Universidad Nacional Autónoma de México.

Facultad de Ciencias

Modelado y Programación

Proyecto 1: WebService

Santiago Díaz Ponton, Mauricio Guerrero Palomares

12 de octubre de 2021

1)*Definición del problema.*
Para este proyecto, debemos hacer que nuestro código sea capaz de leer un archivo tipo *csv*, procesar esa información y apartir de ella consultar el clima de la ciudad en cuestión.
Para poder consultar el clima de la ciuda, tendremos que generar una llave en la API de alguna página que tenga información del clima de ciudades, y apartir de ella hacer peticiones sobre para acceder a la información.
2)*Análisis del problema*
Para realizar el programa, primero debemos leer el archivo *dataset1.csv* y obtener su información. Luego de esto, mediante la API de elección accedemos a los climas de las ciudades. Durante este proceso, debemos generar un cache que guarde los climas de las ciudades que hemos accedido mediante un Request en la API del clima. Si la ciudad ya se encuentra en nuestro caché, entonces tomamos esta información para no hacer un request innecesario. Por último, el clima de las ciudades se mostrará en temrinal, con la ciudad correspondiente, la temperatura media, temperatura mínima/máxima, Humedad y sensación térmica.

3)*Selección de la mejor alternativa*

Para resolver el problema, decidimos usar python por sus bibliotecas **request** y **csv**. La primera funciona para hacer peticiones de API's de una forma sencilla y rápida, la segunda biblioteca funciona alrededor de archivos tipo *csv*, por lo cual fue muy sencillo leer **dataset1.csv**, y también fue sencillo almacenar la información del archivo gracias al método *DictReader()* el cual almacena la información proveniente del csv en un diccionario, lo cual hace mucho más sencillo manejar los datos al querer acceder a la API de algún webService. También, usamos python **Orientado a objetos**, para poder acceder a las variables de clase, y no fue necesario hacer un constructor de clase, así que se uso el constructor que las clases tienen por defecto, ya que no es necesario que reciban parámetros
Un aspecto importante a tomar en cuenta, es que el clima se tiene que tomar en tiempo real. Por esto se tenía que acceder a la API de algún webService, en particular en openWeatherMap se encuentra mucha información sobre cómo ocuparla
También, el uso de diccionarios fue la mejor alternativa en la mayoría de estructuras de datos en el programa. Por ejemplo, el caché es un diccionario, ya que cada llave guarda la **latidud y longitud** de alguna ciudad, y dentro de la llave está la información correspondiente a el clima de esa ciudad, así con el método **containsKey()** y **containsKeyBool()** verificamos si la llave en un diccionario es igual a la latitud y longitud de la ciudad que se está revisando en el csv. Otro uso de diccionarios fue para guardar los Requests, los request con los datos los climas de la ciudad de origen y la ciudad de destino.
En cambio, las cadenas donde se guarda el clima de una ciudad, están guardadas en listas, ya que es más fácil concatenar cadenas en una lista usando el método **append**.
Los diccionarios y listas usados fueron **variables de clase**, para acceder a sus valores de forma más sencilla,por esto cada que se accede a **(cadenasOri,cadenasDest,compDatOrigin,compDatDestin, cache,key y url)** usando *self.*
Otra decisión importante fue el usar coordenadas para acceder a OpenWeatherMap, ya que puedes obtener el clima de una ciudad apartir de el nombre de la ciudad en código IATA, por latitud y longitud, por nombre de la ciudad (este debe ser el código que OpenWeather usa por ciudad y por pais) y otras formas. Usamos las coordenadas porque de esta forma hay menos fallas y es más fácil acceder, aparte de ya tener las coordenadas a revisar en el archivo **dataset1.csv**, y si quisieramos usar otra forma de acceder tendriamos que verificar aparte de **dataset1** ,otro archivo donde estuvieran los códigos IATA de los aeropuertos.
El mayor problema que se presentó en el código fueros los test. Utilicé *Unittest*, que es una librería de python que sirve para realizar pruebas unitarias. El problema que tuve fue el acceder a los módulos que quería probar, ya que estos módulos se encuentran en la carpeta *src*, y los test en la carpeta *tests*, por esto no podía acceder a los módulos, por lo cual no fue posible poderlos ejecutar.
Otro problema que se presentó fue el cache, ya que para implementarlo se tuvieron que implementar 2 métodos, *containsKeyBool* y *containsKey* que se verán más a fondo en la parte de pseudocódigo, pero al final encontramos una forma sencilla de implementarlo y hacer que funcione como debería.

4) *Pseudocódigo* **Clase readerCsv**

readDataSet(file)

Este método lee un archivo tipo *csv*. Primero, accedemos al archivo en cuestión, y con el método dictReader de la clase csv guardamos la información en un diccionario *dic*. Ahora que toda la información está guardada en *dic*, lo que haremos es recorrer *dic* y guardar cada una de las entradas en otro diccionario a, de esta forma en cada lugar de a tendremos la información de un vuelo.

Clase requestKey

En esta clase sólo generamos nuestra llave y verificamos su buen funcionamiento.

Clase Request

1)-printForm(self,weather)

Este método imprime en pantalla un diccionario *weather*, el cual contiene la información del clima de una ciudad. Accede a las posiciones del diccionario donde se encuentran la información correspondiente, y guarda en cadena el nombre de la ciudad (*cadenaCiu*), el clima de la ciudad (*cadenaCli*, dando la temperatura media,máxima y mínima), la sensación térmica (*cadenaSens*), la humedad (*cadenaHum*), y por último concatena todas estas cadenas (*cadenaCompleta*).

2)-containsKeyBool(self,dicCache,dicKey)

Este método revisa si en un diccionario *dicCache* se encuentra un elemento con llave *dicKey*, si se encuentra regresa True, si no False.

3)-ContainsKey(self,dicCache,dicKey)

Este método revisa si en un diccionario *dicCache* se encuentra un elemento con llave *dicKey*, si se encuentra entonces guarda dicho elemento en una lista *keyList* y la regresa.

4)-inList(self,dicRequest)

Este método se encarga de tomar los elementos de *dicRequest* e ir almacenando los valores que nos interesa tomar y descartando los que no. Iremos accediendo a estos valores y los guardaremos en una lista *cadenas*, la cual regresaremos con todos los valores que nos interesa.

5)-checkCache(self,latlonOri, latlonDes,urlCompIda,urlCompDest)

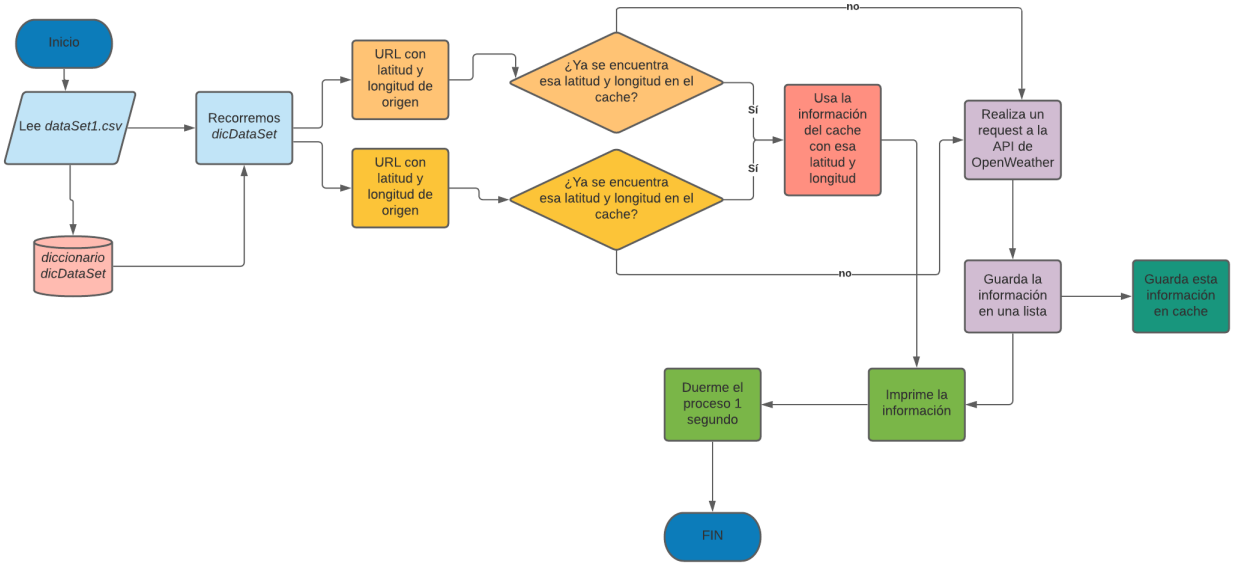
Este método revisa si los climas de ambas ciudades de un vuelo se encuentran en nuestro caché
Lo que hace es preguntar si en el caché ya se encuentra un elemento con la llave *latLonOri* (que es la latitud y longitud de origen), si se encuentra, entonces buscamos el elemento y lo metemos en una lista. Si no se encuentra en el caché, entonces este es un elemento que aún no hemos guardado, por lo cual haremos un request para obtener la información, por último tomaremos la información que sí nos interesa del request y lo imprimiremos en pantalla. Por último, repetiremos este proceso pero con *latLonDest*(que es la latitud y longitud de la ciudad de destino), ya que los casos son análogos
6)-request(self,dicDataSet) Este método se encarga de poner en función todos los demás métodos. Lo que haremos es acceder a nuestro diccionario *dicDataSet* que contiene la información leída apartir del csv, y luego lo iremos recorriendo. Generaremos una url con cada una de las longitudes y latitudes de las ciudades en *dicDataSet*. Luego, mediante el método *checkCache* iremos haciendo peticiones e imprimiéndolas o imprimiendo información de nuestro caché, durmiendo el proceso 1 se-

gundo para poder leer la información y no hacer un uso excesivo de requests.

Pseudocódigo del programa

Lo que haremos para el funcionamiento del programa, es primero leer la información en *dataset1.csv* y guardarla en un diccionario *dicDataSet*, luego iremos recorriendo *dicDataSet*, y generaremos una URL con la latitud y longitud en la posición en la que estemos. Luego de esto, verificaremos si ya tenemos información en nuestro caché respecto a la posición en la que estemos, si no la tenemos entonces haremos un request a la API de OpenWeather para obtener información y la metemos al caché.Esta información se guarda en un diccionario *compDatOrigin* en la ciudad de origen y *compDatDest* de la ciudad de destino. Por último, tomaremos la información que nos sirve apartir del diccionario (es decir, temperatura media/máxima/mínima,humedad,sensación térmica,ciudad), y la imprimiremos.

Diagrama de flujo



5)¿qué mantenimiento crees que podría requerir en un futuro?

Este programa podría mejorar con la implementación de una interfaz gráfica para que sea más amigable con el usuario. También podría mejorar en los tests, con tests más robustos como qué hacer en caso de una falla en internet.

¿cuánto cobrarías por el reto?

Al pensarlo creo que el precio más acertado al trabajo que hemos realizado es de 30,000 pesos. Ya que fueron muchas horas invertidas en buscar información y revisar el funcionamiento correcto del programa.