Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-2

29 de Octubre de 2020
Actividad Formativa

Actividad Formativa 04

Estructuras Nodales I: Árboles

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AF04/`
- **Hora del *push*:** 16:50

Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

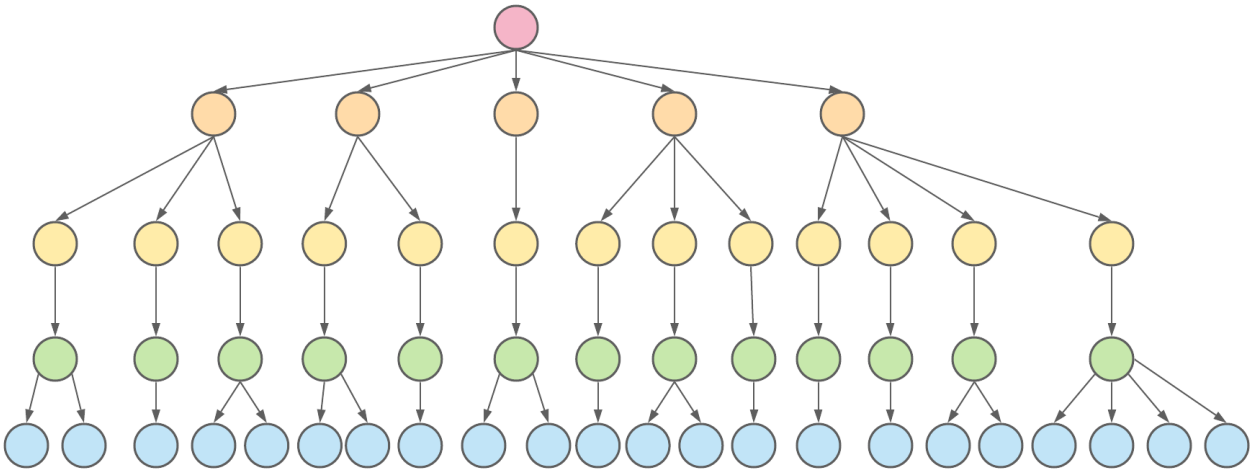
Introducción

A estas alturas del semestre con tanto que programar, ya escuchaste todas las canciones que se te ocurren. Sientes que siempre escuchas lo mismo y es difícil encontrar nuevas canciones, y cuando hablas con tus amigos y amigas te dicen lo mismo. Un día, mientras estabas escuchando música y estudiando la materia de árboles, se te ocurrió una brillante idea... ¿Por qué no hacer un programa con árboles que recomiende música? Y así es como se te ocurrió crear DCConcert.

Flujo del programa

Para este programa se te entregarán tres archivos `.csv` de plataformas de música (Spotify, Youtube Music y Amazon Music), donde cada uno contendrá la información de una de las plataformas. Con esta información deberás crear un árbol por cada plataforma, para posteriormente, realizar consultas en donde tendrás que recorrer los árboles.

A continuación se muestra la estructura que deberán tener los árboles de las plataformas:



La estructura del árbol por niveles desde arriba hacia abajo respeta las siguientes reglas:

- El primer nivel es el nodo raíz que se muestra en **rosado**, representa a la plataforma del árbol que puede ser Spotify, Youtube Music o Amazon Music.
- El segundo nivel tendrá nodos que representan el género musical, que aparecen en **naranjo**.
- El tercer nivel tendrá nodos que representan a artistas, que aparecen en **amarillo**.
- El cuarto nivel tendrá nodos que representan álbumes, que aparecen en **verde**.
- El quinto nivel tendrá nodos que representan el título de la canción, que aparecen en **azul**.

Tu deber es completar el programa dado para que este pueda construir tres de estos árboles a partir de tres bases de datos para plataformas musicales distintas. Una vez lista la construcción del árbol el programa deberá poder realizar distintas consultas sobre este y deberás poner a prueba tus habilidades de búsqueda en árboles. Para aventurarte con este novedoso servicio debes ejecutar el archivo `main.py`. Dentro de él podrás probar todas tus funciones de consultas para los diferentes servicios disponibles.

Archivos recibidos

Base de datos Se entrega una carpeta `data/` con las bases de datos para cada servicio musical. Dentro de ella encontrarás dos set de archvios `facil/` y `dificil/`, los cuales te permitirán trabajar bases de datos de diferentes tamaños. A su vez, dentro de cada uno tendrás los siguientes archivos para cada servicio:

- `spotify.csv`
- `amazon_music.csv`
- `youtube_music.csv`

Puedes cambiar las bases de datos entre `facil` o `dificil` en la línea 19 de `main.py`.

Cada uno de estos archivos contendrá una canción por linea, las cuales cumplirán con el siguiente formato:

`nombre_cancion,nombre_album,nombre_artista,genero_artista`

Puedes cargar estos datos mediante la función `def cargar_servicio(ruta_archivo)` definida en el módulo `cargar_datos.py`. Esta función retornará una lista de diccionarios con el siguiente formato para cada canción:

```
{
    "nombre": "Bad Romance",
    "artista": "Lady Gaga",
    "album": "The Fame Monster",
    "genero": "pop"
}
```

Puedes probar el cargado de los datos ejecutando directamente `cargar_datos.py`, lo cual retornará lo mostrado en la sección Anexos.

Parte 1: Creación del árbol

En esta parte se te entregarán las clases `Nodo` y `Arbol` que se encuentran en el archivo `arboles.py`.

Clase Nodo

Esta clase está completamente implementada, por lo que **no deberás modificarla**. Además, tiene los siguientes atributos:

- `self.tipo`: *String* que representa el tipo de nodo, que puede ser `"plataforma"`, `"genero"`, `"artista"`, `"album"` o `"cancion"`.
- `self.valor`: *String* con información que va depender del tipo de nodo. Ej: Si el nodo es de tipo `"género"`, entonces el valor podría ser `"pop"`, `"rock"`, etc.
- `self.padre`: Nodo padre (otra instancia de `Nodo`).
- `self.hijos`: Lista donde se almacenan los nodos hijos (otras instancias de `Nodo`).

Clase PlataformaMusical

Esta clase representa un árbol que está parcialmente implementado, por lo que deberás modificar solo lo que se te indique. En el método `__init__`, que **no deberás modificar**, se inicializa el siguiente atributo:

- `self.raiz`: Nodo raíz (Instancia de nodo).

A continuación se muestra el método `agregar_cancion` de esta clase, que deberás modificar:

- `def agregar_cancion(info_canciones)`: Recibe un diccionario con la información de una canción, el cual tendrá el formato explicado en la sección Archivos Recibidos. En base a esta información se debe crear el Nodo de tipo `"cancion"` con el nombre de la canción como valor y agregarlo a la lista de hijos del nodo correspondiente al álbum.

  Veamos un ejemplo con la siguiente canción:

  ```
  {
      "nombre": "Bad Romance",
  ```

```
        "artista": "Lady Gaga",
        "album": "The Fame Monster",
        "genero": "pop"
    }
```

Para poder agregar esta canción es necesario que existan los nodos:

- **Album** de valor `"The Fame Monster"`.
- **Artista** de valor `"Lady Gaga"`.
- **Genero** de valor `"pop"`.

Donde se debe buscar primero el genero, luego el artista y luego el álbum para agregar la canción. En caso de que uno de ellos no exista, es necesario agregarlo respetando la jerarquía para poder continuar. Al completar este método pruébalo con la opción 'Visualizar árbol' del menú en el main. Puedes ver un ejemplo en la sección Anexos.

- `def armar_arbol(informacion_canciones)` Este método se encargará de llamar al método ya explicado, `agregar_cancion()`, y entregarle la información necesario para cargar todos los nodos del árbol. **No lo debes modificar.**

Parte 2: Consultas

En esta parte se te pedirá implementar diferentes funciones que realizan consultas. Estas funciones están contenidas en el archivo `consultas.py`.

- `def buscar_info_artista(plataforma, artista_seleccionado)`: Esta función recibe como parámetros `plataforma` (instancia de `Nodo`), que representa la plataforma que se está analizando, y `artista_seleccionado` (`str`), que representa el nombre de un artista, e imprime un resumen de todos los álbumes del artista y la cantidad de canciones que ellos tienen. Por ejemplo, el árbol de `Spotify` en modo difícil, si se consulta por `"U2"`, la consulta podría imprimir algo parecido a lo siguiente

```
> U2
Álbum: Achtung Baby, 1 cancion(es)
Álbum: The Unforgettable Fire (Deluxe Edition Remastered), 1 cancion(es)
```

La forma en cómo se imprime la información queda a tu criterio. **Debes implementar esta función**.

- `def buscar_mejor_plataforma(genero, plataformas)`: Esta función recibe como parámetros `genero` (`str`), que representa el nombre del género musical que se está buscando y `plataformas` (`list`) con las instancias de la clase `PlataformaMusical`. Esta función debe contar la cantidad de canciones de ese género que tiene cada plataforma y luego basado en estos cálculos **retornar el objeto `PlataformaMusical`** con más canciones de ese tipo. **Debes implementar esta función**.

Un ejemplo de la impresión que se ejecuta en el `main.py` en base a esta función sería la siguiente:

```
"Para escuchar música dance pop, definitivamente creemos que debes probar con: Amazon Music"
```

- `def buscar_artistas_parecidos(nombre_cancion, plataforma)`: Esta función recibirá como parámetros `nombre_cancion` (`str`), que representa el nombre de la canción que se usa como referencia, y `plataforma` (instancia de `Nodo`), que representa a la plataforma en la cuál se buscan artistas parecidos. Esta función debe buscar la canción que en su nombre contenga `nombre_cancion`, identificar

a que genero pertenece, y **retornar una lista** con los nombres de los demás artistas que compongan canciones del mismo genero, sin considerar al artista de la canción buscada. **Debes implementar esta función**.

- `def crear_playlist(plataforma, genero_seleccionado, conceptos_canciones)`: Esta función recibe como parámetros `plataforma`, que es un objeto de la clase `Nodo` que representa la plataforma que se está analizando, `genero_seleccionado` (`str`), que representa al nombre del género musical del que se quiere hacer la *playlist* y `conceptos_canciones`, que es una lista con *strings*, que representan palabras que pueden estar contenidas en nombres de canciones. Por ejemplo, si los conceptos son `"amor"` y `"vida"`, se deben seleccionar todas las canciones del género escogido que contengan alguno de estos dos conceptos dentro de sus títulos[1]. Retorna una lista de *strings* con el nombre de todas las canciones que cumplen los criterios mencionados. **Debes implementar esta función**.

Objetivos de la Actividad

- Poblar un árbol que representa un servicio de *streaming* musical.
- Realizar búsquedas dentro de un árbol según diferentes criterios.

[1] No importan mayúsculas ni minúsculas

Anexos

Resultado al cargar `facil/spotify.csv`

Esto es lo que se muestra al ejecutar `cargar_datos.py`.

```
[
    {
        'nombre': 'Meant to Be (feat. Florida Georgia Line)',
        'album': 'All Your Fault: Pt. 2',
        'artista': 'Bebe Rexha',
        'genero': 'pop'
    },
    {
        'nombre': 'Believe',
        'album': 'Believe',
        'artista': 'Cher',
        'genero': 'dance pop'
    },
    {
        'nombre': 'Castle on the Hill',
        'album': 'Castle on the Hill',
        'artista': 'Ed Sheeran',
        'genero': 'pop'
    },
    {
        'nombre': 'Ub3R',
        'album': 'Ub3R',
        'artista': 'C. Malloy',
        'genero': 'dance pop'
    },
    {
        'nombre': '"\'CAN\'\'T STOP THE FEELING! (Original Song from DreamWorks\
                   Animation\'\'s ""TROLLS"")\'"',
        'album': '"\'CAN\'\'T STOP THE FEELING! (Original Song from DreamWorks\
                  Animation\'\'s ""TROLLS"")\'"',
        'artista': 'Justin Timberlake',
        'genero': 'pop'
    },
    {
        'nombre': 'Loco Enamorado',
        'album': 'Loco Enamorado',
        'artista': 'Abraham Mateo',
        'genero': 'reggaeton'
    },
    {
        'nombre': 'Love Me Land',
        'album': 'Love Me Land',
        'artista': 'Zara Larsson',
        'genero': 'dance pop'
    },
```

```
    {
        'nombre': 'Stitches',
        'album': 'Handwritten',
        'artista': 'Shawn Mendes',
        'genero': 'dance pop'
    },
    {
        'nombre': 'Amigos y Enemigos (feat. Bad Bunny & Almighty) - Remix',
        'album': 'Amigos y Enemigos (Remix)',
        'artista': 'Noriel',
        'genero': 'reggaeton'
    },
    {
        'nombre': 'Sigo Extrañándote',
        'album': 'Energía',
        'artista': 'J Balvin',
        'genero': 'reggaeton'
    },
    {
        'nombre': 'Havana - Remix',
        'album': 'Havana (Remix)',
        'artista': 'Camila Cabello',
        'genero': 'pop'
    },
    {
        'nombre': 'Dill Dall (feat. Lothepus & Pilgaard)',
        'album': 'Dill Dall (feat. Lothepus & Pilgaard)',
        'artista': 'Benjamin Beats',
        'genero': 'reggaeton'
    },
    {
        'nombre': 'Somebody',
        'album': 'Strangers / Lovers',
        'artista': 'Dagny',
        'genero': 'pop'
    },
    {
        'nombre': '10,000 Hours (with Justin Bieber)',
        'album': '10,000 Hours (with Justin Bieber)',
        'artista': 'Dan + Shay',
        'genero': 'pop'
    },
    {
        'nombre': 'lovely (with Khalid)',
        'album': 'lovely (with Khalid)',
        'artista': 'Billie Eilish',
        'genero': 'pop'
    }
]
```

Ejemplo de impresión de árbol

Así se ve al imprimir el árbol del servicio `Spotify` en modo `facil` desde el menú del *main*.

```
Spotify
  pop
    Bebe Rexha
      All Your Fault: Pt. 2
        Meant to Be (feat. Florida Georgia Line)
    Ed Sheeran
      Castle on the Hill
        Castle on the Hill
    Justin Timberlake
      "'CAN''T STOP THE FEELING! (Original Song from DreamWorks Animation''s ""TROLLS"")'"
        "'CAN''T STOP THE FEELING! (Original Song from DreamWorks Animation''s ""TROLLS"")'"
    Camila Cabello
      Havana (Remix)
        Havana - Remix
    Dagny
      Strangers / Lovers
        Somebody
    Dan + Shay
      10,000 Hours (with Justin Bieber)
        10,000 Hours (with Justin Bieber)
    Billie Eilish
      lovely (with Khalid)
        lovely (with Khalid)
  dance pop
    Cher
      Believe
        Believe
    C. Malloy
      Ub3R
        Ub3R
    Zara Larsson
      Love Me Land
        Love Me Land
    Shawn Mendes
      Handwritten
        Stitches
  reggaeton
    Abraham Mateo
      Loco Enamorado
        Loco Enamorado
    Noriel
      Amigos y Enemigos (Remix)
        Amigos y Enemigos (feat. Bad Bunny & Almighty) - Remix
    J Balvin
      Energía
        Sigo Extrañándote
    Benjamin Beats
      Dill Dall (feat. Lothepus & Pilgaard)
        Dill Dall (feat. Lothepus & Pilgaard)
```