**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 REQUERIMIENTO 2**

def getPartyMusic(catalog, minEne, maxEne, minDan, maxDan):

    EneValues = om.values(me.getValue(mp.get(catalog['content\_features'], 'energy')),minEne, maxEne)

    DanValues = om.values(me.getValue(mp.get(catalog['content\_features'], 'danceability')), minDan, maxDan)

    #Values en un rango (RBT) => O()

    lstEnergyDance = lt.newList(datastructure='ARRAY\_LIST')

    for dictDance in lt.iterator(DanValues):

        #O(N) donde N es la cantidad de valores de feature en el rango pedido de Danceability.

        trackIdsList\_Dance = mp.valueSet(dictDance['track\_ids'])

        for dictEnergy in lt.iterator(EneValues):

            #O(N') donde N' es la cantidad de valores de feature en el rango pedido de Energy.

            for event in lt.iterator(trackIdsList\_Dance):

                #O(M), donde M es la cantidad de canciones en el rango pedido de Danceability.

                if(mp.contains(dictEnergy['track\_ids'], event['track\_id']) == True):

                    lt.addLast(lstEnergyDance, event)

    return lstEnergyDance

En cuanto a memoria, con el propósito de imprimir 3 canciones aleatorias con su respectivo Energy y danceability, 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 3**

**COMPLEJIDAD REQUERIMIENTO 4**

**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 es 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.