Trabajo Práctico 3: Recomendación de canciones en Spotify

El trabajo práctico número 3 tiene fecha de entrega para el día ** 5/12**.

Contenido

Introducción

El objetivo de este trabajo práctico es el de modelar un sistema de recomendación sobre un sistema de streaming de música: las canciones, playlists y usuarios.

Nos interesa modelar y entender cómo están compuestas las playlists, si un usuario y otro son similares, poder recomendar canciones o playlists enteras.

Datos disponibles

Trabajaremos con una ínfima porción de datos obtenidos de una competencia pública realizada por la plataforma Spotify en 2018. Cuenta con cerca de 2 millones de entradas. Pueden encontrar el archivo ya procesado aquí. El formato del archivo es `tsv`:

```
ID      USER_ID TRACK_NAME      ARTIST  PLAYLIST_ID     PLAYLIST_NAME   GENRES
1       sitevagalume    Eraser  Ed Sheeran      6736120 Ed Sheeran - Divide     Pop,Rock,Pop/Rock
2       sitevagalume    Castle On The Hill      Ed Sheeran      6736120 Ed Sheeran - Divide     Pop,Rock,Pop/Rock
3       sitevagalume    Dive    Ed Sheeran      6736120 Ed Sheeran - Divide     Pop,Rock,Pop/Rock
...
3251    sapatini        Não Foi Em Vão  Sorriso Maroto  1169827 \o/     Romântico,Pagode,Samba
3252    sapatini        Não Mereço Ser Amante   Sorriso Maroto  1169827 \o/     Romântico,Pagode,Samba
...
```

- `ID`: id de la entrada
- `USER_ID`: Id del usuario que creó la playlist en cuestión.
- `TRACK_NAME`: nombre de la canción
- `ARTIST`: nombre del artista (o grupo), compositor/a de la canción.
- `PLAYLIST_ID`: id de la playlist en la que se encuentra esta canción (no necesariamente una canción pertenece a una sóla playlist; algo muy improbable)
- `PLAYLIST_NAME`: nombre de la playlist.
- `GENRES`: géneros a los que pertenece la canción (separados por coma)

Algo a tener en cuenta es que dos artistas/grupos diferentes podrían tener canciones de mismo nombre. Por ejemplo, tanto *Ed Sheeran* como *Nine Inch Nails*, o bien *Coheed and Cambria*, tienen una canción llamada *Eraser*.

Como dicho archivo contiene una enorme cantidad de entradas, les dejamos también un archivo más reducido, para que puedan realizar pruebas más rápidamente.

Para la modelación, se recomienda el siguiente esquema:

1. Tener un grafo no dirigido que relacione si a un usuario le gusta una canción (supongamos que a un usuario le gusta una canción si armó una playlist con ella). Esto formará un grafo bipartito en el cual en un grupo estarán los usuarios, y en el otro las canciones.
2. Tener un grafo no dirigido relacionando canciones si aparecen en playlists de un mismo usuario (no necesariamente ambas en la misma playlist; es decir, si una canción aparece en alguna playlist del usuario X, y otra canción aparece en alguna playlist del mismo usuario X, pudiendo ser la misma playlist, o no). Este es el grafo a tener en cuenta para los útimos 2 comandos (`rango` y

`ciclo`). **Importante**: construir este grafo es costoso. No construirlo si no es necesario.

Consigna

Dado este archivo parseado, se debe modelar el sistema estructuras Grafo considerando los datos disponibles. Esto implica determinar todas las características necesarias para el o los grafos. Se pide implementar un programa que reciba como parámetro la ruta del archivo procesado:

```
$ ./recomendify spotify-procesado.tsv
```

Implementación

El trabajo puede realizarse en lenguaje a elección, siendo aceptados Go, Python y C, y cualquier otro a ser discutido con el corrector asignado.

El trabajo consiste de 3 partes:

1. El TDA Grafo, con sus primitivas completamente agnósticas sobre su uso para modelar el sistema de recomendación de música.
2. Una biblioteca de funciones de grafos, que permitan hacer distintas operaciones sobre un grafo que modela el sistema de música, sin importar cuál es la red específica.
3. El programa `recomendify` que utilice tanto el TDA como la biblioteca para poder implementar todo lo requerido.

Es importante notar que las primeras dos partes deberían poder funcionar en cualquier contexto: El TDA Grafo para cualquier tipo de TP3 (o utilidad); la biblioteca de funciones debe funcionar para aplicar cualquiera de las funciones implementadas sobre cualquier grafo que tenga las características de las de este TP. La tercera parte es la que se encuentra enteramente acoplada al TP en particular.

El programa debe recibir el set de datos por parámetro (`$ ./recomendify spotify-procesado.tsv`), cargarlo en memoria y luego solicitar el ingreso de comandos por entrada estándar, del estilo `<comando> 'parametro'`. Nótese que esto permite tener en un archivo las instrucciones a ser ejecutadas (i.e. `$ ./recomendify spotify-procesado.tsv < entrada.txt`).

A continuación se listarán los comandos junto a ejemplos de entrada y salidas para el caso de la red reducida.

Camino más corto

- Comando: `camino`.
- Parámetros: `origen` y `destino` (separados por `>>>>`). Origen y destino son **canciones**. Se indica el autor en cada caso.
- Utilidad: nos imprime una lista con la cual se conecta (en la menor cantidad de pasos posibles) una canción con otra, considerando los usuarios intermedios y las listas de reproducción en las que aparecen.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(U + C + L)$, siendo U la cantidad de Usuarios, C la cantidad de canciones, y L la cantidad de apariciones totales de canciones en listas de reproducción (las aristas del grafo).
- Ejemplos: Entrada:

```
camino Don't Go Away - Oasis >>>> Quitter - Eminem
camino yarits >>>> Quitter - Eminem
camino Oops!...I Did It Again - Britney Spears >>>> Love Story - Taylor Swift
camino Mr. Brightside - The Killers >>>> Grow Old With Me - Tom Odell
```

Salida:

```
Don't Go Away - Oasis --> aparece en playlist --> misturo tudãao ;x --> de --> 8902446 --> tiene una playlist --> sóc
Tanto el origen como el destino deben ser canciones
Oops!...I Did It Again - Britney Spears --> aparece en playlist --> Britney Spears --> de --> aline_hdb --> tiene una
No se encontro recorrido
```

En caso de que el `origen` o `destino` indicados no sean canciones válidas, se debe imprimir `Tanto el origen como el destino deben ser canciones`.

Canciones más importantes

Utilizaremos el algoritmo de PageRank para implementar este comando, para determinar cuáles son la **canciones** más importantes.

- Comando: `mas_importantes`.
- Parámetros: `n`, la cantidad de canciones más importantes a mostrar.
- Utilidad: nos muestra las `n` **canciones** más centrales/importantes del mundo según el algoritmo de pagerank, ordenadas de mayor importancia a menor importancia. También pueden utilizar otra métrica/algoritmo para detectar esto, como Hubs and Authorities (paper original aquí).
- Ejemplo: Entrada:

```
mas_importantes 20
```

Salida:

```
Bad Romance - Lady Gaga; Poker Face - Lady Gaga; Telephone (feat. Beyoncé) - Lady Gaga; Paparazzi - Lady Gaga; Halo -
```

Importante: Considerar que esto podría pedirse varias veces por ejecución; y no se desea repetir el cálculo (ya que no debería cambiar el valor de pagerank de ningún ítem). Además, dado que hay un factor aleatorio, puede llegar a variar levemente la lista. En las pruebas del curso se tiene bien contemplado esto.

Recomendación (usuarios o canciones)

Usando la idea de PageRank Personalizado, y lo indicado en este paper, implementaremos un algoritmo que permita determinar qué otros usuarios se puede recomendar seguir, o bien qué canciones probablemente pudieran gustar, a partir de un listado de canciones.

La idea será aplicar un PageRank Personalizado en el grafo completo (bipartito) desde el listado de canciones pasadas. Las canciones que tengan mayor PageRank Personalizado serán canciones a recomendar para escuchar (considerar no recomendar una canción que ya esté en la lista pasada), los usuarios con mayor PageRank Personalizado serán usuarios para recomendar seguir.

- Comando: `recomendacion`.
- Parámetros: `usuarios/canciones`, si se espera una recomendación para seguir un usuario o para escuchar una cancion; `n`, la cantidad de usuarios o canciones a recomendar; `cancion1 >>>> cancion2 >>>> ... >>>> cancionK`, las canciones que ya sabemos que le gustan a la persona a recomendar (que podrías ser vos misma ;-)).
- Utilidad: Dar una lista de `n` usuarios o canciones para recomendar, dado el listado de canciones que ya sabemos que le gustan a la persona a la cual recomedar.
- Ejemplo: Entrada:

  ```
  recomendacion canciones 10 Love Story - Taylor Swift >>>> Toxic - Britney Spears >>>> I Wanna Be Yours - Arctic Monke
  recomendacion usuarios 5 Love Story - Taylor Swift >>>> Toxic - Britney Spears >>>> I Wanna Be Yours - Arctic Monkeys
  ```

  Salida:

  ```
  Butterfly - Grimes; Cola - Lana Del Rey; In Time - FKA Twigs; Touch - Troye Sivan; Hurricane - 30 Seconds To Mars; Bo
  lorenafazion; naosoumodinha; hlovato906gmail; tiagogabbana19; extralouca
  ```

Nuevamente, puede haber un cierto factor aleatorio que modifique un poco el orden de la lista, o algún valor específico. Esto es tenido en cuenta por las pruebas del curso.

Ciclo de n canciones

- Comando: `ciclo`.
- Parámetros: `n` y `cancion`.
- Utilidad: permite obtener un ciclo de largo `n` (dentro de la red de canciones) que comience en la canción indicada.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(C^n)$. Es decir, el comando puede tomar mucho tiempo. Para su implementación, recomendamos el uso de un DFS, en el que podamos sacar a los vértices entre los visitados. Es decir, si estoy visitando a un vértice y luego de visitar a todos los suyos, en ningún caso encontré un ciclo de largo `n`, entonces "desmarcar" como visitado al vértice. Esto hace que el mismo vértice sea visitado más de una vez (y de allí que cueste tanto ejecutar el algoritmo).
- Ejemplo: Entrada:

  ```
  ciclo 7 By The Way - Red Hot Chili Peppers
  ciclo 15 Love Me Like You Do - Ellie Goulding
  ```

  Salida:

  ```
  By The Way - Red Hot Chili Peppers --> Fairy Tale - Shaman --> I Hate Everything About You - Three Days Grace --> Viv
  Love Me Like You Do - Ellie Goulding --> Uptown Funk (Feat. Bruno Mars) - Mark Ronson --> Thinking Out Loud - Ed Shee
  ```

En caso de no haber un ciclo de dicho largo empezando desde la página mencionada, debe escribirse por salida estándar `No se encontro recorrido`.

Todas en Rango

- Comando: `rango`.
- Parámetros: `n` y `cancion`.
- Utilidad: permite obtener la cantidad de canciones que se encuentren a **exactamente** `n` saltos desde la `cancion` pasada por parámetro.

- Complejidad: Este comando debe ejecutar en $\mathcal{O}(C + L)$.
- Ejemplo: Entrada:

```
rango 8 Shots - Imagine Dragons
rango 3 Shots - Imagine Dragons
rango 2 After Dark - Asian Kung-fu Generation
rango 4 I'm Yours - Jason Mraz
```

Salida:

```
0
2171
1059
0
```

Entrega

Adicionalmente a los archivos propios del trabajo práctico debe agregarse un archivo `entrega.mk` que contenga la regla `recomendify` para generar el ejecutable de dicho programa (sea compilando o los comandos que fueren necesarios). Por ejemplo, teniendo un TP implementado en Python, podría ser:

```
recomendify: recomendify.py grafo.py biblioteca.py
    cp recomendify.py recomendify
    chmod +x recomendify
```

Importante: En caso de recibir un error `FileNotFoundError: [Errno 2] No such file or directory: './recomendify': './recomendify'`, tener en cuenta que para el caso de enviar código escrito en Python es necesario además indicar la ruta del intérprete. Esto puede hacerse agregando como primera línea del archivo principal (en el ejemplo, sería `recomendify.py`) la línea: `#!/usr/bin/python3`.

Si el TP fuera realizado en Go, un posible ejemplo del archivo podría ser:

```
recomendify:
    cd mi_implementacion_tp3; go build -o ../recomendify
```

Criterios de aprobación

El código entregado debe ser claro y legible y ajustarse a las especificaciones de la consigna. Debe compilar sin advertencias y correr sin errores de memoria.

La entrega incluye, obligatoriamente, los siguientes archivos de código:

- el código del TDA Grafo programado, y cualquier otro TDA que fuere necesario.
- el código de la solución del TP, separado entre los 3 secciones mencionadas en la consigna (que pueden ser separados en archivos, módulos, o como prefieran, pero deben estar claramente distinguibles).

La entrega se realiza en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.