**OBSERVACIONES DEL LA PRACTICA**

Lindsay Vanessa Pinto Morato Cod 202023138

**CONTENIDO**

[EQUIPO DE PRUEBAS 1](#_Toc71306532)

[CONTROLES EN LAS PRUEBAS Y RESULTADOS 2](#_Toc71306533)

[COMPLEJIDAD 2](#_Toc71306534)

[ANÁLISIS DE CONSUMO DE MEMORIA Y TIEMPO 5](#_Toc71306535)

[CONCLUSIONES 8](#_Toc71306536)

**LISTA DE TABLAS**

[Table 1. Especificaciones computador 1](#_Toc69325431)

[Table 2. Datos de tiempo por requerimiento discriminados según el reto 2](#_Toc69325432)

[Table 3. Datos de uso de memoria por requerimiento discriminados según el reto 2](#_Toc69325433)

**LISTA DE ILUSTRACIONES**

[Figure 2. Comparación de rendimiento de tiempo entre los requerimientos de los dos retos 3](#_Toc69325326)

[Figure 3. Comparación de rendimiento de memoria entre los requerimientos de los dos retos 4](#_Toc69325327)

# **EQUIPO DE PRUEBAS**

Las pruebas de rendimiento se realizaron sobre el siguiente equipo:

* Computador Huawei Matebook D14

Table 1. Especificaciones computador

|  |  |
| --- | --- |
|  | Máquina |
| Procesadores | AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10GHZ |
| Memoria RAM (GB) | 8,00GB (6.94 utilizable) |
| Sistema Operativo | Windows 10 Home Single languaje |
| Tipo de sistema | Sistema operativo de 64bits, procesador x64 |

# **CONTROLES EN LAS PRUEBAS Y RESULTADOS**

## **COMPLEJIDAD**

|  |  |
| --- | --- |
| **Requerimiento** | **O(n)** |
| Caracterizar las reproducciones |  |
| Encontrar música para festejar |  |
| Estudiar los géneros musicales |  |

A continuación se muestra el análisis detallado de estas complejidades

**Caracterizar las reproducciones:**

|  |  |
| --- | --- |
| def Reproductions(analyzer, characteristic, minval, maxval):  totevents = 0  artists = 0  for key in lt.iterator(om.keySet(analyzer['artistIndex'])):  charMap = me.getValue(om.get(analyzer['artistIndex'],key))  lst = om.values(charMap[characteristic], minval, maxval)    eventsArtist = 0  for lstevent in lt.iterator(lst):  eventsArtist += lt.size(lstevent['lstevents'])  totevents += eventsArtist  if eventsArtist != 0:  artists += 1    return [totevents,artists] | O(1)  O(1)  O(n)  O(1)  O(log(n)) BST  O(1)  O(n)  O(1)  O(1)  O(1)  O(1) |
| Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades  Resolviendo:  Devolvemos C = 1  Entonces:  Por complejidad asintótica se tiene que la complejidad del algorítmo es de | |

**Encontrar música para festejar**

|  |  |
| --- | --- |
| def getPartyMusic (analyzer, energyMin, energyMax, danceMin, danceMax):    tracks = lt.newList('SINGLE\_LINKED', compareTracks)  lstEnergy = om.values(analyzer["energy"], energyMin, energyMax)  lstDance = om.values(analyzer["danceability"], danceMin, danceMax)  lstResult = lt.newList('SINGLE\_LINKED', compareTracks)  for energyValue in lt.iterator(lstEnergy):  trackIndex1 = energyValue["trackIndex"]  for track in lt.iterator(mp.keySet(trackIndex1)):  for danceValue in lt.iterator(lstDance):  trackIndex2 = danceValue["trackIndex"]  if mp.contains(trackIndex2, track):  lt.addLast(lstResult, me.getValue(mp.get(trackIndex1, track)))  break  tottracks = lt.size(lstResult)    for i in range(5):  pos = random.randint(1,lt.size(lstResult))  lt.addLast(tracks,lt.getElement(lstResult, pos))  lt.deleteElement(lstResult, pos)  return [tottracks, tracks] | O(1)  O(log (n))  O(log (n))  O(1)  O(n)  O(1)  O(n)  O(n)  O(1)  O(1)  O(1)  O(1)  O(n)  O(1)  O(1)  O(1) |
| Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades  Resolviendo:  Devolvemos C = 1  Entonces:  Por complejidad asintótica se tiene que la complejidad del algorítmo es de | |

**Estudiar los géneros musicales**

**(studyGenres)**

|  |  |
| --- | --- |
| def studyGenres(analyzer, genres):  result = []  for genreName in genres:  genreEntry = mp.get(analyzer['genreIndex'], genreName)  genre = me.getValue(genreEntry)  artists = []  totevents = 0  totartists = 0  for key in lt.iterator(om.keySet(analyzer['artistIndex'])):  charMap = me.getValue(om.get(analyzer['artistIndex'],key))  lst = om.values(charMap['tempo'], genre['min'], genre['max'])    eventsArtist = 0  for lstevent in lt.iterator(lst):  eventsArtist += lt.size(lstevent['lstevents'])  totevents += eventsArtist  if eventsArtist != 0:  totartists += 1  for event in lt.iterator(analyzer['lstevents']):  if float(event['tempo']) >= genre['min'] and float(event['tempo']) <= genre['max'] and event['artist\_id'] not in artists:  artists.append(event['artist\_id'])  if len(artists) >= 10:  break  result.append({'genre':genreName, 'count': totevents, 'min':genre['min'], 'max':genre['max'], 'unique\_artists':totartists,'artists':artists})  return result | O(1)  O(n)  O(n) \*  O(1)  O(1)  O(1)  O(1)  O(n)  O(1)  O(1)  O(1)  O(n)  O(1)  O(1)  O(1)  O(1)  O(n)  O(1)  O(1)  O(1)    O(1) |
| \*revisando todas las funciones llamadas se llega a la función isPresent que presenta un ciclo for en su implementación (además de otras funciones) dándo comportamiento asintótica o(n) | |
| Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades  Resolviendo:  Devolvemos C = 1  Entonces:  Por complejidad asintótica se tiene que la complejidad del algorítmo para esta parte del requerimiento es de | |

**Estudiar los géneros musicales**

**(newGenre)**

|  |  |
| --- | --- |
| def newGenre(analyzer, genreName, tempoMin, tempoMax):  genreIndex = analyzer['genreIndex']  if not mp.contains(genreIndex, genreName):  genre = {'name': genreName, 'min': tempoMin, 'max': tempoMax}  updateGenreIndex(genreIndex, genre)  return True  return False | O(1)  O(n)\*  O(1)  O(n)\* |
| \*revisando todas las funciones llamadas se llega a la función isPresent que presenta un ciclo for en su implementación (además de otras funciones) dándo comportamiento asintótica o(n) | |
| Llamaremos a O(1) = C y se realizará suma de cada una de las complejidades  Resolviendo:  Devolvemos C = 1  Entonces:  Por complejidad asintótica se tiene que la complejidad del algorítmo para esta parte del requerimiento es de | |

Esto quiere decir que la complejidad del requerimiento en general es y por complejidad asintótica se traduce en comportamiento

## **ANÁLISIS DE CONSUMO DE MEMORIA Y TIEMPO**

Las diferentes pruebas fueron realizadas utilizando el mismo archivo (videos small) y mismos criterios de búsqueda.

Por ejemplo, para el requerimiento 1, correspondiente a “**Caracterizar las reproducciones**” se utilizó como característica de contenido “instrumentalness” con un valor mínimo y máximo de la característica de contenido 0.75 y 1 respectivamente. Para el requerimiento 2 correspondiente a “**Encontrar música para festejar**” se utilizaron valores mínimos y máximos de energy de 0.50 y 0.75 y de igual manera para danceability se usaron como valores mínimos y máximos 0.75 y 1 respectivamente. Por último, para “**Estudiar los géneros musicales”** se usaron como entrada los géneros reggae, hip-hop y pop.

Se realizaron tres pruebas de cada uno y se registró el promedio obteniendo los siguientes resultados.

Table 2. Datos de tiempo por requerimiento

|  |  |
| --- | --- |
| **Requerimiento** | **Tiempo [ms]** |
| Caracterizar las reproducciones | 367,193 |
| Encontrar música para festejar | 26031,753 |
| Estudiar los géneros musicales | 1278,902 |

Table 3. Datos de uso de memoria por requerimiento

|  |  |
| --- | --- |
| **Requerimiento** | **Memoria [kB]** |
| Caracterizar las reproducciones | 9,321 |
| Encontrar música para festejar | 12,871 |
| Estudiar los géneros musicales | 12,462 |

Se aclara que no se tuvo en cuenta los datos de tiempo y memoria en la carga pues al implementar estas funciones se excedía el tiempo de respuesta del algorítmo y además, el espacio de memoria se agotaba.

Para un mejor análisis, se registran los datos en gráficas a continuación, con su respectivo análisis:

Figure 2. Comparación de rendimiento de tiempo entre los requerimientos.

Para estos datos se puede observar que el requerimiento 2 es el que más tiempo de ejecución toma lo cual se debe a la cantidad de ciclos utilizados para el recorrido y búsqueda de la información además de los llamados realizados a otras funciones tanto de las librerías de DISClib como del código. En términos generales, el tiempo de ejecución es aceptable, pues el requerimiento que más tiempo toma es el número 2 siendo aun de 26 segundos aproximadamente.

Figure 3. Comparación de rendimiento de memoria entre los requerimientos de los dos retos

Los tiempos de memoria para todos los requerimientos fueron similares, no obstante, se nota diferencia en el requerimiento 1 pues es menor aunque no de forma significativa.

# **CONCLUSIONES**

Dependiendo de la construcción que se le de a los árboles, la carga puede ser un poco más costosa en el tiempo pero la mejoría se ve en cada uno de los procesos siguientes los cuales con otras estructuras como listas encadenadas o arreglos podría tomar mucho más tiempo de encontrar