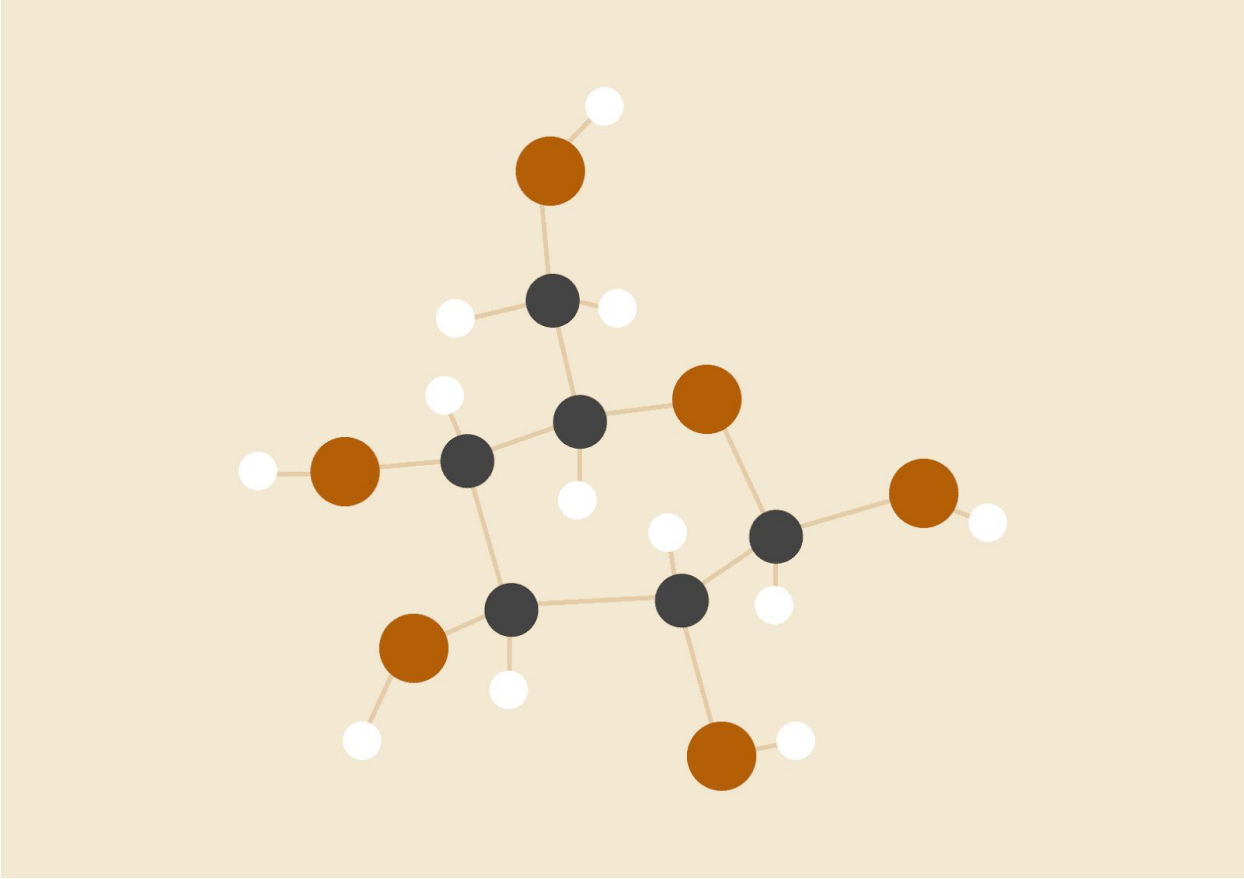
INFORME PARTE 2 TPE PROGRAMACIÓN 3



Maia Venere- venere.maia@gmail.com

Brenda Flamminio- brendulu@gmail.com

Primera parte : correcciones

Comenzamos por corregir el código que lee el archivo CSVReader para luego recorrerlo y añadir los datos al grafo, con esto pudimos testear el resto del código corrigiendo errores de java null pointer exception.

Luego solucionamos el servicio 2 corrigiendo el código utilizando un DFS que al comenzar setea todos los aeropuertos en “blanco” (no visitado), y luego visita al aeropuerto de origen, donde a su vez este visita a cada uno de sus vecinos, etc. En esta “visita” el codigo va verificando si llegó al destino, para así devolver una lista de rutas posibles de origen a destino. La función del DFS retorna una lista de lista de rutas (ArrayList<ArrayList<Ruta>>) con todos los caminos posibles de un origen a un destino, sin tomar la aerolínea determinada.

Modificamos todos los servicios, de VOID a Array<String> o Array<ArrayList<String>> para que en vez de que los muestre por consola, retorne este dato.

Luego terminamos por incorporar el código de CSVWritter, que lo introdujimos en el main.java como dos funciones distintas (una que recibe como parámetro un ArrayList<String> y otra que recibe por parámetro un ArrayList<ArrayList<String>>) donde ambas reciben también por parámetro, un string con el nombre del servicio, para asi la funcion crea un archivo por servicio a realizar, y no los borra dejando así el último ejecutado. Nos pareció la mejor opción ya que anteriormente habíamos diferenciado los servicios por “retorna una lista” o “retorna una lista de lista”, si no deberíamos haber escrito una función de CSVWritter por servicio a mostrar.

SEGUNDA PARTE

Introducción al problema:

Para la segunda parte del enunciado tuvimos que resolver el problema de recorrer todos los nodos del grafo (aeropuertos) sin pasar dos veces por el mismo, y volver al nodo de

donde se partió al comienzo. Debimos resolver dicho problema de dos maneras: Utilizando backtracking y utilizando greedy.

Análisis:

Para realizar la segunda parte del trabajo comenzamos con el algoritmo de Greedy, ya que nos pareció más importante debido a que ya habíamos realizado un ejercicio de Backtracking en la primer parte del enunciado y ya teníamos una idea más arraigada, por ende pensamos que Greedy nos iba a llevar más tiempo que con el algoritmo de Backtracking.

Antes de comenzar el trabajo, reorganizamos los archivos de datasets, de forma tal que exista una solución posible al problema, ya que en el grafo original, existían nodos que solo contenían un único vecino, por ende si se los visitaba, se llegaba a un "callejón sin salida" y no se encontraba solución posible con ninguno de los dos algoritmos.

Luego además de "recortar" estos aeropuertos sin salida, achicamos dicho grafo, para poder tener un seguimiento más claro de los posibles caminos para la resolución del problema. Resultado: Realizamos las pruebas con un grafo de 5 aeropuertos (creando dos archivos nuevos: uno para aeropuertos, y otro para rutas)

- Jorge Newbery;CABA;ARG
- Ministro Pistarini;Ezeiza;ARG
- Comodoro Benitez;Santiago;CHI
- Pucon; Pucon;CHI
- John F. Kennedy;New York;USA

GREEDY:
Para plantear visitar a todos los aeropuertos sin pasar dos veces por el mismo, y volver al origen, realizamos el código con Greedy, seteando todos los aeropuertos en "sin visitar" y calculando los kilómetros entre el aeropuerto de origen y sus vecinos, y optando por el aeropuerto más cercano a este (el de menor kilómetros). Una vez elegida esta opción y elegido el aeropuerto vecino de menor distancia, se visitaba a este aeropuerto y se volvía a hacer el mismo cálculo. Este algoritmo no garantiza la mejor solución, ni garantiza UNA solución. En nuestro código, si encuentra una solución, guarda cada ruta en el ArrayList<Ruta> camino, y luego se muestra aeropuerto por aeropuerto, y si no se llego a encontrar ninguna solución, vacía el ArrayList de camino y no muestra nada.

El código de Greedy nos pareció mucho más limpio y conciso a diferencia del Backtracking, ya que este algoritmo toma una decisión (la elección de un aeropuerto determinado) y no vuelve hacia atrás, si o si continua hacia adelante, eligiendo el siguiente aeropuerto. Debido a que no puede volver para atrás, encuentra una única solución (o no la encuentra, si llega a un nodo que no puede visitar a más vecinos) y esto no garantiza la solución más óptima al problema a resolver, pero si es efectivo en tiempo de desarrollo y de procesamiento.

BACKTRACKING:
Para realizar la solución al problema con este algoritmo, habiendo ya realizado un backtracking del mismo grafo para encontrar todas las rutas entre un origen y un destino, pudimos realizarlo más rápido, pero de todas formas, tardamos más en resolverlo que lo que tardamos con Greedy.
Lo que hicimos fue buscar todas las soluciones que existen para resolver el problema del enunciado, y una vez encontradas todas las disponibles, filtramos solamente la que más nos convenía, es decir la que menos kilómetros realizaba en el total del recorrido, esto lo resolvimos recorriendo todos los caminos posibles y guardando en una variable la opción más recomendable hasta el momento.

El código fue muy similar al del enunciado de la primera parte, incluso más simple ya que solo nos importaba los aeropuertos visitados o no visitados, y no todos los datos de por ejemplo, si había asientos disponibles, si era directo o no era directo.
Dentro de dicha modificación tuvimos que agregar la lógica de chequear que se hayan visitado TODOS los aeropuertos, y que a su vez una vez todos los aeropuertos visitados, chequear que el último aeropuerto tenga como vecino al primer aeropuerto de donde partimos a realizar el Backtracking.

Con respecto a las ventajas de este algoritmo es que nos garantiza la solución más óptima ya que busca todas las opciones disponibles, y elegimos la que más nos conviene para resolver el problema. Esto lo realiza a costa de que el tiempo de procesamiento es mayor debido a que en la alternativa de Greedy realiza solo una solución, mientras que éste busca todas.