Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999
Juan Diego Yepes - 202022391

Reto III

Índice:

1 Construcción del Analizador

Para construir el analizador, utilizamos en principio un diccionario con las llaves de los mapas, tablas de hash y listas que usaremos después:

Analyzer {
- 'listening_events': datastructure='ARRAY_LIST
- 'artists': maptype='PROBING'
- 'tracks': maptype='PROBING'
- 'instrumentalness': omaptype='RBT'
- 'acousticness': omaptype='RBT'
- 'liveness': omaptype='RBT'
- 'speechiness': omaptype='RBT'
- 'energy': omaptype='RBT'
- 'danceability': omaptype='RBT'
- 'valence': omaptype='RBT'
- 'tempo': omaptype='RBT'
- 'created_at': omaptype='RBT'
- 'hashtags': maptype='PROBING'
- 'vaders': maptype='PROBING'

Fue necesario iterar los tres archivos para la construcción de distintas estructuras de datos necesarias para la resolución de los requerimientos. En primera instancia, con el archivo **"Context Content Features"** se construyó:

- **Array List 'listening_events'**: Lista que guardara cada entrada del archivo csv
- **Mapa 'tracks'**: Un mapa llave "track id" y valor "id"
- **Mapa 'artists'**: Un mapa llave "Artist id" y valor "id"
- **RBT 'instrumentalness'**: Un árbol RBT de llave/nodo "instrumentalness" y valor igual a la entrada correspondiente del csv
- **RBT 'acousticness'**: Un árbol RBT de llave/nodo "acousticness" y valor igual a la entrada correspondiente del csv
- **RBT 'liveness'**: Un árbol RBT de llave/nodo "liveness" y valor igual a la entrada correspondiente del csv
- **RBT 'speechiness'**: Un árbol RBT de llave/nodo "speechiness" y valor igual a la entrada correspondiente del csv
- **RBT 'energy'**: Un árbol RBT de llave/nodo "energy" y valor igual a la entrada correspondiente del csv

- **RBT 'danceability'**: Un árbol RBT de llave/nodo "danceability" y valor igual a la entrada correspondiente del csv
- **RBT 'valence'**: Un árbol RBT de llave/nodo "valence" y valor igual a la entrada correspondiente del csv
- **RBT 'tempo'**: Un árbol RBT de llave/nodo "tempo" y valor igual a la entrada correspondiente del csv
- **RBT 'created_at'**: Un árbol RBT de llave/nodo "created_at" igual al rango de segundos (para facilitar el cáclculo) y valor igual a la entrada correspondiente del csv

De igual forma, con el archivo **"User Track Hashtags"** creamos un mapa correspondiente a:

- **Mapa 'hashtags'**: Un mapa llave un id único concatenando los elementos de "user id", "track id" y "created at"y valor "hashtag"

Finalmente, con el archivo **"Sentiment Values"** creamos un mapa correspondiente a:

- **Mapa 'vader'**: Un mapa llave "hashtag" y valor "vader average"

Con ese propósito, se utilizan las siguientes funciones:

1.1 Funciones del View

```
    if int(inputs[0]) == 1:
        analyzer = controller.init()
        print("Cargando informacion de los archivos ....")
        answer = controller.loadData(
            analyzer, events_analysis_file, hashtag_file, sentiment_values)
        print('Registro de eventos Cargados: ' + str(controller.eventsSize(
            analyzer)))
        print('Artistas unicos Cargados: ' + str(controller.artistsSize(
            analyzer)))
        print('Pistas unicas Cargados: ' + str(controller.tracksSize(
            analyzer)))
        print('\n')
        print('Primeros y ultimos 5 cargados, respectivamente: ')
        printfirstandlast5(analyzer)
        print('\n')
        print(
            "Tiempo [ms]: ",
            f"{answer[0]:.3f}", "  ||  ",
            "Memoria [kB]: ",
            f"{answer[1]:.3f}")
```

1.2 Funciones del Controller

```
def loadData(analyzer, file1, file2, file3):
    """
    Carga los datos de los archivos CSV en el modelo
    """
    delta_time = -1.0
    delta_memory = -1.0

    tracemalloc.start()
    start_time = getTime()
    start_memory = getMemory()

    loadEvents(analyzer, file1)
    loadHashtags(analyzer, file2)
    loadVader(analyzer, file3)

```

```
    stop_memory = getMemory()
    stop_time = getTime()
    tracemalloc.stop()

    delta_time = stop_time - start_time
    delta_memory = deltaMemory(start_memory, stop_memory)

    return delta_time, delta_memory
```

```
def loadEvents(analyzer, file):
    """
    Itera cada elemento del archivo csv
    """
    analysis_file = cf.data_dir + file
    input_file = csv.DictReader(open(analysis_file, encoding="utf-8"),
                                delimiter=",")
    for event in input_file:
        model.addEvent(analyzer, event)
```

```
def loadHashtags(analyzer, file):
    """
    Itera cada elemento del archivo csv
    """
    analysis_file = cf.data_dir + file
    input_file = csv.DictReader(open(analysis_file, encoding="utf-8"),
                                delimiter=",")
    for event in input_file:
        for event in input_file:
         key = event['user_id'] + event['track_id'] + event['created_at']
         model.addOnMap(analyzer, event['hashtag'], key, 'hashtags')
        model.addOnMap(analyzer, event['hashtag'], key, 'hashtags')
```

```
def loadVader(analyzer, file):
    """
    Itera cada elemento del archivo csv
    """
    analysis_file = cf.data_dir + file
    input_file = csv.DictReader(open(analysis_file, encoding="utf-8"),
                                delimiter=",")
    for vader in input_file:
        model.addOnMap(
            analyzer, vader['vader_avg'], vader['hashtag'], 'vaders')
```

1.3 Funciones del Model

```
def newAnalyzer():
    """ Inicializa el analizador

    Retorna el analizador inicializado.
    """
    analyzer = {'listening_events': None,
                'artists': None,
                'tracks': None,
                'instrumentalness': None,
                'acousticness': None,
                'liveness': None,
                'speechiness': None,
                'energy': None,
                'danceability': None,
                'valence': None,
                'tempo': None,
                'created_at': None,
                'hashtags': None,
                'vaders': None
                }

```

```
    analyzer['listening_events'] = lt.newList(datastructure='ARRAY_LIST')
    analyzer['artists'] = mp.newMap(
        numelements=40000, maptype='PROBING')
    analyzer['tracks'] = mp.newMap(
        numelements=40000, maptype='PROBING')
    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['created_at'] = om.newMap(omaptype='RBT')
    analyzer['hashtags'] = mp.newMap(
        numelements=100000, maptype='PROBING')
    analyzer['vaders'] = mp.newMap(
        numelements=10000, maptype='PROBING')

    return analyzer
```

```
def addEvent(analyzer, event):
    '''
    Agrega individualmente el evento al analyzer, en
    cada uno de sus mapas
    '''
    lt.addLast(analyzer['listening_events'], event)
    mp.put(analyzer['artists'], event['artist_id'], 0)
    mp.put(analyzer['tracks'], event['track_id'], 0)
    juancarlos(analyzer, event)
    addTimedEvent(
        analyzer, event['created_at'], event, 'created_at')
```

```
def juancarlos(analyzer, event):
    '''
    La funcion juancarlos() itera las caracteristicas
    y agrega el evento individual al mapa correspondiente
    '''
    yourtimeline = [
        'instrumentalness', 'acousticness',
        'liveness', 'speechiness', 'energy',
        'danceability', 'valence', 'tempo']
    for soundtrack in yourtimeline:
        addEventOnOrderedRBTMap(
            analyzer, float(event[soundtrack]),
            event, soundtrack)
```

```
def addOnMap(analyzer, event, key, map_name):
    '''
    Agrega los hashtags y los vaders a sus mapas individuales
    '''
    mp.put(analyzer[map_name], key, event)
```

```
def addEventOnProbingMap(analyzer, int_input, event, map_key):
    """
    La funcion de addEventOnProbingMap() adiciona el video al mapa
    tipo PROBING que se ha seleccionado.
    Args:
        analyzer: Analizador de eventos
        int_input: Llave a analizar
        video: Video a a adir
        map_key: Especifica cual mapa
    """
    selected_map = analyzer[map_key]
    existkey = mp.contains(selected_map, int_input)
    if existkey:
        entry = mp.get(selected_map, int_input)
```

```
        value = me.getValue(entry)
    else:
        value = newSeparator(int_input, map_key)
        mp.put(selected_map, int_input, value)
    lt.addLast(value['events'], event)
```

```
def addEventOnOrderedRBTMap(analyzer, int_input, event, map_key):
    """
    La funcion de addEventOnOrderedRBTMap() adiciona el video al arbol
    tipo RBT que se ha seleccionado.
    Args:
        analyzer: Analizador de eventos
        int_input: Llave a analizar
        video: Video a a adir
        map_key: Especifica cual mapa
    """
    selected_map = analyzer[map_key]
    entry = om.get(selected_map, int_input)
    if entry is not None:
        value = me.getValue(entry)
    else:
        value = newDataEntry()
        om.put(selected_map, int_input, value)
    lt.addLast(value['events'], event)
```

```
def newDataEntry():
    '''
    Crea un bucket para guardar todos los eventos dentro de
    la categoria
    '''
    entry = {'events': None}
    entry['events'] = lt.newList('ARRAY_LIST')
    return entry
```

```
def addTimedEvent(analyzer, int_input, event, map_key):
    '''
    Adiciona un evento a un arbol tipo RBT usando el
    tiempo de creacion del evento, con los segundos como
    llave
    '''
    time = int_input.split(" ")
    time = time[1].split(':')
    time = int(time[0])*3600 + int(time[1])*60 + int(time[2])
    addEventOnOrderedRBTMap(
        analyzer, time,
        event, map_key)
```

2 Primer Requerimiento

Se desea conocer cuántas reproducciones (eventos de escucha) se tienen en el sistema de recomendación basado en una característica de contenido y con un rango determinado, El sistema debe indicar el total de canciones y el número de artistas (sin repeticiones). Para dar respuesta a este requerimiento se debe recibir como entrada la siguiente información:

- La característica de contenido (ej.: valencia, sonoridad, etc.).
- El valor mínimo de la característica de contenido.
- El valor máximo de la característica de contenido.

Y como respuesta debe presentar en consola la siguiente información:

- El Total de los eventos de escucha o reproducciones.
- El número de artistas unicos (sin repeticiones)

2.1 Funciones del View

```
elif int(inputs[0]) == 2:
        criteria = input("Ingrese el criterio a evaluar: ")
        initial = float(input("Ingrese el limite inferior: "))
        final = float(input("Ingrese el limite superior: "))
        print("Buscando en la base de datos ....")
        result = controller.getEventsByRange(
            analyzer, criteria, initial, final)
        print('Registro de eventos Cargados: ' + str(result[0]))
        print('Artistas unicos Cargados: ' + str(result[1]))
```

2.2 Funciones del Controller

```
def getEventsByRange(analyzer, criteria, initial, final):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getEventsByRange(analyzer, criteria, initial, final)
```

2.3 Funciones del Model

```
def getEventsByRange(analyzer, criteria, initial, final):
    '''
    Retorna los varias caracteristicas de los
    eventos dado un criterio y rango en el mismo,
    buscandolos en un arbol
    Args:
        analyzer: Analizador de eventos
        criteria: Llave del analyzer a analizar
        initial: Inicio del rango
        final: Fin del rango
    '''
    lst = om.values(analyzer[criteria], initial, final)
    events = 0
    artists = mp.newMap(maptype='PROBING')
    tracks = mp.newMap(maptype='PROBING')

    for lstevents in lt.iterator(lst):
        events += lt.size(lstevents['events'])
        for soundtrackyourtimeline in lt.iterator(lstevents['events']):
            mp.put(artists, soundtrackyourtimeline['artist_id'], 1)
            mp.put(tracks, soundtrackyourtimeline['track_id'], 1)

    artists_size = mp.size(artists)
    tracks_size = mp.size(tracks)

    return events, artists_size, tracks_size, artists, tracks
```

3 Segundo Requerimiento (Juan Diego Yepes)

Se desea encontrar las pistas en el sistema de recomendación que pueden utilizarse en una fiesta que se tendrá próximamente. Se desea encontrar las canciones que tengan en cuenta las variables (Energy, y Danceability). El usuario podrá indicar los valores (rangos) para esos parámetros. Para dar respuesta a este requerimiento se debe recibir como entrada la siguiente información:

- El Valor mínimo de la característica Energy.
- El Valor máximo de la característica Energy.
- El Valor mínimo de la característica Danceability.

- El Valor máximo de la característica Danceability.

Y como respuesta debe presentar en consola la siguiente información:

- El total de pistas únicas (sin repeticiones).
- La información de 5 pistas seleccionadas aleatoriamente (incluya los valores Energy y Danceability para cada uno).

3.1 Funciones del View

```
elif int(inputs[0]) == 3:
        initialenergy = float(input(
            "Ingrese el limite inferior para la energ a: "))
        finalenergy = float(input(
            "Ingrese el limite superior para la energ a: "))
        energyrange = (initialenergy, finalenergy)
        initialdanceability = float(input(
            "Ingrese el limite inferior para la perreabilidad: "))
        finaldanceability = float(input(
            "Ingrese el limite superior para la perreabilidad: "))
        danceabilityrange = (initialdanceability, finaldanceability)
        print("Buscando en la base de datos ....")
        result = controller.getMusicToParty(
            analyzer, energyrange, danceabilityrange)
        print('Artistas unicos Cargados:', str(result[0]))
        print('Tracks  nicas  Cargadas:', str(result[1]))
        printRandom5(result[2], 'energy', 'danceability')
```

3.2 Funciones del Controller

```
def getMusicToParty(analyzer, energyrange, danceabilityrange):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getTrcForTwoCriteria(
        analyzer, energyrange, 'energy', danceabilityrange, 'danceability')
```

3.3 Funciones del Model

```
def getTrcForTwoCriteria(analyzer, criteria1range, str1, criteria2range, str2):
    '''
    Retorna los varias caracteristicas de los
    eventos dado dos criterios y rangos en el mismo,
    busc ndolos en ambos  rboles . El retorno son los eventos
    que cumplen con ambas caracteristicas
    Args:
        analyzer: Analizador de eventos
        criteria1range: Rango del criterio 1
        str1: Llave del analyzer del criterio 1
        criteria2range: Rango del criterio 2
        str2: Llave del analyzer del criterio 2
    '''
    criteria1 = om.values(analyzer[str1], criteria1range[0], criteria1range[1])
    submap = {'events': None}
    submap[str2] = om.newMap(omaptype='RBT')
    for event0 in lt.iterator(criteria1):
        for event0 in lt.iterator(event0['events']):
            addEvent0nOrderedRBTMap(submap, float(event0[str2]), event0, str2)
    result = om.values(submap[str2], criteria2range[0], criteria2range[1])
    artists = mp.newMap(maptype='PROBING')
```

```
    tracks = mp.newMap(maptype='PROBING')
    for event1 in lt.iterator(result):
        for eventi in lt.iterator(event1['events']):
            mp.put(artists, eventi['artist_id'], 1)
            mp.put(
                tracks, eventi['track_id'],
                (eventi[str1], eventi[str2]))
    return (mp.size(artists), mp.size(tracks), tracks)
```

4 Tercer Requerimiento (Jose Luis Tavera)

Se desea conocer las pistas en el sistema de recomendación que podrían ayudar a los usuarios en su periodo de estudio, para este fin se debe tener en cuenta tengan en cuenta las variables (Instrumentalness, y Tempo). Por ejemplo, para un grupo de estudio que desea trabajar de forma tranquila; en este caso se prefieren canciones instrumentales, sin letra o muy poca y de preferencia con un Tempo Largo (40–60 BPM) porque se sabe que favorece el aprendizaje.Para dar respuesta a este requerimiento se recibe como entrada la siguiente información:

- El valor mínimo del rango para Instrumentalness.
- El valor máximo del rango para Instrumentalness.
- El valor mínimo del rango para el Tempo.
- El valor máximo del rango para el Tempo.

Y como respuesta se espera que la consola presente la siguiente información:

- El total de pistas únicas (sin repeticiones).
- La información de 5 pistas seleccionadas aleatoriamente (incluya los valores de Instrumentalness y Tempo para cada uno).

4.1 Funciones del View

```
elif int(inputs[0]) == 4:
        initialinstrumentalness = float(input(
            "Ingrese el limite inferior para la instrumentalidad: "))
        finalinstrumentalness = float(input(
            "Ingrese el limite superior para la instrumentalidad: "))
        instrumentalnessrange = (
            initialinstrumentalness, finalinstrumentalness)
        initialtempo = float(input(
            "Ingrese el limite inferior para el tempo: "))
        finaltempo = float(input(
            "Ingrese el limite superior para el tempo: "))
        temporange = (initialtempo, finaltempo)
        print("Buscando en la base de datos ....")
        result = controller.getMusicToStudy(
            analyzer, instrumentalnessrange, temporange)
        print('Artistas unicos Cargados:', str(result[0]))
        print('Tracks  nicas  Cargadas:', str(result[1]))
        printRandom5(result[2], 'instrumentalness', 'tempo')
```

4.2 Funciones del Controller

```
def getMusicToStudy(analyzer, instrumentalnessrange, temporange):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getTrcForTwoCriteria(
        analyzer,
        instrumentalnessrange, 'instrumentalness', temporange, 'tempo')
```

4.3 Funciones del Model

```
def getTrcForTwoCriteria(analyzer, criteria1range, str1, criteria2range, str2):
    '''
    Retorna los varias caracteristicas de los
    eventos dado dos criterios y rangos en el mismo,
    busc ndolos en ambos  rboles . El retorno son los eventos
    que cumplen con ambas caracteristicas
    Args:
        analyzer: Analizador de eventos
        criteria1range: Rango del criterio 1
        str1: Llave del analyzer del criterio 1
        criteria2range: Rango del criterio 2
        str2: Llave del analyzer del criterio 2
    '''
    criteria1 = om.values(analyzer[str1], criteria1range[0], criteria1range[1])
    submap = {'events': None}
    submap[str2] = om.newMap(omaptype='RBT')
    for event0 in lt.iterator(criteria1):
        for event0 in lt.iterator(event0['events']):
            addEventOnOrderedRBTMap(submap, float(event0[str2]), event0, str2)
    result = om.values(submap[str2], criteria2range[0], criteria2range[1])
    artists = mp.newMap(maptype='PROBING')
    tracks = mp.newMap(maptype='PROBING')
    for event1 in lt.iterator(result):
        for eventi in lt.iterator(event1['events']):
            mp.put(artists, eventi['artist_id'], 1)
            mp.put(
                tracks, eventi['track_id'],
                (eventi[str1], eventi[str2]))
    return (mp.size(artists), mp.size(tracks), tracks)
```

5 Cuarto Requerimiento

Dada la siguiente tabla, que relaciona el Tempo con el género musical, se desea saber cuántas canciones se tienen por cada género (definidos como un rango de Tempo como se ve en la Tabla 4); adicionalmente, cuántos artistas se tienen en cada género. Para dar respuesta a este requerimiento se recibe como entrada la siguiente información:

- La lista de géneros musicales que se desea buscar. (ej.: Reggae, Hip-hop, Pop.).

En caso de desearlo, el usuario puede agregar un nuevo género musical en la búsqueda con las siguientes variables de entrada:

- Nombre único para el nuevo género musical.
- Valor mínimo del Tempo del nuevo género musical.
- Valor máximo del Tempo del nuevo género musical.

Y como respuesta se espera que la consola presente la siguiente información:

- El Total de los eventos de escucha o reproducciones.
- El Total de los eventos de escucha o reproducciones en cada género.
- El número de artistas unicos (sin repeticiones) en cada uno de los géneros musicales, y el ID de los primeros 10 artistas

5.1 Funciones del View

```
elif int(inputs[0]) == 5:
        genero = input("Ingrese el nombre del genero musical: ")
        lim_inf = int(input("Ingrese el limite inferior del Tempo: "))
        lim_sup = int(input("Ingrese el limite superior del Tempo: "))
        n = len(genre.keys()) + 1
        llave = str(n) + "- " + str(genero)
        genre[llave] = (lim_inf, lim_sup)
        print("El genero " + str(genero) + " ha sido agregado con exito!")
        print(genre)
        print('esta es la base de datos actualizada:')
        printgenresdict(genre)
```

```
elif int(inputs[0]) == 6:
        print('estos son los generos disponibles para consulta:')
        printgenresdict(genre)
        lista_generos = []
        generos = "1"
        print("Ingrese el n mero de cada uno de los generos a consultar.")
        print("Para dejar de agregar generos a la b squeda, presione enter.")
        while len(generos) > 0:
            generos = input('~')
            if len(generos) != 0:
                lista_generos.append(generos)
        ranges = controller.getRanges(lista_generos, genre)
        dicc = controller.getEventsByRangeGenres(
            analyzer, 'tempo', genre, lista_generos)
        printgenre(dicc)
        printfortotal(analyzer, ranges)
```

5.2 Funciones del Controller

```
def getRanges(lista_generos, genre):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getRanges(lista_generos, genre)
```

5.3 Funciones del Model

```
def getRanges(lista_generos, dicc):
    '''
    Retorna los rangos dados los generos
    '''
    llaves = []
    lim_inf = 1000
    lim_sup = 0

    for llave in dicc:
        llaves.append(llave[0])
    for i in lista_generos:
        for llave in dicc:
            if i in llave:
                lim = dicc[llave]
                if lim[0] <= lim_inf:
                    lim_inf = lim[0]
                if lim[1] >= lim_sup:
                    lim_sup = lim[1]

    ranges = []
    n = 0
    while n < lim_sup:
        ranges.append(0)
```

```
        n += 1

    for x in lista_generos:
        for llave in dicc:
            if x in llave:
                lim = dicc[llave]
                h = lim[0]
                while h < lim[1]:
                    ranges[h] = 1
                    h += 1

    ranges.append(0)
    resultados = []

    for pos in range(0, len(ranges)):
        if ranges[pos] == 1 and ranges[pos-1] == 0:
            inferior = pos
        elif ranges[pos] == 1 and ranges[pos+1] == 0:
            superior = pos + 1
            resultados.append((inferior, superior))

    return resultados
```

```
def getEventsByRange(analyzer, criteria, initial, final):
    '''
    Retorna los varias caracteristicas de los
    eventos dado un criterio y rango en el mismo,
    buscandolos en un  rbol
    Args:
        analyzer: Analizador de eventos
        criteria: Llave del analyzer a analizar
        initial: Inicio del rango
        final: Fin del rango
    '''
    lst = om.values(analyzer[criteria], initial, final)
    events = 0
    artists = mp.newMap(maptype='PROBING')
    tracks = mp.newMap(maptype='PROBING')

    for lstevents in lt.iterator(lst):
        events += lt.size(lstevents['events'])
        for soundtrackyourtimeline in lt.iterator(lstevents['events']):
            mp.put(artists, soundtrackyourtimeline['artist_id'], 1)
            mp.put(tracks, soundtrackyourtimeline['track_id'], 1)

    artists_size = mp.size(artists)
    tracks_size = mp.size(tracks)

    return events, artists_size, tracks_size, artists, tracks
```

6 Quinto Requerimiento

Dado un rango de horas (ej.: de 10:00 am a 10:30 am) 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 de análisis de sentimiento, en las canciones de dicho rango.Para dar respuesta a este requerimiento se recibe como entrada la siguiente información:

- El valor mínimo de la hora del día.
- El valor máximo de la hora del día.

Y como respuesta debe presentar en consola la siguiente información:

- Género más referenciado en el rango de horas.

- Para el género más referenciado, calcular el valor promedio VADER de las pistas que contiene. tome como base para del cálculo el campo vader_avg de las etiquetas (Hashtag).

6.1 Funciones del View

```
elif int(inputs[0]) == 7:
        print("Ingrese la hora de inicio en formato 24h: (Ej. 12:34:56)")
        tiempo_inicio = input("~")
        print("Ingrese la hora final en formato 24h: (Ej. 12:34:56)")
        tiempo_final = input("~")
        result = controller.getTemposByTime(
            analyzer, tiempo_inicio, tiempo_final)
        bestgenre = controller.getBestGenre(result, genre)
        printTopGenres(bestgenre[0])
        uniqueIDs = controller.getUniqueIDs(
            result, genre, bestgenre[1])
        result2 = controller.getSentimentAnalysis(uniqueIDs, analyzer)
        printtop10tracks(result2, bestgenre[1])
```

6.2 Funciones del Controller

```
def getTemposByTime(analyzer, tiempo_inicio, tiempo_final):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getTemposByTime(analyzer, tiempo_inicio, tiempo_final)
```

```
def getBestGenre(minimap, genredicc):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getBestGenre(minimap, genredicc)
```

```
def getUniqueIDs(minimap, generos, bestgenre):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getUniqueIDs(minimap, generos, bestgenre)
```

```
def getSentimentAnalysis(unique_ids, analyzer):
    '''
    Funcion puente entre las funciones homonimas entre el model y view
    '''
    return model.getSentimentAnalysis(unique_ids, analyzer)
```

6.3 Funciones del Model

```
def getTemposByTime(analyzer, tiempo_inicio, tiempo_final):
    '''
    Retorna los eventos dados los tiempos al usar la
    funcion getEventsByRangeTempoReturn()
    '''
    realstarttime = tiempo_inicio.split(':')
    realstarttime = (
        int(realstarttime[0])*3600 + int(realstarttime[1])*60
        + int(realstarttime[2]))
    realfinishtime = tiempo_final.split(':')
    realfinishtime = (
        int(realfinishtime[0])*3600 + int(realfinishtime[1])*60
        + int(realfinishtime[2]))
    return getEventsByRangeTempoReturn(
        analyzer, 'created_at', realstarttime, realfinishtime)
```

```
def getBestGenre(minimap, genredicc):
    '''
    Retorna un diccionario con el top de
    los generos dadas las repeticiones
    '''
    asqueroso_top = {}
    bestgenre = None
    mayor = 0
    for genre in genredicc:
        lim = genredicc[genre]
        events = getTotalEventsByRangeGenre(
            minimap, 'tempo_map', lim[0], lim[1])
        asqueroso_top[genre] = events
        if events > mayor:
            mayor = events
            bestgenre = genre

    return asqueroso_top, bestgenre
```

```
def getUniqueIDs(minimap, generos, bestgenre):
    '''
    Retorna los eventos unicos dada la llave concatenada
    '''
    lim = generos[bestgenre]
    lst = om.values(minimap['tempo_map'], lim[0], lim[1])
    tracks = {'data': None}
    tracks['data'] = mp.newMap(maptype='PROBING')
    events = 0

    for lstevents in lt.iterator(lst):
        events += lt.size(lstevents['events'])
        for soundtrackyourtimeline in lt.iterator(lstevents['events']):
            unique_id = (
                soundtrackyourtimeline['user_id']
                + soundtrackyourtimeline['track_id'])
            addEventOnProbingMap(
                tracks, soundtrackyourtimeline['track_id'], unique_id, 'data')

    tracks_size = mp.size(tracks['data'])

    return tracks['data'], tracks_size, events
```

```
def getSentimentAnalysis(unique_ids, analyzer):
    '''
    Retorna los eventos que tienen un hashtag
    determinado dada la llave concatenada
    '''
    hashtags = analyzer['hashtags']
    vaders = analyzer['vaders']
    llaves = mp.keySet(unique_ids[0])
    tracks = mp.newMap(maptype="PROBING")

    for llave in lt.iterator(llaves):
        ids = mp.get(unique_ids[0], llave)
        vaderavg = 0
        lista = me.getValue(ids)
        for each_id in lt.iterator(lista['events']):
            hashtag = mp.get(hashtags, each_id)
            hashtag_value = me.getValue(hashtag)
            vader = mp.get(vaders, hashtag_value)
            if (vader is not None):
                vader_val = me.getValue(vader)
                if (vader_val is not None) and (vader_val != ''):
                    vaderavg += float(vader_val)

        if vaderavg != 0.0:
            listaX = me.getValue(ids)
            n = lt.size(listaX['events'])
```

```
27              vaderavg = vaderavg/n
28              mp.put(tracks, llave, (vaderavg, n))
29
30      return tracks
```

7 Tiempos & Espacio de Requerimientos

Requerimiento	Tiempo (ms)	Espacio (kb)
1	374,765	3880,113
2	463,262	1403,305
3	158,279	25,93
4	3326,327	27043,884
5	233,394	760,262

Tabla 1: Análisis de Tiempo y Espacio por requermiento.

8 Cálculos de Ordenes de Crecimiento

En el siguiente apartado analizaremos los cálculos de los ordenes de crecimiento de cada una de las funciones. En primera instancia, es necesaria definir distintos "tamaños de datos" para cada archivo y filtración usada en los análisis, ya que sería negligente referirse a un solo tamaño "N" de datos. Por lo tanto, definiremos las siguientes variables para su mejor y peor caso:

- **CCF**: Indica el tamaño de entradas dentro del archivo .csv correspondiente a "Context Content Features".
- **SV**: Indica el tamaño de entradas dentro del archivo .csv correspondiente a "Sentiment Values".
- **UTH**: Indica el tamaño de entradas dentro del archivo .csv correspondiente a "User Track Hashtags".

Como los anterriores tienen un número constante de entradas, no tienen peor ni mejor caso. Por otro lado, definiremos un tamaño de datos de filtración por rango para cualquiera de las características músicales, y u subtamaño correspondiente a una segunda filtración que nos serviran para analizar los ordenes de crecimiento de los requerimientos:

- **R1**: Es el tamaño de promedio de los elementos de un rango dado de CCF con cualquiera de las distintas características músicales. Su peor caso es CCF ó $R1_W$ (cuando se indica el rango que contiene todas las entradas) y su mejor caso lo indicaremos con $R1_B$.
- **R2**: Es el tamaño promedio de un de los elementos de un rango dado de $R1$ con cualquiera de las distintas características músicales. Su peor caso es $R1$ (cuando se indica el rango que contiene todas las entradas) y su mejor caso lo indicaremos con $R2_B$.
- **R3**: Es el tamaño promedio de un de los elementos de un rango dado de $R2$ con cualquiera de las distintas características músicales. Su peor caso es $R2$ (cuando se indica el rango que contiene todas las entradas) y su mejor caso lo indicaremos con $R3_B$.
- **H**: Es la altura promedio de los rangos filtrados $R1$, que en su peor caso es igual a la altura de los árboles, señalada por H_W y en su mejor caso diremos que es igual a H_B.
- **H**: Es la altura promedio de los rangos filtrados $R2$, que en su peor caso es igual a la altura de los árboles, señalada por h_W y en su mejor caso diremos que es igual a h_B.
- **G**: Es el total de géneros presentes.

8.1 Analizador

Para la construcción del analizador fue necesario iterar los tres archivos para la construcción de distintas estructuras de datos necesarias para la resolución de los requerimientos. En primera instancia, con el archivo **"Context Content Features"** se construyó:

- **Array List 'listening_events'**: Lista que guardara cada entrada del archivo csv, y se demora CCF en construirse.
- **Mapa 'tracks'**: Un mapa llave "track id" y valor "id" y, asumiendo que no hay rehash, se demora CCF construirse.
- **Mapa 'artists'**: Un mapa llave "Artist id" y valor "id" y, asumiendo que no hay rehash, se demora CCF construirse.
- **RBT 'instrumentalness'**: Un árbol RBT de llave/nodo "instrumentalness" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 16.
- **RBT 'acousticness'**: Un árbol RBT de llave/nodo "acousticness" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 15.
- **RBT 'liveness'**: Un árbol RBT de llave/nodo "liveness" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 13.
- **RBT 'speechiness'**: Un árbol RBT de llave/nodo "speechiness" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 13.
- **RBT 'energy'**: Un árbol RBT de llave/nodo "energy" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 12.
- **RBT 'danceability'**: Un árbol RBT de llave/nodo "danceability" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 13.
- **RBT 'valence'**: Un árbol RBT de llave/nodo "valence" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 14.
- **RBT 'tempo'**: Un árbol RBT de llave/nodo "tempo" y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 19.
- **RBT 'created_at'**: Un árbol RBT de llave/nodo "created_at" igual al rango de segundos (para facilitar el cáclculo) y valor igual a la entrada correspondiente del csv, el cual se demora $CCF \cdot \log CCF$ en construirse y de altura 21.

De igual forma, con el archivo **"User Track Hashtags"** creamos un mapa correspondiente a:

- **Mapa 'hashtags'**: Un mapa llave un id único concatenando los elementos de "user id", "track id" y "created at"y valor "hashtag" y, asumiendo que no hay rehash, se demora UTH construirse.

Finalmente, con el archivo **"Sentiment Values"** creamos un mapa correspondiente a:

- **Mapa 'vader'**: Un mapa llave "hashtag" y valor "vader average" y, asumiendo que no hay rehash, se demora SV construirse.

Por lo tanto, la construcción del analizador tendrá una complejidad igual a la suma de las complejidades pasadas:

$$A = 3 \cdot CCF + 9 \cdot CCF \cdot \log CCF + UTH + SV$$

8.2 Primer Requerimiento

En este requerimiento es necesario hacer una busqueda de rangos haciendo uso de los árboles RBT creados en el apartado anterior. Para lo cual, debemos hacer uso de los tamaños de los rangos definidos. Luego, como el requerimiento 1 solo busca encontrar las entradas de un rango determinado su orden de crecimiento será igual a lo que se demore en llegar al rango, más el recorrido de las entradas. Por lo tanto su orden de crecimiento será igual a H más $R1$.

8.3 Segundo Requerimiento

Similar al caso anterior, en este caso es necesario obtener las entradas de un rango particulado. Posterior a esto, usamos esas entradas para la construcción de un nuevo árbol y obtener las entradas de un nuevo rango.

- El costo de obtener las entradas de la primera filtración es H más $R1$.
- La construcción del nuevo árbol de tamaño $R1$ será $R1 \cdot \log(R1)$
- El costo de obtener las segundas entradas del arbol recién creado es h más $R2$.

El orden de crecimiento total corresponde a la suma de estos tres:

$$Req2 = (H + R1) + (R1 \cdot \log(R1)) + (h + R2)$$

8.4 Tercer Requerimiento

Análogamente, en este caso es necesario obtener las entradas de un rango particulado. Posterior a esto, usamos esas entradas para la construcción de un nuevo árbol y obtener las entradas de un nuevo rango.

- El costo de obtener las entradas de la primera filtración es H más $R1$.
- La construcción del nuevo árbol de tamaño $R1$ será $R1 \cdot \log(R1)$
- El costo de obtener las segundas entradas del arbol recién creado es h más $R2$.

El orden de crecimiento total corresponde a la suma de estos tres:

$$Req3 = (H + R1) + (R1 \cdot \log(R1)) + (h + R2)$$

8.5 Cuarto Requerimiento

En el cuarto requerimiento es necesario usar el árbol de Tempo creado en el analyzer y buscar las entradas de un rango específico, cada rango está dado por el género y recordemos que G es el número de géneros totales, ya que se pueden agregar nuevos géneros al diccionario. Por lo tanto, el órden de crecimiento será G veces de H más $R1$.

8.6 Quinto Requermiento

Finalmente, para este requerimiento fue necesario seguir los siguientes pasos:

1. Haciendo uso del árbol de created_at , obtenemos las pistas dentro del rango de horas, lo que tiene un órden H más $R1$.
2. Posterior a esta filtración inicial construimos un nuevo árbol de tempos para encontrar el género con más reproducciones, lo que se demora $R1 \cdot \log(R1)$
3. Para los G géneros hacemos el recorrido por el árbol recién construido para obtener el género más escuchado, lo que tiene un órden de crecimiento igual a G veces de h más $R2$

4. Ahora bien, para hallar los vaders de los hashtags correspondientes a los tracks del género más escuchados es necesario iterar los valores del árbol que tiene un tamaño de $R3$. Con lo que crearemos un mapa de llaves track_id y valor una lista de los unique_id con los que creamos los mapas de hashtags, esto tiene una duración de $R3$ siempre y cuando no se haga rehash.

5. Una vez obtenido el mapa, es necesario iterar ese nuevo mapa para obtener por cada hashtag su vader correspondiente. Esto último, tiene un órden de crecimiento de $R3$ siempre y cuando no se haga rehash.

Recordemos que gracias a la creación de los mapas de vaders y hashtags creados al principio es posible obtener cada valor correspondiente en un promedio de $O(1.5)$ por lo que los pasos anteriores son la forma más eficiente de resolver el problema de los vaders. Por lo tanto, el órden de crecimiento será igual a la suma de cada uno de los pasos:

$$Req5 = (H + R1) + (R1 \cdot \log(R1)) + G(h + R2) + 2 \cdot R3$$

9 Resultados

	Analizador	**Requerimiento 1**
Notacion Tilda	$\sim N(3 \cdot CCF + 9 \cdot CCF \cdot \log CCF + UTH + SV)$	$\sim N(H + R1)$
Big Theta	$\Theta(CCF \cdot \log CCF)$	$\Theta(R1)$
Big O	$O(CCF \cdot \log CCF))$	$O(R1_W)$
Big Omega	$\Omega(CCF \cdot \log CCF))$	$\Omega(R1_B)$
	Requerimiento 2	**Requerimiento 3**
Notacion Tilda	$\sim N((H + R1) + (R1 \cdot \log(R1)) + (h + R2))$	$\sim N((H + R1) + (R1 \cdot \log(R1)) + (h + R2))$
Big Theta	$\Theta(R1 \cdot \log(R1))$	$\Theta(R1 \cdot \log(R1))$
Big O	$O(R1_W \cdot \log(R1_W))$	$O(R1_W \cdot \log(R1_W))$
Big Omega	$\Omega(R1_B \cdot \log(R1_B))$	$\Omega(R1_B \cdot \log(R1_B))$
	Requerimiento 4	**Requerimiento 5**
Notacion Tilda	$\sim N(G(H + R1))$	
Big Theta	$\Theta(R1))$	
Big O	$O(R1_W))$	
Big Omega	$\Omega(R1_B)$	