en el preprocesamiento se pueden pasar al procesamiento y los tiempos siguen siendo veloces.

Carga de datos

	Tiempo de ejecución [ms]	**Consumo de memoría [kB]**
Reto 1	**435111.439**	**1332714.598**
Reto 2	**59566.47**	**1333890.275**

Nuestro catálogo consiste de diferentes estructuras que nos ayudan a resolver los problemas propuestos en el reto.

1. catalog[‘videos’]: esta es una lista de tipo ARRAY_LIST, el propósito de esta lista es guardar todos los videos del archivo “videos-large.csv”.

2. catalog[‘categories_normal’]: esta es una lista de tipo ARRAY_LIST, el propósito de esta lista es guardar todas las categorías del archivo “category-id.csv”.

3. catalog[‘pure_country’]: este es un mapa de tipo PROBING, el propósito de este mapa es guardar como llave los diferentes países que se encuentren en el archivo ‘videos-large.csv’ y guardar como valor una lista con los videos pertenecientes al país. El tamaño de este mapa es de 400 y el factor de carga de 0.5, el tamaño ideal lo encontramos despejando la siguiente ecuación $\alpha = N/M$ en donde α es el factor de carga, Nes la cantidad de parejas (llave-valor) y Mes el

tamaño del mapa. 0.5 se escogió como factor de carga ya que en la documentación encontramos que este número era óptimo para un mapa de tipo PROBING y la cantidad de parejas llave-valor se sacó del hecho de que en el archivo "videos-large.csv" hay aproximadamente 200 países.

4. catalog['categories']: este es un mapa de tipo PROBING, el propósito de este mapa es guardar como llave las diferentes categorías que se encuentran en el archivo 'videos-large.csv" y guardar como valor una lista con los videos pertenecientes a la categoría. El tamaño de este mapa es de 64 y este número se halló de la misma manera que el catálogo anterior, teniendo como datos para despejar nuestra ecuación, un factor de carga de 0.5 y una cantidad de parejas llave-valor de 32 debido a que solo hay 32 categorías.

5. catalog['countries']: este mapa es igual al de catalog['pure_country'], la única diferencia es que este no es un mapa en donde las llaves tienen como valor a una lista, por el contrario, tienen como valor a otro mapa, este otro mapa es idéntico al mapa de catalog['categories'], de este modo podemos acceder a videos pertenecientes de un cierto país y una cierta categoría de manera instantánea.

Requerimiento 1

	Tiempo de ejecución [ms]	**Consumo de memoría [kB]**
Reto 1	**82.463**	**0.199**
Reto 2	**440.795**	**20.281**

Para el requerimiento 1 le pedimos al usuario un país, una categoría y un número de videos a enlistar, primero encontramos el valor numérico de la categoría que nos dio el usuario mediante la función find_position_category(), después recurrimos a la función sortVideosByViews() que accede al mapa catalog['countries'] en donde se busca como llave el país que introdujo el usuario y así obtenemos el valor, el cual es otro mapa en donde buscamos la llave que sea igual a la categoría que nos dio el usuario, después accedemos al valor de esta llave que es una lista con videos, después ordenamos esta lista con el algoritmo de ordenamiento merge sort, esto tiene una complejidad de O(n(log(n))) y acceder a esta lista es O(1) ya que acceder a las llaves en un hash map es constante, por último se utiliza la función find_videos_views_country() para imprimir los n videos que quiere listar el usuario.

Requerimiento 2 (Juan Camilo Colmenares Ortiz - 202011866)

	Tiempo de ejecución [ms]	**Consumo de memoría [kB]**
Reto 1	**98.568**	**2.383**
Reto 2	**2507.490**	**8.524**

Para el requerimiento 2 le pedimos al usuario un país, después recurrimos a la función sortVideosID() la cual accede al mapa catalog[‘pure_country’] y busca la llave igual al país introducido por el usuario, después accede al valor de esta llave que es una lista con todos los videos del país que le interesa al usuario, después por medio del algoritmo de ordenamiento merge sort de complejidad O(n(log(n))) ordenamos la lista en base a la id de los videos, después utilizamos la función find_trending_video() la cual recorre la lista ordenada una sola vez siendo O(n) y retornan los datos del video con más días de trending, posterior a esto se imprimen los datos de este video.

Requerimiento 3 (Juan Andrés Ospina Sabogal - 202021727)

	Tiempo de ejecución [ms]	**Consumo de memoría [kB]**
Reto 1	**1013.287**	**288.250**
Reto 2	**565.902**	**3.539**

El requerimiento 3 recibe del usuario el nombre de una categoría y retorna el video que más días ha estado en tendencia para dicha categoría. Esto se hace enteramente por la función video_most_trending_days_category, ubicada en el modelo. Primero, se usa la función strip() para eliminar caracteres innecesarios antes o después del nombre de categoría ingresado por el usuario. Posteriormente, mediante un ciclo básico que hace uso de la lista catalog[“categories_normal”], se encuentra el id de la categoría en cuestión. Una vez se tiene el category_id, se procede a obtener la lista de videos de dicha categoría. Para esto se emplean las funciones get y getValue, las cuales permiten acceder al elemento deseado en una tabla de símbolos (en este caso, el elemento deseado es la lista con todos los videos de la categoría que seleccionó el usuario). Una vez obtenida, se procede a determinar cuál de los videos de la misma está repetido más veces. Una aparición en el archivo “videos-large.csv” implica un solo día como tendencia. Por ende, al no haber una columna que contenga los días de un video como tendencia, se deben contar las apariciones de cada video y determinar cual se repite más. Esto se hace con un ciclo que recorre la lista obtenida, y guarda los títulos de los videos con sus respectivas repeticiones en un diccionario de Python. Una vez recorrida la lista por completo, se procede a extraer del diccionario creado a partir de la lista el video más repetido. La complejidad del algoritmo en su peor caso es O(n), siendo n la cantidad de videos en la categoría especificada por el usuario. La complejidad temporal de obtener la sublista de videos tiene valor constante, pues extraer un elemento de una tabla de símbolos a partir de una llave es O(1). Se debe recorrer toda la sublista para determinar la cantidad de veces que se repite cada video, por lo cual la complejidad es O(n). La implementación de este mismo requerimiento en el Reto 1 fue en su totalidad basada en ciclos. Aunque los ciclos fueron implementados de la manera más eficiente posible, la complejidad O(1) de obtener elementos de una tabla de símbolos es mucho menos costosa de tiempo. Esto explica la diferencia de tiempos entre el Reto 1 y el Reto 2.

Requerimiento 4

	Tiempo de ejecución [ms]	Consumo de memoría [kB]
Reto 1	**1.683**	**0.199**
Reto 2	**3124.051**	**2.824**

El requerimiento 4

Para el requerimiento 4 le pedimos al usuario un país, la cantidad de videos a listar y un tag, después recurrimos a la función sortVideosLikes() la cual accede al mapa catalog[‘pure_country’] y busca la llave que sea igual al país que ingresó el usuario, después accede al valor de esta llave el cual es una lista la cual mediante el algoritmo de ordenamiento merge sort de complejidad O(n(log(n))) será ordenada en base a los likes que tenga cada video, posterior a esto se hace uso de la función likes_tags() la cual va a filtrar los videos por el tag que pasó el usuario y va retornar los videos con mas likes que contienen el tag que mencionó el usuario, este algoritmo será de O(n).