Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2018-2

16 de agosto de 2018
Evaluada

Actividad 01

Estructuras de Datos: *built-ins*

Introducción

¡Hay subasta en el DCC (Departamento del Carrete y Comida) y todos están más que invitados! Como el odiado Fernando (el malvado líder del equipo Rocket) quiere que nada ni nadie se quede sin su pokémon esclavo, ha maquinado un plan para que se venda absolutamente todo y aprovechar de experimentar diferentes formas de vender y ver qué pasa. Se te ha encomendado la labor de realizar un programa que lea una serie de archivos y a partir de estos lograr hacer consultas para un posterior estudio de la subasta.

La subasta

Esta subasta se considera libre del mundo capitalista ya que no involucra traspasos de dinero, sino que sólo importa el orden en que se solicitan los pokémones.

Existen varias solicitudes asociadas a cada pokémon, hechas por distintos entrenadores. Al subastar un pokémon, las solicitudes asociadas a éste se van leyendo una a una. Cada pokémon tiene una cantidad máxima de solicitudes que, cuando es alcanzada, el último entrenador es el que gana el pokémon. Por ejemplo, si un pokémon tiene 30 solicitudes pero su cantidad de solicitudes máxima es 2, el entrenador asociado a la segunda solicitud leída se gana el pokémon independiente de las otras 28 solicitudes restantes.

Además, esta subasta tiene dos modalidades para ir leyendo las solicitudes. Estas son en orden de llegada o bien en orden inverso de llegada.

Archivos

Junto al enunciado se subieron varios archivos que se utilizarán en esta actividad. A continuación se describirán brevemente sus contenidos.

- `main.py`: En este archivo se tiene el esqueleto del programa que debes realizar. ¡Procura editar sólo las cosas necesarias!
- `entrenadores.txt`: En este archivo se tiene la información de los entrenadores que participarán en la subasta. Cada línea de este archivo tiene el formato: `id_entrenador;nombre;apellido`.
- `pokemones.txt`: En este archivo se tiene la información de los distintos pokémones que serán subastados. Cada línea de este archivo tiene el formato: `id_pokemon;nombre;tipo;max_solicitudes`. **Pueden existir varias instancias de un mismo pokémon**, sin embargo, el `id` de cada uno será único.

- `solicitudes.txt`: En este archivo se tienen las distintas solicitudes de los entrenadores sobre los pokémones. Cada línea de este archivo tiene el formato: `id_entrenador;id_pokemon`.
- `tests.py`: Este módulo contiene una serie de pruebas para que vean si sus funciones de consultas funcionan correctamente.
- `results_1.txt`: Contiene las respuestas correctas a los llamados de `tests.py` cuando se selecciona el primero de los modos.
- `results_2.txt`: Contiene las respuestas correctas a los llamados de `tests.py` cuando se selecciona el segundo de los modos.

Programa

En el módulo `main.py` entregado ya están creadas las entidades `Entrenador`, `Pokemon` y `Solicitud`, siendo modeladas con `namedtuple`. Es por esto que **no debes crearlas**. Además, note que tanto el `id` de un entrenador como el `id` de los pokémones son datos **únicos** y no son atributos ni de los entrenadores ni de los pokémones.

Para lograr satisfacer al odiado Fernando, debes realizar lo pedido a continuación para asegurar el correcto funcionamiento del programa.

Poblar el sistema

Es necesario cargar el contenido de los distintos archivos al programa. Para lograr esto debes completar las funciones `cargar_entrenadores`, `cargar_pokemones` y `cargar_solicitudes` como corresponde. El odiado Fernando desea que la información básica de cada entidad se guarde de manera eficiente por lo que ha exigido que se guarden como *named tuples*, a partir de lo dado en el archivo `main.py`.

- `cargar_entrenadores(ruta_archivo)`: Esta función recibe la ruta al archivo `entrenadores.txt`. Se debe cargar el contenido del archivo guardando la información de cada línea de manera que, luego de cargar todos los entrenadores, se pueda obtener cada uno de manera eficiente. Esta estructura debe ser retornada.
- `cargar_pokemones(ruta_archivo)`: Esta función recibe la ruta al archivo `pokemones.txt`. Se debe cargar el contenido del archivo guardando la información de cada línea de manera que, luego de cargar todos los pokémones, se pueda obtener cada uno de manera eficiente. Esta estructura debe ser retornada.
- `cargar_solicitudes(ruta_archivo)`: Esta función recibe la ruta al archivo `solicitudes.txt`. Esta función debe cargar el contenido del archivo guardando la información de cada línea en una estructura de datos adecuada, de forma que sea rápido obtener las solicitudes asociadas al id de un pokémon. Recuerde que existen varias solicitudes por pokémon. Además, la forma de almacenar los conjuntos de solicitudes debe ser eficiente para ambas modalidades de la subasta. Por último, su estructura debe estar preparada para consultas en las que no exista un pokémon solicitado. Esta estructura debe ser retornada

Note que en el archivo `main.py` ya se hacen los llamados a las funciones, por lo que **no debes hacerlo**.

Lógica del sistema

Como bien se dijo, existen dos modalidades de subasta. El modo "1" es en orden de llegada, es decir, se lee de la primera solicitud a la última arribada. Por otro lado, el modo "2" es el inverso, es decir, se lee desde la última solicitud llegada a la primera.

Para representar esto, debes completar la función `sistema` entregada en el archivo `main.py`. Esta función recibe un modo y las estructuras creadas anteriormente que contienen a los entrenadores, pokémones y solicitudes.

El sistema debe subastar a todos los pokémones existentes. Por lo tanto, se espera que para cada pokémon, según el modo escogido por el usuario, se vayan extrayendo las solicitudes hasta el máximo permitido por dicho pokémon. El entrenador asociado a la última solicitud extraída es el que se gana al pokémon. **Puedes asumir que siempre habrán solicitudes suficientes para cada pokémon**, es decir, nunca el número de solicitudes de un pokémon va a ser inferior al número máximo de solicitudes permitidas por el mismo.

Es posible que un entrenador haya ganado dos veces el mismo pokémon, pues pudo haber ganado dos instancias distintas de este. No obstante, al odiado Fernando no le interesan las repeticiones. Por lo tanto, para cada conjunto de pokémones ganados por cada entrenador, debes evitar que hayan pokémones repetidos.

La estructura en que se almacenan los pokémones ganados por cada entrenador debe estar preparada para el caso en que se consulte por los pokémones ganados por un entrenador no existente. Esta estructura debe ser retornada por la función.

Para facilitar la interacción con los usuarios del programa, en el archivo `main.py` se te entregó un menú simple en que se realiza el llamado a la función. Por lo que **no debes hacerlo**.

Consultas

El odiado Fernando te solicitó completar algunas funciones que le agreguen utilidad al programa que tú estás haciendo. A continuación se entrega una pequeña descripción de cada una.

- `pokemones_por_entrenador(id_entrenador, resultado_simulacion)`: Esta función recibe el `id` de un entrenador y debe retornar una lista de los pokémones que ese entrenador ganó en la subasta.
- `mismos_pokemones(id_entrenador1, id_entrenador2, resultado_simulacion)`: Esta función recibe 2 `id`s y debe retornar todos los pokémones que comparten ambos entrenadores, sin repetirlos. Finalmente, este conjunto debe ser retornado como una lista.
- `diferentes_pokemones(id_entrenador1, id_entrenador2, resultado_simulacion)`: Esta función recibe dos `id`s y debe retornar todos los pokémones que ganó el entrenador1 y que no ganó el entrenador2, sin repetirlos. Finalmente, este conjunto debe ser retornado como una lista.

Si deseas comprobar que tus consultas están funcionando de manera correcta, puedes ejecutar el módulo `tests.py`. Sin embargo, esto no garantiza que tengas el puntaje completo en cada uno de los requerimientos, dado que no sólo importa el resultado final, sino que también el cómo lo obtuviste.

Importante

En esta actividad está **absolutamente prohibido trabajar con clases personalizadas, con el uso de excepciones y con variables globales**. Además, deben utilizar las *named tuples* entregadas para cargar la información de los archivos.

No sólo se espera que usen las estructuras de datos adecuadas para guardar la información, sino también que las utilicen de manera correcta.

Por último, deben completar de manera correcta el módulo `main.py`, ya que se utilizará este módulo para corregir su actividad.

Notas

- Recuerde lo visto en los contenidos y en ayudantía. Específicamente las estructuras disponibles para guardar información: `namedtuple`, `deque`, `dict`, `defaultdict` y `set`. Piense bien cuál necesita en cada caso.
- Si tiene problemas para abrir los archivos, considere cambiar el *encoding* (opciones: `'utf-8'` y `'latin-1'`).
- Dos instancias de `namedtuple`s se consideran iguales si sus atributos lo son. Un ejemplo sería si se tiene una entidad `('Huesped', 'nombre apellido')` y se tienen dos instancias con nombre `Fernando` y apellido `Pieressa`. Si se realizara `print(instancia1 == instancia2)`, se imprimiría `True`.

Requerimientos

- (2.40 pts) Poblar el sistema
  - (0.60 pts) `cargar_entrenadores`.
    - (0.30 pts) Carga de manera correcta todos los entrenadores del archivo.
    - (0.30 pts) Retorna la estructura deseada.
  - (0.60 pts) `cargar_pokemones`.
    - (0.30 pts) Carga de manera correcta todos los pokémones del archivo.
    - (0.30 pts) Retorna la estructura deseada.
  - (1.2 pts) `cargar_solicitudes`.
    - (0.30 pts) Carga de manera correcta todas las solicitudes del archivo.
    - (0.90 pts) Retorna la estructura deseada.
- (2.00 pts) Lógica del sistema
  - (1.20 pts) Funcionan correctamente ambos modos de subasta.
    - (1.00 pts) El resultado es el correcto.
    - (0.20 pts) Cada modo extrae de manera correcta las solicitudes.
  - (0.20 pts) Se evita de manera correcta tener pokémones repetidos.
  - (0.60 pts) La función `sistema` retorna la estructura deseada.
- (1.60 pts) Consultas
  - (0.20 pts) La función `pokemones_por_entrenador` retorna lo deseado en el formato correcto.
  - (0.70 pts) La función `mismos_tipos` retorna lo deseado en el formato correcto.
  - (0.70 pts) La función `diferentes_tipos` retorna lo deseado en el formato correcto.

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AC01/`
- **Hora límite del último *push*:** 16:40