ANÁLISIS RETO 3

David Leonardo Almanza Márquez – 202011293 – d.almanza@uniandes.edu.co
Laura Daniela Arias Flórez – 202020621 – l.ariasf@uniandes.edu.co

COMPLEJIDAD REQUERIMIENTO 1

```
def getCharacteristicReproductions(catalog, characteristic, minrange, toprange):
    tree = me.getValue(mp.get(catalog['content_features'], characteristic))
    #get => O(1)
    total = 0
    artists = lt.newList('ARRAY')
    for value in lt.iterator(om.values(tree, minrange, toprange)):
        #O(N) donde N es la cantidad de valores de feature en el rango pedido.
        events = value['valueevents']
        total += lt.size(events)

        for event in lt.iterator(events):
            #O(M), donde M es la cantidad de canciones en el rango pedido del Feature
            if not lt.isPresent(artists, event['artist_id']):
                lt.addLast(artists, event['artist_id'])
    total2 = lt.size(artists)
    artists.clear()
    return total, total2
```

En cuanto a memoria, para resolver este requerimiento se crea un arreglo que contiene los Ids de los artistas. Este requerimiento toma alrededor de 5 segundos en ejecutarse.

COMPLEJIDAD REQUERIMIENTOS 2 Y 3 (FUNCIÓN GENERALIZADA PARA AMBOS)

```
def getMusic(catalog, min1, max1, min2, max2, name1, name2):
    OneValues = om.values(me.getValue(mp.get(catalog['content_features'], name1)),min1, max1)
    TwoValues = om.values(me.getValue(mp.get(catalog['content_features'], name2)), min2, max2)
    #Values en un rango (RBT) => O()
    lst = lt.newList(datastructure='ARRAY_LIST')

    for dictTwo in lt.iterator(TwoValues):
        #O(N) donde N es la cantidad de valores de feature en el rango pedido de Danceability/Tempo.
        trackIdsList_Two = mp.valueSet(dictTwo['track_ids'])
        for dictOne in lt.iterator(OneValues):
            #O(N') donde N' es la cantidad de valores de feature en el rango pedido de Energy/Instrumental
ness.
            for event in lt.iterator(trackIdsList_Two):
```

```
            #O(M), donde M es la cantidad de canciones en el rango pedido de Danceability/Tempo.
            if(mp.contains(dictOne['track_ids'], event['track_id']) == True):
                lt.addLast(lst, event)

    return lst
```

En cuanto a memoria, con el propósito de imprimir 5 canciones aleatorias con sus respectivos valores de característica, se crea un arreglo con todos los eventos que cumplen con las condiciones del requerimiento. Este requerimiento tarda alrededor de 4 segundos en ejecutarse.

COMPLEJIDAD REQUERIMIENTO 4

View:

```
    elif int(inputs[0]) == 5:
        genreslist = input("Nombre de los géneros (separados por coma sin espacios): ").lower().split(',')
        totalcount = 0
        countlist = lt.newList('ARRAY_LIST')
        artists = lt.newList('ARRAY_LIST')
        artistcountlist = lt.newList('ARRAY_LIST')
        for genrename in genreslist:
            #O(N) donde N es el número de géneros ingresados por el usuario
            if not mp.contains(catalog['genres'], genrename.lower()):
                mintempo = float(input("Valor mínimo de tempo del género " + genrename + ": "))
                maxtempo = float(input("Valor máximo de tempo del género" + genrename + ": "))
                newgenrecount = controller.addUserGenre(catalog, genrename, mintempo, maxtempo)
                totalcount += newgenrecount[0]
                lt.addLast(countlist, newgenrecount[0])
                lt.addLast(artists, newgenrecount[1])
                lt.addLast(artistcountlist, lt.size(newgenrecount[1]))
            else:
                result = controller.getGenreReproductions(catalog, genrename)
                totalcount += result[0]
                lt.addLast(countlist, result[0])
                lt.addLast(artists, result[1])
                lt.addLast(artistcountlist, lt.size(result[1]))
        printGenresInfo(genreslist, totalcount, countlist, artists, artistcountlist)
```

Model:

```
def addUserGenre(catalog, genrename, mintempo, maxtempo):
    count = 0
    genre = newGenre(genrename, mintempo, maxtempo)
    mp.put(catalog['genres'], genrename.lower(), genre)
    tempotree = me.getValue(mp.get(catalog['content_features'], "tempo"))
```

```
    eventslists = om.values(tempotree, mintempo, maxtempo)
    artists = lt.newList('ARRAY_LIST')
    for lst in lt.iterator(eventslists):
        #O(N) donde N es la cantidad de tempos diferentes que hay en el rango dado
        for event in lt.iterator(lst):
            #O(M) donde M es la cantidad de eventos con el mismo tempo
            lt.addLast(genre['events'], event)
            count += 1
            if not lt.isPresent(artists, event['artist_id']):
                lt.addLast(artists, event['artist_id'])
    return count, artists
```

```
def getGenreReproductions(catalog, genrename):
    genre = me.getValue(mp.get(catalog['genres'], genrename))
    count = lt.size(genre['events'])
    artists = lt.newList('ARRAY_LIST')
    for event in lt.iterator(genre['events']):
        #O(N) donde N es la cantidad de eventos que hacen parte del género
        if not lt.isPresent(artists, event['artist_id']):
            lt.addLast(artists, event['artist_id'])
    return count, artists
```

Con respecto a la memoria, en el view se crean 3 arreglos para guardar la información del número de eventos por género, los artistas de los géneros, y el número de artistas por género respectivamente. Ya en model, en addUserGenre se adicionan los datos requeridos al catálogo si el usuario ingresó un género nuevo, y en getGenreReproductions se crea un arreglo para guardar los artistas únicos del género pre-existente.

COMPLEJIDAD REQUERIMIENTO 5

```
def generosEnRango(catalog, minHour, maxHour):
    DateMinHour = dt.datetime.strptime(minHour, '%H:%M:%S')
    minHour = DateMinHour.time()
    DateMaxHour = dt.datetime.strptime(maxHour, '%H:%M:%S')
    maxHour = DateMaxHour.time()
    #Cada funcion de la libreria datetime usada es O(1)
(se definen 9 variables)
    treeValues = om.values(catalog['hourTree'], minHour, maxHour)
    #Values en un rango (RBT) => O()
    for value in lt.iterator(treeValues):
        #O(N) donde N es la cantidad de valores en el rango pedido de tiempo.
        for genre in lt.iterator(mp.valueSet(catalog['genres'])):
            #O(C) donde C es la cantidad de generos, hay 9 generos.
            genreName = genre['name']
            generoLista = mp.get(value, genreName)
```

```
            if generoLista is not None:
                Lista = me.getValue(generoLista)
                #get => O(1)
                Total += lt.size(Lista)
                #size => O(1)
                          (Una gran cantidad de condicionales)
    generos = (Reggae, Down_Tempo, Chill_out, hip_hop, Jazz_and_Funk, Pop, RyB, Rock, Metal)
    genero = maxVariable(generos)
    #Esta es una funcion que defini por fuera para evitar ocupar mucho espacio en una sola funcion, pero e
s O(1)

    eventosConVader = om.newMap(omaptype='RBT', comparefunction=compareVader)
    for value in lt.iterator(treeValues):
        #O(N) donde N es la cantidad de valores en el rango pedido de tiempo.
        generoBuscado = mp.get(value, genero)
        if generoBuscado is not None:
            Lista = me.getValue(generoBuscado)
            for event in lt.iterator(Lista):
                #O(M) donde M es la cantidad de eventos asocioados al genero con mas reproducciones
                #En cada llave del RBT que tiene como llaves una hora.
                key = (event['user_id'], event['track_id'], event['created_at'])
                pareja = mp.get(catalog['UserHashtags'], key)
                hashtags = me.getValue(pareja)
                seenHashtags = 0
                for hashtag in lt.iterator(hashtags):
                    #O(H) Donde H es la cantidad de Hashtags asociados al evento.
                    promedioVader = 0
                    Hashtag_Vader = mp.get(catalog['hashtags'], hashtag)
                    if Hashtag_Vader is not None:
                        Vader = me.getValue(Hashtag_Vader)
                        promedioVader += Vader
                        seenHashtags += 1
                    if hashtag == lt.lastElement(hashtags):
                        if(seenHashtags != 0):
                            promedioVader = (promedioVader/seenHashtags)

                if(promedioVader != 0):
                    om.put(eventosConVader, promedioVader, event)
    return generos, genero, Total, eventosConVader
```

En cuanto a memoria, solo para este requerimiento se crean dos tablas de Hash: Una basada en el archivo de sentiment values y otra con el archivo que guarda el hashtag de cada reproducción. Como, contrario a los anteriores requerimientos, este es el único requerimiento que usa estos 2 archivos, este es el requerimiento que más uso de memoria hace.

El tiempo de ejecución de este requerimiento es alrededor de 3 segundos.