Examen final 2020-10-03

95.14/75.40 - Algoritmos y Programación I - Curso Essaya

Objetivo

Implementar la clase `AeroDB`, una base de datos de aeropuertos y rutas aéreas de todo el mundo.

Se dispone de los siguientes archivos:

- `aerodb.py`: implementación (inicialmente vacía) de la clase `AeroDB`
- `pruebas.py`: Pruebas automáticas
- `aeropuertos.csv`: CSV con datos de aeropuertos de todo el mundo
- `rutas.csv`: CSV con datos de rutas aéreas

El archivo `pruebas.py` efectúa una serie de pruebas para verificar el correcto funcionamiento de la clase `AeroDB`.

La idea es agregar en `aerodb.py` todo el código necesario para que las pruebas automáticas pasen.

Hay 5 ejercicios. Es condición necesaria (pero no suficiente) para aprobar el examen que haya 3 ejercicios OK.

Salida del programa

Al ejecutar el programa (`python3 pruebas.py`), se ejecutan todas las pruebas, y se imprime el resultado de cada ejercicio (OK o FAIL), junto con la cantidad de ejercicios OK. Ejemplo:

```
$ python3 pruebas.py
ejercicio_1: OK
ejercicio_2: OK
ejercicio_3: OK

Traceback (most recent call last):
File "pruebas.py", line 237, in main
    ejercicio()
File "pruebas.py", line 148, in ejercicio_4
    assert ok
AssertionError

ejercicio_4: FAIL
ejercicio_5: OK
Cantidad de ejercicios OK: 4
```

Recordar: es condición necesaria[1] (pero no suficiente[2]) para aprobar el examen que haya al menos 3 ejercicios OK.

[1] Es posible que un ejercicio sea considerado ”bien” aun cuando la prueba informa ”FAIL”; por ejemplo si hay un error trivial en el código que se arreglaría haciendo un pequeño cambio.

[2] Es posible (pero poco probable) que un ejercicio sea considerado ”mal” aun cuando la prueba informa ”OK”; por ejemplo si hay errores conceptuales.

La clase AeroDB

La clase `AeroDB` modela una base de datos de aeropuertos y rutas aéreas.

- Un **aeropuerto** tiene:
  - Una **designación** única llamada "código IATA", formada por 3 letras. Ejemplo: "EZE".
  - Un **nombre**. Ejemplo: "Ministro Pistarini International Airport".
  - Una **ciudad**. Ejemplo: "Buenos Aires".
  - Un **país**. Ejemplo: "Argentina".
  - Una **ubicación**, indicada en forma de coordenadas geodésicas (**latitud** y **longitud**).

La designación del aeropuerto es única; es decir que no hay dos aeropuertos con la misma designación; y la misma se utiliza para identificar al aeropuerto.

- Una **ruta aérea** tiene:
  - Un **código de vuelo**. Ejemplo: "AR412".
  - Un aeropuerto de **origen** (indicado por su designación). Ejemplo: "EZE".
  - Un aeropuerto de **destino** (indicado por su designación). Ejemplo: "BRC".

La terna de `(código, origen, destino)` identifica unívocamente a la ruta. Ejemplo: La ruta `(AR412, EZE, BRC)` no puede estar duplicada; pero sí puede existir al mismo tiempo una ruta `(AR412, BRC, EZE)`.

Descripción de las pruebas

`ejercicio_1`: Funcionamiento básico

Prueba el funcionamiento básico de la clase `AeroDB`. Sin esta prueba funcionando probablemente no se pueda pasar ninguna de las otras pruebas.

Métodos a implementar:

- `__init__`: Crea una instancia de `AeroDB` con 0 aeropuertos y 0 rutas.
- `aeropuerto_agregar`: Recibe la designación, el nombre, la ciudad, el pais, y las coordenadas (latitud y longitud) de un aeropuerto, y lo agrega a la base de datos.
- `cantidad_aeropuertos`: Devuelve la cantidad de aeropuertos existentes en la base de datos.
- `aeropuerto_get_nombre`, `aeropuerto_get_ciudad`, `aeropuerto_get_pais`: Reciben la designación de un aeropuerto y devuelven respectivamente el nombre, la ciudad y el país del aeropuerto correspondiente.
- `aeropuerto_get_coords`: Recibe la designación de un aeropuerto y devuelve una tupla con la latitud y la longitud del mismo.
- `ruta_agregar`: Recibe un código de vuelo y las designaciones de los aeropuertos origen y destino. Agerga la ruta a la base de datos.
- `cantidad_rutas`: Devuelve la cantidad de rutas existentes en la base de datos.

`ejercicio_2`: Búsqueda de rutas

Prueba que podamos listar todas las rutas que salen de una ciudad y que llegan a una ciudad.

- `rutas_desde_ciudad`: Recibe el nombre de una ciudad, y devuelve la lista de las rutas cuyo aeropuerto de origen corresponde a la ciudad indicada.
- `rutas_hacia_ciudad`: Recibe el nombre de una ciudad, y devuelve la lista de las rutas cuyo aeropuerto de destino corresponde a la ciudad indicada.

Ambas funciones devuelven una lista de rutas, y el formato de cada ruta debe ser una tupla `(código, origen, destino)`.

ejercicio_3: Cargar CSV

Prueba que podamos cargar la base de datos completa a partir de dos archivos CSV (uno para los aeropuertos y otro para las rutas), y que podamos averiguar cuál es el aeropuerto con más rutas del mundo.

Funciones a implementar:

- `cargar`: Recibe las rutas de los archivos CSV correspondientes a los listados de aeropuertos y rutas. Devuelve una instancia de `AeroDB` conteniendo toda la información de los archivos.

  Nota: `cargar` es una función suelta, no es un método de `AeroDB`.

  El CSV de aeropuertos tiene el formato `designación|nombre|ciudad|país|latitud|longitud` (sin cabecera).

  El CSV de rutas tiene el formato `codigo|origen|destino` (sin cabecera).

- `aeropuerto_con_mas_rutas`: Devuelve una tupla `(designación, cantidad)`, donde `designación` es la designación del aeropuerto que sirve más rutas, y `cantidad` es la cantidad de rutas correspondientes a ese aeropuerto.

  Un aeropuerto sirve una ruta tanto si es el origen o destino de la misma. Dicho de otra manera, una ruta que va de A a B es una ruta de ambos aeropuertos A y B.

ejercicio_4: Ordenar aeropuertos

Funciones a implementar:

- `aeropuertos_ordenados_por_distancia`: Recibe una latitud y una longitud, y devuelve la lista de designaciones de todos los aeropuertos, ordenada según la distancia de cada aeropuerto al punto indicado (el aeropuerto más cercano primero).

  Nota: el cálculo de distancias con coordenadas geodésicas no es trivial; a los efectos de este ejercicio vamos a simplificar y calcularlas como si fueran coordenadas cartesianas en un plano. Si x es la latitud e y es la longitud:

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

ejercicio_5: Itinerario

Si queremos viajar de la ciudad A a la ciudad B, es posible que no exista una ruta `(*, A, B)` que cubra el viaje en un tramo. Sin embargo, podemos buscar un *itinerario*, formado por una secuencia de rutas que permitan viajar de A a B haciendo escalas en otras ciudades, por ejemplo `(*, A, C) -> (*, C, D) -> (*, D, B)`.

Funciones a implementar:

- `armar_itinerario`: Recibe los nombres de las ciudades de origen y destino, y devuelve una lista con las rutas que forman un itinerario posible entre dichas ciudades, de la forma `[(código_0, origen, destino_0), ..., (código_n, origen_n, destino)]`.

  Devuelve `None` si no hay ningún itinerario posible entre ambas ciudades.

  Si hay más de un itinerario posible, es indistinto cuál de ellos es devuelto.

Ayuda: una forma de implementar esta función es mediante el algoritmo de *backtracking*. Este algoritmo es recursivo. En cada paso necesitamos:

- una lista `R` de rutas que conforman un itinerario posible a partir de la ciudad origen
- un conjunto `V` de ciudades visitadas

Descripción del algoritmo:

Sea `C` la ciudad de destino de la última ruta de `R` (o la ciudad origen si el itinerario `R` está vacío). Esta es la ciudad en la que estamos ”parados” actualmente.

Agregamos `C` al conjunto de ciudades visitadas `V`.

Si `C` es la ciudad destino, `R` es un itinerario que resuelve el problema.

En caso contrario, para cada ruta `r` existente con origen `C`, y con un destino no visitado aun, llamamos recursivamente al algoritmo, con `R = R + [r]`.