Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2019-2

10 de octubre de 2019
Actividad Formativa

Actividad 07

Árboles y Listas Ligadas

DCCivil War

Figura 1

Después de muchos incidentes públicos del cuerpo de ayudantes de Programación Avanzada, los grandes lideres universitarios piensan que hay que restringirlos para que sigan haciendo el bien pero sin dejar daños colaterales cada vez que hagan lo suyo. Esto ha creado un increíble quiebre dentro del equipo de ayudantes, en dos grupos muy marcados.

Por un lado esta el equipo de Docencia, liderado por ~~*Iron Man*~~ el jefe de Docencia, quien le encuentra la razón a la universidad y cree en que ellos tienen que supervisar las acciones de los ayudantes. Por el otro lado, los Tareos, liderados por ~~*Capitana America*~~ la jefa de Tareas, esta convecida de que el cuerpo de ayudantes no debería ser un títere de la universidad.

Para evitar una real guerra civil se decide solucionar el conflicto mediante una simulación para calcular cuál de los dos grupos es más ~~*poderoso*~~ eficiente y que éste sea el que tome la decisión final.

¡Está en tus manos el evitar una verdadera catástrofe mediante tus nuevos ~~*superpoderes*~~ conocimientos de nodos y árboles!

Cuerpo de Ayudantes

El cuerpo de ayudantes esta compuesto por un ayudante Coordinador, dos ~~*Capos*~~ ayudantes Jefes, uno del área de Tareas y otro del área de Docencia y finalmente por varios ayudantes de rango menor: Mentor o Novato. Cada uno de estos últimos pertenece a uno de dos grupos, Tareos (área Tareas) o Docencios (área Docencia).

Debido a la estructura jerárquica de estos roles y al hecho de que cada integrante del cuerpo de ayudantes solo tiene un jefe directo, podemos representar el cuerpo de ayudantes como un árbol.

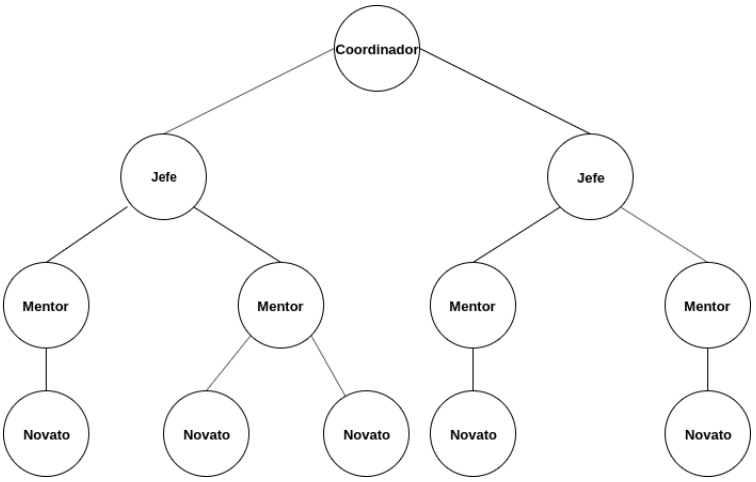

Figura 2: Árbol del Cuerpo de Ayudantes

Atributos de las Entidades

Todas las entidades a modelar son de la clase `Ayudante`, y tendrán los siguientes atributos:

- `self.nombre`: es un `str` que indica el nombre del ayudante.
- `self.rango`: es un `str` que indica el rango del ayudante. Puede ser `"Coordinador"`, `"Jefe"`, `"Mentor"` o `"Novato"`.
- `self.tipo`: es un `str` que indica el tipo de ayudante que es. Puede ser `"Tareo"` o `"Docencio"`. Si es coordinador, el valor es `"Coordinador"`.
- `self.afinidad`: es un `int` que representa la afinidad del ayudante.
- `self.eficiencia`: es un `int` que representa la eficiencia de un ayudante.
- `self.subordinados`: es una `list` con todos los ayudantes que tiene a cargo. Solo contiene instancias de la clase `Ayudante`.

Armar el cuerpo de Ayudantes

El proceso para armar el cuerpo de ayudantes es el siguiente:

1. Se crea al Ayudante Coordinador.
2. Se agrega a cada Jefe como subordinado del Coordinador.

3. Se agregan de a uno a los Ayudantes Mentores. Estos serán subordinados del Jefe de Tareas si son del grupo Tareos, o del Jefe de Docencia si son del grupo Docencios.
4. Se agregan de a uno a los Ayudantes Novatos. Cada Ayudante Novato será subordinado del Mentor con el valor de afinidad más cercano al suyo, siempre y cuanto sea de su mismo tipo (de la misma área).

Para esto se te pide implementar el siguiente método dentro de la clase `Ayudante`:

- `def agregar_ayudante(self, ayudante):` recibe un ayudante y decide si agregarlo como subordinado directo o entregarlo al subordinado más adecuado y que éste lo ingrese.

Consultas

Una vez terminado tu árbol, deberás dejar en claro de una vez por todas qué grupo es más eficiente: ¿Docencios o Tareos?. Antes de responder esta pregunta debes corroborar que tu árbol fue construído correctamente, para esto deberás diseñar la siguiente función:

- `def imprimir_grupo(ayudante):` recibe un ayudante e imprime su nombre y el de todos sus subordinados **en orden jerárquico**. Si `ayudante` es de tipo Coordinador, entonces hay que imprimir su nombre, el de los Jefes, el de los Mentores y el de los Novatos, en ese orden. En cambio si `ayudante` fuera un Mentor, solo habría que imprimir su nombre y el de los Novatos asociados a él, en ese orden.

Ahora que sabes que tu árbol fue construido correctamente puedes hacer uso de la siguiente fórmula para definir que grupo es el mejor:

$$E_{grupo} = 4 \times E_{coordinador} + 3 \times E_{jefe} + 2 \times E_{mentores} + E_{novatos}$$

- E_{grupo}: Eficiencia total del grupo.
- $E_{coordinador}$: Valor `self.eficiencia` del coordinador.
- E_{jefe}: Valor `self.eficiencia` del jefe (de Docencia o de Tareas).
- $E_{mentores}$: Suma de la eficiencia de todos los mentores que pertenecen al grupo.
- $E_{novatos}$: Suma de la eficiencia de todos los novatos que pertenecen al grupo.

Para resolver esto tendrás que diseñar las siguientes consultas sobre el sistema:

- `def grupo_mayor_eficiencia(coordinador):` recibe como único argumento a un ayudante coordinador (nodo raíz del arbol). Debe imprimir al grupo más eficiente (Docencios o Tareos) junto con el valor de su eficiencia grupal. Para esto puedes utilizar la función `imprimir_grupo`.

Estas consultas deben ser implementadas como funciones dentro del `main` (no como métodos en una clase).

Un ejemplo de salida de estas consultas puede ser el siguiente.

```
>> imprimir_grupo(ayudante)
- Coordinador: Enzo Tamburini
- Jefe: Dante Pinto
- Mentores: Juan Aguillón, Mario Reinike, Benjamín Martínez, Roberto Negrin, Ian Basly ...
- Novatos: Josefina Fernández, Maximiliano Narea, Pablo Araneda, Nicolas Orellana, ...
```

```
>> grupo_mayor_eficiencia(coordinador)

Grupo más eficiente: Tareos
Eficencia total grupal: 128
```

Ten en cuenta que esta salida es **sólo un ejemplo**. Puedes mostrar la información en otro formato siempre y cuando se mantenga el orden pedido (Coordinador, Jefe, Mentores y luego Novatos).

Archivos

Junto con el enunciado de esta actividad, se adjuntan los archivos `leer_archivos.py`, `ayudantes.csv` y `main.py`.

El archivo `leer_archivos.py` tiene el método `read_file()`, el cual se encarga de cargar a los ayudantes de `ayudantes.csv` para que puedas utilizarlos. No es necesario que revisen en detalles estos archivos.

El archivo `main.py` **es el único archivo que debes editar**. Este contiene la clase `Ayudante`, donde deberás completar el `__init__` y el método `ingresar_ayudante(self, ayudante)`. También contiene las consultas `imprimir_grupo(ayudante)` y `grupo_mayor_eficiencia(sistema)` que debes implementar, y el método (ya implementado) `instanciar_cuerpo_ayudantes(ayudantes)` que se encarga de instanciar a los ayudantes recibidos de `ayudantes.csv`.

Notas

- Debe utilizar la estructura recursiva del árbol para implementar los métodos y funciones solicitados.

Requerimientos

- (2.00 pts) `class Ayudante`
  - (0.50 pts) La clase tiene los atributos correspondientes.
  - (1.50 pts) La función `ingresar_ayudantes` es implementada correctamente.
- (4.00 pts) Consultas:
  - (2.00 pts) Consulta `calcular_eficiencia` implementada correctamente.
  - (2.00 pts) Consulta `imprimir_mas_eficiente` implementada correctamente.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC07/`
- **Hora del *push*:** 16:30

Auto-evaluación

Como esta corresponde a una actividad formativa, te extendemos la instancia de responder, después de terminada la actividad, una auto-evaluación de tu desempeño. Esta se habilitará a las **16:50 de jueves 10 de octubre** y tendrás plazo para responderla hasta las **23:59 del día siguiente**. Puedes acceder al formulario mediante el siguiente enlace:

`https://forms.gle/GsytdktgCHsnRNL58`

El asistir, realizar la actividad y responder la auto-evaluación otorgará como bonificación al alumno 2 décimas para sumar en su peor actividad sumativa del semestre.