María Castro 202020850 m.castroi@uniandes.edu.co
Valentina Calderón 202020771 v.calderonm@uniandes.edu.co

Sustentación Reto 3

Función 3 / Requerimiento 1:

Objetivo: Se desea conocer cuántas reproducciones (eventos de escucha) se tienen en el sistema de recomendación basado en una característica de contenido y con un rango determinado, El sistema debe indicar el total de canciones y el número de artistas (sin repeticiones).

MVC:

View:

1. Se solicita al usuario que ingrese la categoría de contenido que desea consultar.
2. Se llama a getCategory() en el controller revisa si la categoría existe en el mapa contentCtegory(). Que es un mapa que tiene todas las posibles categorías que se pueden consultar.
3. Si getCategory() es None, la categoría de contenido no es válida. En caso de que no sea None, verifica que el mínimo y máximo ingresado sean válidos. Para esto se verifica que la resta entre min_range y max_range no sea mayor a 0, que el min_range sea mayor a 0, y que el max_range sea menor a 1. En caso de que estas condiciones no se cumpla, solicita ingresar nuevos rangos.
4. En caso de que el rango si sea válido, llama a la función del controller categoryCaracterization con el fin de devolver el total de canciones y el número de artistas para la categoría específica. Esto lo devuelve en un arbol.

Controller:

1. La función getCategory, llama a la función getCategory en el controller
2. La función categoryCaracterization llama a la función categoryCaracterization en el controller

Model:

1. La función getCatgeory primero revisa en el mapa contentcategory con .get si existe la categoría ingresada por parámetro, en caso que sí, retorna la información de la categoría en un arbol, en caso opuesto; en caso de que no, retorna None
2. La función categoryCaracterization
3. Mapa ordenado
    a. Se usa lista de listas con un mapa ordenado siguiendo la categoría, y los dos rangos.
    b. Se crea un nuevo mapa con compare function según las categorías
    c. Se establece un contador.
    d. Se hace un doble recorrido en la lista de listas para comparar los items.

e. Se llama a la función del modelo checkWithUser para revisar que el evento también esté dentro de los datos de usertrackinghashtags. Si esto sucede, se suma 1 al total y se agrega el artista en el mapa de artistas.
f. Se retorna una tupla, con el total de canciones y el tamaño del mapa de artistas, que sería la cantidad única de artistas.

4. checkWithUser()
    a. Convierte la fecha del evento a una datetime.
    b. Usa esa fecha para obtener los valores correspondientes en el árbol de fechas del usertrackinghastag, - esto retorna una pareja llave-valor cuyo valor es una lista con todos los eventos del usertrackinghashmaps que ocurrieron en la misma fecha del evento actual
    c. Se recorre la lista de eventos para la fecha -
        i. Si un evento tiene el mismo user_id y y track_id (la fecha ya se reviso con el .get del árbol) se retorna True – el evento se encuentra en ambos archivos
            1. De lo contrario se retornará False

Complejidad:

- .values() **$\log_2(n)$** -> n es la cantidad de nodos en el árbol de la categoría ingresada
- .values() devuelve una lista cuyo tamaño es **m** ie. hay m sublistas
    - Caso general: m < n
        - **$\log_2(n) + O(m \times e')$** , e' es la cantidad de eventos por cada sublista **m**
        - Cada valor de e' varia por cada m
    - **Peor caso: m = n (valores para la categoría ingresada 0.0 a 1.0)**
        - **$\log_2(n) + n^2$**
            - Toca mirar todos los elementos del árbol
- Entre más cerca m este de n, es decir entre más valores diferentes ie. nodos, toque buscar, más grande será la complejidad
- CheckWithUser() O(e) → la cantidad de eventos de usertrackinghashtags en una fecha dada

Función 4 / Requerimiento 2 (Maria):

Objetivo:

Se desea encontrar las pistas en el sistema de recomendación que pueden utilizarse en una fiesta que se tendrá próximamente. Se desea encontrar las canciones que tengan en cuenta las variables (Energy, y Danceability). El usuario podrá indicar los valores (rangos) para esos parámetros.

MVC:

View:

1. Se le pregunta al usuario la siguiente información: valor mínimo para Energy, valor máximo para Energy, valor mínimo para Danceability, valor máximo para Danceability
    a. Se verifica que estos datos tengan un rango válido
2. Se llama a la función del controller partyMusic()
3. Se imprime la cantidad de tracks únicos dentro de los parámetros
4. La función printTracks() imprime 5 tracks aleatorios y sus respectivos valores de Energy y Danceability

Controller:

1. partyMusic() llama a la función del modelo y retorna el resultado de esta

Model:

1. partyMusc()
    a. Del Mapa de content categoría se saca la pareja-llave valor para Energy (**constante)** y el valor de esta pareja que es un RBT por valor de Energy
    b. Se usa la función om.values() para obtener una lista de todos los elementos en el rango dado para Energy
        i. Esto retorna una lista de listas ya que cada valor para los nodos es una lista con todos los videos que tienen ese valor
    c. Se crean dos estructuras, un mapa para guardar los tracks únicos y una lista que tendrá los elementos finales
    d. Se hace un recorrido doble para llegar a cada evento dentro de los rangos (cada valor de la lista retornada por values es una lista)
        i. Este recorrido depende de S - la cantidad de valores entre el dado menor y mayor para energy y de E - la cantidad de eventos que tiene cada lista
    e. Se usa checkWithUser() para revisar que el evento también esté dentro de los datos de usertrackinghashtags
        i. Si esto se cumple, se revisa el valor de danceability para el evento actual, si este valor está dentro del rango deseado se agrega a la lista de elementos finales y se agrega el track al mapa de tracks
2. checkWithUser()
    a. Convierte la fecha del evento a una datetime.
    b. Usa esa fecha para obtener los valores correspondientes en el árbol de fechas del usertrackinghastag, - esto retorna una pareja llave-valor cuyo valor es una lista con todos los eventos del usertrackinghashmaps que ocurrieron en la misma fecha del evento actual
    c. Se recorre la lista de eventos para la fecha -
        i. Si un evento tiene el mismo user_id y y track_id (la fecha ya se reviso con el .get del árbol) se retorna True – el evento se encuentra en ambos archivos
            1. De lo contrario se retornará False

Complejidad:

- .values() **$\log_2(n)$** -> n es la cantidad de nodos en el árbol
- .values() devuelve una lista cuyo tamaño es m ie. hay m sublistas
    - Caso general: m < n
        - **$\log_2(n) + O(m \times e')$** , e' es la cantidad de eventos por cada sublista m
        - Cada valor de e' varia por cada m
    - **Peor caso: m = n (valores para Energy y Danceability de 0.0 a 1.0)**
        - **$\log_2(n) + n^2$**
            - Toca mirar todos los elementos del árbol
- Entre más cerca m este de n, es decir entre más valores diferentes ie. nodos, toque buscar, más grande será la complejidad
- CheckWithUser() O(e) → la cantidad de eventos de usertrackinghashtags en una fecha dada

Función 5 / Requerimiento 3 (Valentina):

Objetivo:

Se desea conocer las pistas en el sistema de recomendación que podrían ayudar a los usuarios en su periodo de estudio, para este fin se debe tener en cuenta tengan en cuenta las variables (Instrumentalness, y Tempo)

MVC:

View:

1. Se le pregunta al usuario la siguiente información: valor mínimo para Instrumentalness, valor máximo para Instrumentalness, valor mínimo para tempo, valor máximo para tempo
    a. Se verifica que estos datos tengan un rango válido
2. Se llama a la función del controller relaxingMusic()
3. Se imprime la cantidad de tracks únicos dentro de los parámetros
4. La función printTracks2() imprime 5 tracks aleatorios y sus respectivos valores de Instrumentalness y tempo

Controller:

1. relaxingMusic() llama a la función del modelo y retorna el resultado de esta

Model:

3. relaxingMusic()

a. Del Mapa de content categoría se saca la pareja-llave valor para Instrumentalness (**constante)** y el valor de esta pareja que es un RBT por valor de Instrumentalness
b. Se usa la función om.values() para obtener una lista de todos los elementos en el rango dado para Instrumentalness
    i. Esto retorna una lista de listas ya que cada valor para los nodos es una lista con todos los videos que tienen ese valor
c. Se crean dos estructuras, un mapa para guardar los tracks únicos y una lista que tendrá los elementos finales
d. Se hace un recorrido doble para llegar a cada evento dentro de los rangos (cada valor de la lista retornada por values es una lista)
    i. Este recorrido depende de la cantidad de valores entre el dado menor y mayor y de la cantidad de eventos que tiene cada lista
e. Se usa checkWithUser() para revisar que el evento también esté dentro de los datos de usertrackinghashtags
    i. Si esto se cumple, se revisa el valor de tempo para el evento actual, si este valor está dentro del rango deseado se agrega a la lista de elementos finales y se agrega el track al mapa de tracks

4. checkWithUser()
    a. Convierte la fecha del evento a una datetime.
    b. Usa esa fecha para obtener los valores correspondientes en el árbol de fechas del usertrackinghastag, - esto retorna una pareja llave-valor cuyo valor es una lista con todos los eventos del usertrackinghashmaps que ocurrieron en la misma fecha del evento actual
    c. Se recorre la lista de eventos para la fecha -
        i. Si un evento tiene el mismo user_id y y track_id (la fecha ya se reviso con el .get del árbol) se retorna True – el evento se encuentra en ambos archivos
            1. De lo contrario se retornará False

Complejidad:

- .values() **$\log_2(n)$** -> n es la cantidad de nodos en el árbol de Instrumentales
- .values() devuelve una lista cuyo tamaño es m ie. hay m sublistas
    - Caso general: m < n
        - **$\log_2(n) + O(m \times e')$** , e' es la cantidad de eventos por cada sublista m
        - Cada valor de e' varia por cada m
    - **Peor caso: m = n (valores para Instrumentales y tempo de 0.0 a 1.0)**
        - **$\log_2(n) + n^2$**
            - Toca mirar todos los elementos del árbol

- Entre más cerca m este de n, es decir entre más valores diferentes ie. nodos, toque buscar, más grande será la complejidad
- CheckWithUser() O(e) → la cantidad de eventos de usertrackinghashtags en una fecha dada

Función 6 / <u>Requerimiento 4</u>:

<u>Objetivo:</u>

Se desea saber cuántas canciones se tienen por cada género y cuántos artistas se tienen en cada género.

<u>MVC:</u>

View:

1. Se le pide al usuario que ingrese los géneros que desea consultar
2. Revisa que todos los géneros ingresados existan en el diccionario de géneros
    a. Si alguno de los géneros no existe, le pregunta al usuario si quiere crear un nuevo género con el que no existe
        i. Si se desea crear un nuevo género se llama a la función del controller newGenre()
        ii. Si no se desea crear el nuevo género se le informa al usuario que ese género es válido y que no se puede proceder.
3. Si todos los géneros fueron validados se llama a la función genresStudy() del controller
4. Para cada género se imprime su tempo, la cantidad de reproducciones y artistas únicos y 10 artistas del género.

Controller:

1. newGenre() llama a la función del modelo newGenre()
2. genresStudy() llama a la función del model genresStudy()

Model:

1. newGenre(): con los datos ingresados (nombre, mínimo y máximo del Tempo) crea un nuevo género y lo agrega al mapa (diccionario de géneros)
2. genresStudy()
    a. Crea un nuevo mapa y obtiene el árbol por tempos (del mapa de content categories)
    b. Para cada género ingresado por el usuario saca su valores correspondientes de tempo mínimo y máximo
    c. Del árbol de tempos saca los eventos que correspondientes al género actual
    d. Se hace un recorrido doble para llegar a cada evento dentro de los rangos (cada valor de la lista retornada por values es una lista)

i. Se llama a la función checkWithUser() para revisar que el evento actual también está en usertrackinghashtags
    1. Si lo está, este evento se añade a una lista y el artista asociado a un mapa
ii. Al final se añade el género a un mapa y su valor asociado es una lista con sus eventos y el mapa con artistas únicos

Complejidad:

- Primer loop → O(g): cantidad de géneros que ingresó el usuario
- .values() **$\log_2(n)$** -> n es la cantidad de nodos en el árbol de Instrumentales
- .values() devuelve una lista cuyo tamaño es m ie. hay m sublistas
    - Caso general: m < n
        - **$\log_2(n) + O(m \times e')$** , e' es la cantidad de eventos por cada sublista m
        - Cada valor de e' varia por cada m
    - **Peor caso: m = n (valores de género son todos los géneros que hay en el diccionario de género)**
        - **$\log_2(n) + n^2$**
            - Toca mirar todos los elementos del árbol
- Entre más cerca m este de n, es decir entre más valores diferentes ie. nodos, toque buscar, más grande será la complejidad
- CheckWithUser() O(e) → la cantidad de eventos de usertrackinghashtags en una fecha dada

Función 7 / Requerimiento 5:

Objetivo:

Dado un rango de horas, se desea indicar el género de música más escuchado en dicho rango teniendo en cuenta todos los días disponibles e informar el promedio para cada uno de los valores de análisis de sentimiento, en las canciones de dicho rango.

MVC:

View:

1. Se le pide al usuario el valor mínimo y máximo de la hora del día.
2. Verifica que el rango de horas sea correcto
3. Llama a la función genreMostListened() del controller
4. Se imprimen los resultados de la busqueda
5. La función print_genre_reps() in imprime todos los géneros con sus respectivas reproducciones de orden de mayor a menor
6. print_tracks() imprime los 10 tracks aleatorios del género con más reproducciones y sus

Controller:

1. genreMostListened() llama a la función del modelo genreMostListened()

Model:

1. genreMostListened()
    a. Usa om.values () para sacar del arbol de “content_time” los eventos que ocurren dentro del rango de tiempos solicitados
    b. Se crea un mapa llamado genre reps que guardara la cantidad de reproducciones de cada género
    c. Se hace un recorrido doble para llegar a cada evento dentro de los rangos (cada valor de la lista retornada por values es una lista)
    d. Se hace una revisión con los datos de usertrackinghash usando la función checkWithUser2()
    e. Se llama a la función matchTempo()
    f. Para cada llave en genre_reps (es decir para cada género) se saca su valor asociado (las reproducciones) y se agrega a una lista.
        i. Dentro de una variable total, se va sumando las reproducciones de cada género para obtener el total de reproducciones dentro del rango de horas
    g. Se ordena la lista que se creó
        i. Esta lista se ordena usando la función de comparación cmp Genre,
    h. El primer elemento de esta lista es el género que más fue escuchado dentro del rango de horas
    i. Se llama a la función info_top_genre()
    j. Se retorna el total de reproducciones, el género con más reproducciones, la lista con los géneros y sus reproducciones ordenadas y la lista con los 10 tracks cuyos hashtags se van a mostrar.
2. checkWithUser2()
    a. Recibe un evento de content context. Con la hora de ese evento, se obtienen los eventos correspondientes del árbol por tiempos usertrackinghastags
    b. Para cada evento con esa hora se revisa que el user_id y track_id del evento en ambos archivos sea el mismo.
        i. Si lo es se llama a la función updateTrackHashtags() y se retorna True
        ii. De lo contrario se retorna False
3. updateTrackHashtags()
    a. Recibe un evento de usertrckinghashtag.
    b. Saca el track correspondiente de “tracks_hastag”, si no existe se crea una lista y se pone en el mapa
    c. Si si existe se saca la lista de hashtags asociados a ese track id.
    d. A la lista de hashtags se le añade el hashtag del evento actual y se vuelve a poner dentro del mapa
4. matchTempo()
    a. Para cada género dentro del diccionario de géneros (mapa) se revisa si el evento está dentro del rango de Tempo asociado al género actual.

i. Si lo está, del mapa genre reps se obtiene la cantidad de repeticiones asociadas a un género o se inicializa en 0 si todavía no hay
ii. También se obtiene o se crea una mapa que tendrá todos los tracks asociados a un género
iii. Se le agrega uno a la cantidad de reproducciones del género y se agrega el track actual a la mapa de tracks

5. info_top_genre()
    a. Se obtiene el mapa de los tracks del género con más reproducciones. Se crean dos listas y un mapa.
    b. Se recorre cada track del género y para cada uno se obtiene los hashtags asociados (del mapa de tracks_hahtag)
    c. Para cada uno de los hashtags se obtiene su valor de VADER del mapa de sentiment values.
    d. Se revisa que el VADER no sea 0 o un string vacío.
        i. Si no lo es este valor se suma a un contador
        ii. De lo contrario, no se tiene en cuenta ese hashtag y resta 1 al número de total de hashtags
    e. Después del recorrido se revisa que el número de hashtags no sea menor que uno y se saca el promedio.
    f. La información para cada track se guarda en una lista
    g. Se seleccionan 10 elementos aleatorios de la lista final de los tracks y se guardan en una última lista
    h. Esta lista se ordena y se retorna.
        i. Mergesort

Complejidad:

- .values() **$\log_2(n)$** -> n es la cantidad de nodos en el árbol por horas de contentcontext
- .values() devuelve una lista cuyo tamaño es m ie. hay m sublistas
    - Caso general: m < n
        - **$\log_2(n) + O(m \times e')$** , e' es la cantidad de eventos por cada sublista m
        - Cada valor de e' varia por cada m
    - **Peor caso: m = n (valores para hora del dia del 0:00:00 a 23:59:59)**
        - **$\log_2(n) + n^2$**
            - Toca mirar todos los elementos del árbol
- Entre más cerca m este de n, es decir entre más valores diferentes ie. nodos, toque buscar, más grande será la complejidad
- O(g) → depende de la cantidad de géneros que hayan en el 'genre_dictionary'
    - Mejor casos sería O(8), pero si el usuario agrega géneros este número aumenta
- CheckWithUser2(): O(e) → la cantidad de eventos de usertrackinghashtags en una fecha dada

- matchTempo(): O(g) → depende de la cantidad de géneros que hayan en el 'genre_dictionary'
    - Mejor casos sería O(8), pero si el usuario agrega géneros este número aumenta
- info_top_genre(): O(t) → # de tracks únicos pero un género (varía con el género)
- **Mergesort: O(nlogn)**

Parte 4: Análisis de Resultados

Requerimiento	Caso de Prueba	Dato	Reto 3
Req 0		Tiempo de Ejecución [ms]:	86851.0843
		Consumo de Memoria[kB]:	427238.288
Req 1	Category: instrumentalness Min: 0.75 Max: 1.0	Tiempo de Ejecución [ms]:	454.897
		Consumo de Memoria[kB]:	14.892
Req 2	Min Energy: 0.5 Max Energy: 0.75 Min Danceability: .75 Max Danceability: 1.0	Tiempo de Ejecución [ms]:	1887.3913
		Consumo de Memoria[kB]:	38.38
Req 3	Min Instrumentalness: 0.6 Max Instrumentalness: 0.9 Min Tempo: 40 Max Tempo: 60	Tiempo de Ejecución [ms]:	578.764
		Consumo de Memoria[kB]:	8.744
Req 4	Genres: Reggae Hip-hop Pop	Tiempo de Ejecución [ms]:	7335.463
		Consumo de Memoria[kB]:	20926.3773
Req 5	Min Hora: 7:15:00 Max Hora: 9:45:00	Tiempo de Ejecución [ms]:	6466.88
		Consumo de Memoria[kB]:	2423.468