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 que salen el mismo día que se corre el algoritmo. No es necesaria la interacción con el usuario, sólo la impresión de los siguientes datos del clima de cada ciudad: nombre de la ciudad o el aeropuerto, la descripción general del clima, temperatura, sensación térmica, temperaturas mínima y máxima, presión y humedad.

2 Análisis del problema

Para este problema los datos con que se cuentan son los siguientes:

- **Datos de entrada:**
  - Dataset1.csv que contiene las ciudades o países de origen y destino escritos en clave IATA correspondiente a la ISO 3166, además contiene las latitudes y longitudes del origen y del destino.
  - Dataset2.csv que contiene el destino, hora de salida, hora de llegada y la fecha de salida.

Y los datos que queremos que nos de el programa son:

- **Datos de salida:**
  - Informe del clima. Que contenga: nombre de la ciudad, descripción del clima, temperatura mínima, temperatura máxima y humedad humedad de la ciudad origen.

Se necesita una manera eficaz de leer los 3 mil datos distintos, guardarlos y poder accesar 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 *dataset1* 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 consideró que el API de OpenWeatherMap es una herramienta que devuelve los datos necesarios para este problema, además es de muy fácil acceso, gratuita y se maneja de manera sencila con Python.

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. Para esa espera, se utilizó la librería *waiters* de Python, ya que sirve para esta ocasión y es la que se conocía. La información de cada petición se empareja en otro dicionario con el nombre de la ciudad, este diccionario servirá para la impresión de la información requerida.

Como las peticiones regresan más información de la pedida, en el método *impresión()* se filtra, y sólo son impresos los datos requeridos. Como no es necesaria la distición entre ciudad origen y destino, simplemente se imprime el clima de toda ciudad que aparezca en los *dataset*.

4 Diagrama de flujo o pseudocódigo

Proyecto1
Inicio:
    Leer archivo csv.
        diccionario = {}
        if dataset1 existe
            diccionario ← {dataset1.ciudadOrigen : [ciudad.latitud, ciudad.longitud]}
            diccionario ← {dataset1.ciudadDestino : [ciudad.latitud,ciudad.longitud]}
        elif dataset2 existe.
            diccionario ← {dataset2.destino : []}
        else:
            print Archivo no encontrado.
    Peticiones diccionario
        $i = 0$
        diccionarioNuevo = {}
        for llave en diccionario
            if $i >= 30$
                Esperar 30 segundos
                $i = 0$
            Hacer petición ← petición a API con llave
            if petición == **True**

```
            diccionarioNuevo ← {llave : peticiónClave}
            i += 1
        elif petición == False
            diccionarioNuevo ← {llave : peticiónCoordenadas}
            i += 1
        else
            diccionarioNuevo ← {llave : error}
            i += 1
    Imprimir diccionario
        for llave in diccionario
            if diccionario[llave] != error
                Print diccionario[llave] + información de clima
            else
                Print diccionario[llave] + error
Fin
```

5 Pensamiento a futuro

Algunas mejoras que se le pueden hacer al proyecto es solucionar todos los problemas que presenta, como que el API no encontró el idioma para algunas ciudades. Por otro lado, a los datos impresos del clima se le puede agregar más información, como la zona horaria, el porcentaje de nubes, la visibilidad y la velocidad del viento. Esta mejora es sencilla y se puede hacer en poco tiempo, de acuerdo a lo que decida el cliente.

Usando varios hilos de ejecución se pueden hacer más peticiones en menos tiempo y así el programa sería más veloz. También, al ser dos programadores trabajando, se podría usar ambas llaves de acceso al API al mismo tiempo.

Con una membresía diferente de OpenWeatherMap, que tiene costo, se podría tomar en cuenta el tiempo de vuelo y pedir predicciones del clima a la hora de llegada del vuelo a la ciudad en cuestión.

También se podría construir una interfaz de usuario que permita ingresar el nombre de la ciudad y se imprima la información del clima uno a uno, o, que se pueda elegir de qué ciudades se quiere ver la información, en lugar de que se impriman todos al mismo tiempo.

Por último están las actualizaciones a nuestro programa y mantenimiento que este pueda llegar a necesitar, ya sea por cambios en la API de OpenWeatherMap y como ésta es utilizada o futuras incompatibilidades que se puedan llegar a presentar en la ejecución con nuevas versiones del lenguaje.

En relación al costo del código se propone la cantidad de 3500.00 pesos mexicanos y un costo extra para cada actualización o mantenimiento que llegue a necesitar el programa, según se determine con el cliente.