LABORATORIO No. 7: Mecanismos de Colisión

Objetivos

Comprender la implementación del Tipo Abstracto de Datos Tabla de Símbolos no ordenadas (Map) y su uso para la solución de problemas.

Al finalizar este laboratorio el estudiante estará en capacidad de:

- Entender a usar las tablas de hash como estructura de datos y su impacto en los órdenes de crecimiento temporal.
- Familiarizarse con las estrategias de manejo de colisiones en tablas de hash
- Comparar los tiempos de respuesta de las tablas de hash cuando su factor de carga cambia

Fecha Límite de Entrega

Miércoles **31 de marzo** antes de la media noche (11:59 p.m.).

Preparación del Laboratorio

- Revisar el API del TAD Map ubicado en **`DISClib\ADT\map.py`**
- Revisar las estructuras de datos **`chaininghashtable.py`** y **`probinghashtable.py`** ubicado en **`DISClib\DataStructures`**
- Instalar las extensiones de VS Code para medir el consumo de recursos y explorar marcas ("TODO") dentro del código.
    - **Todo Tree,** URL: Gruntfuggly.todo-tree
    - **Resource Monitor,** URL: mutantdino.resourcemonitor

Trabajo Propuesto

PASO 1: Copiar el ejemplo en su organización

Copie/Haga **Fork** del repositorio del laboratorio en su organización con el procedimiento aprendido en las prácticas anteriores.

El repositorio del proyecto base que utiliza este laboratorio es el siguiente:

- https://github.com/ISIS1225DEVS/ISIS1225-SampleCollision.git

Antes de clonar el repositorio en su computador diríjase a su organización (Ej.: ***EDA2021-1-SEC02-G01*** para el primer grupo de la sección 2 del curso) y cambie el nombre del repositorio de acuerdo con el

esquema **LabCollision-S<<XX>>-G<<YY>>** donde **XX** es el número de la semana de la práctica y donde **YY** es el número del grupo de trabajo. (Ej.: **LabCollision-S07-G01** para este **séptimo laboratorio** hecho por el **grupo 1** de la **sección 2).**

Recuerde que **NO necesita** agregar la sección o el semestre en este nombre porque ya está identificado en su organización.

PASO 2: Descargar el ejemplo

Después de renombrar el proyecto dentro de su organización ya puede clonar el proyecto. Descargue el código en su máquina local **(`git clone`)** siguiendo lo aprendido en las practicas anteriores.

Recuerde modificar el **`README`** del repositorio para incluir los nombres de los integrantes del grupo.

PASO 3: Ejecutar y explorar el ejemplo

El proyecto **`SampleCollision`** busca ayudarnos a medir el comportamiento de las Tablas de Símbolos no ordenadas (Map), específicamente el tiempo de ejecución y la cantidad de memoria consumida.

Antes de iniciar a explorar y modificar el ejemplo, recuerde descargar los datos de trabajo **`Goodreads`** disponibles en el portal oficial del curso en BrightSpace. Descargue el **Zip,** descomprímalo y guarde los archivos CSV en la carpeta **`*/Data/GoodReads/`** de su copia local de código.

Diríjase al archivo **`view.py`** y ejecútelo, y seleccione secuencialmente la **opción 1** y **2** para iniciar el catálogo y cargar información respectivamente. Al finalizar estos pasos la consola mostrará estadísticas del número de registros cargados como se muestra a continuación.

```
Seleccione una opción para continuar
2
Cargando información de los archivos ....
Libros cargados: 10000
Autores cargados: 5841
Géneros cargados: 34252
Tiempo [ms]:  52191.528   ||   Memoria [kB]:  311420.532
Bienvenido
1- Inicializar Catálogo
2- Cargar información en el catálogo
3- Consultar los libros de un año
4- Consultar los libros de un autor
5- Consultar los Libros por etiqueta
6- Ordenar mejores libros de un año
0- Salir
Seleccione una opción para continuar
```

En la anterior imagen observamos la memoria y el tiempo utilizado para completar la carga del catálogo. En este caso el tiempo de ejecución fue de aproximadamente **52191.53 ms** y la memoria utilizada de **311420.53 kB**.

Ahora ejecute la **opción 6**, con el año **2011**, con una fracción de **0.5** de libros en el año y **siete (7)** como numero de libros dentro del escalafón. Como resultado aparecerán los siguientes libros junto su correspondiente información de tiempo de ejecución y cantidad de memoria utilizada.

```
Bienvenido
1- Inicializar Catálogo
2- Cargar información en el catálogo
3- Consultar los libros de un año
4- Consultar los libros de un autor
5- Consultar los Libros por etiqueta
6- Ordenar mejores libros de un año
0- Salir
Seleccione una opción para continuar
6
Buscando libros del año?: 2011
Fraccion de libros en el año? (entre 0.0 y 1.0): 0.5
Cuantos libros en el escalafon? (mayor a 0): 7
Total de libros en 2011: 556
Muestra de libros: 278
 Estos son los mejores libros:
Titulo: A Song of Ice and Fire (A Song of Ice and Fire, #1-4)  ISBN: 345529057 Rating: 4.63
Titulo: The Wise Man's Fear (The Kingkiller Chronicle, #2)  ISBN: 756404738 Rating: 4.57
Titulo: Harry Potter Page to Screen: The Complete Filmmaking Journey  ISBN: 62101897 Rating: 4.56
Titulo: A Game of Thrones: The Graphic Novel, Vol. 1  ISBN: 044042321X Rating: 4.48
Titulo: Blood Song (Raven's Shadow, #1)  ISBN:  Rating: 4.47
Titulo: Clockwork Prince (The Infernal Devices, #2)  ISBN: 1416975888 Rating: 4.46
Titulo: Shadowfever (Fever, #5)  ISBN: 385341679 Rating: 4.46


Time [ms]:  77.1170 Memory [Byte]:  28047.00
```

Note que el tiempo de ejecución fue de aproximadamente **77.12 ms** y la cantidad de memoria utilizada fue de **28047 kB**.

Para observar detalladamente cómo se implementaron estas opciones diríjase al **model.py** y en la función **sortBooksByYear()** encontrara el código documentado de este ordenamiento que integra el uso de los **TAD List** y **TAD Map** del curso.

Ahora, para entender como medir el tiempo de ejecución y la memoria consumida de las funcionalidades diríjase al **controller.py**, allí encontrará 3 funciones básicas para esta tarea, cuyos nombres son: **getTime()**, **getMemory()** y **deltaMemory().** Para entenderlas mejor a continuación se realizará la descripción de cada una de estas.

La función **getTime()** toma el tiempo del reloj del procesador utilizando la librería **"time"** y devuelve su resultado en milisegundos [ms] como flotante.

```python
def getTime():
    """
    devuelve el instante tiempo de procesamiento en milisegundos
    """
    return float(time.perf_counter()*1000)
```

por su parte la función **getMemory()** utiliza la librería nativa **"tracemalloc"** de Python para tomar una imagen de la memoria asignada por el programa en un instante de tiempo.

```python
def getMemory():
    """
    toma una muestra de la memoria alocada en instante de tiempo
    """
    return tracemalloc.take_snapshot()
```

por último, **deltaMemory()** utiliza **"tracemalloc"**, toma dos muestras de memoria (resultados de invocar **getMemory()**) y calcula la memoria asignada por Python sumando las diferencias entre las dos muestras.

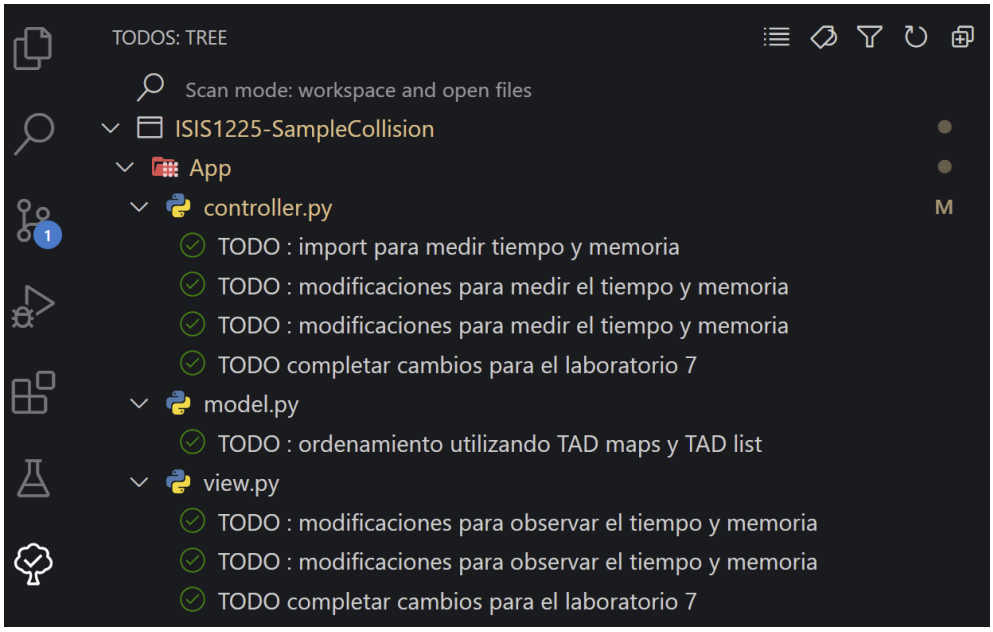
```
def deltaMemory(start_memory, stop_memory):
    """
    calcula la diferencia en memoria alocada del programa entre dos
    instantes de tiempo y devuelve el resultado en kBytes (ej.: 2100.0 kB)
    """
    memory_diff = stop_memory.compare_to(start_memory, "filename")
    delta_memory = 0.0

    # suma de las diferencias en uso de memoria
    for stat in memory_diff:
        delta_memory = delta_memory + stat.size_diff
    # de Byte -> kByte
    delta_memory = delta_memory/1024.0
    return delta_memory
```

Estas cuatro funciones (**getTime()**, **deltaTime()**, **getMemory()** y **deltaMemory()**) se deben incluir en la función **sortBooksByYear()** en el controlador para calcular el tiempo de ejecución y la cantidad de memoria utilizada.

Dentro de la implementación de **sortBooksByYear()** es importante recordar que la librería de **"tracemalloc"** debe iniciar el proceso de toma de datos antes de ejecutar la función del **model.py** con el comando **start()** y debe finalizar subsecuentemente su proceso con **stop()**.

PASO 4: Entendiendo las modificaciones

Para entender bien como se ejecutan las mediciones en las **opciones 2 y 3** e integrar los cambios a un código funcional exploraremos el proyecto con la extensión **Todo Tree** de **VS Code** para localizar las notas ***"TODO"*** dentro de la implementación como se ve a continuación.



Primero diríjase al **controller.py** y examine los **"TODO"** de los **cuatro (4)** primeros cambios.

a) Aquí se incluyen las importaciones de las librerías **"time"** y **"tracemalloc"**.

```
# TODO: import para medir tiempo y memoria
import time
import tracemalloc
import config as cf
import model
import csv
```

b) La carga de datos en el catálogo en **loadData()** incluye las modificaciones para medir tiempo de ejecución y memoria utilizada se muestra a continuación.

```
def loadData(ctrlr):
    """
    Carga los datos de los archivos y cargar los datos en la
    estructura de datos
    """
    # TODO: modificaciones para medir el tiempo y memoria
    # inicializa el proceso para medir memoria
    tracemalloc.start()

    # toma de tiempo y memoria al inicio del proceso
    start_time = getTime()
    start_memory = getMemory()

    loadBooks(ctrlr)
    loadTags(ctrlr)
    loadBooksTags(ctrlr)

    # toma de tiempo y memoria al final del proceso
    stop_memory = getMemory()
    stop_time = getTime()
    # finaliza el proceso para medir memoria
    tracemalloc.stop()

    # calculando la diferencia de tiempo y memoria
    delta_time = deltaTime(stop_time, start_time)
    delta_memory = deltaMemory(stop_memory, start_memory)

    return delta_time, delta_memory
```

Recuerde que la función **getBooksYear()** también incluye estas modificaciones para medir tiempo de ejecución y memoria utilizada.

Por último, vaya al archivo **view.py** y examine las **tres (3)** últimas modificaciones.

a) Por ejemplo, el bloque de código para la **opción 2** permite imprimir en consola los resultados de la carga del catálogo, el tiempo de ejecución y la memoria utilizada como se ve a continuación.

```
    elif int(inputs[0]) == 2:
        # TODO: modificaciones para observar el tiempo y memoria
        print("Cargando información de los archivos ....")
        answer = controller.loadData(ctrlr)
        print('Libros cargados: ' + str(controller.booksSize(ctrlr)))
        print('Autores cargados: ' + str(controller.authorsSize(ctrlr)))
        print('Géneros cargados: ' + str(controller.tagsSize(ctrlr)))
        print("Tiempo [ms]: ", f"{answer[0]:.3f}", "||",
              "Memoria [kB]: ", f"{answer[1]:.3f}")
```

Recuerde que el código para la **opción 3** también tiene las modificaciones necesarias para imprimir el resultado de la función, más el tiempo de ejecución y la memoria trabajo.

Con esto, todas las funciones en el **`view.py`** despliegan el mismo tipo de información al completar su ejecución. Los resultados del tiempo de ejecución y la memoria utilizada se aproximan a la tercera cifra decimal para obtener datos con una precisión adecuada como se observa a continuación al ejecuta la **opción 3.**

```
Proper Gauge (Wool, #2)
The Virgin Cure
The Tycoon's Revenge (Baby for the Billionaire, #1)
Cinderella Ate My Daughter: Dispatches from the Frontlines of the New Girlie-Girl Culture


Tiempo [ms]:  0.386   ||   Memoria [kB]:  0.594
Bienvenido
1- Inicializar Catálogo
2- Cargar información en el catálogo
3- Consultar los libros de un año
4- Consultar los libros de un autor
5- Consultar los Libros por etiqueta
6- Ordenar mejores libros de un año
0- Salir
Seleccione una opción para continuar
```

Para finalizar esta etapa de la practica considere, responda las preguntas y registre su respuesta en el documento de observaciones:

a) ¿Por qué en la función **`getTime()`** se utiliza **`time.perf_counter()`** en vez de otras funciones como **`time.process_time()`**?
b) ¿Por qué son importantes las funciones **`start()`** y **`stop()`** de la librería **`tracemalloc`**?

PASO 6: Aplicando las modificaciones

Ahora, aplique las modificaciones estudiadas sobre la **opción 6.** Para ello utilice la extensión **Todo Tree** de **VS Code** y localice las notas ***"TODO"*** con el mensaje *"completar cambios para el laboratorio 7".*

Primero diríjase al **`controller.py`** den la función **`sortBooksByYear()`** y complete el código de la siguiente manera:

```
def sortBooksByYear(ctrlr, year, rank):
    """
    Retorna los libros que fueron publicados
    en un año ordenados por rating
    """
    # TODO completar cambios para el laboratorio 7
    # respuesta por defecto
    books = None
    # inicializa el proceso para medir memoria
    tracemalloc.start()

    # toma de tiempo y memoria al inicio del proceso
    start_time = getTime()
    start_memory = getMemory()
    books = model.sortBooksByYear(ctrlr['model'], year, rank)

    # toma de tiempo y memoria al final del proceso
    stop_memory = getMemory()
    stop_time = getTime()

    # finaliza el procesos para medir memoria
    tracemalloc.stop()

    # calculando la diferencia de tiempo y memoria
    delta_time = deltaTime(stop_time, start_time)
    delta_memory = deltaMemory(stop_memory, start_memory)
    return books, delta_time, delta_memory
```

Y segundo, modifique la marca *"TODO"* en el **model.py** como se indica a continuación para hacer que la **opción 6** del programa devuelva los valores de tiempo y memoria adecuadamente.

```
    elif int(inputs[0]) == 6:
        number = input("Buscando libros del año?: ")
        fraction = input("Fraccion de libros en el año? (entre 0.0 y 1.0): ")
        rank = input("Cuantos libros en el escalafon? (mayor a 0): ")
        number = int(number)
        fraction = float(fraction)
        rank = int(rank)
        answer = controller.sortBooksByYear(ctrlr, number, fraction, rank)
        # TODO completar cambios para el laboratorio 7
        printBestBooks(answer[0])
        print("Tiempo [ms]: ", f"{answer[1]:.3f}", "||",
              "Memoria [kB]: ", f"{answer[2]:.3f}")
```

Al finalizar estos cambios y reiniciar la aplicación la **opción 6** del menú debe ejecutar sin problemas.

Finalmente, asegúrese de hacer **commit** y **push** de los cambios en el repositorio con el comentario *"modificaciones de tiempos y memorias – Laboratorio 7"*.

Factores de Carga

Ahora que se entiende el uso del TAD Map, los factores de carga y cómo podemos medir la memoria que utilizan las tablas de hash en la carga de datos podemos. Aplicaremos todo esto en la implementación del reto.

PASO 1: Copiar la plantilla del Reto No. 2 en su organización

Copie/Haga **Fork** del repositorio del reto en su organización con el procedimiento aprendido previamente. El repositorio del proyecto base que utiliza este módulo es:

https://github.com/ISIS1225DEVS/Reto2-Template

Antes de clonar el repositorio diríjase a su organización (Ej.: ***EDA2021-1-SEC02-G01*** para el primer grupo de la sección 2 del curso) y cambie el nombre del repositorio según el esquema **Reto2-G<<XX>>** donde **XX** es el número del grupo de trabajo. (Ej.: **Reto2-G01** para el **grupo 1** de la **sección 2).**

Recuerde que el repositorio **debe ser privado** y **NO necesita** agregar la sección o el semestre en este nombre.

PASO 2: Descargue el Reto No. 2

Después de renombrar el proyecto dentro de su organización ya puede clonar el proyecto. Descargue el código en su máquina local siguiendo lo aprendido en las practicas anteriores.

Recuerde modificar el **README** principal del repositorio con los nombres de los integrantes del grupo e identificar claramente cual miembro implementara cual requerimiento individuo. Por ejemplo:

- *Req. 2 - Santiago Arteaga, 200411086, sa-artea@uniandes.edu.co*
- *Req. 3 - Carlos Lozano, 200211089, calozanog@uniandes.edu.co*

PASO 3: Aplicar Maps en el Reto No. 2

Tome como base el código implementado para resolver el Reto 1 (archivos **`view.py`, `controller.py`,** y **`model.py`**) y utilícelo para iniciar a implementar Reto 2.

Específicamente en el archivo **`model.py`** del reto 2 realice las siguientes modificaciones:

1) Agregue las importaciones apropiadas para utilizar ADT Map en el **`model.py`** de su reto.
2) Modifique su catálogo para que al cargar los artistas se cree un índice por géneros musicales ("genres") utilizando la librería **Maps.py** de **DISClib.**

PASO 4: Actualizar el repositorio en la rama principal

Confirme los cambios con los comandos **`Commit`** y **`Push`** en la rama ***main*** local y de GitHub con el comentario *"Primer avance – Reto 2"* antes de la fecha límite de entrega.

PASO 5: Preparar las pruebas

Antes de iniciar las pruebas sobre la implementación de su Reto No. 2 recomendamos tener en cuenta las siguientes instrucciones:

1) Cada estudiante debe ejecutar las pruebas propuestas en sus máquinas.

2) Para la toma de datos se debe utilizar el archivo de mayor tamaño que los computadores de ambos integrantes del equipo puedan procesar. Por decir, si los dos integrantes pueden cargar el archivo **large** deben usar este tamaño para la toma de datos. mientras que si el computador de un estudiante solo puede cargar el **50pct** ambos deberán utilizar este subconjunto.
3) Diligenciar la Tabla 1 en el documento **`Observaciones-lab7.docx`** con la información de las máquinas de cómputo donde cada estudiante ejecutará las pruebas de carga de datos.

	Máquina 1	Máquina 2
Procesadores		
Memoria RAM (GB)		
Sistema Operativo		

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de carga de datos.

Por ejemplo:

	Máquina 1	Máquina 2
Procesadores	Intel(R) Core (TM) i7-55000 CPU @2.40GHz 2.40GHZ	Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
Memoria RAM (GB)	16.0 GBGB	32.0 GB
Sistema Operativo	Windows 10 Pro 64-bits	Windows 10 Pro 64-bits

4) Ejecute todas las pruebas de rendimiento propuestas en la misma máquina que reporto.
5) Cierre todas las aplicaciones que puedan afectar la toma de datos y sean **innecesarias** de su máquina, ej.: **Discord**, **OneDrive**, **Dropbox**, **Word** y en especial el navegador de internet (**Edge**, **Chrome**, **Firefox**, **Safari, ...**).
6) Se recomienda que cada prueba se ejecute por lo menos tres veces para poder obtener un promedio o consolidado consistente de las pruebas. Ej. Para un experimento para cargar el catálogo con factor de carga 0.5 se debe ejecutar por lo menos 3 veces (ideal 5 veces).
7) evitando errores dentro de la máquina como lo son actualizaciones o escaneos de seguridad, para registrar resultados adecuados en las tablas provistas.
8) Cada uno de los tiempos debe estas registrado en **milisegundos (ms)** y con 2 cifras decimales redondeado hacia arriba desde la mitad. Ej.: **43494.498587 ms** se aproxima a **43494.50 ms.**

PASO 6: Medir tiempo y memoria en el Reto No. 2

En esta sección evaluaremos el efecto en el uso de la memoria y tiempo de ejecución para **TAD Map** en la implementación del reto.

Para ello, integre al código del Reto No. 2 los cambios estudiados en la primera parte de la práctica. Inclúyalos para medir el tiempo de ejecución y la memoria utilizada en la carga del catálogo de videos, recuerde que en la práctica anterior creo un índice por categorías con **TAD Map.**

Para ello siga las siguientes indicaciones:

1) Incluir las importaciones para medir tiempo y memoria en el **`controller.py.`**
2) Integrar las funciones **`getTime()`, `deltaTime()`, `getMemory()`** y **`deltaMemory()`** en el **`controller.py`**
3) Modificar la su función de carga del catálogo Spotify para recuperar el tiempo de ejecución y la memoria utilizada en el proceso.

Al completar las modificaciones del Reto No. 2, diligencie la `Tabla 2` y la `Tabla 3` variando el factor de carga y los mecanismos de colisiones entre **"PROBING"** y **"CHAINING"** para la Tabla de Hash en el índice de géneros previamente implementado (e.j.: *`maptype='PROBING'`* y *`maptype='CHAINING'`*).

Carga de Catálogo PROBING

Factor de Carga (PROBING)	Consumo de Datos [kB]	Tiempo de Ejecución [ms]
0.10		
0.50		
0.90		

Tabla 2. Mediciones de tiempo y datos para diferentes factores de carga en PROBING.

Carga de Catálogo CHAINING

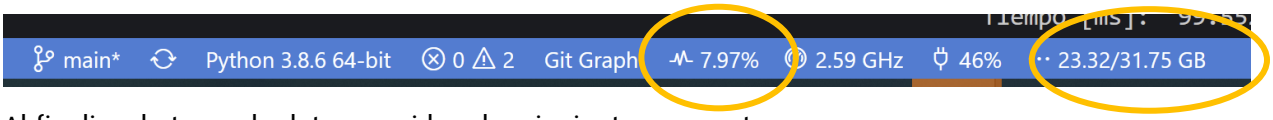
Factor de Carga (CHAINING)	Consumo de Datos [kB]	Tiempo de Ejecución [ms]
2.00		
4.00		
8.00		

Tabla 3. Mediciones de tiempo y datos para diferentes factores de carga en CHAINING.

Durante la ejecución de estas pruebas recomendamos escoger el archivo de datos más apropiado a las capacidades de sus máquinas de trabajo. Y utilizar el **Resource Monitor** de **VS Code** para monitorear la memoria disponible de su computador.

Recuerde que las pruebas tienen un **máximo de duración de 10.0 minutos,** en caso de que esto suceda aborte el procesamiento y reduzca la cantidad de datos que su aplicación planea manejar.

Aborte la prueba en caso de que la memoria se acerque a su capacidad máxima o en caso de que su procesador este sobre cargado por mucho tiempo. Ambos datos se pueden ver en la barra inferior de información del IDE como se muestra a continuación.



Al finalizar la toma de datos considere las siguientes preguntas:

a) Teniendo en cuenta cada uno de los requerimientos del reto ¿Cuántos índices implementaría en el Reto? y ¿Por qué?
b) Según los índices propuestos ¿en qué caso usaría **Linear Probing** o **Separate Chaining** en estos índices? y ¿Por qué?
c) Dado el número de elementos de los archivos Spotify (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
d) ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el factor de carga máximo para cargar el catálogo de Spotify?
e) ¿Qué cambios percibe en el **consumo de memoria** al modificar el factor de carga máximo para cargar el catálogo de Spotify?
f) ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.
g) ¿Qué cambios percibe en el **consumo de memoria** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.
h) ¿Qué configuración de ideal ADT Map escogería para el **índice géneros musicales**?, especifique el mecanismo de colisión, el factor de carga y el numero inicial de elementos.

Las preguntas de análisis y las tablas de datos deben estar en el documento **observaciones-lab7.docx,** al completarlas guarde una copia de este documento en formato **PDF** para evitar problemas de lectura en la evaluación y cópielo en el repositorio de la práctica.

PASO 7: Actualizar los repositorios

Para el repositorio del laboratorio confirme los cambios con los comandos **`Commit`** y **`Push`** en la rama **main** local y de GitHub con el comentario *"laboratorio 7 - Entrega final"* antes de la fecha límite de entrega.

Para el repositorio del Reto confirme los cambios con los comandos **`Commit`** y **`Push`** en la rama **main** local y de GitHub con el comentario *"Reto 2 – Avance en índices"* antes de la fecha límite de entrega.

PASO 5: Revisar entregables de la practica

Finalmente, para realizar la entrega del laboratorio revise que sus entregables de la practica estén completos. Para ello, siga las siguientes indicaciones:

1) Acceso a la organización GitHub del grupo para su profesor y monitores de la sección de laboratorio.
2) **README** de ambos repositorios con los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
3) Enlace al repositorio GitHub **Reto2-G<<XX>>** con rama ***Main*** actualizada con el comentario *"Reto 2 – Avance en índices"* antes del límite de entrega.
4) Enlace al repositorio GitHub **LabCollision -S<<XX>>-G<<YY>>** con rama ***Main*** actualizada con el comentario *"Laboratorio 7 – Entrega final"* antes del límite de entrega.
5) Incluir en repositorio del laboratorio en la carpeta **Docs** el documento **observaciones-lab7.pdf** con las respuestas a las preguntas de observación.

    a) Los datos reportados en la Tabla 1, Tabla 2 y Tabla 3.
    b) La grafica comparativa generada por los resultados de las pruebas de carga de datos.
    c) Las respuestas a las preguntas de observación:

        a) ¿Por qué en la función **`getTime()`** se utiliza **`time.perf_counter()`** en vez de otras funciones como **`time.process_time()`**?
        b) ¿Por qué son importantes las funciones **`start()`** y **`stop()`** de la librería **`tracemalloc`**?
        c) Teniendo en cuenta cada uno de los requerimientos del reto ¿Cuántos índices implementaría en el Reto? y ¿Por qué?
        d) Según los índices propuestos ¿en qué caso usaría **Linear Probing** o **Separate Chaining** en estos índices? y ¿Por qué?
        e) Dado el número de elementos de los archivos Spotify (large), ¿Cuál sería el factor de carga para estos índices según su mecanismo de colisión?
        f) ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el factor de carga máximo para cargar el catálogo de Spotify?
        g) ¿Qué cambios percibe en el **consumo de memoria** al modificar el factor de carga máximo para cargar el catálogo de Spotify?
        h) ¿Qué cambios percibe en el **tiempo de ejecución** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.

i) ¿Qué cambios percibe en el **consumo de memoria** al modificar el esquema de colisiones?, si los percibe, describa las diferencias y argumente su respuesta.
j) ¿Qué configuración ideal de ADT Map escogería para el **índice géneros musicales**?, especifique el mecanismo de colisión, el factor de carga y el numero inicial de elementos.

PASO 9: Compartir resultados con los evaluadores

Envíe el **enlace (URL)** del repositorio por **BrightSpace** antes de la fecha límite de entrega.

Recuerden que cualquier documento solicitado durante la práctica debe incluirse dentro del repositorio GIT y solo se calificarán los entregables hasta el último ***COMMIT*** realizado previo a la media noche **(11:59 PM) del 31 de marzo de 2022.**