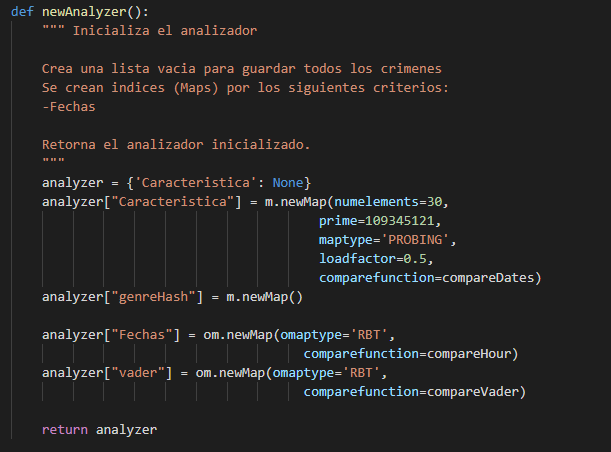
**Factor de carga de cada elemento**

**Reto 3**

Felipe Rueda Rivera 202010903

Julian Rivera 202013033

**Carga de datos:**



**Dentro del analyzer cargamos 2 llaves:**

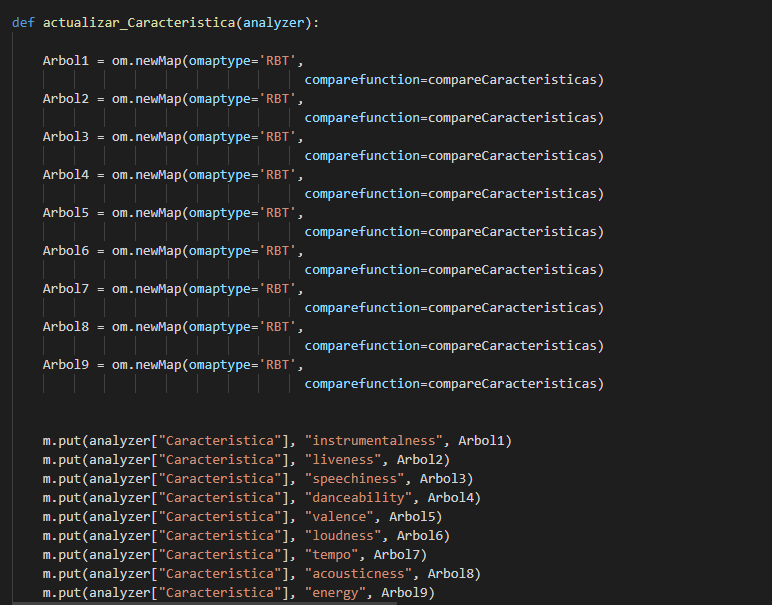
* Características
* Genrehash
* Fechas
* vader

Como se puede evidenciar en Características se guardo una tabla de hash de tipo probing, por lo que si hay colisiones no guarda en un bucket sino que encuentra la siguiente posición vacía. Tenemos pensado guardar como llave el nombre de la característica (instrumentalness, tempo, etc) y como valor un árbol BST para que quede organizado y poder cumplir los requerimientos con la mayor eficiencia y exitosamente.

Como se puede observar también en genrehash se guardo una tabla de hash predeterminada pues en esta tabla vamos a guardar los géneros para el requerimiento 4, como llaves se guarda el nombre del genero y como valores se guarda el rango de tempo que tiene dichos géneros, este rango se guardó por medio de una tupla.

Así mismo, las llaves “Fechas” y “Vader” se usaron para crear el requerimiento 5, sus valores son arboles binarios que se ordenan, como su nombre lo indicada, con respecto a el orden de las fechas y el valor del Vader\_avg respectivamente.

**Características (llave 1 de analyzer):**



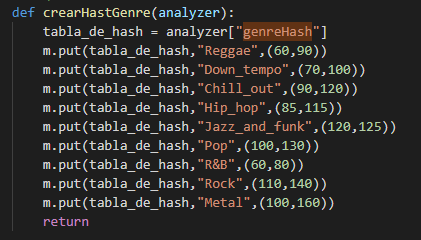
Como se dijo antes, el valor de característica es una tabla de hash que guarda 9 parejas llave/valor, las llaves son las diferentes características de las canciones y como valor un árbol BTS. En total hay nueve árboles pues hay 9 características del documento “context\_content\_features.csv”

.



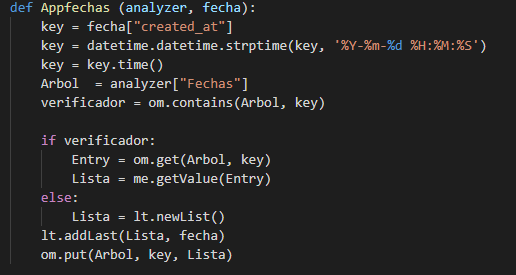
Dentro de cada árbol BST se guardan como llave todos los valores en orden que cumplen con esa característica y como valor, una lista con el número total de autores que pertenecen a ese valor del documento “context\_content\_features.csv”.

**Genrehash (llave 2 de analyzer):**



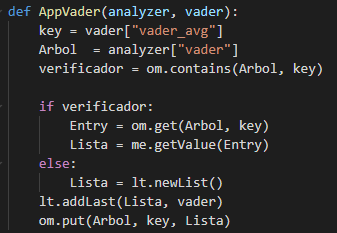
El valor de Genrehash es una tabla de hash que tiene como llave todos los géneros de las canciones y cómo valor una tupla que contiene el rango BPM típico.

**Fechas (llave 3 de analyzer):**



**El valor es un árbol BST que tiene como llaves todas las fechas de las canciones y como valor toda** la fila con las características de la respectiva canción del documento “context\_content\_features.csv”.

**vader (llave 4 de analyzer):**

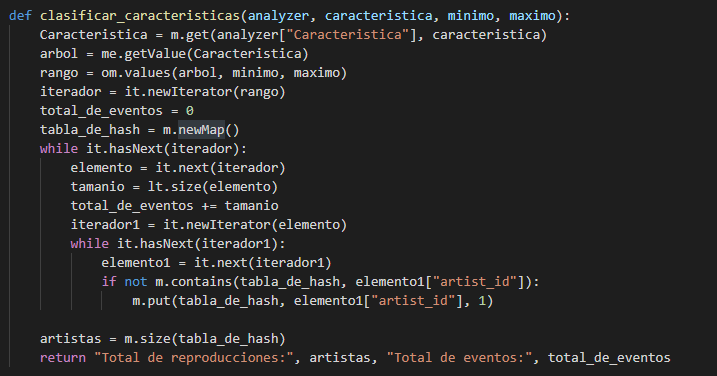


El valor es un árbol BST que tiene como llaves todas las opciones de Vader\_avg y como valor toda la fila con las características de la respectiva canción del documento “sentiment\_values.cvs”.

Tiempo en la carga de datos (Core i5 decima generación/ 8 de ram):

|  |  |
| --- | --- |
| Archivo | Tiempo en segundos para la carga de datos |
| Small | 16.42 |
| 10% | 312.21 |
| 30% | Tiempo excedido |

**Requerimiento 1:**



En el requerimiento uno toma el árbol que se busca, selecciona el rango solicitado y por medio de un “While” toma el número total de reproducciones, para sacar el número total de eventos sin repetir, se agregaron los eventos a una tabla de hash, posteriormente se sacó el número de llaves de la tabla de hash para sacar los autores sin repetir. Se utilizaron dos “while” porque había una lista dentro de una lista, entonces para sacar las listas de información de las pistas musicales se hizo este método.

De igual forma se pensó hacerlo de forma eficiente por lo que en vez de comparar con listas se comparó los artistas por medio de una tabla de hash que tiene complejidad O(1), además aprovechamos que en la tabla de hash no puede haber dos llaves iguales entonces por medio del m.size se calculó la cantidad de llaves que están en la tabla de hash por lo que devuelve el número de artistas sin repeticiones, esto tiene una complejidad de O(1). Finalmente, para hallar el total de eventos se calculó el size pero de la listas lista y se la sumamos a una variable que tiene 0 de parámetro pues solo coge el size de una lista entonces en el peor de los casos seria O(1)

**Requerimiento 2:**

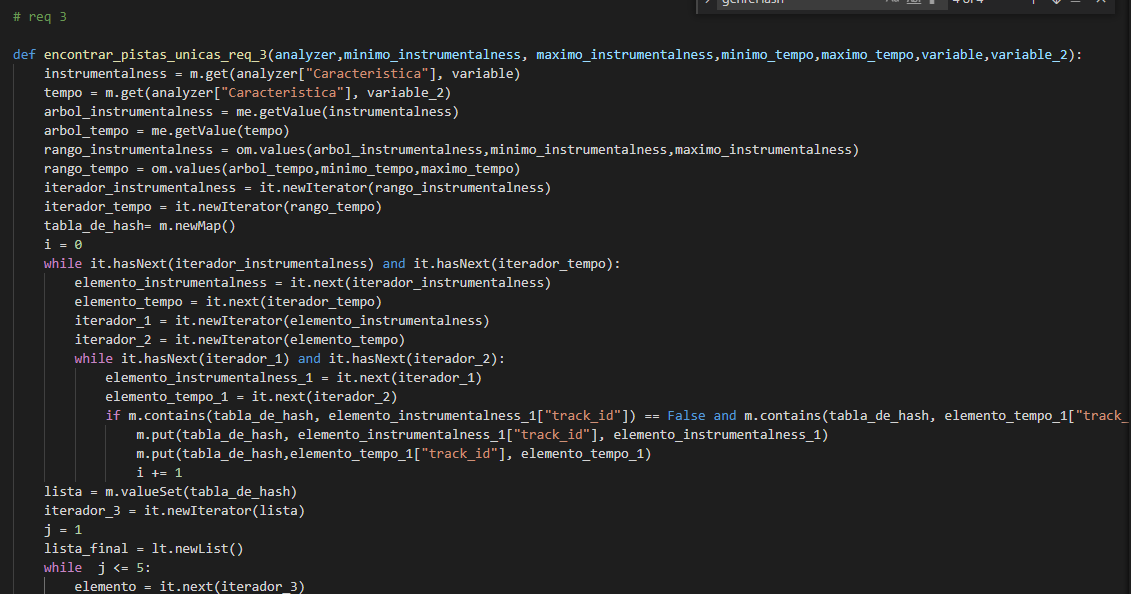
**Texto

Descripción generada automáticamente**

Para el requerimiento dos, se tomó el rango solicitado de “energy” y “danceability” respectivamente, por un lado, se recorrieron todas las opciones de “Track” dentro del rango por medio de dos “while” (por ser una lista de listas) para sacar el número total, por otro lado, para sacar 5 pistas aleatorias se hizo un “while” de 5 iteraciones que recorría, además se recorrieron los dos árboles al mismo tiempo.

Igual que el requerimiento 1, este requerimiento se pensó para que sea lo más eficiente posible, por lo cual para calcular el total de pistas únicas y sin repeticiones lo que se hizo fue comparar el “track\_id” de las pistas musicales por medio de una tabla de hash y para esto se aprovechó que en la tabla de hash no se pueden guardar dos llaves con el mismo valor además de que para encontrar un elemento su complejidad es de O(1), aunque en el peor caso para añadir esa información en la tabla de hash en el peor caso al ser un mapa lineal probing será alrededor de O(n), por lo que esta comparación para encontrar el total de pistas únicas, además se utilizó m.size para encontrar cuantas llaves tiene esa tabla de hash. También para obtener los valores se utilizo el m.valueset() para obtener todos los valores pero solo se cogieron cinco (5) valores entonces la complejidad es de O(constante (5))

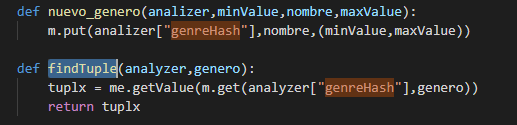
**Requerimiento 3:**



Para el requerimiento tres, se tomó el rango solicitado de “Instrumentalness” y “Tempo” respectivamente, por un lado, se recorrieron todas las opciones de “Track” dentro del rango por medio de dos “while” (por ser una lista de listas) para sacar el número total, por otro lado, para sacar 5 pistas aleatorias se hizo un “while” de 5 iteraciones que recorría, además se recorrió ambos árboles al mismo tiempo para hacer que a la hora de imprimir los valores se hiciera aleatoriamente.

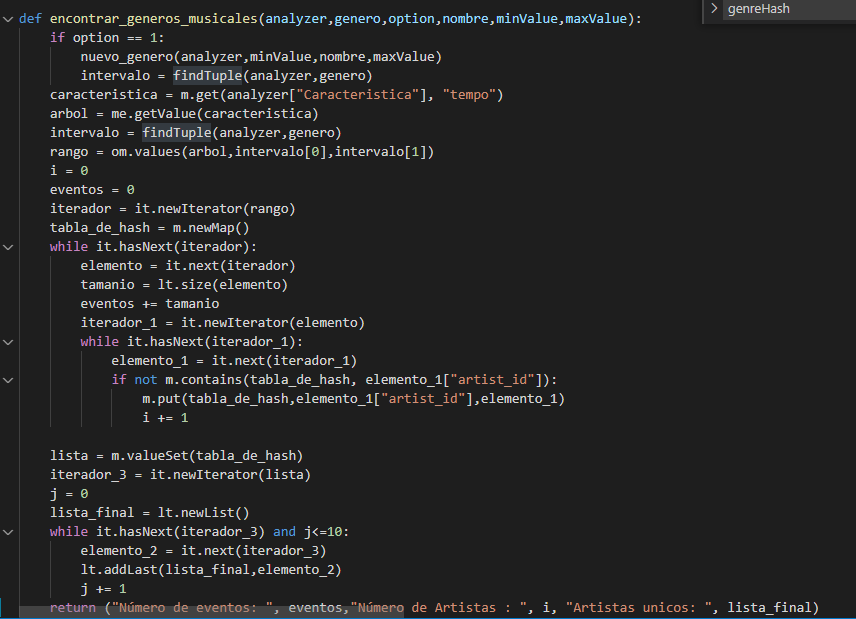
Este requerimiento básicamente se pensó igual que el requerimiento 2, entonces se hizo una tabla de hash aprovechando las dos propiedades descritas en el requerimiento y con esto calculamos el total de pistas únicas con una complejidad de O(1), después esta información se agrega a la tabla de hash que en el peor caso seria muy cercano a O(n). Y para encontrar el size del mapa su complejidad es de O(1). Para retornar la información de las pistas de música se usa un m.valueset() pero se busca los primeros 5 listas, por lo que su complejidad es de O(constante(5)).

**Requerimiento 4:**



La función “nuevo\_genero” agrega a la tabla de hash, dentro de Genrehash de analyzer, un nuevo genero con sus valores máximos y mínimos. Esta función se pensó para que el usuario en dado caso quiera meter un nuevo género, esta función mete información a la tabla de hash, por lo que en el peor caso seria una complejidad muy cercana a O(n)

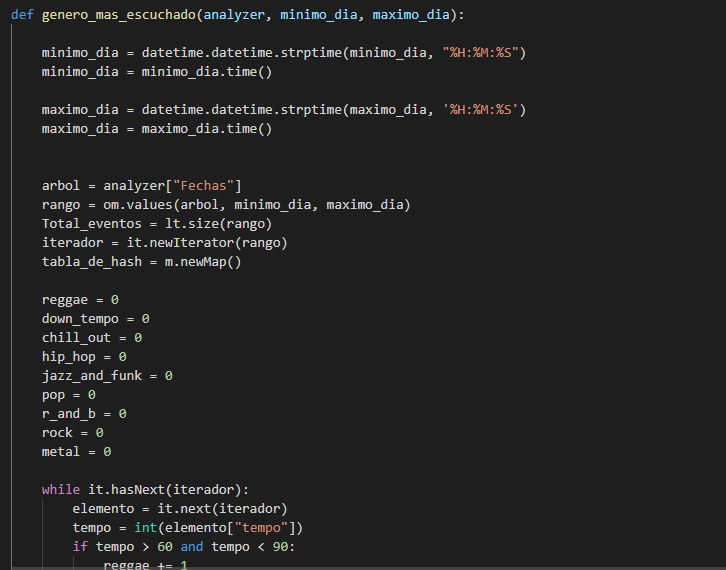
La función “findTuple” busca en la tabla de hash, dentro de Genrehash de analyzer, el rango de algún genero por medio de una tupla. Esta función se hace para encontrar el rango de cada género (igualmente en el caso de que el usuario seleccione si quiere adicionar un nuevo género). La complejidad de buscar un elemento en una tabla de hash es de O(1).



Con la tabla de hash de géneros en la llave Genrehash de analyzer, se recorrieron los tempos para hacer un rango con el valor mínimo del género solicitado, por medio de un “while” se contaron todos los eventos, dentro del mismo se creó otro “while” que filtraba todos los eventos que cumplían el rango del evento solicitado, por último, para los autores hizo otro “while” para sacar lun top 10 de autores para cada género.

Este requerimiento al igual que los anteriores se pensó para ser lo mas eficientemente posible, por lo que para encontrar el total de eventos de escucha se saco el size de cada lista para sumarlo a una variable esto es de O(1), para encontrar el total de eventos de escucha de cada genero se hizo una variable y se le sumo cada vez que se agregaba a la tabla de hash cada vez que el genero se guardaba por lo que su complejidad es cercana a O(n). Finalmente, para encontrar los artistas únicos se guardo como llave el “artista\_id” en la tabla de hash y con base a las propiedades descritas anteriormente esto es de complejidad O(1) y al guardar esta información en una lista tiene una complejidad de O(constante(10))

**Requerimiento 5:**



Se intento sacar el rango de las fechas solicitadas con en el árbol binario dentro de “analyzer[“Fechas”]”, posteriormente, con el valor y la llave “tempo” se clasifico en número de géneros que aparecen en el dominio solicitado por medio de un while.