Universidad Nacional Autónoma de México

Facultad de Ciencias

Proyecto 1

Modelado y Programación

Johann Ramón Gordillo Guzmán

418046090

Proyecto presentado como parte del curso de **Modelado y Programación** impartido por el profesor **José de Jesús Galaviz Casas**.

20 de Agosto del 2019

Link al código fuente: https://github.com/JohannGordillo/

1. Definición del Problema

El cliente, Aeropuerto Internacional de la Ciudad de México, nos ha solicitado elaborar un programa para obtener un informe del clima en tiempo real en las ciudades origen y destino de $3,000$ vuelos.

La información de los vuelos residirá en la computadora del cliente y contendrá el nombre de las ciudades origen y destino, así como sus coordenadas geográficas en un archivo .csv. El cliente, además, quiere poder compartir el programa con varios aeropuertos en el país.

2. Análisis del Problema

1. **Requisitos Funcionales**

   Dado un archivo .csv con los apróximadamente $3,000$ vuelos, se debe entregar como salida un informe en tiempo real del clima en las ciudades de origen y destino por la salida estándar.

   Se debe llevar a cabo la obtención de dichos informes de alguna aplicación externa para procesarlos y regresar al usuario lo que solicita.

2. **Requisitos No Funcionales**

   El cliente es el Aeropuerto Internacional de la Ciudad de México, por lo que se supone el personal estará calificado para manejar el directorio de archivos de la computadora y poder ubicar sencillamente los archivos necesarios por medio de la interfaz proporcionada por el programa. Dicha interfaz será **amigable** y sencilla de usar para el personal del aeropuerto.

   El programa deberá poseer **tolerancia a fallas**, pues el Aeropuerto invierte millones de pesos en sus usuarios y quiere que éstos estén satisfechos con los servicios proporcionados, entre ellos el conocer las diferencia de clima entre la ciudad de donde viajan y a la que viajan.

   El programa no será **escalable** si el Aeropuerto no proporciona un servicio de paga en una API para obtener el clima, como por ejemplo Open Weather Map o Yahoo Weather.

   Se cumplirá con la **eficiencia** necesaria en un aeropuerto de talla internacional, asegurándose el programa de que los datos sean obtenidos del servidor del web service de manera rápida, y de la misma manera impresas en la salida estándar.

Considerando la complejidad que conlleva realizar una API que nos provea el clima en tiempo real de una cantidad razonable de ubicaciones geográficas, y la falta de presupuesto y medios para la realización de la misma, se opta por utilizar un servicio web que nos proporcioné dichos datos.

Sin embargo, los datos serán pasados al programa en un archivo .csv (comma separated values), por lo que se necesitará un proceso intermedio, antes de hacer solicitudes al web service, en el que procesemos los datos del archivo y una vez que obtengamos lo necesario, por ejemplo las coordenadas geográficas, podamos hacer solicitudes al servidor y obtener el informe del clima para dicha ubicación.

De una manera más particular, podemos dividir hacer el analísis del problema de la siguiente manera:

2.1. Entrada

Uno o varios archivos .csv (comma separated values) con los datos de la ciudad de origen, ciudad destino, y la latitud y longitud de ambas ciudades. El archivo se seleccionará del directorio de archivos del usuario con ayuda de una interfaz gráfica (GUI).

1. **Formato**
   Archivos de formato .csv (Comma Separated Values). Con el siguiente formato:

   origin, destination, origin_latitude, origin_longitude, destination_latitude, destination_longitude

2. **Tamaño**
   Cada archivo con, a lo más, $3,000$ vuelos.

   $1 \leq vuelos \leq 3,000$

3. **Cantidad**
   Uno o más archivos.

   $1 \leq archivos$

4. **Fuente**
   Directorio de archivos del usuario.

2.2. Salida

Un informe del clima en tiempo real de las ciudades origen y destino, por la salida estándar.

1. **Formato**
   Texto plano.

2. **Cantidad**
   Un informe del clima por cada vuelo del archivo.

3. **Destino**
   Salida estándar.

3. Selección de la mejor alternativa

Se ha decidido usar el lenguaje de programación Python para la elaboración del proyecto. Python es un lenguaje interpretado muy fácil de usar, además de muy poderoso, y con muchísimas librerias, como 'requests' y 'pytest', que podrían ser de mucha ayuda en el programa. Además de que nos permite usar un paradigma de **programación orientado a objetos** y al mismo tiempo nos proveé del poder de la programación **funcional**.

Como se ha mencionado en la sección de **Análisis del Problema**, no es factible crear una API para obtener el clima en tiempo real. En su defecto, se ha decidido usar el web service **Open Weather Map**, que nos proporcionará un informe del clima en el lugar con coordenadas geográficas dadas en el archivo .csv.

Para poder utilizar la API, se ha registrado una cuenta gratuita, que nos permitirá realizar un número pequeño de solicitudes (miles, aún así) de informes de clima al servidor. La API key obtenida se ha colocado en la carpeta data/ del proyecto, pues es buena práctica mantenerla fuea del código principal.

Para las pruebas unitarias, se ha elegido **pytest**, pues está muy bien documentado y más legible que **unittest**, lo que va acorde con el **Zen de Python**.

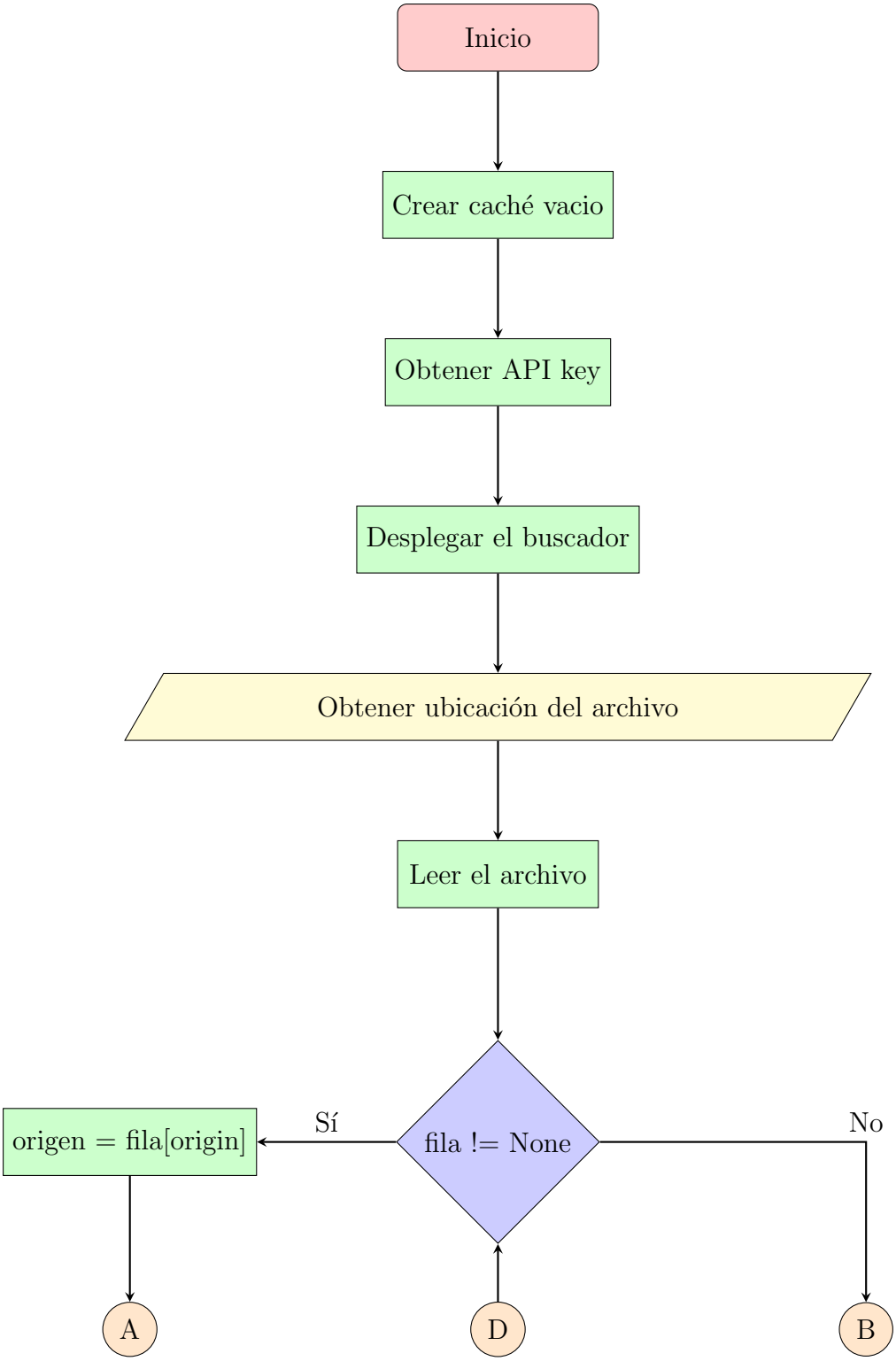
El desarrollador del proyecto está consciente de que muchos vuelos podrían estar repetidos en el archivo proporcionado, por lo que se ha optado por utilizar un **diccionario** como **caché**. El diccionario nos da una **complejidad en tiempo amortizada de O(1)** en sus operaciones, por lo que es la mejor opción entre las distintas estructuras de datos disponibles.

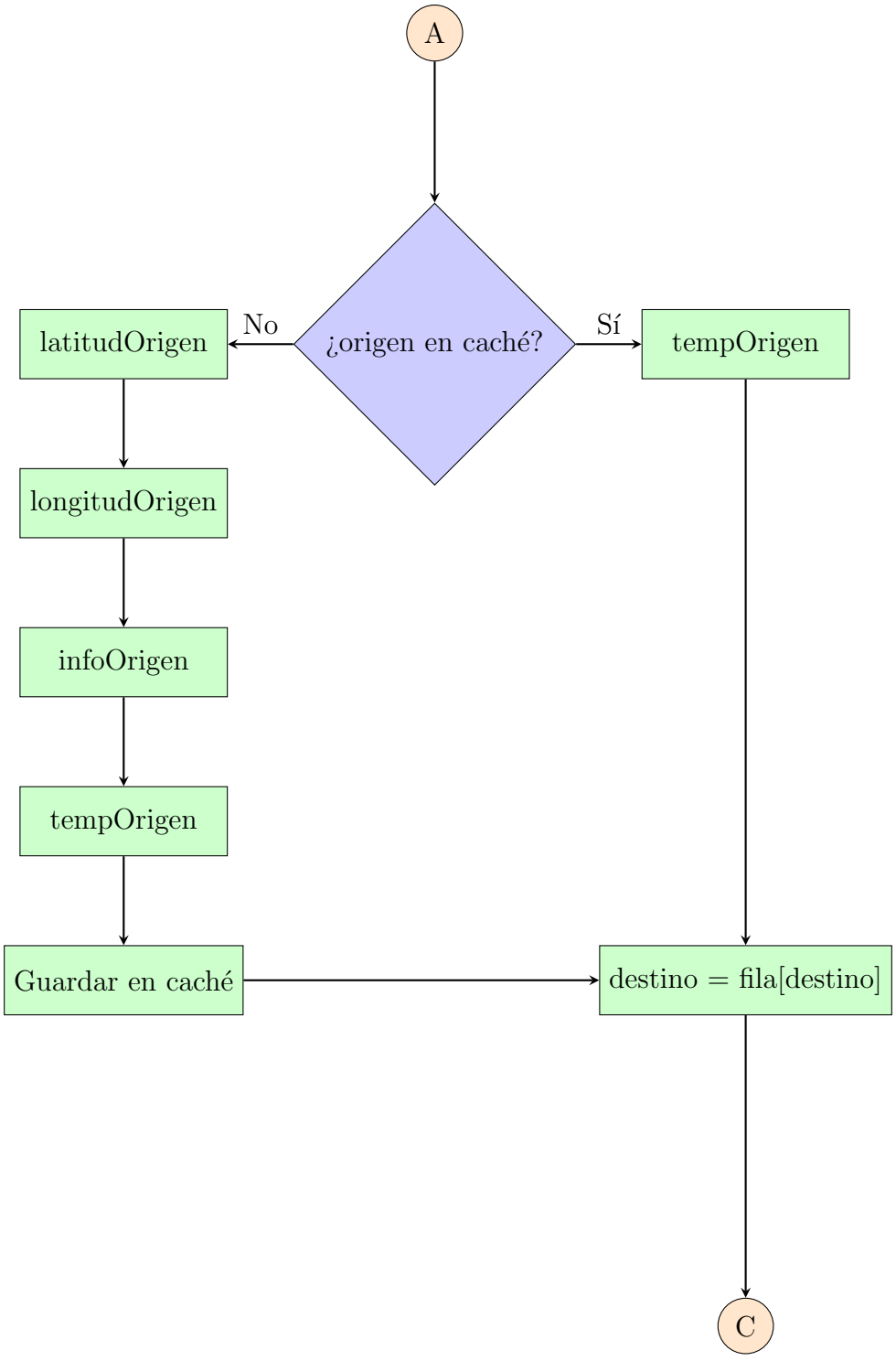
El modelo con el cual trabajar elegido se ajusta a los requisitos funcionales y al esbozo general de la solución y el programador no ve una mejor solución por el momento. Sin embargo, si en algún futuro encuentra una mejor manera de hacerlo, notificará al cliente.
[**véase Mantenimiento**].

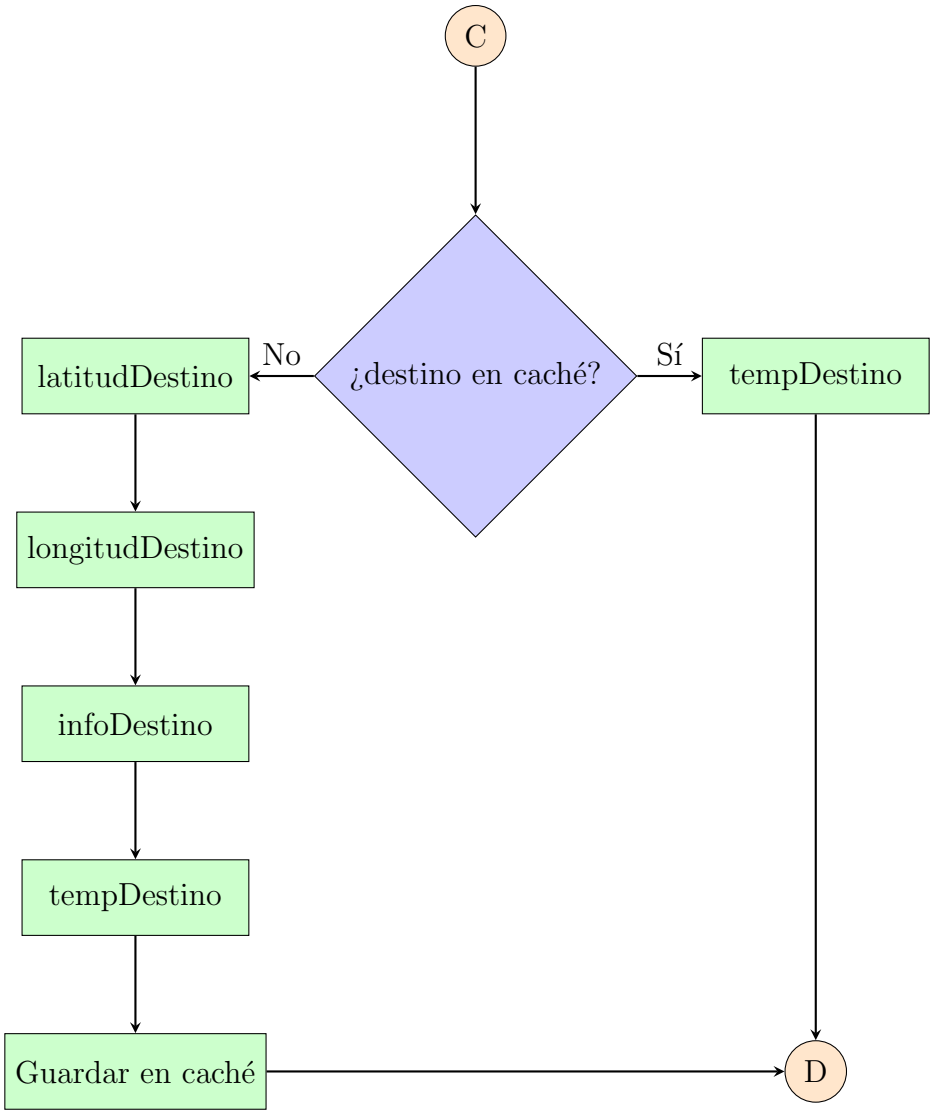
Sin embargo, si hay maneras menos eficientes de resolver el problema, por ejemplo usando una pila o una cola como caché. La complejidad en tiempo sería peor y sería muy difícil, sino imposible, utilizarlo como caché.

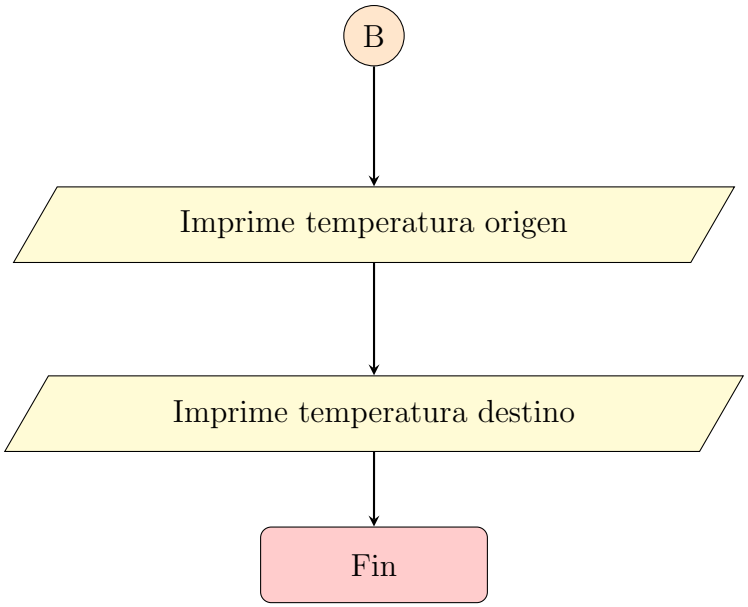
Como se mencionó anteriormente en este documento, se utilizará **interfaz gráfica**. Para ello, se ha elegido usar **TKinter** sobre otras alternativas, por su facilidad de uso y su excelente documentación.

4. Diagrama de Flujo




A
¿origen en caché?
No
Sí
latitudOrigen
longitudOrigen
infoOrigen
tempOrigen
tempOrigen
Guardar en caché
destino = fila[destino]
C


C
¿destino en caché?
No
Sí
latitudDestino
tempDestino
longitudDestino
infoDestino
tempDestino
Guardar en caché
D



4.1. Explicación

1. Iniciamos la ejecución del programa.

2. Primero creamos un caché donde guardaremos los datos solicitados al servidor de Open Weather Map, para evitar hacer consultas repetidas.

3. El siguiente paso es obtener la key de la API, para poder acceder al servicio web.

4. Una vez hecho esto, desplegamos un buscador de archivos (GUI) para que el usuario seleccione el archivo a procesar.

5. Ya con el archivo identificado, obtenemos su ruta para posteriormente leerlo.

6. Checamos que la fila del archivo leido no sea vacia, es decir, que haya algo que leer en el archivo. De ser así, pasa a «**8**». En caso contrario pasa a «**7**».

7. Ya se ha leido todo el archivo, y por ende se ha procesado ya toda la información requerida por el proyecto, imprime el clima de las ciudades origen y destino y pasa a «**16**».

8. Hay que leer los datos para la ciudad de origen, para ello, primero hacemos a la varibale origen igual a la columna ciudad que le corresponde en el archivo.

9. Si esta ciudad ya está dentro del caché, únicamente la extraemos y la guardamos en la variable **temp_origen**.

10. Si no lo está, buscamos su latitud y su longitud en el archivo leido.

11. Ya con la latitud y longitud ubicadas, hacemos la petición al servicio web para que nos regrese la información del clima en estas coordenadas geográficas, y guardamos los datos recibidos en una variable **info_origen.**

12. Extraemos de la variable **info_origen** la temperatura de la ciudad origen y la guadamos en la variable **temp_origen.**

13. Guardamos en el cache la información correspondiente a esta ciudad.

14. Repetimos los pasos «**8**» a «**13**», pero ahora con la ciudad destino.

15. Regresa al paso «**5**».

16. Finalizamos el programa.

5. Sobre el mantenimiento del código

5.1. Nuevas funcionalidades

El programa solicitado por el cliente solicita únicamente la temperatura de las ciudades de origen y destino. Sin embargo, Open Weather Map otorga más datos, como la temperatura máxima, la temperatura mínima y el pronóstico del clima; por lo que si el cliente así lo desea, el desarrollador podría agregar estas funcionalidades en futuras versiones.

Si el cliente en algún momento decide almacenar los datos obtenidos en un archivo, también se puede implementar sin ningún problema.

5.2. Bugs

Ningún programa es libre de bugs, y el desarrollador está conciente de ello, por lo que ante descubrimientos de bugs y errores en el código, el desarrollador se encargará de corregirlos y agregarlos a las pruebas unitarias.

5.3. Eficiencia

Si el programador descubre una manera más eficiente de implementar la solución al problema, notificará al cliente para estimar el costo de nuevas implementaciones. Por ejemplo, si se encuentra un algoritmo que permita reducir el tiempo en el que se hacen peticiones al servidor de Open Weather Map.

5.4. Escalabilidad

Si el cliente necesita que se procesen más solicitudes al servidor de la API, éste necesita proporcionar al desarrollador del proyecto una subscripción de paga al web service. Pues la subscripción gratuita tiene un límite de solicitudes antes de saturarlo.

6. Costo final del proyecto

El costo del proyecto fue calculado con base en las horas que el desarrollador del mismo, **Johann Gordillo**, tuvo que invertir para verificar que la solución al problema planteado por el usuario, **Aeropuerto Internacional de la Ciudad de México**, fuera eficiente; y el código de la implementación, robusto.

Se implementaron pruebas para que en un futuro, desarrolladores que trabajen en el proyecto puedan implementar nuevas funcionalidades y aún así verificar si el código sigue funcionando correctamente bajo ciertos casos planteados estrategicamente por el desarrollador.

El costo final es de **$500.00** pesos mexicanos.

Para aclaraciones del proyecto y del costo del mismo contactar con **jgordillo@ciencias.unam.mx**