Algoritmos 2

Licenciatura en Ciencia de Datos - UNSAM

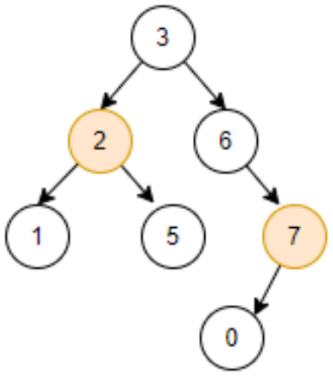
Examen parcial

Formato de entrega: Código fuente de módulos y/o paquetes python
Criterio mínimo de aprobación: Alcanzar al menos 5 puntos

Resolver los ejercicios utilizando los principios del Paradigma Orientado a Objetos.
Entregar cada ejercicio en su propio paquete o módulo (ejercicio1 y ejercicio2).

1) Se necesita modelar un **árbol binario** con nodos que se identifican con números enteros que no se repiten (no está necesariamente ordenado), similar al ArbBin(Entero) que vimos en la cursada, pero con una particularidad. Este árbol puede tener algunos nodos especiales que, cuando se realiza un recorrido especial (le diremos **recorrido parcial**) estos nodos actúan como si fueran hojas, es decir, como si no tuvieran descendientes. Hay que contemplar que estos nodos especiales son árboles **no** vacíos que responden como cualquier otro nodo común en los recorridos clásicos sobre árboles binarios que hemos visto, por lo cual **sólo se diferencian al momento de hacer un recorrido especial**.

   Por ejemplo:



   Nodos especiales: 2 y 7

   Recorrido DFS preorder: 3, 2, 1, 5, 6, 7, 0

   Recorrido DFS preorder especial: 3. 2. 6. 7

   Se solicita realizar lo siguiente:

   a) Definir la clase **ArbEsp** que soporte el modelo propuesto en el enunciado, incluyendo otras clases adicionales si es necesario. *(1 pt)*

   b) Implementar la operación **preorder_especial** que, dado un árbol **ArbEsp**, genere una **lista de enteros con el recorrido preorder especial del árbol**, **contemplando los nodos especiales como si fueran hojas**. *(1 pt)*

   c) Implementar la operación **es_especial** que, dado un árbol **ArbEsp** y un **número entero**, devuelva **si existe o no** en el árbol un **nodo especial** identificado con ese número. *(1 pt)*

d) Implementar la operación **podados** que, dado un árbol **ArbEsp**, devuelva la **cantidad de nodos** del árbol **que no se recorren cuando se realiza un recorrido especial**, es decir, la cantidad de nodos que son ignorados en ese tipo de recorrido. *(1 pt)*

2) La plataforma de streaming ChauFlix recolecta información de la **cantidad de películas** que ven sus usuarixs, con el objeto de realizar futuras recomendaciones de películas a usuarixs que tengan un perfil similar. Las películas tienen un título, duración (minutos) y **uno o más géneros** (comedia, acción y drama). Cuando un usuarix mira una película, **se actualiza su perfil con los géneros que tiene esa película**. Esta actualización **incrementa en uno cada uno de los géneros en su perfil**.

   Por ejemplo, si Ana tiene un perfil {acción: 0, comedia: 1, drama: 3} y luego mira una película de acción y comedia, su perfil se actualiza a: {acción: 1, comedia: 2, drama: 3}.

   Se considera que dos usuarixs tienen **un perfil similar** cuando **las diferencias de cada género visto difieren como máximo en 1**.
   Por ejemplo:

```
perfil 1: {acción: 0, comedia: 1, drama: 3}
perfil 2: {acción: 2, comedia: 1, drama: 3}
perfil 3: {acción: 1, comedia: 0, drama: 4}
El perfil1 es similar al perfil3 y viceversa.
El perfil1 no es similar al perfil2 y viceversa.
El perfil2 es similar al perfil3 y viceversa.
```

   Se solicita realizar lo siguiente:

a) Definir la clase **ChauFlix** que soporte el modelo del enunciado, contemplando el escenario de prueba propuesto e incluyendo otras clases adicionales, si es necesario. *(1 pt)*

b) Implementar una operación privada en ChauFlix llamada **actualizar_relaciones** donde se recorran todos los usuarixs de la plataforma y se actualicen las relaciones que indican que un usuarix está relacionado con otrx, es decir, **tiene un perfil similar**. *(2 pts)*

c) Implementar la función recursiva **tienen_alguna_relación** que, dados dos usuarixs, devuelva si se relacionan directa o indirectamente entre sí. La relación directa se da cuando tienen perfiles similares. La relación indirecta puede darse por transitividad. *(3 pts)*

   Por ejemplo:
   - Juana se relaciona con Ana, tienen perfil similar.
   - Juana no se relaciona con María, no tienen perfil similar.
   - Ana se relaciona con María, tienen perfil similar.
   - Juana se relaciona con María indirectamente.

Escenario de prueba:
Se provee el siguiente escenario para probar y validar los resultados esperados. Se recomienda entregar este escenario como ejecución dentro del main.

```
def main():
    chau_flix = ChauFlix()

    # Users
    juana = User("Juana")
    ana = User("Ana")
    maria = User("Maria")
    pedro = User("Pedro")
    chau_flix.agregar_usuarix(juana)
    chau_flix.agregar_usuarix(ana)
    chau_flix.agregar_usuarix(maria)
    chau_flix.agregar_usuarix(pedro)

    # Peliculas
    vengadores = Pelicula("Vengadores", [Genero.ACCION], 180)
    deadpool = Pelicula("Deadpool", [Genero.ACCION, Genero.COMEDIA], 120)
    shawshank = Pelicula("The Shawshank Redemption", [Genero.DRAMA], 142)
    lalaland = Pelicula("La La Land", [Genero.COMEDIA, Genero.DRAMA], 128)
    elpadrino = Pelicula("El Padrino", [Genero.DRAMA], 175)

    chau_flix.agregar_pelicula(vengadores)
    chau_flix.agregar_pelicula(deadpool)
    chau_flix.agregar_pelicula(shawshank)
    chau_flix.agregar_pelicula(lalaland)
    chau_flix.agregar_pelicula(elpadrino)

    # Mirar peliculas: La operación mirar_pelicula() invoca a
actualizar_relaciones()
    chau_flix.mirar_pelicula(juana, vengadores)
    chau_flix.mirar_pelicula(juana, vengadores)
    chau_flix.mirar_pelicula(ana, deadpool)
    chau_flix.mirar_pelicula(ana, elpadrino)
    chau_flix.mirar_pelicula(maria, shawshank)
    chau_flix.mirar_pelicula(pedro, lalaland)
    chau_flix.mirar_pelicula(pedro, elpadrino)
    chau_flix.mirar_pelicula(pedro, elpadrino)

    # Relaciones directas
```

```
    print(f'Relación directa de Juana con Ana:
{chau_flix.tienen_relacion(juana, ana)}')    # True
    print(f'Relación directa de Juana con Maria:
{chau_flix.tienen_relacion(juana, maria)}')    # False
    print(f'Relación directa de Juana con Pedro:
{chau_flix.tienen_relacion(juana, pedro)}')    # False
    print(f'Relación directa de Ana con Maria:
{chau_flix.tienen_relacion(ana, maria)}')    # True
    print(f'Relación directa de Ana con Pedro:
{chau_flix.tienen_relacion(ana, pedro)}')    # False
    print(f'Relación directa de Maria con Pedro:
{chau_flix.tienen_relacion(maria, pedro)}')    # False

    # Relaciones indirectas
    print(f'Relación alguna de Juana con Ana:
{chau_flix.tienen_alguna_relacion(juana, ana)}')  # True
    print(f'Relación alguna de Juana con Maria:
{chau_flix.tienen_alguna_relacion(juana, maria)}')  # True
    print(f'Relación alguna de Juana con Pedro:
{chau_flix.tienen_alguna_relacion(juana, pedro)}')  # False
    print(f'Relación alguna de Ana con Maria:
{chau_flix.tienen_alguna_relacion(ana, maria)}')  # True
    print(f'Relación alguna de Ana con Pedro:
{chau_flix.tienen_alguna_relacion(ana, pedro)}')  # False
    print(f'Relación alguna de Maria con Pedro:
{chau_flix.tienen_alguna_relacion(maria, pedro)}')  # False
```