Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2023-1

18 de Mayo de 2023
Actividad

Actividad 3

Estructuras Nodales

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/A3/`
- **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

Al darse cuenta de la popularidad de su deporte favorito, *DCCaptura la tortuga*, **Lily416** decide crear una liga profesional y, para ello, comienza a armar diferentes equipos con los jugadores que tiene disponibles.

Para conseguir este objetivo, **Lily416** estudió las capacidades e interacciones entre sus jugadores y te pide a ti que la ayudes a representar esas interacciones para formar los equipos de la mejor manera.

Flujo del programa

El programa consiste en un grafo dirigido que contiene la información individual de cada uno de los jugadores, junto con las interacciones entre ellos y permite hacerle consultas sobre los datos que contiene. Decimos que un jugador es amigo de otro, si tiene una arista hacia él, decimos que dos jugadores son conocidos si es que existe un camino entre ellos y, finalmente, decimos que dos jugadores son compañeros si pertenecen al mismo equipo (sin importar si están conectados o no).

Dentro del grafo, cada jugador corresponde a un nodo, que guarda su nombre y velocidad, y existirá una arista entre dos jugadores si es que el jugador A ve al jugador B como un amigo. Cómo es un grafo dirigido, que A vea a B como amigo, no implica que B también vea a A como amigo... la amistad no es recíproca.

Archivos

Archivos de código

En el directorio de la actividad encontrarás los siguientes archivos con código:

- Modificar `equipo.py`: Aquí encontrarás la definición básica de las clases que debes implementar y completar, junto con un pequeño *main* que prueba las consultas.
- No modificar `test.py`: Este archivo ejecuta los *tests* asociados a las distintas consultas.

Entidades

No modificar `class Jugador`:
Clase que representa un nodo del grafo. Cada jugador guarda su información personal y tiene los siguientes métodos:

- No modificar `def __init__(self, nombre: str, velocidad: int) -> None`:
  Inicializador de la clase. Asigna los siguientes atributos:

`self.nombre`	Un `str` que representa el nombre del jugador.
`self.velocidad`	Un `int` que representa la velocidad del jugador para atrapar la tortuga.

- No modificar `def __repr__(self) -> None`:
  Método encargado del `repr` de la clase.

Modificar `class Equipo`:
Clase que representa al grafo con los datos de los jugadores. Posee los siguientes métodos:

- No modificar `def __init__(self) -> None`:
  Inicializador de la clase. Define los siguientes atributos:

`self.jugadores`	Un `dict` que asocia el `id_jugador` de los jugadores con la instancia `Jugador` correspondiente.
`self.dict_adyacencia`	Un `defaultdict` que asocia cada `id_jugador` con un `set` formado por los `id_jugador` de sus vecinos. En este caso, los vecinos de un jugador A serían todos los jugadores dentro del equipo que A considera amigos.

- Modificar `def agregar_jugador(self, id_jugador: int, jugador: Jugador) -> bool`:
  Método encargado de agregar un jugador al grafo. Recibe el `id_jugador` y la instancia del jugador e intenta agregarlo al grafo.
  - Si ya existe un jugador con ese `id_jugador` en el equipo, no lo agrega y retorna `False`.
  - Si no existe un jugador con ese `id_jugador`, lo añade al diccionario de jugadores y retorna `True`.
- Modificar `def agregar_vecinos(self, id_jugador: int, vecinos: list[int]) -> int`:
  Método encargado de agregar los vecinos de un jugador al grafo. Recibe el `id_jugador` y una lista con los `id_jugador` de sus vecinos e intenta agregar todos los vecinos de la lista al diccionario de adyacencia.
  - En caso que el jugador no se encuentra en el diccionario de adyacencia, no se agrega ningún vecino y el método retorna -1.

- Si el jugador se encuentra en el diccionario de adyacencia, se agregan los vecinos y retorna la cantidad de nuevos vecinos agregados. En este ultimo caso, si algún `id_jugador` de la lista `vecinos` ya existe previamente entre los vecinos del jugador, estos `id_jugador` **no debe ser añadido nuevamente**, solamente se añaden los nuevos vecinos que no estaban previamente.

- Modificar `def mejor_amigo(self, id_jugador: int) -> Jugador:`
  Un jugador se dice "mejor amigo" de otro si: es su vecino directo y además la diferencia entre sus velocidades es la menor entre todos los vecinos directos. El método recibe el `id_jugador` de un jugador, busca entre sus vecinos a su mejor amigo y lo retorna.

  Puedes asumir que el `id_jugador` está en el equipo y que nunca existirá empate en la diferencia de velocidades. Si el jugador no tiene vecinos, el método debe retornar `None`.

- Modificar `def peor_compañero(self, id_jugador: int) -> Jugador:`
  Un jugador será el "peor compañero" de otro si es que la diferencia entre sus velocidad es la mayor en **todo el equipo**. Este método recibe el `id_jugador` de un jugador y retorna a su peor compañero en el equipo.

  Puedes asumir que el `id_jugador` está en el equipo y que nunca existirá empate en la diferencia de velocidades. Si el jugador es el único miembro del equipo, el método debe retornar `None`.

- Modificar `def peor_conocido(self, id_jugador: int) -> Jugador:`
  Decimos que dos jugadores están conectados si existe un camino entre ellos en el grafo. Un jugador será el "peor conocido" de otro si es que la diferencia entre sus velocidad es la mayor entre todos los jugadores conectados a este. Este método recibe el `id_jugador` de un jugador y retorna a su peor conocido en el equipo.

  Puedes asumir que el `id_jugador` está en el equipo y que nunca existirá empate en la diferencia de velocidades. Si el jugador no tiene vecinos, el método debe retornar `None`.

- Modificar `def distancia(self, id_jugador_1: int, id_jugador_2: int) -> int:`
  Definimos la distancia entre dos jugadores como el tamaño del camino más corto entre ellos y decimos que la distancia entre un nodo y sí mismo es 0. Este método recibe los *ids* de dos jugadores y calcula la distancia entre estos. Si no existe camino entre los dos nodos, el método debe retornar -1.

  Puedes asumir que ambos `id_jugador` están en el equipo.

Notas

- Recuerda que la ubicación de tu entrega es en tu **repositorio personal**. Verifica que no estés trabajando en el **Syllabus**.
- Recuerda que esta evaluación presenta corrección **automatizada**. Si entregas un código que se cae al momento de correr los *tests*, será evaluado con 0 puntos.
- Se recomienda completar la actividad en el orden del enunciado.
- Si aparece un error inesperado, ¡léelo! Intenta interpretarlo y/o buscarlo en Google.