Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2021-2

25 de Noviembre de 2021
Actividad Sumativa

Actividad Sumativa 4

Estructuras Nodales I y II

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS4/`
- **Hora del *push*:** 16:40

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

Se acerca el fin de semestre y junto a ello el fin de año. Época en que se reúnen familiares y amigos, pero lamentablemente este año se cayó Facebook, WhatsApp e Instagram, dificultando las comunicaciones entre personas. Es por esto que, como especialista en el área, decides aplicar, no una, sino que hasta tres estructuras de datos para crear la nueva y mejorada red social DCCelebrity, en la cual cada usuario maneja sus distintas amistades y puede crear eventos, por ejemplo, como amigos secretos. Además, DCCelebrity es muy popular entre los famosos, así que puedes saber a cuántas amistades de distancia estás de alguna DCCelebridad!

Figura 1: Logo DCCelebrity

Archivos

- AS4
  - usuario.py **No debes modificarlo**:Contiene a la clase Usuario, las instancias de esta clase van a formar los nodos que se usaran en las estructuras de datos
  - cargar_usuarios.py **No debes modificarlo**:Archivo que carga un diccionario con los usuarios
  - dccelebrity.py **No debes modificarlo**: Contiene a la clase DCCelebrity que representa a toda la red social, y donde se instancian los distintos tipos de estructuras
  - grafo.py **Debes modificarlo**:Contiene a la clase NodoGrafo, que se usa para el **grafo no dirigido** para manejar las amistades de un usuario
  - arbol.py **Debes modificarlo**: Contiene a la clase ArbolBinario que representa un **árbol binario** para ordenar a las instancias de la clase NodoFama según su nivel de fama
  - lista_ligada.py **Debes modificarlo**: Define la clase NodoAmigoSecreto que representa los nodos de la **lista ligada** que se usa para el amigo secreto
  - main.py **No debes modificarlo**: Instancia DCCelebrity, se debe ejecutar para probar la actividad
  - parametros.py **No debes modificarlo**: Contiene los parámetros
  - data **No debes modificarlo**: Tiene los datos para probar el programa
    - data.json **No debes modificarlo**: contiene información de los usuarios junto a sus amistades y sus niveles de fama
    - regalos.csv **No debes modificarlo**: contiene información sobre regalos a entregar entre los usuarios

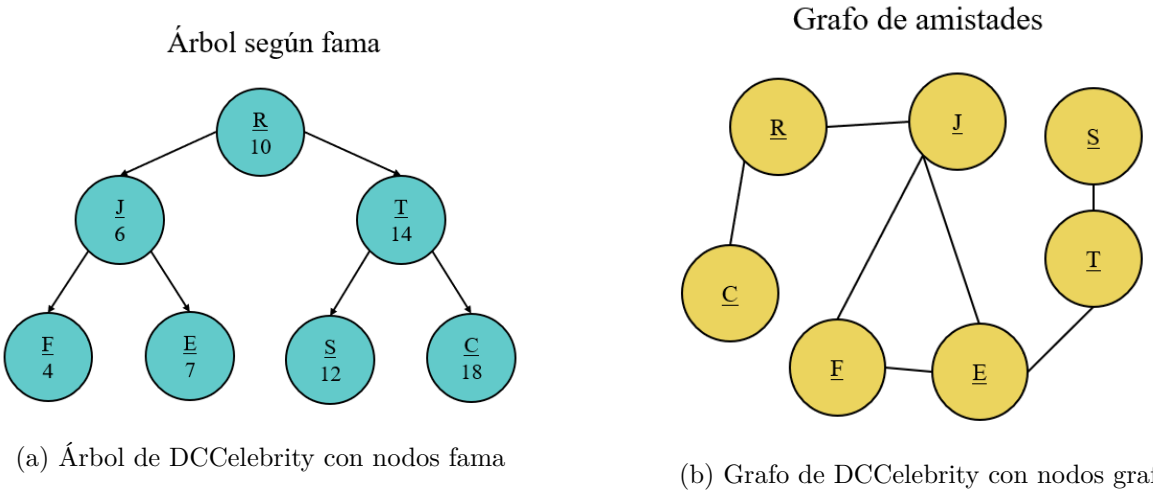
Flujo del Programa

DCCelebrity es una red social como la mayoría en la que cada usuario tiene un nivel de fama único que los distingue y que en caso de superar un umbral ¡será considerado como famoso! Como usuario de DCCelebrity puedes iniciar sesión, ver todos los usuarios de la red, ver y modificar mis amistades, pedir recomendaciones de amistades, buscar al famoso que esté más cerca e incluso organizar un amigo secreto. Tú como estudiante puedes usar uno de los nombres disponibles de la red social para ingresar como usuario y probar los distintos métodos que vas a ir implementando.

DCCelebrity reúne a sus usuarios en tres estructuras de datos principales:

- Un **árbol binario** con **nodos de fama**, en el cual cada nodo almacena un usuario y se organiza según su nivel de fama respectivo. Además, posee un hijo izquierdo (una instancia de `NodoFama`) que tiene un usuario con menor nivel de fama; y un hijo derecho (que también es un `NodoFama`) con un usuario de mayor nivel de fama. La idea de este árbol es que sea capaz de insertar y buscar nodos en base a un nivel de fama entregado. Este árbol se define en el archivo `arbol.py`.
- Un **grafo no dirigido** construido en base a **nodos grafo**, los cuales almacenan un usuario y se conectan mediante amistades que tienen entre ellos. En este grafo se espera poder modificar las amistades (agregar y eliminar), pero también realizar una búsqueda de recomendaciones de posibles amistades a formar. Los nodos de este grafo se representan con la clase `NodoGrafo` del archivo `grafo.py`.
- Una **lista ligada** construida con la clase `NodoAmigoSecreto` los cuales se conectan mediante al atributo siguiente de estos. En esta estructura se espera poder agregar mas usuarios a la lista manteniendo el orden, y que los usuarios puedan interactuar con su amigo secreto. Los nodos del amigo secreto están definidos en el archivo `lista_ligada.py`.

Para que entiendas la estructura del árbol y grafo, la siguiente imagen refleja como **cada tipo de nodo forman dos estructuras distintas**:



(a) Árbol de DCCelebrity con nodos fama

(b) Grafo de DCCelebrity con nodos grafo

En este ejemplo, cada círculo representa un nodo distinto de las distintas estructuras que corresponde a un usuario. En el caso del árbol binario, la posición del nodo nos ayuda a identificar cuál es su nivel de fama. Para este ejemplo todos los nodos a la izquierda de R tienen una fama menor a la suya (10). Recordar que cada nodo tiene a lo mas 2 nodos hijos, con uno con fama estrictamente menor y otro con nivel de fama estrictamente mayor.

Por el otro lado, el grafo de amistades representa las amistades entre usuarios. De este diagrama, uno puede concluir que R y J se tienen agregados como amigos, pero que R y S, no están guardados como amigos. De la misma manera, se puede identificar que R y E están a una distancia de 1 de ser amigos, dado que R y J son amigos y J y E son amigos.

Es importante recalcar que en ambas figuras, los nodos iguales representan al mismo usuario que es parte de distintas estructuras.

Por último, en el archivo `dccelebrity.py` la clase `DCCelebrity`, instancia todas las estructuras con el método `crear_red` y maneja la interacción entre el usuario y la red social a través de menús en consola.

Parte 1: Amigo secreto con lista ligada

Para calentar motores en esta actividad, debes saber que en DCCelebrity uno de los eventos más populares es el amigo secreto. En este juego a cada `Usuario` se le asigna un amigo a quien hacerle un regalo y dependiendo de qué tan bueno sea, existe la posibilidad de que se formen nuevas amistades! Este juego se logra a través de una **lista ligada**, donde cada `NodoAmigoSecreto` tiene un atributo `siguiente` que indica quién es su amigo secreto. Una particularidad de esta lista ligada es su *circularidad*: El último nodo está conectado al siguiente. Deberás completar el código del archivo `lista_ligada.py` para esta parte, teniendo en cuenta que algunas de las reglas aprendidas de **listas ligadas** funcionarán un poco distinto.

- `def __init__(self, usuario: Usuario, siguiente: Usuario=None):` No debes modificarlo
  Recibe el `Usuario` que guardará el `NodoAmigoSecreto` y opcionalmente un `NodoAmigoSecreto` que será el valor **siguiente**, es decir, que contiene al amigo secreto. Además, tiene un atributo `regalo_entregado` que parte en `False` y, eventualmente, se torna `True` cuando ha entregado el regalo a `self.siguiente`. **Como esta lista es circular, inicialmente el valor `self.siguiente` del primer nodo de la lista es `self` (él mismo).**

- `def insertar_amigo_secreto(self, nuevo_nodo: NodoAmigoSecreto, posicion: int, posicion_actual: int = 0):` **Debes modificarlo**
  Recibe un `NodoAmigoSecreto` a insertar y la `posicion` en la que debe agregarlo. Deberás recorrer la lista ligada desde el nodo inicial hasta llegar a esa posición y crear un `NodoAmigoSecreto` que contenga al `usuario` y asignarlo al atributo `siguiente` del último nodo visitado. ¡Procura emplear lo que has aprendido de listas ligadas para que esta mantenga su estructura!

- `def entregar_regalos(self):` **No debes modificarlo**
  Este método comienza el amigo secreto, entregando los regalos recursivamente.

Parte 2: Árbol binario

En esta parte deberás completar algunos métodos de `ArbolBinario` que se encuentran en `arbol.py` y que modelan el árbol binario en base al nivel de fama de cada `Usuario`. Para esto deberás completar la inserción y búsqueda que te permitirán construir el árbol e iniciar sesión con un usuario.

La clase `NodoFama` tiene los siguientes métodos que no debes modificar:

- `def __init__(self, usuario: Usuario, padre: NodoFama=None):` **No debes modificarlo**
  Inicializa un nodo de fama. Asigna al atributo `self.usuario` la instancia `usuario` que viene de argumento y el atributo `self.padre` el argumento opcional `padre`. Además, inicializa los valores de los atributos `hijo_izquierdo` e `hijo_derecho` como `None`.

Por su parte la clase `ArbolBinario` tiene los siguientes métodos de los cuales algunos debes modificar:

- `def __init__(self):` **No debes modificarlo**
  Inicializa el árbol binario. Asigna al atributo `raiz` el valor `None`.

- `def crear_arbol(self, nodos_fama: list):` **No debes modificarlo**
  Método que recibe una lista de `NodosFama` y la recorre para construir el árbol mediante el método `insertar_nodo`.

- `def insertar_nodo(self, nuevo_nodo: NodoFama, padre: NodoFama = None):` **Debes modificarlo**
  Método que recibe un nodo a agregar (`nuevo_nodo`) y un nodo `padre`. Debe insertar `nuevo_nodo` en el árbol a partir del atributo `fama` del usuario que contiene. El parámetro `padre` lo puedes usar para ir avanzando por los distintos nodos del árbol (como pivote). La implementación de este método puede ser iterativa o recursiva según estimes conveniente, pero debes tener en consideración que debes ir construyendo el árbol a partir de la raíz, donde cada `hijo_izquierdo` tiene menor fama que su padre y cada `hijo_derecho` tiene mayor fama que su padre.

  Como recomendación, revisa la imagen de ejemplo del árbol binario para entender cómo se construyó.

- `def buscar_nodo(self, fama: int, padre: NodoFama=None): -> NodoFama` **Debes modificarlo**
  Recibe un `int` que representa la fama de un `Usuario`. Debes usar ese valor para encontrar el `NodoFama` correspondiente en el árbol y retornarlo. Recuerda que tu búsqueda parte de la raíz y en caso de que no encuentre un usuario con ese nivel de fama, el método debe retornar `None`. Nuevamente, la implementación de este método puede ser iterativa o recursiva según estimes conveniente.

Parte 3: Grafo de amistades

En esta parte deberás completar métodos de la clase `NodoGrafo` ubicada en `grafo.py` y funciones del mismo archivo, con el fin de que los usuarios puedan tener acceso a las funcionalidades que ofrece la aplicación.

La clase `NodoGrafo` posee los siguientes métodos:

- `def __init__(self, usuario: Usuario):` **No debes modificarlo**
  Recibe una instancia de usuario y la almacena como atributo, además inicializa su atributo `self.amistades` como `None`.
- `def formar_amistad(self, nueva_amistad: NodoGrafo):` **Debes modificarlo**
  Recibe una instancia de `NodoGrafo`, con la cual debes agregar mutuamente los nodos a la lista de amistades de quienes estén formando esta amistad, solamente si esta **no existe** previamente. Puedes agregar los `print` necesarios para cada caso.

  Hint: Recuerda que es un grafo no dirigido.
- `def eliminar_amistad(self, ex_amistad: NodoGrafo):` **Debes modificarlo**
  Recibe una instancia de `NodoGrafo` y debes eliminarla mutuamente de las listas de amistades del nodo original y del nodo a eliminar, solamente si la amistad **ya existe**. Puedes agregar los `print` necesarios para cada caso. *Hint*: Recuerda que es un grafo no dirigido.

Luego de esto, es necesario habilitar ciertas búsquedas en el grafo. Para ello es necesario que completes la siguiente función ubicada en `grafo.py`.

- `def recomendar_amistades(self, nodo_inicial: NodoGrafo, profundidad: int) -> list:`
  Debes modificarlo
  Recibe un nodo de inicio `nodo_inicial` y una profundidad máxima `profundidad`; retorna una lista con todos los `NodoGrafo` que se encuentren a una profundidad igual o menor a la máxima. Es importante que solo se recomienden `NodoGrafo` que no sean parte de las amistades ya existentes.

  Como sugerencia revisa el ejemplo de grafo de amistades y supone que partimos desde el nodo R. Si la profundidad es 1 debemos buscar en las amistades de mis amistades, en este caso serían únicamente F y E, si es profundidad 2 avanzamos un nivel más, es decir, serían F, E y T, mientras que si es profundidad 3 serían F, E, T y S.

Bonus

Como se señaló en la introducción, una de las grandes cualidades de DDCelebrity es que un `Usuario` puede saber a cuánta distancia en amistades está de un Usuario famoso. Para cumplir con la esencia de la red social, necesitamos que nos ayudes a implementar la siguiente función ubicada en `grafo.py`:

- `def busqueda_famoso(nodo_inicial: NodoGrafo, visitados=None, distancia_max=80)-> tuple:`
  Debes modificarlo
  Recibe un `NodoGrafo` a partir del cual deberás usar un método de búsqueda a través de las amistades para encontrar el `Usuario` famoso más cercano al que se pueda llegar.

  Para ello puedes usar la *property* `es_famoso` de la clase `Usuario`, la cual compara su nivel de fama con el parámetro `COTA_FAMA`, retornando `True` en caso de superar ese umbral y `False` en otro caso. Deberás retornar una tupla de la forma `(distancia, nodo_famoso)` donde `distancia` es la distancia en amistades que hay entre el nodo inicial y el nodo con `Usuario` famoso.

  En caso de no encontrar un famoso debes retornar la misma tupla, pero con la distancia máxima por defecto y `None` en vez de un nodo. Mientras que si hay dos o más `Usuarios` famosos con la misma distancia, puedes retornar el primero que encuentres.

Para probar tu implementación del bonus, debes descomentar las líneas **120-126** en `dccelebrity.py` e iniciar sesión con un usuario que no sea famoso (puedes revisar el archivo de `datos.csv` para ver qué usuarios no son famosos).

Notas

- La recolección de la actividad se hará en la rama principal (`main`) de tu repositorio.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo.
- Siéntete libre de agregar nuevos `print` en cualquier lugar de tu código para encontrar errores. Es una herramienta muy útil.
- Recuerda especificar tus dudas en el Discord, para que podamos ayudar y encontrar las dudas más frecuentes.

Requerimientos

- (0.50 pts) Clase `NodoAmigoSecreto`
  - (0.50 pt) Completar `def insertar_amigo_secreto()`.
- (3.00 pts) Clase `ArbolBinario`
  - (1.50 pt) Completar `def insertar_nodo()`.
  - (1.50 pt) Completar `def buscar_nodo()`.
- (2.50 pts) Clase `NodoGrafo`
  - (0.50 pts) Completar `def formar_amistad()`.
  - (0.50 pts) Completar `def eliminar_amistad()`.
  - (1.50 pts) Completar `def recomendar_amistades()`.
- (1.00 pts) Bonus
  - (1.00 pts) Completar `def busqueda_famoso()`. (no hay puntaje parcial).