Trabajo Práctico 3
Listas enlazadas y Árboles binarios de búsqueda

Estructuras de datos
Comision A (Turno Mañana)
Comision B (Turno Noche)
Segundo cuatrimestre de 2020

Docentes	Sergio Gonzalez Román García Ariel Clocchiatti
Email	sergio.gonzalez@unahur.edu.ar roman.garcia.uni@gmail.com aclocchi@gmail.com
Fecha de entrega	20 / 11 / 2020

Índice

1. Introducción

En este trabajo práctico vamos a trabajar con la integración de los temas de la segunda parte de la materia, centrándonos en las estructuras de datos Lista enlazada y Árboles binarios de búsqueda. Van a tener que utilizar los conocimientos que fueron adquiriendo hasta ahora, para resolver un problema e implementar esta solución en el lenguaje de programación *Python*.

1.1. Presentación del problema

Los sistemas de escucha de música online almacenan ademas de los archivos multimedia, información asociada para darnos resultados de nuestras búsquedas muy rápidamente, teniendo en cuenta que deben recorrer toda la información que tienen almacenada para hacerlo (cerca de **35 millones** de canciones en el caso de *Spotify*). Lo que ocurre es que la aplicación no recorre toda la información cada vez que realiza una búsqueda, sino que recorre una estructura de datos en la que tiene almacenadas todos los nombres de las canciones con sus respectivos intérpretes (banda, cantante, etc). En este trabajo práctico vamos a implementar un árbol binario que almacene información básica y distintas operaciones para poder hacer busquedas dentro de él.

Para el modelado de la estructura, se deben tener en cuenta estas condiciones:

- El árbol se ordena usando los nombres de los interpretes, es decir, las claves de los nodos del árbol son los nombres.
- En cada nodo del árbol, ademas del nombre del intérprete, vamos a tener una lista con los nombres de sus canciones.

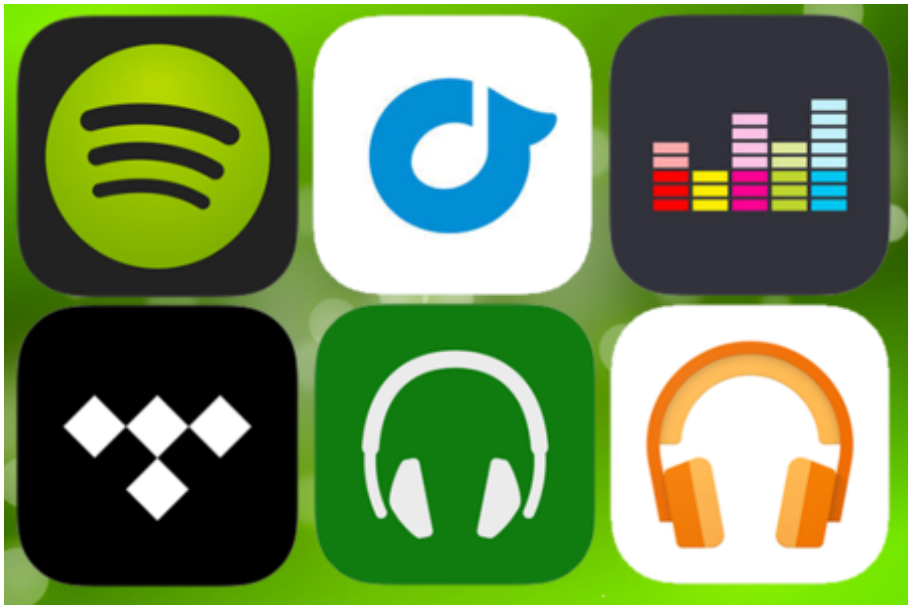

Figura 1: Algunas de las aplicaciones de escucha de muúsica *online* actuales.

2. Objetivos

Definir un TDA que represente al árbol binario de búsqueda para almacenar los interpretes y canciones que necesita la aplicación. Cada vez que necesiten una **Lista** deben utilizar su propio TDA "Lista Enlazada", la implementación que hicieron en la guía de ejercicios (Recursiva o Iterativa, la que quieran), si necesitan alguna operación que no tienen en ese TDA, deben agregarla a la implementación. **No utilizar la implementación de Listas de Python**

El TDA ***ArbolDeCanciones*** (con sus nodos de tipo ***NodoArbolDeCanciones***) debe incluir las siguientes operaciones:

- **insertarCanciones(listaCanciones, nombreInterprete):** Inserta cada una de las canciones de la lista en el árbol. Si el intérprete ya existe en el árbol, agrega cada canción a la lista de canciones (se debe verificar previamente si la canción está en la lista, para no duplicar información). Si el intérprete no existe en el árbol, agrega un nuevo nodo con el nuevo intérprete y las canciones en el lugar correspondiente. *Nota: La lista de canciones de entrada **NO** tiene canciones duplicadas.*
- **interpretesDeCancion(nombreCancion):** Recibe el nombre de una canción y retorna una lista de todos los interpretes que tienen una canción con ese nombre (pueden ser uno o más intérpretes). Si no hay ninguna canción con ese nombre almacenada en el árbol, retorna una lista vacía.
- **buscarCanciones(listaInterpretes):** Recibe una lista con los intérpretes buscados. Retorna una lista con los nombres de las canciones compartidas por **todos** los intérpretes de la lista de entrada. Si no hay ninguna canción compatida por **todos** los intérpretes, debe retornar una lista vacía.
- **eliminarInterprete(nombreInterprete):** Elimina del árbol el intérprete que recibe por parámetros.
- **eliminarCanción(nombreCancion):** Elimina del árbol la canción que recibe por parámetro. Debe eliminarla de todos los nodos del árbol donde se encuentre.
- **cantidadTotalInterpretes(Palabra):** Recibe una palabra por parámetro y retorna la cantidad **total** de intérpretes almacenados en el árbol que tienen esa palabra formando parte de su nombre (completa o como parte de otra. Ej: La palabra *jugando* forma parte de la palabra *conjugando*).
- **raizBalanceada():** Retorna *True* si la diferencia de altura entre los subárboles hijos de la raiz es menor o igual a 1 y *False* en caso contrario.
- **cancionesEnNivel(nivel):** Recibe un nivel y retorna una lista con las canciones de todos los intérpretes que están en ese nivel del árbol. La lista no debe tener canciones repetidas. Si en ese nivel no hay nada, retorna una lista vacía.
- **interpretesConMasCanciones(cantidadCancionesMinima):** Recibe una cantidad de canciones por parámetro y retorna la cantidad de intérpretes que tienen esa cantidad de canciones o más almacenadas en el árbol.
- **internosAlfabetico():** Retorna una lista con los intérpretes que se encuentran en un nodo interno del árbol (no hojas). La lista debe contener los nombres de los intérpretes en orden alfabético.

3. Datos de prueba

Previo a la entrega del trabajo práctico, deben controlar que su implementación funcione con el lote de datos de prueba. El lote de prueba consta de un *script* en lenguaje *Python* y un archivo *.csv* con los datos que el *script* usa para generar el árbol de prueba.

Link para *script*:
https://colab.research.google.com/drive/1R6yO_CJAIYUn7IgU36_LsmDGNpHAMzJQ?usp=sharing

Link Para archivo *.csv*:
https://drive.google.com/file/d/1im8le4_KYIN_6l6AtSPpoORQibX2QDgA/view?usp=sharing

Deben respetar los nombres de las operaciones y de los TDAs (***Lista***, ***ArbolDeCanciones*** y ***NodoArbolDeCanciones***). En la parte superior del *script* de prueba tienen las indicaciones de nombres de variables y nombres de operaciones que deben respetar. El mismo *script* contiene un bloque de código con la función *treePlot* para gráficar los árboles.

4. Entrega

La entrega del trabajo práctico debe ser:

- Un informe escrito (doc, pdf. etc), incluyendo:
  - Descripción de cada una de las **estructuras de datos** diseñadas e implementadas. Incluir una descripción escrita de los algoritmos. Pueden incluir diagramas de flujo.
  - Descripción de la implementación en *Python*. Explicar **claramente** que hace cada función y procedimiento implementados.
- Código completo y comentado de la implementación.
- Opcional: Video explicando cómo funciona el algoritmo.

Luego de la entrega, les vamos a hacer preguntas a las/los integrantes del grupo sobre el trabajo, asi que les aconsejamos no copiar código que encuentren en internet.