Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 2

Tarea 0

Fecha de entrega código: 30 de Agosto del 2022

Objetivos

- Comprender las diferencias entre Arrays y Listas ligadas
- Familiarizarse con el uso de punteros y del manejo de memoria
- Aplicar funciones de Join y Sort de Listas ligadas

Introducción

Vas caminando por el campus San Joaquín llegando tarde a tu clase de "Estructuras de Datos y Algoritmos", cuando de la nada te atropella una manada de furros [1]. Te noquea y al despertar te das cuenta qué te transportaste por el multiverso y ahora te encuentras en el universo de **DCComics**. En este lugar te encuentras con Batman, el caballero oscuro de la noche, quien luego de una persecución nocturna contra su archienemigo ha perdido su batiMP3.

Al ver que provienes de otro mundo, Batman te pone a prueba para aceptarte como Robin, pidiéndote que reconstruyas su batiMP3 y así pueda salir a patrullar con "temazos".

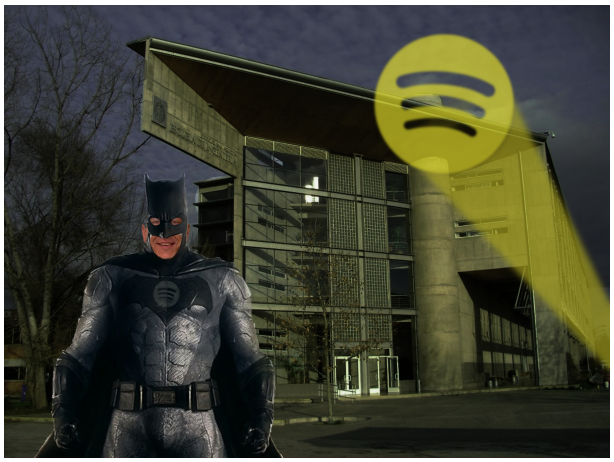

Figura 1: Batman con la cara de Yadran llamándote a reconstruir su batiMP3.

[1] Los furros son figuras antropomórficas de animales que abundan en el DCC, tal como las de este video.

Problema

El problema consiste en reconstruir la batiMP3 de Batman. En el universo **DCComics** existe una cantidad `D` de discos que están disponibles, donde cada disco tiene su propia cantidad de canciones, y cada canción tiene su duración y rating. El batiMP3 tiene una capacidad de `P` batiplaylists, donde cada una de ellas puede guardar una cantidad variable de canciones. Ten en consideración que una batiplaylist no puede contener canciones repetidas. Para esto deberás modelar el funcionamiento del batiMP3 según los eventos mencionados en la sección de eventos.

Eventos

Para facilitar la implementación, los eventos estarán divididos en tres categorías:

A. Batidiscos (20 %)

Esta etapa contempla todo lo que es la creación de los Batidiscos. Se tiene una cantidad `D` de discos, donde cada uno de ellos tiene una cantidad fija N de canciones. Los `songID` de las canciones son relativos a su disco. Los eventos a considerar para esta etapa son los siguientes:

1. `CREAR-DISCO N`

El evento crea un nuevo disco en la primera posición disponible. Este posee una capacidad de `N` canciones. Luego se entregan `N` líneas que corresponden a cada canción del disco en el formato `length rating`.

```
CREAR-DISCO 4
20 3
50 5
35 7
60 2
```

Se ha de imprimir el ID de la canción agregada en el siguiente formato **cada vez** que se agregue una canción:

```
CANCION AGREGADA songId diskId
```

2. `IMPRIMIR-DISCO diskID`

Se ha de imprimir el estado del disco `diskID` solicitado en el siguiente formato:

```
ESTADO DISCO diskID
    N SLOTS
    SLOTS RESTANTES
    MAX RATING songID
    MIN RATING songID
    TOTAL LENGTH
    CANCIONES
        ID_CANCION 1
        ID_CANCION 2
        -
        ID_CANCION 4
        ...
FIN ESTADO
```

Donde `MAX RATING songID` es el máximo rating entre las canciones y su respectivo `songID`, `MIN RATING songID` es el mínimo rating entre las canciones y su respectivo `songID` y `TOTAL LENGTH` es la duración total de las canciones en el disco y por último, `SLOTS RESTANTES` es la cantidad de espacio libre que el disco `diskID` tiene.
En caso que exista un slot vacío, se debe imprimir el carácter "–".

3. IMPRIMIR-CANCION diskID songID

Debes encontrar la canción `songID` en el disco `diskID`. Se debe imprimir el estado de la canción solicitado en el siguiente formato:

```
ESTADO CANCION songID
    LENGTH
    RATING
FIN ESTADO
```

Tu algoritmo de búsqueda no debe superar la complejidad $\mathcal{O}(1)$ y puedes asumir que siempre se va a preguntar por una canción existente.

B. Batiplaylists (35 %)

Esta etapa contempla todo lo que es crear las batiplaylists con el cual Batman va a combatir el crimen de Ciudad Gótica. Se tiene una Batilista que contiene un número arbitrario de batiplaylists, donde cada batiplaylists puede tener una cantidad variable de canciones. Los eventos a considerar para esta etapa son los siguientes:

1. CREAR-BATIPLAYLIST playlistID

Inicializa una batiplaylist `playlistID` que puede almacenar canciones individuales (inicialmente vacía). Debe imprimir lo siguiente:

```
BATIPLAYLIST CREATED playlistID
```

2. AGREGAR-CANCION-BATIPLAYLIST playlistID diskID songID

Agrega la canción `songID` del disco `diskID` a la batiplaylist `playlistID`. Se ha de imprimir el evento en el siguiente formato:

```
NEW SONG ADDED songId diskId playlistId
```

En caso que la canción no exista se debe imprimir

```
SONG NOT FOUND
```

3. ELIMINAR-CANCION-BATIPLAYLIST playlistID diskID songID

Elimina la canción `songID` correspondiente al disco `diskID` de la batiplaylist `playlistID`. Y se ha de imprimir el ID de la canción eliminada en el siguiente formato:

```
ELIMINADO songID diskID playlistID
```

Ojo que ese evento elimina la canción de la batiplaylist, pero la canción sigue existiendo.
Si la cancion no existe en la playlist deberás imprimir

```
SONG NOT FOUND ON PLAYLIST
```

4. AGREGAR-DISCO-BATIPLAYLIST playlistID diskID

Agrega todas las canciones del disco `diskID` al final de la batiplaylist `playlistID`. Debe imprimir la cantidad de canciones agregadas en el siguiente formato:

```
AGREGADO N_SONGS diskID playlistID
```

Donde `N_SONGS` corresponde a la cantidad de canciones del disco que fueron agregadas a la playlist. En el caso de encontrar canciones que ya se encuentran en la playlist, dichas canciones se ignoran y siguen manteniendo su posición original en la playlist.

5. PLAY-BATIPLAYLIST playlistID

Imprime las todas las canciones de la batiplaylist `playlistID`. Se ha de imprimir el estado de la batiplaylist solicitada en el siguiente formato:

```
ESTADO BATIPLAYLIST playlistID
    N CANCIONES
    CANCIONES
        ID_CANCION 1 diskID
        ID_CANCION 2 diskID
        -
        -
        ...
FIN ESTADO
```

6. RATE-BATIPLAYLIST playlistID

Imprime el `rating` promedio de todas las canciones de la batiplaylist `playlistID`. Se ha de imprimir el estado de la batiplaylist solicitada en el siguiente formato:

```
BATIPLAYLIST playlistID: RATING_PROMEDIO
```

Recuerda, el `rating` puede ser un decimal, por lo tanto, lo debes redondear a **dos** decimales.[2]

C. Baticaprichos (45 %)

1. ELIMINAR-BATIPLAYLIST playlistID

Elimina la batiplaylist `playlistID` correspondiente sin afectar el funcionamiento del resto del programa.[3]. Como output debes imprimir lo siguiente:

```
BATIPLAYLIST DELETED playlistID N_SONGS
```

Donde `N_SONGS` corresponde a la cantidad de canciones que tenía la batiplaylist solicitada al momento de eliminarse.

2. UNIR-BATIPLAYLIST playlistId1 playlistId2

Agrega las canciones de `playlistId2` al final la `playlistId1`. Luego elimina `playlistId2`. Se ha de imprimir en el siguiente formato :

```
JOINED playlistId1 AND playlistId2
```

[2] Puedes usar el formato %.2f para aproximar al printear.
[3] Ojo. no elimina las canciones, solo la playlist

3. SPLIT-BATIPLAYLIST playlistID NewPlaylistID position

Divide la batiplaylist en 2 partes, **desde** la posición de `position` en adelante (inclusive). La nueva playlist es agregada con id `NewPlaylistId`. Puedes asumir que `NewPlaylistID` es un espacio vacío.

4. ORDENAR-BATIPLAYLIST playlistId

Este comando ordena las canciones de una batiplaylist basándonos en el criterio length de cada una de forma creciente. Se ha de imprimir en el siguiente formato:

```
SORTED BATIPLAYLIST playlistId
   CANCION songId
   CANCION songId
   ...
END SORTED
```

Hint: estudiar Insertion Sort o Selection Sort para la EDD que corresponda

5. PURGAR-BATIPLAYLIST playlistID rating

Elimina todas las canciones de la playlist que tengan un rating menor[4] a `rating`. Debe imprimir lo siguiente:

```
BATIPLAYLIST PURGED playlistID N_SONGS
```

Donde `N_SONGS` corresponde a la cantidad de canciones eliminadas.

[4] Menor estricto

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `dccomics` que se ejecuta con el siguiente comando:

```
./dccomics input.txt output.txt
```

Donde input será un archivo con los eventos a simular y output el archivo a guardar los resultados.
Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

El archivo de input comenzará con el número de batiplaylist `P`, la siguiente línea sería un número `D` que corresponde a la cantidad de discos, eso significa que habrá(n) `D` evento(s) `CREAR-DISCO` (cada uno seguido de input de canciones respectivos, *ver detalle en la parte A*) y el resto del input de manera indefinida corresponderá a cualquier evento de la tarea, terminando con la secuencia `FIN` que indica el final del programa.

Se verían con el siguiente formato:

```
P
D
CREAR-DISCO N
...
IMPRIMIR-DISCO n
FIN
```

Ejemplo 01

```
1
1
CREAR-DISCO 4
20 3
50 5
35 10
60 20
IMPRIMIR-DISCO 0
FIN
```

Es decir, hay 1 batiplaylist, 1 disco, a su vez este tiene 4 canciones con un `length` y `rating` respectivo. Luego se manda a imprimir al disco en la posición 0 y por último se finaliza la ejecución con el comando `FIN`.

Output

Tu output debe consistir de un archivo con la información solicitada por cada evento en el archivo de input.

Evaluación

La nota de tu tarea es calculada a partir de testcases e Input/Output. Y se descompone de la siguiente forma

- 90 % a la nota entre las partes A, B y C
  - 20 % a la nota de los tests de la parte A
  - 35 % a la nota de los tests de la parte B
  - 45 % a la nota de los tests de la parte C

  Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1.2 GB de ram[5]. De lo contrario, recibirás 0 puntos en ese test.
- 5 % Por no poseer leaks de memoria
- 5 % Por no poseer errores de memoria

Recomendación de tus ayudantes

Las tareas requieren de mucha dedicación de tiempo generalmente, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos. Así mismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, leas el enunciado y te dediques a entender de manera profunda lo que te pedimos. Una vez que hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación y modelación de tu solución, para posteriormente poder programar de manera eficiente. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo :)

Entrega

Código: GIT - Rama master del repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Atraso: Esta tarea no considera cupones ni política de atrasos, cualquier atraso debe ser justificado con el equipo docente del curso.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** por el alumno y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

[5] Puedes revisarlo con el comando htop o con el servidor