Programación III

Trabajo Práctico Especial

Integrantes: Carpinetti, Heroel - Careri, Nicolas

Github: https://github.com/HeroelC/tpeProgramacionIII

Fecha de entrega: Viernes 17 de Marzo de 2019

Lugar de entrega: Laboratorio de ISISTAN

Introducción

Se desea implementar un sistema centralizado, para proveer información a viajeros frecuentes, este deben conocer los aeropuertos y los vuelos correspondientes. Los mismos poseen diferentes aerolíneas a la hora de optar por un viaje. A su vez las aerolíneas poseen una cantidad de butacas disponibles, la cual está dada por la capacidad total menos la cantidad de reservas.

El sistema debe poder listar todos los aeropuertos, las reservas realizadas y además poder consultar sobre diferentes tipos de vuelo.

Objetivo del trabajo

Introducirnos en los conceptos de estructuras de datos (grafos) para poder manipularlos, entender su complejidad y eficiencia.

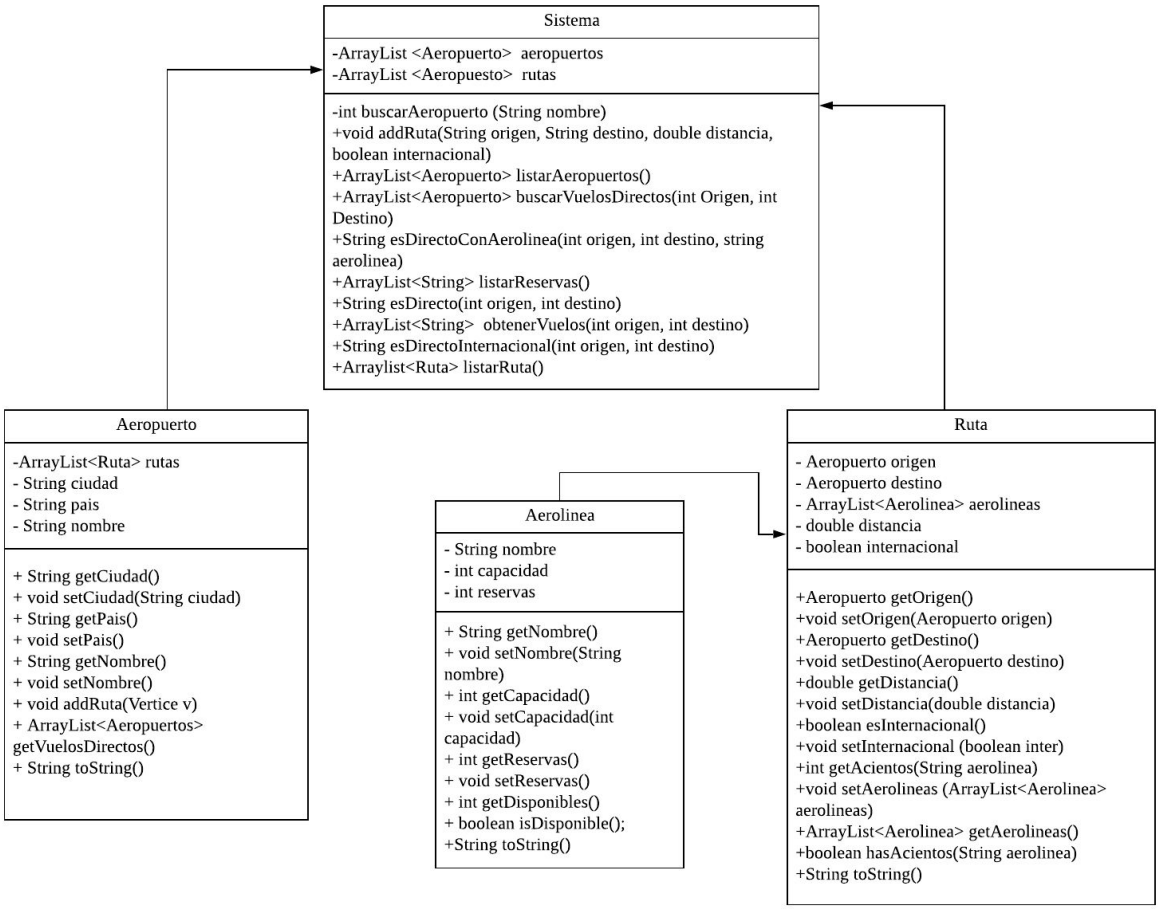
Utilizar las diferentes estructuras de datos, ya sea las proporcionadas por el lenguaje java o implementando las propias, entendiendo el costo computacional que estas representan.

Poder utilizar los contenidos brindados por la cátedra de manera eficiente de tal forma que se pueda implementar los requerimientos que se plantean en la introducción en el tiempo estimado.

Tareas involucradas en el desarrollo

La primer tarea que realizamos para poder iniciar el desarrollo del sistema fue investigar cómo era la estructura de un grafo, cuales eran sus métodos y la forma de implementarlo en java.
Realizamos la implementación de un grafo genérico en java para comprender las diferencias entre implementar un grafo dirigido de uno no dirigido y sus diferentes comportamientos.

Entendiendo lo anterior, comenzamos a reconocer las distintas clases que iban a ser parte del sistema y que función iba a cumplir cada una. Con esto realizamos el primer diagrama UML como se puede observar en siguiente imagen.



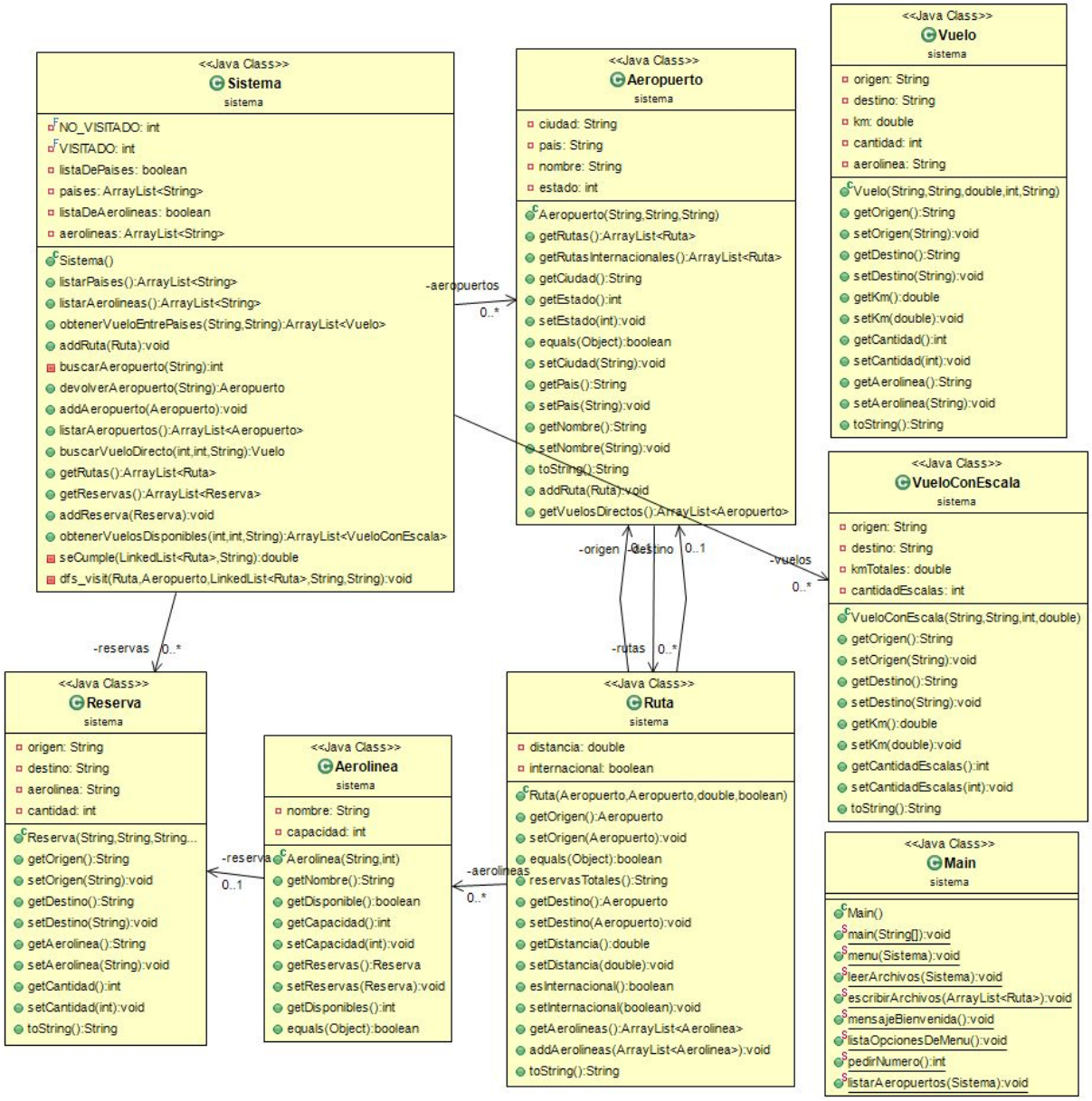
1.0 Primer diagrama UML

Generando este diagrama comenzamos con la implementación de clases, métodos y algoritmos. Luego de varias pruebas, cambios y nuevas implementaciones decidimos hacer varias modificaciones al UML que generamos en un principio.

Otra de las tareas que tuvimos que realizar fue analizar el funcionamiento del código que fue provisto por la cátedra para cargar el sistema de información por medio de archivos csv. Y a su vez analizar otro código el cual nos permite escribir en un archivo csv información de los servicios que fueron pedidos durante la ejecución del sistema.

Modelado del problema

Explicación de los tipos de datos elegidos y su relación entre sí



2.0 Ultimo diagrama UML

Como se puede ver en la imágen, utilizamos un grafo no dirigido para la estructura general del sistema la cual se representa de la siguiente manera:
Los vértices representan los aeropuertos guardando la información del mismo y su lista de adyacencia.

Las aristas están representadas por las rutas que almacenan información del vuelo entre dos aeropuertos. Dentro de las mismas se encuentran dos tipos de datos más, que son las aerolíneas con sus respectivas reservas. Dentro de este sistema también se encuentran almacenadas:

- Lista de países de los distintos aeropuertos sin repetir
- Lista de las reservas realizadas
- Lista de las distintas aerolíneas disponibles

Cada una de éstas tiene la funcionalidad de poder acceder a ellas de manera más eficiente a costa de utilizar más memoria.

Para retornar la información de los distintos servicios, creamos un tipo de dato llamado "Vuelo" que este almacena la información necesaria para su retorno.

Estructura de datos elegidas

Grafo, ArrayList, LinkedList

Ventajas y desventajas de las estructuras de datos elegidas

Grafo

A la hora de implementar el grafo escogimos la lista de adyacencia, esta forma de implementar el grafo nos provee de ciertas ventajas y desventajas a la hora de acceder, insertar o recorrer el mismo.

Las ventajas que nos provee la lista de adyacencia es que requiere un espacio proporcional a la suma entre la cantidad de vértices y la cantidad de arcos, esto trae un buen uso a la cantidad de memoria utilizada. Esta implementación se ve favorecida ya que el grafo planteado en el sistema es un grafo más disperso que denso, en el caso contrario hubiésemos optado por la matriz de adyacencia.

Las desventajas de la lista de adyacencia proviene al encontrarse con un grafo denso (demasiados enlaces entre los vértices) ya que estamos obligados a la hora de determinar si existe o no, un vértice o una arista, en el peor de los casos a recorrer uno por uno los vértices y las aristas. Por esta razón en caso de implementar un grafo denso se optaría por la matriz de adyacencia que provee un tiempo fijo y constante a la hora de acceder a costa de más memoria. La desventaja en caso de querer examinar la matriz, hace que nos encontremos ante una complejidad computacional de $O(n^2)$

Carga de datos

Para la carga de información mediante csv decidimos utilizar ArrayList, dado que es una lista que nos permite insertar la cantidad de elementos que fueran necesarios de forma dinámica y luego accederlos de forma directa (ya que al usuario en el momento de mostrarle las posibles opciones le damos la lista

que tenemos en el sistema y los números que el usuario visualiza en realidad son las posiciones donde están guardados nuestros datos).

Una manera en la cual esta implementación podría haber sido más eficiente es la de recorrer la cantidad de información que se desea insertar y crear un array estático con estas cantidades. Esto beneficiaría en gran medida al uso de procesamiento que se está utilizando ya que, el ArrayList crea en un primer momento un array estático de 10 elementos y a medida que va necesitando más lugar crea otro array estático con más capacidad y recorre todos sus elementos para copiarlos al nuevo, haciendo esta iteración cada 10 elementos que se inserten en ese ArrayList obteniendo como resultado una complejidad de O(n) siendo n la cantidad de elementos de la lista en cada iteración.

Recolección de datos para los servicios

Para realizar la recolección de datos de las aerolíneas y de los países disponibles dentro del sistema, decidimos realizar un solo recorrido cuando se realiza la primera consulta sobre el servicio que necesite de estos y guardarlos en una variable de tipo ArrayList la misma está acompañada de una variable de tipo boolean la cual indica si el ArrayList fue cargado o no previamente. Esta implementación hace que solo se recorra toda la lista una sola vez (la primera vez que se ejecuta el servicio)

Implementación de los servicios

Servicio 1: Verificar vuelo directo:

Selección y justificación del algoritmo elegido:

Para la resolución del mismo decidimos recorrer la lista de adyacencia del aeropuerto origen recibido, filtrando por la aerolínea elegida por el usuario, obteniendo exclusivamente los resultados válidos. En caso de no existir la petición, retornamos nulo.

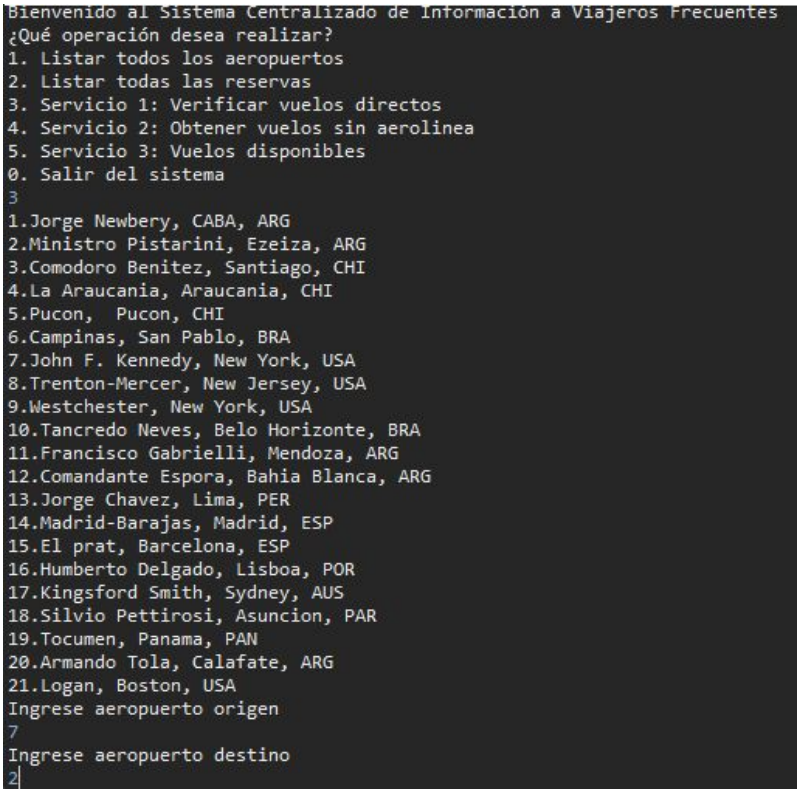
Utilización de los tipos de datos:

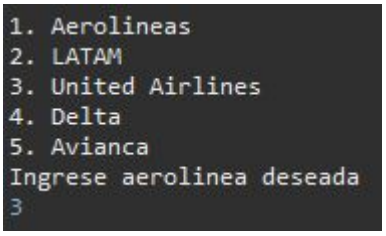
Decidimos utilizar el ArrayList de las adyacencias del aeropuerto origen y la utilizamos para recorrerla para así encontrar el destino indicado, como también el ArrayList de aerolíneas que posee la ruta de dichos aeropuertos.

Complejidad del algoritmo utilizado:

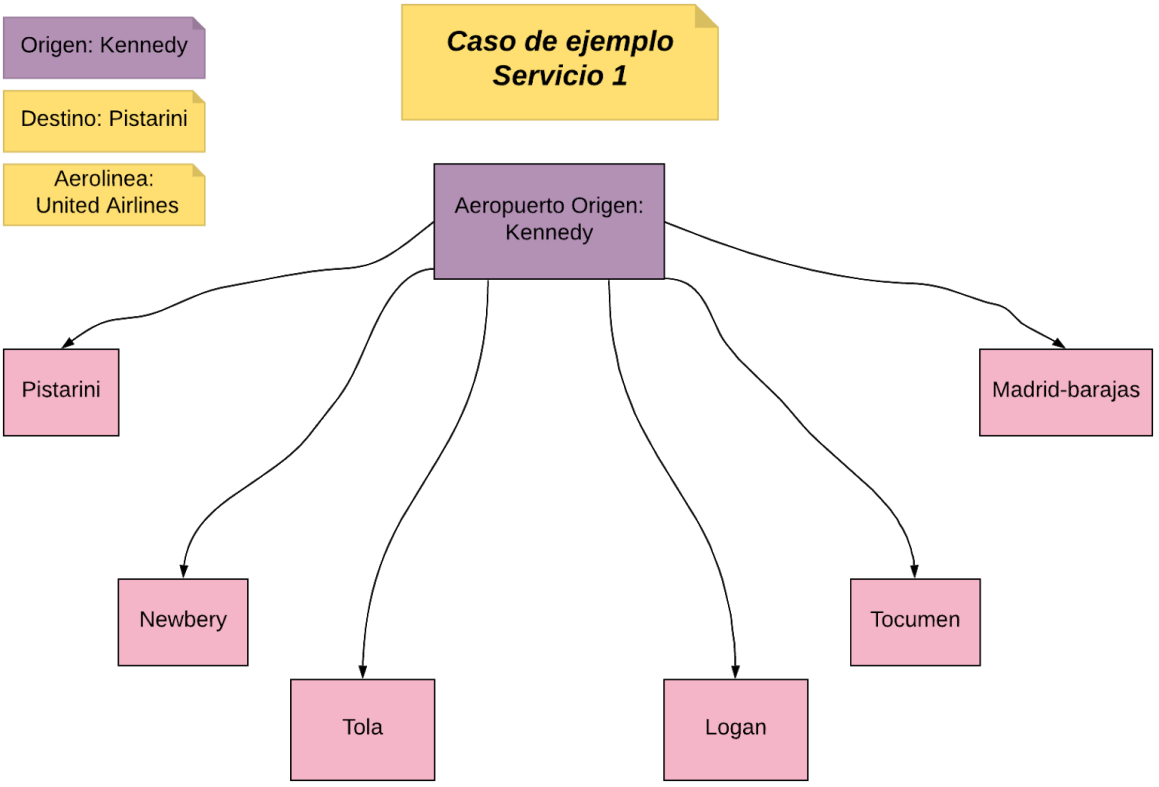
Ésta implementación en el peor de los casos tiene una complejidad computacional de $O(n^2)$ siendo “n” el tamaño de las lista de adyacencia del aeropuerto origen y el tamaño de la lista de aerolíneas de la ruta encontrada, debido al recorrido de las mismas

Caso de ejemplo:

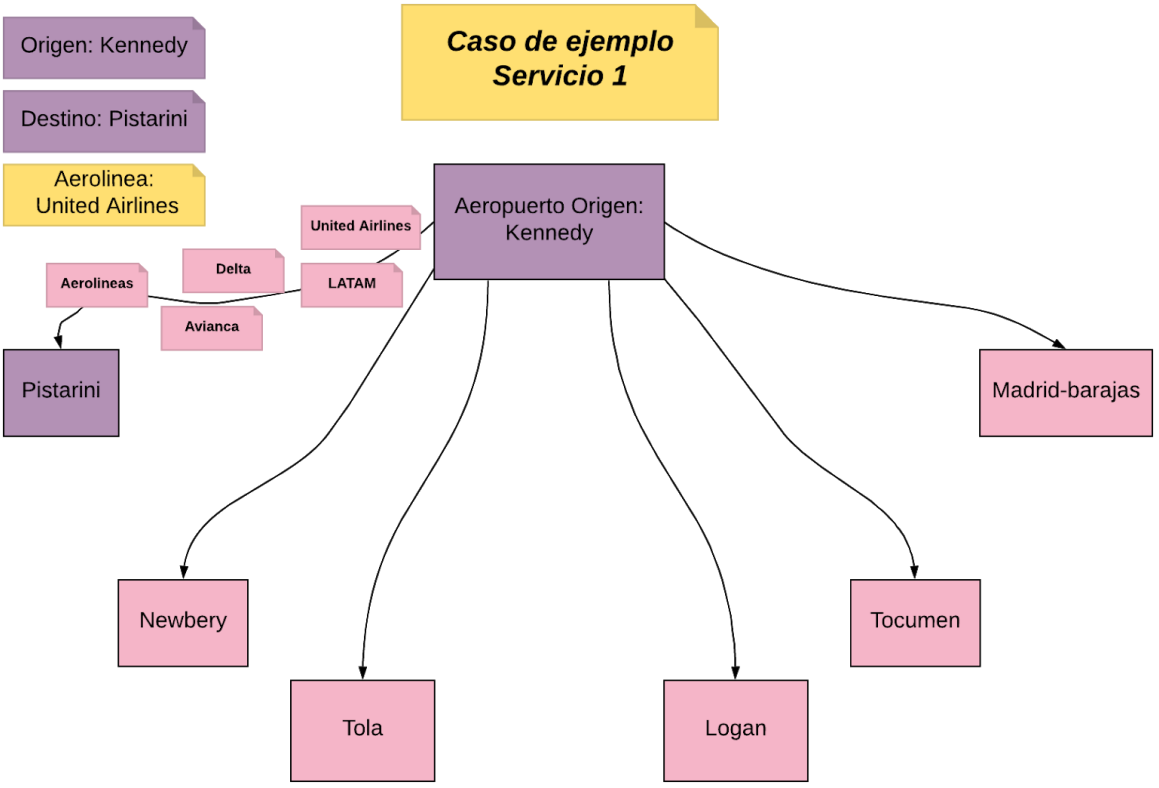




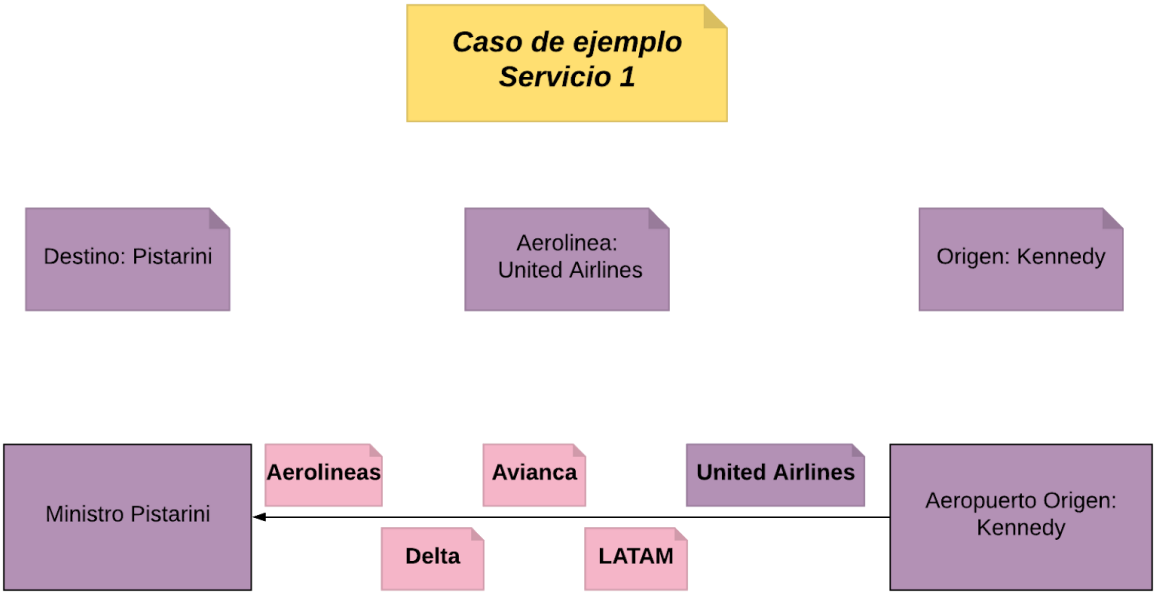
Elegimos el servicio 1: Verificar vuelos directos. Luego seleccionamos el aeropuerto “John F. Kennedy” como origen y el aeropuerto “Ministro Pistarini” como destino, y como última instancia la aerolínea “United Airlines”



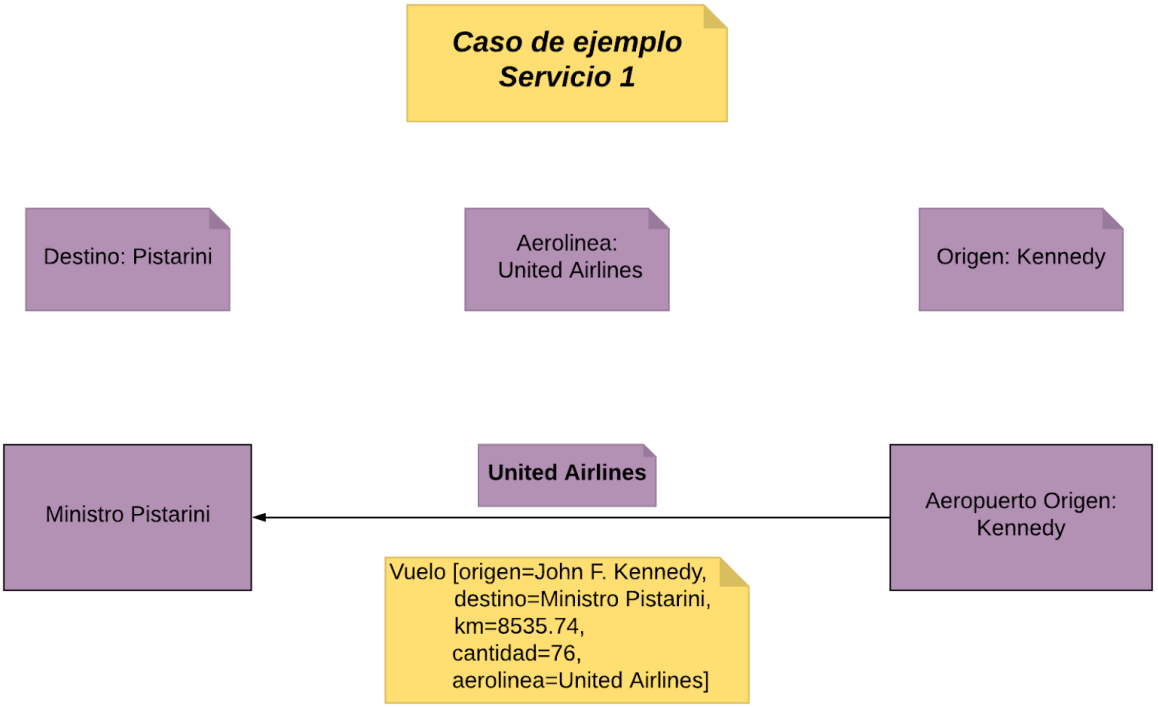
Obtenemos las rutas adyacentes del aeropuerto origen



Iteramos hasta encontrar en la ruta el aeropuerto destino “Ministro Pistarini” y le pedimos a la ruta encontrada las aerolíneas disponibles



Preguntamos por la aerolínea “United Airlines”



El sistema nos devuelve que encontró un vuelo entre los aeropuertos especificados con sus respectivas características.

Servicio 2: Obtener vuelos sin aerolínea

Selección y justificación del algoritmo elegido:

El algoritmo elegido es un DFS con dos colores (muy parecido a un backtracking con poda)

Utilización de los tipos de datos:

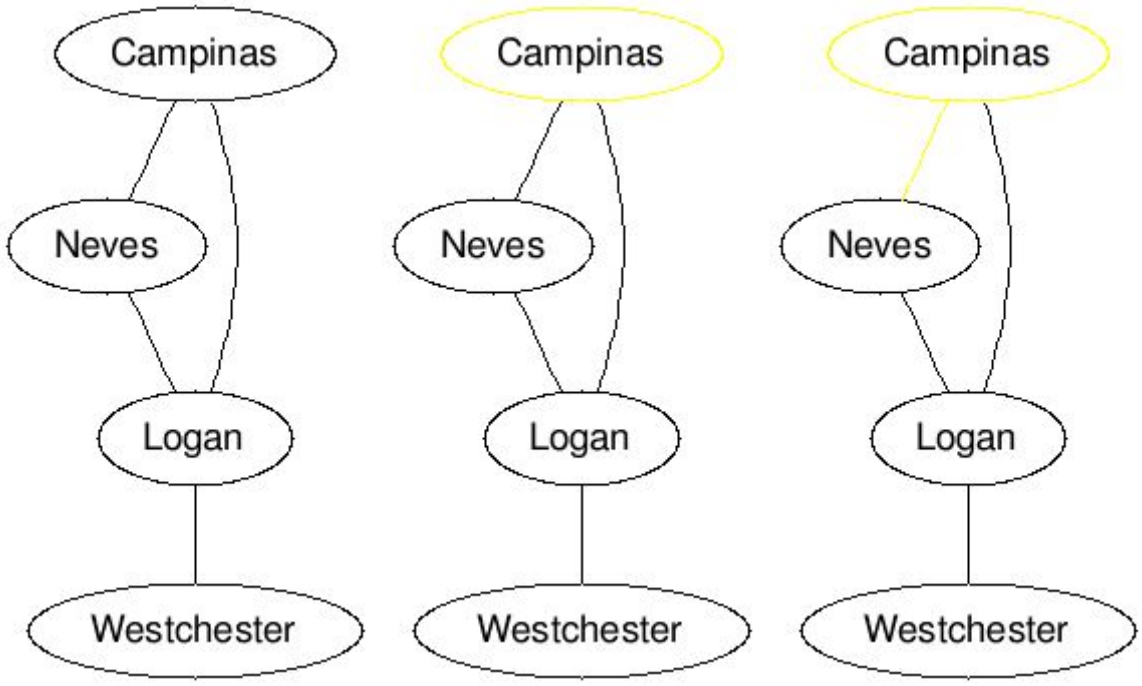
Decidimos utilizar el ArrayList de las adyacencias del aeropuerto origen y los adyacentes de los aeropuerto que encuentre. La utilizamos para recorrerla y así encontrar el destino indicado, como también el ArrayList de aerolíneas que posee la ruta de dichos aeropuertos. El camino que va siguiendo el algoritmo en forma de pila lo representamos con una LinkedList. El retorno del algoritmo es mediante un objeto llamado “VueloConEscala”.

Complejidad del algoritmo utilizado:

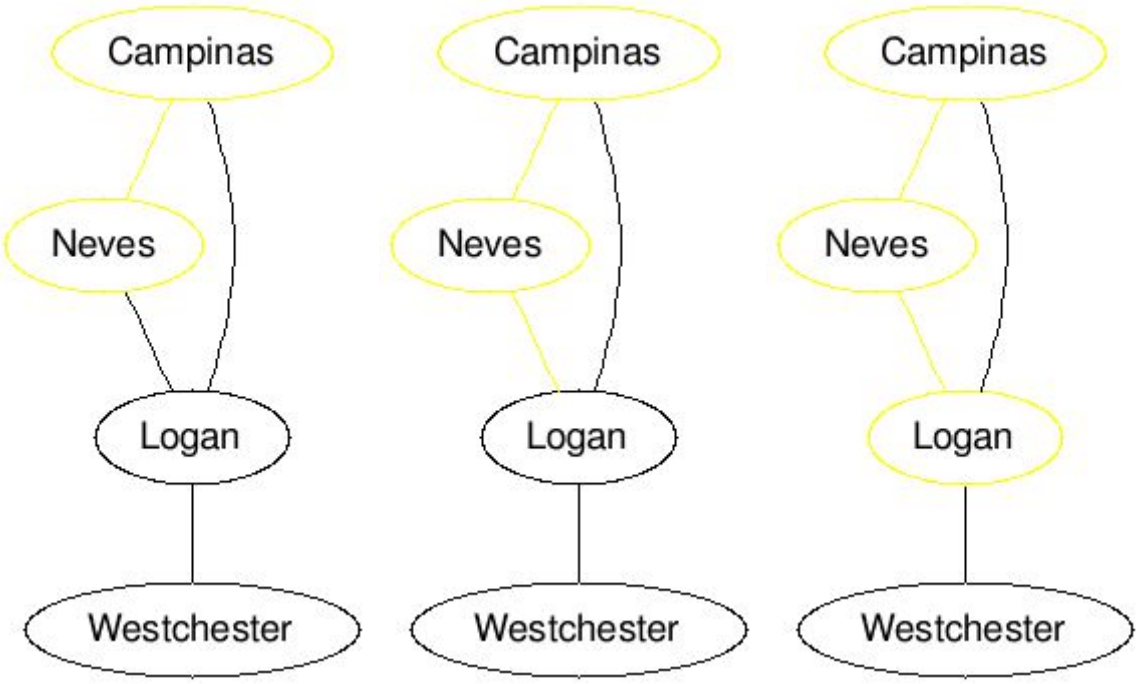
Este servicio se encuentra con una complejidad computacional de $O(x^n)$ siendo ‘x’ el tamaño del grafo y ‘n’ la cantidad de “hijos” de cada nodo.

Caso de ejemplo:

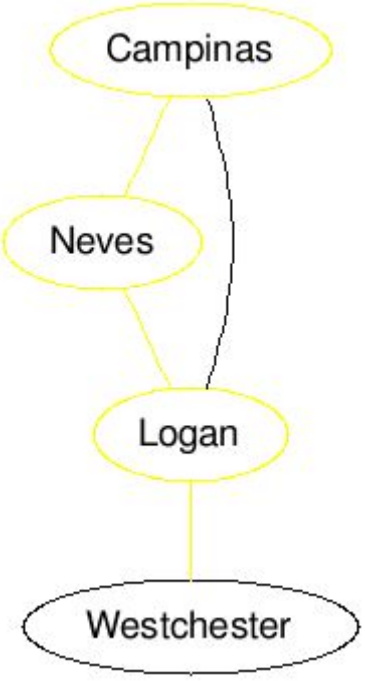
El caso de ejemplo es con origen Campinas y destino Westchester.
El algoritmo pediría todos los adyacentes al origen y empezaría tomando la primera ruta que encuentre, marcando al origen como visitado.



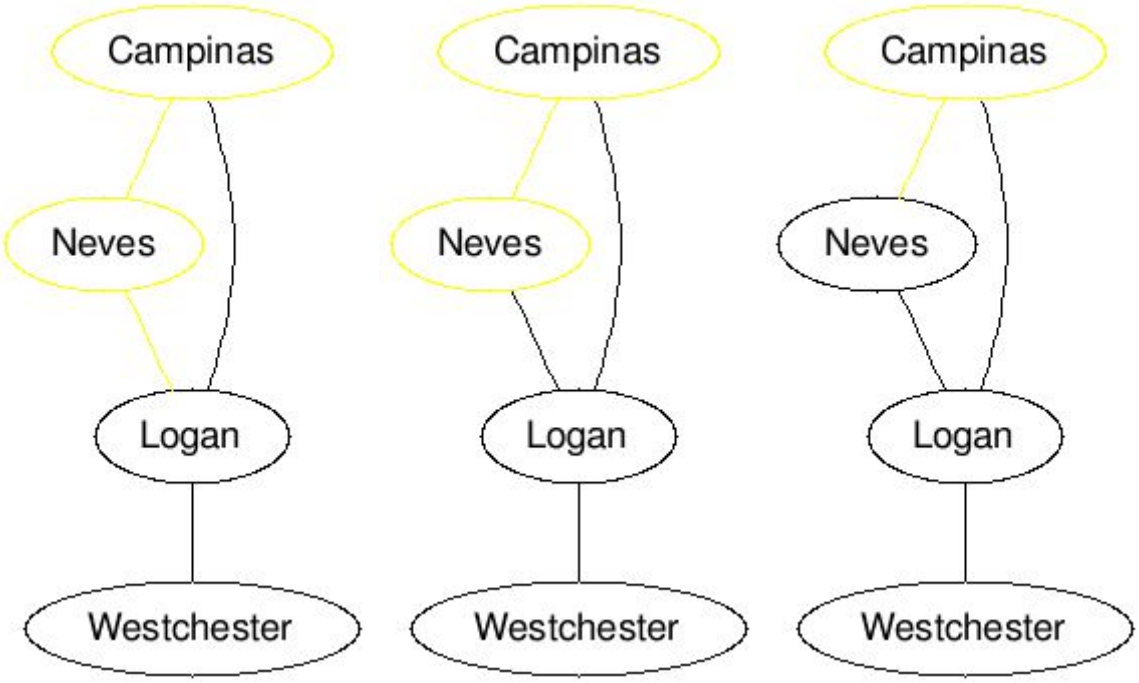
Le pregunta al destino de esa ruta si es el destino solicitado, en este caso no lo sería. Por lo tanto le pediría los adyacentes a Neves y entraría a recursionar. Marcando como visitado a Neves.



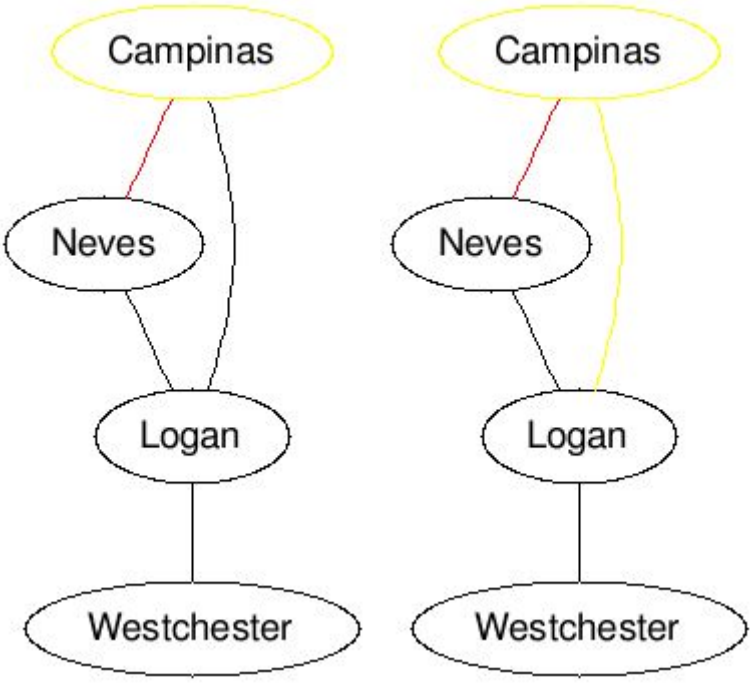
Encuentra la ruta con Logan le pregunta si el destino es el destino que está buscando, como no lo es, sigue recursionando, pidiendo los adyacentes de Logan.



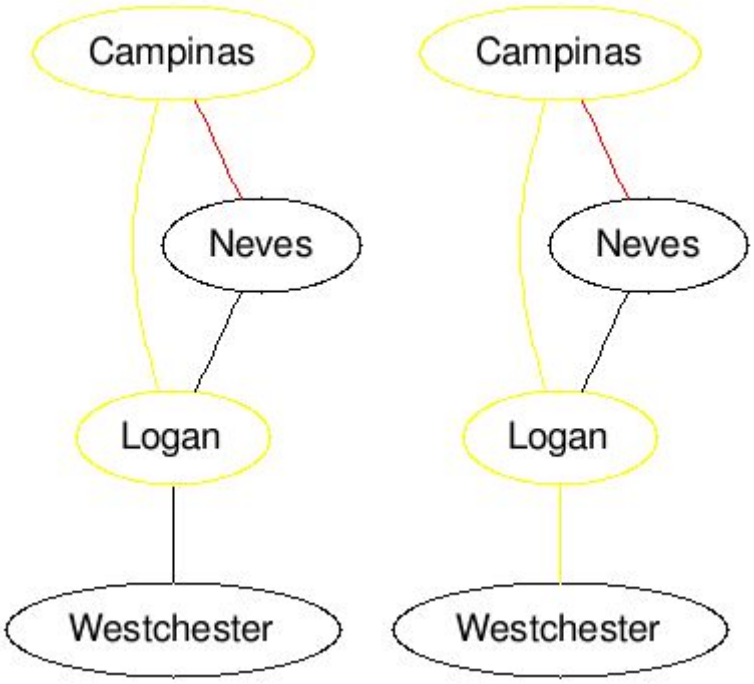
Cuando elige el único adyacente que tiene logan, le pregunta al destino si es el que está buscando, este dirá que si, se agrega a la lista de salida y se comienzan a desmarcar los aeropuertos que se habían marcado como visitados. (En la vuelta, si Logan hubiera tenido más adyacentes se hubieran seguido iterando, lo mismo neves, pero este es un caso de ejemplo reducido)



Como ya recorrió toda la primera rama de campinas, va iterar la siguiente adyacencia, para ver si encuentra otras rutas para el mismo destino solicitado.



Pregunta si ese destino es el buscado, como no lo es, entra a la recursión marcando a Logan como visitado y pidiendo a este sus adyacentes.



El único adyacentes que tiene logan pregunta si su destino es el buscado, este va a serlo por lo tanto se agrega a la lista de retorno.

No hay más adyacentes por recorrer por lo tanto devuelve la lista con la cantidad de rutas que llegan al mismo destino solicitado. Estas serían Campinas -> Neves -> Logan -> Westchester y Campinas -> Logan -> Westchester.

Servicio 3: Vuelos disponibles

Selección y justificación del algoritmo elegido:

Seleccionamos un algoritmo que itera los aeropuertos cargados en el sistema extrayendo específicamente los que residen en el país origen, para luego recorrer las adyacencias de los mismos quedándonos con las rutas que llegan a los aeropuertos en el país destino, por cada ruta encontrada preguntamos por la disponibilidad de sus aerolíneas, así retornando cada posible vuelo entre los países seleccionados por el usuario

Utilización de los tipos de datos:

Para la implementación, usamos los ArrayList que contienen los aeropuertos, las rutas de los mismos y las aerolíneas de cada ruta, para luego recorrer cada ArrayList e ir filtrando por los parámetros deseados

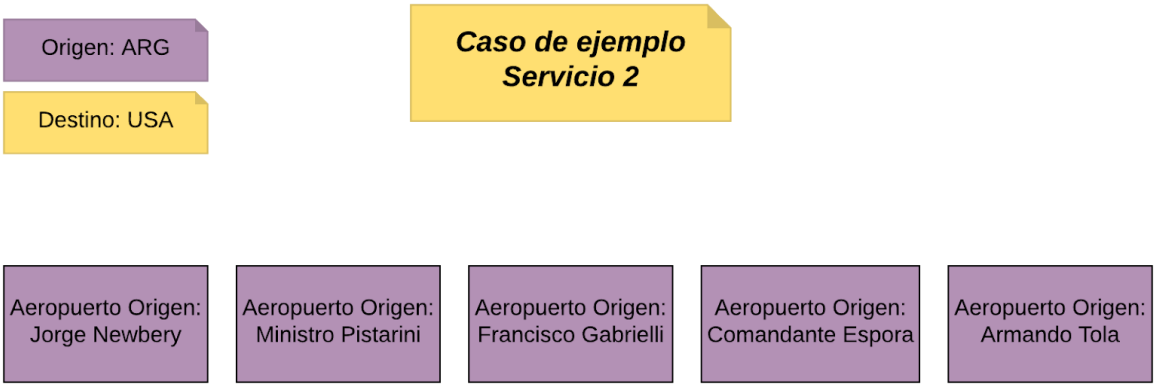
Complejidad del algoritmo utilizado:

La complejidad de este servicio se podría decir que es $O(n^2)$ ya que en un principio se recorre toda la lista de Aeropuertos para encontrar los que son del país de origen. Y luego se les pide los adyacentes a estos, para saber si van al país de destino.

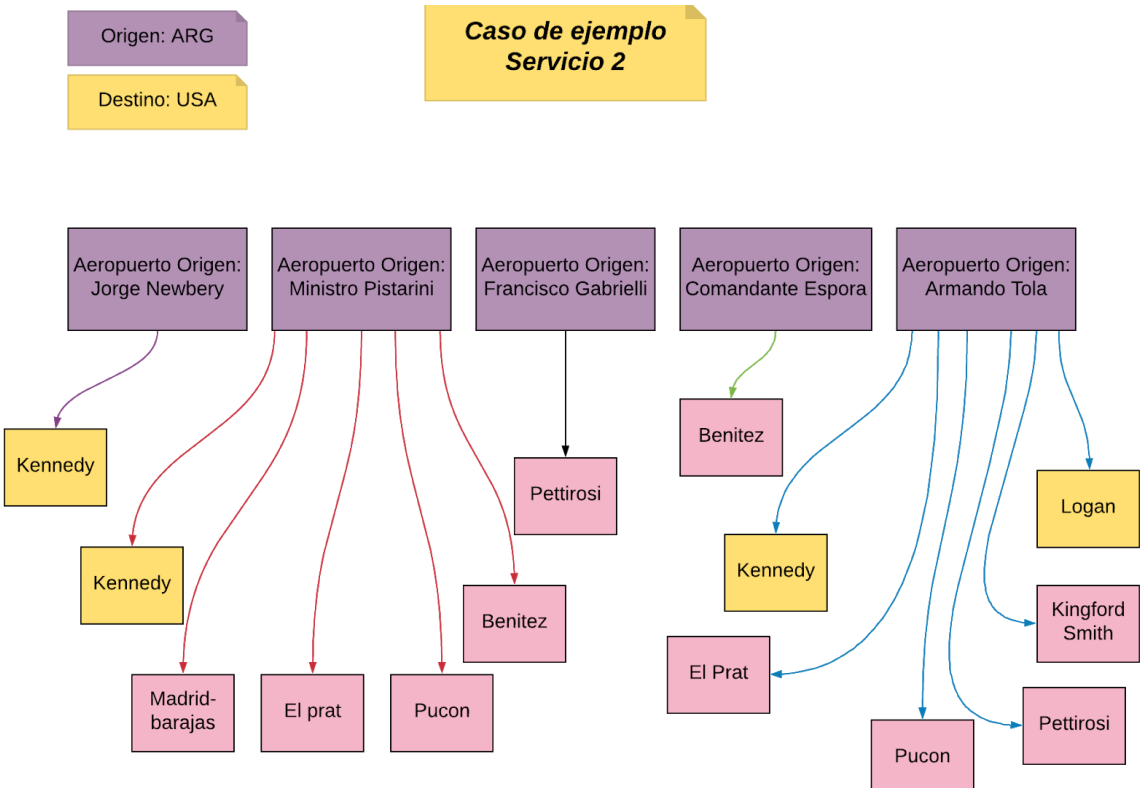
Caso de ejemplo:

```
Bienvenido al Sistema Centralizado de Información a Viajeros Frecuentes
¿Qué operación desea realizar?
1. Listar todos los aeropuertos
2. Listar todas las reservas
3. Servicio 1: Verificar vuelos directos
4. Servicio 2: Obtener vuelos sin aerolinea
5. Servicio 3: Vuelos disponibles
0. Salir del sistema
5
1. ARG
2. CHI
3. BRA
4. USA
5. PER
6. ESP
7. POR
8. AUS
9. PAR
10. PAN
Ingrese país origen
1
Ingrese país destino
4
```

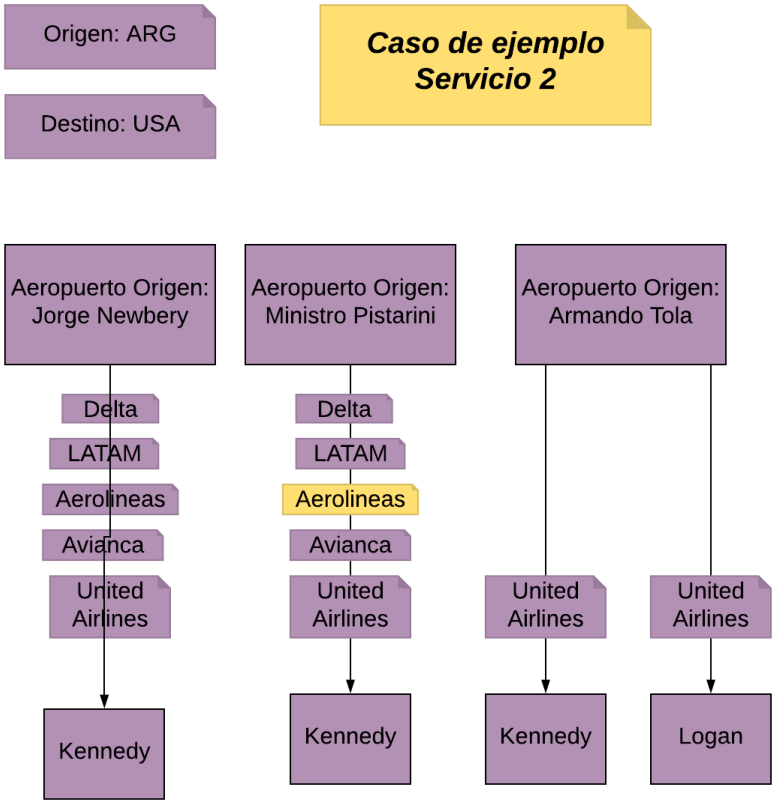
Escogemos en el menú el Servicio 3: Vuelos disponibles.
Luego seleccionamos el país de origen “ARG” y el país destino “USA”.



Recorremos los aeropuertos del sistema y nos quedamos con los del país origen



Obtenemos e iteramos las rutas internacionales de los aeropuertos origen y seleccionamos las que lleguen al país destino



Por cada ruta encontrada iteramos sus aerolíneas y preguntamos por su disponibilidad

```
Vuelo [origen=Jorge Newbery, destino=John F. Kennedy, km=8516.0, cantidad=96,
        aerolinea=United Airlines]

Vuelo [origen=Jorge Newbery, destino=John F. Kennedy, km=8516.0, cantidad=152, aerolinea=Delta]

Vuelo [origen=Jorge Newbery, destino=John F. Kennedy, km=8516.0, cantidad=210, aerolinea=LATAM]

Vuelo [origen=Jorge Newbery, destino=John F. Kennedy, km=8516.0, cantidad=276, aerolinea=Aerolineas]

Vuelo [origen=Ministro Pistarini, destino=John F. Kennedy, km=8535.74, cantidad=76,
        aerolinea=United Airlines]

Vuelo [origen=Ministro Pistarini, destino=John F. Kennedy, km=8535.74, cantidad=210, aerolinea=LATAM]

Vuelo [origen=Ministro Pistarini, destino=John F. Kennedy, km=8535.74, cantidad=276, aerolinea=Aerolineas]

Vuelo [origen=Ministro Pistarini, destino=John F. Kennedy, km=8535.74, cantidad=250, aerolinea=Avianca]

Vuelo [origen=Armando Tola, destino=John F. Kennedy, km=10137.15, cantidad=146, aerolinea=United Airlines]

Vuelo [origen=Armando Tola, destino=Logan, km=10319.77, cantidad=146, aerolinea=United Airlines]
```

Nos quedamos con los vuelos donde haya disponibilidad de asientos. El sistema nos informa que encontró una lista de vuelos disponibles

Funcionamiento del sistema

Cuando inicia el sistema se muestra un menú con los diferentes servicios que ofrece el sistema estos son:

- Listar todos los aeropuertos.
- Listar todas las reservas realizadas.
- Servicio 1: Verificar vuelos directos.
- Servicio 2: Obtener vuelos sin aerolineas.
- Servicio 3: Vuelos disponibles.

Dependiendo del servicio que el usuario elija se va a desplegar una lista con las diferentes opciones en la cual el usuario va a tener que elegir mediante números. Una vez finalizada la petición del servicio, se vuelve a preguntar si se desea utilizar otro servicio.

Conclusiones

Llegando al final de este informe, podemos concluir en varias reflexiones:

Por un lado podemos observar que a la hora de querer mejorar la eficiencia de los algoritmos o las funcionalidades en sí, por lo general esto conlleva más memoria para tener menor cantidad de accesos, más tiempo de implementación y dificultad.

También se podría decir que el tiempo que tuvimos que manejar a la hora de realizar el trabajo se vió claramente reducido por el hecho de las demás responsabilidades con las materias restantes y sus respectivos trabajos y parciales. Lo cual en caso de haber tenido más tiempo podríamos habernos enfocado más, haber optimizado y hecho de manera eficiente ciertas partes del sistema, como por ejemplo, el uso de los ArrayList en la carga de datos del sistema, como especificamos más arriba.