README

Armando Ramírez González - 317158225
Cecilia Villatoro Ramos - 419002938

October 9, 2020

1 Definición del problema

Fuimos contratados por el aeropuerto de la Ciudad de México, para realizar un programa que devolviera el clima de las ciudades destino de 3 mil tickets diferentes, fueron entregados en dos arcivos *.csv*, uno de vuelos nacionales, llamado *dataset1* y uno de vuelos internacionales llamado *dataset2*. No es necesaria la interacción con el usuario, sólo la impresión de los climas.

2 Análisis del problema

Se necesita una maera eficaz de leer los 3 mil datos distintos, guardarlos y poder accedar a ellos de manera rápida y directa. De la misma manera, es necesaria una forma de emparejar el clima con su ciudad correspondiente.

Para devolver el clima, se hizo uso de el API de OpenWeatherMap. Este API solo permite hasta 60 peticiones por minuto, así que se necesita una manera de dosificar las peticiones y medir el tiempo.

Como no es necesaria la interacción con el usuario, se necesita una manera de informar los climas de todas las ciudades sin distinción del ticket en el que aparezcan. Es decir, sin emparejar ciudad origen con destino. Por lo tanto, una opción es recolectar todas las ciudades diferentes que aparezcan en los tickets y emparejarlas con su clima, así se evitan hacer dos llamadas para una misma ciudad.

Al ver los *dataset* recibidos, se observan que *dataset2* incluye datos innecesarios para resolver el problema, como los horarios de salida y llegada. Además, se nota que algunos tickes de *dataset1* incluyen un origen diferente a la Ciudad de México, iguamente se tomaron esas ciudades en cuenta para devolver el clima. Para los vuelos nacionales las ciudades están en código IATA, pero como el API escogida no usa esos datos, se usan las coordenadas que el *dataset* incluye para cada ciudad.

Al hacer peticiones, hay que tomar en cuenta varios casos. Si la petición es exitosa, se manipulan los datos recibidos para devolver sólo lo necesario. Si la petición tiene algún problema, se le avisa al usuario para que vuelva a correr el programa.

Otra fase del problema es la impresión del clima. Para imprimir los datos, es necesario un formato legible por cualquier persona. De ser posible, en español.

En conclusión, podemos observar tres etapas principales del problema: el manejo, almacenamiento y acceso de los datos recibidos, el proceso de realizar una petición a OpenWeatherMap y la impresión de la información requerida.

3 Selección de la mejor alternativa

Se escogió el lenguaje Python por su versatilidad, ya que al ser un lenguaje multiparadigma se puede construir el proyecto con una estructura Orientada a Objetos, también nos provee de herramientas sencillas y útiles para trabajar con web services. Además incluye todas las esctructuras que fueron necesarias para resolver el problema, como los diccionarios.

Como ya fue mencionado antes, el nombre de las ciudades nacionales estaba en código IATA, lo que no hace posible hacer una petición a OpenWeatherMap con este dato, pero el *dataset1* también contaba con las coordenadas de cada ciudad, un dato con el que sí es posible hacer una petición. Por esta razón era necesario guardar los dos datos para cada ciudad, para eso se usaron diccionarios, donde la clave de búsqueda es el IATA de la ciudad y el contenido es una lista de sus coordenadas.

Los diccionarios son convenientes en este caso, porque si la ciudad se repite, sólo se reescriben las coordenadas, pero sigue existiendo una única entrada para cada ciudad.

Las ciudades de *dataset2* también se guardaron en un diccionario, pero con entradas vacías porque este no contenía otros datos relevantes a parte del nombre de las ciudades.

En el código, la lectura de los archivos y la escritura en los diccionarios se hizo con un sólo método *lectura()* que distingue entre *dataset1.csv* y *dataset2.csv*.

Para la etapa de las peticiones, se realizó un método llamado *peticiones()* que recibe un diccionario, en este caso recibe la unión de los dos diccionarios creados con anterioridad a partir de los *dataset*. Con un ciclo *for* realiza una petición por entrada del diccionario, si la petición suelta un error por el nombre en código IATA, hace otra petición con las coordenadas. Se verificó que en el caso de las ciudades del *dataset2* no ocurriera ningún error por el nombre de la ciudad. Se auxilia con un contador para no realizar más de 30 peticiones cada 30 segundos, así el programa espera 30 segundos en lugar de un minuto y no satura de peticiones al API. La información de cada petición se empareja en otro dicionario con el nombre de la ciudad, este diccionario servirá para la impresión de la infromación requerida.

4 Diagrama de flujo o pseudocódigo

5 Pensamiento a futuro