**RETO 3**

**Estructuras de Datos y Algoritmos**

***Estudiante A:*** *Daniel Andrés Bernal Cáceres 202020706*

***Estudiante B:*** *Juan Martín Santos Ayala 202013610*

***Correos:*** *da.bernalc1@uniandes.edu.co*

*j.santosa* *@uniandes.edu.co*

1. **Preparación del ambiente de trabajo:**

Para la preparación del ambiente de trabajo, creamos los archivos correspondientes al modelo “MVC” para independizar los cambios hechos en código y que no se vieran reflejados en el producto entregado al usuario.

* 1. **Carga de archivos**

Para cargar los archivos primero creamos un analizador para guardar la información de un archivo de forma organizada. Tal organización fue:

def newAnalyzer():

    analyzer = {'event': None,

                'tracks': None,

                'artists': None,

                'instrumentalness': None,

                'acousticness': None,

                'liveness':  None,

                'speechiness': None,

                'energy': None,

                'danceability': None,

                'valence': None,

                'tempo': None,

                'hashtag': None

                }

Creamos una lista vacía con todas las pistas cargadas para facilitar la consulta del tamaño total de datos cargados del archivo CSV.

    analyzer['event'] = lt.newList('ARRAY\_LIST')

Se crean índices (Maps) por los siguientes criterios: ‘track\_id’

    analyzer['tracks'] = om.newMap(omaptype='RBT')

    analyzer['artists'] = om.newMap(omaptype='RBT')

    analyzer['instrumentalness'] = om.newMap(omaptype='RBT')

    analyzer['acousticness'] = om.newMap(omaptype='RBT')

    analyzer['liveness'] = om.newMap(omaptype='RBT')

    analyzer['speechiness'] = om.newMap(omaptype='RBT')

    analyzer['energy'] = om.newMap(omaptype='RBT')

    analyzer['danceability'] = om.newMap(omaptype='RBT')

    analyzer['valence'] = om.newMap(omaptype='RBT')

    analyzer['tempo'] = om.newMap(omaptype='RBT')

    analyzer['hashtag'] = mp.newMap(numelements=17,

                                    maptype='CHAINING',

                                    loadfactor=0.5)

Después de crear el catálogo lo que hacemos es agregar toda la información del archivo en una lista de tipo ARRAY\_LIST, esta la usamos para conocer el size que lo necesitamos en varias funciones.

def addVideoToCat(catalog, video):

    lt.addLast(catalog['videos'], video)

Después de crear el catálogo lo que hacemos es adicionar el track\_id de cada evento al map. Si el track\_id ya está en el árbol, se adiciona a su lista respectiva y se actualiza el index. Si no se encuentra creado un nodo para ese id en el árbol, se crea y se actualiza el indeice del id de las pistas.

Actualiza el track\_id de cada evento y se adiciona al map.

def updateIdIndex(maps, event):

    eventId = event['track\_id']

    entry = om.get(maps, eventId)

    if entry is None:

        datantry = newDataEntry(event)

        om.put(maps, eventId, datantry)

    else:

        datantry = me.getValue(entry)

    addEntry(datantry, event)

    return maps

Actualiza el artist\_id de cada evento y se adiciona al map.

def updateartistsIndex(maps, event):

    eventId = event['artist\_id']

    entry = om.get(maps, eventId)

    if entry is None:

        datantry = newDataEntry(event)

        om.put(maps, eventId, datantry)

    else:

        datantry = me.getValue(entry)

    addEntry(datantry, event)

    return maps

Después de crear el catálogo lo que hacemos es actualizar los arboles de cada una de las características del contenido. Así mismo, agregamos una lista con todos los eventos de escucha al árbol dependiendo del valor que estos tienen en esa característica.

def updateContCara(maps, event):

    caracteristics = ['instrumentalness', 'acousticness',

                      'liveness', 'speechiness', 'energy',

                      'danceability', 'valence', 'tempo']

    for i in caracteristics:

        entry = om.get(maps[i], float(event[i]))

        if entry is None:

            datantry = newDataEntry(event)

            om.put(maps[i], float(event[i]), datantry)

        else:

            datantry = me.getValue(entry)

        addEntry(datantry, event)

    return maps

Después añadimos un evento a la lista que es valor en cada uno de los mapas del analyzer.

def newDataEntry(event):

    entry = {'lstevent': None}

    entry['lstevent'] = lt.newList('ARRAY\_LIST')

    return entry

Después creamos una entrada en el índice por track\_id, es decir en el árbol binario.

def newDataEntry(event):

    entry = {'lstevent': None}

    entry['lstevent'] = lt.newList('ARRAY\_LIST')

    return entry

* 1. **Mostrar la información al usuario**

Por último, antes de comenzar a realizar los requerimientos, imprimimos la información al usuario. Acá fue donde nos dimos cuenta de que era mejor tener todos los videos en una lista en específico. Esto nos serviría para conocer el size del archivo a leer y en primera medida todos los datos del csv.

**La complejidad de nuestro catálogo es de O(n logn).**

1. **Requerimiento 1**

El requerimiento 1 nos exigía encontrar cuántas reproducciones se tienen en el sistema de recomendación basado en una característica de contenido y con un rango determinado.

def caracterizeReproductions(maps, characteristic, keylo, keyhi):

    exists = characteristic in maps

    if exists is True:

        events = om.values(maps[characteristic], keylo, keyhi)

*# creamos una tabla de has para guardar a los artistas únicos*

        artistsMap = mp.newMap(34500,

                               maptype='PROBING',

                               loadfactor=0.5)

        eventsq = 0

        iterator = ite.newIterator(events)

        while ite.hasNext(iterator):

*# este elemento tiene la lista con los eventos*

            eventlist = ite.next(iterator)

*# Se suman la cantidad de eventos*

            eventsq += lt.size(eventlist['lstevent'])

            newiterator = ite.newIterator(eventlist['lstevent'])

            while ite.hasNext(newiterator):

*# Este ya es el evento con sus caracteristicas*

                event = ite.next(newiterator)

*# Se agrega a una tabla de hash los id's de los artistas*

                mp.put(artistsMap, event['artist\_id'], 0)

*# Es la cantidad de artistas unicos del requerimiento*

        artistsq = mp.size(artistsMap)

        return eventsq, artistsq

    else:

        return 0

Al final de esta función obtenemos la cantidad de eventos que hay y la cantidad de artistas únicos, y en el view sólo imprimimos los que nos pide el usuario.

**La complejidad del requerimiento 1 es de O( 2n log(n) ).**

1. **Requerimiento 2 (Estudiante A)**

El requerimiento 2 nos exigía las pistas en el sistema de recomendación que podrían utilizarse en una fiesta.

def partyMusic(maps, keylo1, keyhi1, keylo2, keyhi2):

    easylist = studyMap(maps, keylo1, keyhi1, keylo2, keyhi2,

                        'energy', 'danceability')

    if easylist is None:

        return 0

    else:

        tracksMap = mp.newMap(34500,

                              maptype='PROBING',

                              loadfactor=0.5)

        iterator = ite.newIterator(easylist)

        while ite.hasNext(iterator):

            events = ite.next(iterator)

            mp.put(tracksMap, events['track\_id'], events)

        size = lt.size(tracksMap)

        if size > 5:

            tracklist = 0

        else:

            lista = mp.valueSet(tracksMap)

            tracklist = get5artists(tracksMap, lista, size)

        return size, tracklist

Busca dentro del mapa deseado, en un rango especificado por el usuario, los eventos con una pista única que cumplen el rango especificado.

def studyMap(maps, keylo1, keyhi1, keylo2, keyhi2, caract1, caract2):

*# Aca se organizaran los eventos por segunda vez*

    eventsmap = om.newMap(omaptype='RBT')

*# Es la lista con todos los eventos de instrumentalness*

    eventslist = om.values(maps[caract1], keylo1, keyhi1)

    if lt.isEmpty(eventslist):

        return None

    else:

        iterator = ite.newIterator(eventslist)

        while ite.hasNext(iterator):

            events = ite.next(iterator)

            newiterator = ite.newIterator(events['lstevent'])

            while ite.hasNext(newiterator):

                event = ite.next(newiterator)

                om.put(eventsmap, float(event[caract2]), event)

        return om.values(eventsmap, keylo2, keyhi2)

Función que me retorna 5 valores al azar de una lista de listas pasada por parámetro

def get5artists(map, lista, size):

    finallist = lt.newList("ARRAY\_LIST")

    eventlist = random.sample(range(size), 5)

    for i in eventlist:

        keya = lt.getElement(lista, i)

        lt.addLast(finallist, keya)

    return finallist

Al final de esta función obtenemos la cantidad de eventos de escucha que cumple la característica dada.

**La complejidad de nuestro requerimiento 2 es de O( n ( 3log(n)+ 2 ) ).**

1. **Requerimiento 3 (Estudiante B)**

El requerimiento 3 nos exigía las pistas en el sistema de recomendación que podrían ayudar a los usuarios en su periodo de estudio.

def studyMusic(maps, keylo1, keyhi1, keylo2, keyhi2):

    caract1 = 'instrumentalness'

    caract2 = 'tempo'

    easylist = studyMap(maps, keylo1, keyhi1, keylo2, keyhi2, caract1, caract2)

    if easylist is None:

        return 0

    else:

        tracksMap = mp.newMap(34500,

                              maptype='PROBING',

                              loadfactor=0.5)

        iterator = ite.newIterator(easylist)

        while ite.hasNext(iterator):

            events = ite.next(iterator)

            mp.put(tracksMap, events['track\_id'], events)

        size = lt.size(tracksMap)

        if size > 5:

            tracklist = 0

        else:

            lista = mp.valueSet(tracksMap)

            tracklist = get5artists(tracksMap, lista, size)

        return size, tracklist

Busca dentro del mapa deseado, en un rango especificado por el usuario, los eventos con una pista única que cumplen el rango especificado.

def studyMap(maps, keylo1, keyhi1, keylo2, keyhi2, caract1, caract2):

*# Aca se organizaran los eventos por segunda vez*

    eventsmap = om.newMap(omaptype='RBT')

*# Es la lista con todos los eventos de instrumentalness*

    eventslist = om.values(maps[caract1], keylo1, keyhi1)

    if lt.isEmpty(eventslist):

        return None

    else:

        iterator = ite.newIterator(eventslist)

        while ite.hasNext(iterator):

            events = ite.next(iterator)

            newiterator = ite.newIterator(events['lstevent'])

            while ite.hasNext(newiterator):

                event = ite.next(newiterator)

                om.put(eventsmap, float(event[caract2]), event)

        return om.values(eventsmap, keylo2, keyhi2)

Función que me retorna 5 valores al azar de una lista de listas pasada por parámetro

def get5artists(map, lista, size):

    finallist = lt.newList("ARRAY\_LIST")

    eventlist = random.sample(range(size), 5)

    for i in eventlist:

        keya = lt.getElement(lista, i)

        lt.addLast(finallist, keya)

    return finallist

Al final de esta función obtenemos la cantidad de eventos de escucha que cumple la característica dada.

**La complejidad de nuestro requerimiento 3 es de O( n ( 3log(n) + 2 ) ).**

1. **Requerimiento 4**

El requerimiento 4 nos exigía saber cuantas cancones se tienen por cada género y cuantos artistas de tienen en cada género.

Función que me retorna 10 valores al azar de una lista de listas pasada por parámetro.

def get10artists(artistmap, lista, size):

    finallist = lt.newList("ARRAY\_LIST")

    eventlist = random.sample(range(size), 10)

    for i in eventlist:

        keya = lt.getElement(lista, i)

        lt.addLast(finallist, keya)

    return finallist

Función que retorna la cantidad de reproducciones de canciones de genero reggae, la cantidad de artistas únicos y 10 id's de artistas al azar.

def reggae(maps):

*# Aca se agregaran los artistas unicos*

    reggaemap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    reggae = om.values(maps['tempo'], 60, 90)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(reggae)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(reggaemap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(reggaemap)

    size = lt.size(lista)

    artistslist = 10(reggaemap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero Down-tempo, la cantidad de artistas únicos y 10 id's de artistas al azar.

def down(maps):

*# Aca se agregaran los artistas unicos*

    downmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    down = om.values(maps["tempo"], 70, 100)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(down)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(downmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(downmap)

    size = lt.size(lista)

    artistslist = 10(downmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero Chill-out, la cantidad de artistas unicos y 10 id's de artistas al azar.

def chill(maps):

*# Aca se agregaran los artistas unicos*

    chillmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    chill = om.values(maps["tempo"], 90, 120)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(chill)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(chillmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(chillmap)

    size = lt.size(lista)

    artistslist = 10(chillmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero Hip-hop, la cantidad de artistas únicos y 10 id's de artistas al azar.

def hiphop(maps):

*# Aca se agregaran los artistas unicos*

    hiphopmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    hiphop = om.values(maps["tempo"], 85, 115)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(hiphop)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(hiphopmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(hiphopmap)

    size = lt.size(lista)

    artistslist = 10(hiphopmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero Jazz and funk, la cantidad de artistas únicos y 10 id's de artistas al azar.

def jazzfunk(maps):

*# Aca se agregaran los artistas unicos*

    jazzfunkmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    jazzfunk = om.values(maps["tempo"], 120, 125)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(jazzfunk)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(jazzfunkmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(jazzfunkmap)

    size = lt.size(lista)

    artistslist = 10(jazzfunkmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de género Pop, la cantidad de artistas únicos y 10 id's de artistas al azar.

def pop(maps):

*# Aca se agregaran los artistas unicos*

    popmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    pop = om.values(maps["tempo"], 100, 130)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(pop)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(popmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(popmap)

    size = lt.size(lista)

    artistslist = 10(popmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero R&B, la cantidad de artistas únicos y 10 id's de artistas al azar.

def ryb(maps):

*# Aca se agregaran los artistas unicos*

    rybmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    ryb = om.values(maps["tempo"], 60, 80)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(ryb)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(rybmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(rybmap)

    size = lt.size(lista)

    artistslist = 10(rybmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero rock, la cantidad de artistas únicos y 10 id's de artistas al azar.

def rock(maps):

*# Aca se agregaran los artistas unicos*

    rockmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    rock = om.values(maps["tempo"], 110, 140)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(rock)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(rockmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(rockmap)

    size = lt.size(lista)

    artistslist = 10(rockmap, lista, size)

    return reproductions, artistslist, size

Función que retorna la cantidad de reproducciones de canciones de genero metal, la cantidad de artistas únicos y 10 id's de artistas al azar.

def metal(maps):

*# Aca se agregaran los artistas unicos*

    metalmap = om.newMap(omaptype='RBT')

    reproductions = 0

*# Se obtienen los eventos en el rango del genero*

    metal = om.values(maps["tempo"], 100, 160)

*# Recorre el arbol y busca los valores del genero*

    iterator = ite.newIterator(metal)

    while ite.hasNext(iterator):

        eventlist = ite.next(iterator)

*# agrega la cantidad de videos que hay dentro de cada valor*

        reproductions += lt.size(eventlist['lstevent'])

        newiterator = ite.newIterator(eventlist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

*# Agrega a los artistas a un mapa para saber los unicos*

            om.put(metalmap, event['artist\_id'], event)

*# Es una lista con 5 artists\_id al azar*

    lista = om.keySet(metalmap)

    size = lt.size(lista)

    artistslist = 10(metalmap, lista, size)

    return reproductions, artistslist, size

Al final de esta función obtenemos 10 llaves al azar y la cantidad de reproducciones totales de cada género.

**La complejidad de nuestro requerimiento 4 es de O( 9n (log(n) + n + 1 ) )**

1. **Requerimiento 5**

El requerimiento 5 nos exigía 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.

Función que busca los eventos en un rango de tiempo.

Retorna la cantidad de eventos de escucha totales y un mapa con los eventos

def musicInTime(maps, time1, time2):

    init = seconds(time1)

    final = seconds(time2)

    if init < 0 or final < 0:

        return 0

    else:

*# thelist = lt.newList("ARRAY\_LIST")*

        themap = om.newMap(omaptype='RBT')

        eventsq = 0

*# Sacamos lo eventos en el rango deseado por el usuario*

        getusers = om.values(maps['time'], init, final)

*# Recorremos la lista de las listas de los eventos*

        iterator = ite.newIterator(getusers)

        while ite.hasNext(iterator):

            eventslist = ite.next(iterator)

*# Para encontrar el numero total de eventos en el rango*

            newiterator = ite.newIterator(eventslist['lstevent'])

            while ite.hasNext(newiterator):

                event = ite.next(newiterator)

                eventsq += 1

                entry = om.get(themap, float(event['tempo']))

                if entry is None:

                    dataentry = newDataEntry(event)

                    om.put(themap, float(event['tempo']), dataentry)

                else:

                    dataentry = me.getValue(entry)

                    addEntry(dataentry, event)

        return eventsq, themap

Función que encuentra la cantidad de reproducciones por género.

def getgenres(maps):

    thehash = lt.newList("ARRAY\_LIST")

    reggae = om.values(maps, 60, 90)

    reggaesize = getsize(reggae)

    lt.addLast(thehash, ('Reggae', reggaesize))

    down = om.values(maps, 70, 100)

    downsize = getsize(down)

    lt.addLast(thehash, ('Down-tempo', downsize))

    chill = om.values(maps, 90, 120)

    chillsize = getsize(chill)

    lt.addLast(thehash, ('Chill-out', chillsize))

    hiphop = om.values(maps, 85, 115)

    hiphopsize = getsize(hiphop)

    lt.addLast(thehash, ('Hip-hop', hiphopsize))

    jazzfunk = om.values(maps, 120, 125)

    jazzfunksize = getsize(jazzfunk)

    lt.addLast(thehash, ('Jazz and Funk', jazzfunksize))

    pop = om.values(maps, 100, 130)

    popsize = getsize(pop)

    lt.addLast(thehash, ('Pop', popsize))

    ryb = om.values(maps, 60, 80)

    rybsize = getsize(ryb)

    lt.addLast(thehash, ('R&B', rybsize))

    rock = om.values(maps, 110, 140)

    rocksize = getsize(rock)

    lt.addLast(thehash, ('Rock', rocksize))

    metal = om.values(maps, 100, 160)

    metalsize = getsize(metal)

    lt.addLast(thehash, ('Metal', metalsize))

    thehash = sortgenres(thehash)

    return thehash

Función que busca cual es el género que tuvo más reproducciones.

def getgenre(maps, genretuple, table, table2):

    newmap = om.newMap(omaptype='RBT')

    if genretuple[0] == "Reggae":

        value = om.values(maps, 60, 90)

    elif genretuple[0] == "Down-tempo":

        value = om.values(maps, 70, 100)

    elif genretuple[0] == "Chill-out":

        value = om.values(maps, 90, 120)

    elif genretuple[0] == "Hip-hop":

        value = om.values(maps, 85, 115)

    elif genretuple[0] == "Jazz and Funk":

        value = om.values(maps, 120, 125)

    elif genretuple[0] == "Pop":

        value = om.values(maps, 100, 130)

    elif genretuple[0] == "R&B":

        value = om.values(maps, 60, 80)

    elif genretuple[0] == "Rock":

        value = om.values(maps, 110, 140)

    elif genretuple[0] == "Metal":

        value = om.values(maps, 100, 160)

    iterator = ite.newIterator(value)

    while ite.hasNext(iterator):

        eventslist = ite.next(iterator)

        newiterator = ite.newIterator(eventslist['lstevent'])

        while ite.hasNext(newiterator):

            event = ite.next(newiterator)

            entry = om.get(newmap, event['track\_id'])

            if entry is None:

                dataentry = newDataEntry(event)

                om.put(newmap, event['track\_id'], dataentry)

            else:

                dataentry = me.getValue(entry)

                addEntry(dataentry, event)

    tracksid = om.keySet(newmap)

    finallist = lt.newList("ARRAY\_LIST")

    tracksiterator = ite.newIterator(tracksid)

    while ite.hasNext(tracksiterator):

        info = ite.next(tracksiterator)

        entry = mp.get(table, info)

        entryvalue = me.getValue(entry)

        hasht = mp.newMap(numelements=15, maptype='PROBING',

                          loadfactor=0.5)

        newiterator = ite.newIterator(entryvalue['lstevent'])

        while ite.hasNext(newiterator):

            trackhash = ite.next(newiterator)

            hashtag = trackhash['hashtag'].lower()

            mp.put(hasht, hashtag, 0)

            vader = mp.get(table2, hashtag)

            if vader is None:

                break

            else:

                vadevalue = me.getValue(vader)

                juandiego = ite.newIterator(vadevalue['lstevent'])

                sumvader = 0

                while ite.hasNext(juandiego):

                    pos = ite.next(juandiego)

                    if pos['vader\_avg'] == '':

                        break

                    else:

                        averagevader = float(pos['vader\_avg'])

                        sumvader += averagevader

        nohash = mp.size(hasht)

        avgvader = sumvader / nohash

        lt.addLast(finallist, (info, nohash, avgvader))

    return finallist

Al final de esta función obtenemos una lista con los valores de track\_id, cantidad de Hashtags y Vader promedio.

La complejidad de nuestro requerimiento 5 es de O( n ( 5log(n) + 2n + 10 ) ).

* Comparar la complejidad de los requerimientos implementados en el Reto No. 1 con los implementados en este reto.

**Requerimiento 1:**

En nuestro reto 1 la complejidad era de O(n logn) y en el reto 2 es de O(n logn).

**Requerimiento 2:**

En nuestro reto 1 la complejidad era de O(2n^2) y en el reto 2 es de O(n).

**Requerimiento 3:**

En nuestro reto 1 la complejidad era de O(2n^2) y en el reto 2 es de O(n).

**Requerimiento 4:**

En nuestro reto 1 la complejidad era de O(2n) y en el reto 2 es de O(n).

Estas complejidades fueron calculadas teniendo en cuenta el peor de los casos.

Dicho esto, se puede concluir que los diferentes algoritmos utilizados para el reto 2 implementando los maps (tablas de Hash) son claramente más eficientes tanto en la búsqueda como también para agregar elementos, permitiéndonos tener búsquedas con complejidades logarítmicas. Conllevando, a que los tiempos fueran menores a los del reto 1.