**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 de crear el catálogo lo que hacemos es agregar la información de los videos en el catálogo en la llave "country" según al país al que pertenecen. La llave mencionada es un mapa por lo que el nombre del país va a ser la llave y el valor va a ser una lista con todos los videos y su información que pertenecen a ese país.

* 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 la n cantidad de videos con mejores vistas o que fueron tendencia, en un país y una categoría específica.

def getVideosCat(catalog, category, country):

*# Tiene un espacio porque los nombres de las categorias se guardaron asi*

    parameter = ' ' + category

*# Obtenemos la pareja llave valor de la categoria*

    pair = mp.get(catalog['category'], parameter)

    if pair is None:

        newlist = 1

    else:

*# Sacamos la informacion de la pareja y es la lista con los videos*

        category\_list = me.getValue(pair)

        countryList = lt.newList('ARRAY\_LIST')

        iterator = ite.newIterator(category\_list['videos'])

        while ite.hasNext(iterator):

            info = ite.next(iterator)

            if info['country'] == country:

                lt.addLast(countryList, info)

*# Organizamos la lista de los videos por la cantidad de likes*

        newlist = sortVideos(countryList, cmpVideosByViews)

    return newlist

Al final de esta función obtenemos una lista con todos los elementos que cumplen los argumentos ordenados, y en el view sólo imprimimos los que nos pide el usuario.

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

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

El requerimiento 2 nos exigía encontrar el video con más días en tendencia, en un país.

def mostTrendingVideoCountry(catalog, country):

*# Obtenemos la pareja llave valor de la categoria*

    pair = mp.get(catalog['country'], country)

    if pair is None:

        newlist = 1

        days = 0

    else:

*# Sacamos la informacion de la pareja y es la lista con los videos*

        country\_list = me.getValue(pair)

        dictionary = {}

        iterator = ite.newIterator(country\_list['videos'])

        while ite.hasNext(iterator):

            info = ite.next(iterator)

*# Seleccionamos los valores que son utiles para la busqueda*

            newinfo = (info['title'], info['channel\_title'],

                       info['country'])

            llist = [1]

            if newinfo in dictionary:

                dictionary[newinfo].append(1)

            else:

                dictionary[newinfo] = llist

*# Sacamos la llave del dicionario cuyo valor es el mayor*

        newlist = max(dictionary, key=dictionary.get)

        days = dictionary[newlist]

    return newlist, days

Al final de esta función obtenemos una tupla, donde en la primera posición se encuentra la llave con la información del video más tendencia y en la segunda posición se encuentra una lista con la cantidad de veces que el video fue tendencia.

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

1. **Requerimiento 3**

El requerimiento 3 nos exigía encontrar el video con más días en tendencia, en una categoría.

def mostTrendingVideoCat(catalog, category):

*# Tiene un espacio porque los nombres de las categorias se guardaron asi*

    parameter = ' ' + category

*# Obtenemos la pareja llave valor de la categoria*

    pair = mp.get(catalog['category'], parameter)

    if pair is None:

        newlist = 1

        days = 0

    else:

*# Sacamos la informacion de la pareja y es la lista con los videos*

        category\_list = me.getValue(pair)

        dictionary = {}

        iterator = ite.newIterator(category\_list['videos'])

        while ite.hasNext(iterator):

            info = ite.next(iterator)

*# Seleccionamos los valores que son utiles para la busqueda*

            newinfo = (info['title'], info['channel\_title'],

                       info['category\_id'])

            llist = [1]

            if newinfo in dictionary:

                dictionary[newinfo].append(1)

            else:

                dictionary[newinfo] = llist

*# Sacamos la llave del dicionario cuyo valor es el mayor*

        newlist = max(dictionary, key=dictionary.get)

        days = dictionary[newlist]

    return newlist, days

Al final de esta función obtenemos una tupla, donde en la primera posición se encuentra la llave con la información del video más tendencia y en la segunda posición se encuentra una lista con la cantidad de veces que el video fue tendencia.

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

1. **Requerimiento 4**

El requerimiento 4 nos exigía encontrar los videos con más likes en un país y con una etiqueta en específico.

def mostLikedVideosCountryTag(catalog, country, tag):

*# Obtenemos la pareja llave valor de la categoria*

    pair = mp.get(catalog['country'], country)

    if pair is None:

        newList = 1

    else:

*# Obtenemos la lista con los videos de ese pais*

        countryList = me.getValue(pair)

        finalList = lt.newList('ARRAY\_LIST')

        iterator = ite.newIterator(countryList['videos'])

        while ite.hasNext(iterator):

            info = ite.next(iterator)

*# Buscamos los tags que desea el usuario*

            if tag in info['tags']:

*# Verificamos que no hayan repeticiones de videos*

                lt.addLast(finalList, info)

*# Le hacemos un sort a la lista dependiendo de los likes*

        newList = sortVideos(finalList, cmpVideosByLikes)

        return newList

Al final de esta función obtenemos una lista, donde se encuentran los videos con más likes del país dado por parámetro y que tienen la etiqueta dada.

**La complejidad de nuestro requerimiento 4 es de O(n)**

* 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.