Trabajo Práctico 3

El Trabajo Práctico número 3 es de elaboración grupal, y tiene fecha de entrega para el ** 5/12**.

Contenido

Introducción

Debido al éxito rotundo del nuevo software en el aeropuerto de Algueiza, las distintas aerolíneas del mundo (!) nos han pedido que desarollemos un programa que permita dar información sobre las conexiones entre los aeropuertos. Son tantos, pero tantos los pedidos que será necesario limitar la cantidad de *features* a implementar para poder llegar con la fecha de entrega. Eso sí, será necesario lograr que el programa tenga el suficiente valor para poder dejar a todos medianamente contentos.

Análisis de la situación

Aerolíneas de distintas partes del mundo, requieren distintos tipos de funcionalidades:

- Todas las aerolíneas quieren que los clientes puedan buscar el camino más rápido, el más barato, o el de menor cantidad de escalas, a gusto.
- Para poder entender dónde conviene invertir en mayor infraestructura, se quiere obtener cuáles son los aeropuertos más importantes, teniendo en cuenta en cuáles es más frecuente terminar haciendo escalas para poder ir entre cualquier parte del mundo a otra.
- Una nueva aerolínea (RositAIR Airlines, con sede principal en el aeropuerto de Lanús) quiere poder conectar absolutamente todo el mundo, pero quiere utilizar las rutas más baratas posibles para ello. Entre tantos datos que hay, no saben por dónde empezar!
- Otras aerolíneas pidieron poder ofrecer un viaje cultural: que se pueda indicar qué ciudades se quieren conocer, y además si hay alguna ciudad que deba visitarse antes que otra (para poder disfrutar más del viaje).
- Muchas de las aerolíneas quisieran poder exportar un mapa que nos permita visualizar las rutas de vuelo.

Datos disponibles

Se cuenta con los datos de los aeropuertos de Estados Unidos, así como de más de medio millón de vuelos durante el año 2015.

Una consultora uso un innovador script para parsear dichos datos en los siguientes archivos:

- El archivo aeropuertos.csv con el formato:

  ```
  ciudad,codigo_aeropuerto,latitud,longitud
  ```

- El archivo vuelos.csv con el formato:

  ```
  aeropuerto_i,aeropuerto_j,tiempo_promedio,precio,cant_vuelos_entre_aeropuertos
  ```

Adicionalmente, para poder realizar algunas pruebas rápidas (especialmente en los algoritmos que más tiempo puedan demorar) se crea este otro set con aeropuertos y vuelos inventados.

Aclaraciones

- Una ciudad puede tener **uno o más aeropuertos** asociados.
- El tiempo promedio es el promedio (redondeado) de todos los tiempos de los vuelos entre ese par de ciudades.
- Los precios han sido inventados completamente (puede verse en el script generador) y no tienen ninguna relación con la realidad. El curso no se hace responsable por las recomendaciones que pueda hacer alguno de los programas implementados.

- Sería de mayor interés tener los datos disponibles de todo el mundo, o de diversos países. De todas formas, hay que considerar que se tratan de casi 6 millones de vuelos sólo considerando un país, en un año.
- Más allá de tener los datos de un país en especial, el programa debería funcionar para cualquier parte del mundo.

Implementación

El trabajo puede realizarse en lenguaje a elección, siendo aceptados Go, Python y C, y cualquier otro a ser discutido con el corrector asignado.

El trabajo consiste de 3 partes:

1. El TDA Grafo, con sus primitivas.
2. Una biblioteca de funciones de grafos, que permitan hacer distintas operaciones sobre un grafo de aeropuertos del mundo, sin importar cuál sea el aeropuerto ni país al que pertence.
3. El programa `FlyCombi` que utilice tanto el TDA como la biblioteca para poder implementar todo lo requerido.

El programa debe recibir por parámetro y cargar en memoria el set de datos (`$ ./flycombi aeropuertos.csv vuelos.csv`) y luego solicitar el ingreso de comandos por entrada estándar, del estilo `<comando> 'parametro'`. Notar que esto permite tener un archivo de instrucciones a ser ejecutadas (i.e. `$ ./flycombi aeropuertos.csv vuelos.csv < entrada.txt`).

Caminos mínimos

Camino más barato y camino más rápido

- Comando: `camino_mas`
- Parámetros: `barato` ó `rapido`, `origen` y `destino`. Origen y destino son **ciudades**.
- Utilidad: nos imprime una lista con los **aeropuertos** (código) con los cuales vamos de la **ciudad** `origen` a la **ciudad** `destino` de la forma más rápida o más barata, según corresponda. Tener en cuenta que tanto la ciudad `origen` como la `destino` pueden tener más de un aeropuerto, y nos interesa la mejor forma (rápida o barata) entre todas las combinaciones posibles.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(F \log(A))$, siendo A la cantidad de aeropuertos, y F la cantidad de vuelos entre aeropuertos (sin contar frecuencia). Podemos considerar que la cantidad de aeropuertos que tenga cada ciudad es despreciable.
- Ejemplo: Entrada:

  ```
  camino_mas rapido,San Diego,New York
  camino_mas barato,San Diego,New York
  ```

  Salida:

  ```
  SAN -> JFK
  SAN -> DEN -> JFK
  ```

Camino con menor cantidad de escalas

- Comando: `camino_escalas`
- Parámetros: `origen` y `destino`.
- Utilidad: nos imprime una lista con los **aeropuertos** (código) con los cuales vamos de la **ciudad** `origen` a la **ciudad** `destino` con la menor cantidad de escalas. Tener en cuenta que tanto la ciudad `origen` como la `destino` pueden tener más de un aeropuerto, y nos interesa la mejor forma (en cantidad de escalas) entre todas las combinaciones posibles.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(F + A)$, siendo A la cantidad de aeropuertos, y F la cantidad de vuelos entre aeropuertos (sin contar frecuencia). Podemos considerar que la cantidad de aeropuertos que tenga cada ciudad es despreciable.
- Ejemplos: Entrada:

  ```
  camino_escalas San Diego,New York
  ```

  Salida:

  ```
  SAN -> JFK
  ```

Aeropuertos más importantes

Utlizaremos este comando para poder obtener cuáles son los aeropuertos más importantes. Esto lo podemos hacer obteniendo cuáles son los aeropuertos más centrales. Es necesario tener en cuenta además la frecuencia de vuelos: no es lo mismo un aeropuerto que se conecte contra todos con un vuelo al año, que uno que se conecte al 80% con muchos vuelos al año. ¡Cuantos más vuelos, mejor!

Dejamos un apunte para la explicación de qué es y cómo se calcula el Betweeness Centrality.

- Comando: `centralidad`
- Parámetros: `n`, la cantidad de aeropuertos más importantes a mostrar.

- Utilidad: nos muestra los `n` aeropuertos más centrales/importantes del mundo, de mayor importancia a menor importancia.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(A \times F \log(A))$.
- Ejemplo: Entrada:

  ```
  centralidad 5
  ```

  Salida:

  ```
  ATL, ORD, LAX, DFW, DEN
  ```

Optimización de rutas para nueva aerolínea

- Comando: `nueva_aerolinea`.
- Parámetros: `ruta`, ruta del archivo de salida.
- Utilidad: exportar un archivo con las rutas que permitan implementar una nueva aerolínea tal que se pueda comunicar a todos los aeropuertos (en esta primera iteración, sólo de Estados Unidos) con dicha aerolínea, pero que el costo total de la licitación de las rutas aéreas sea mínima. Se considera que el costo de las rutas es proporcional al costo de los pasajes, por lo que se puede trabajar directamente con dicho costo. Se busca que la nueva aerolínea pueda llegar a todos los **aeropuertos** de alguna forma. La salida de este comando debe ser un archivo con el mismo formato del archivo `vuelos.csv`, pero únicamente con las rutas de vuelo a utilizar por ésta aerolínea.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(F \log(A))$.
- Ejemplo: Entrada:

  ```
  nueva_aerolinea RositAIR.csv
  ```

  Salida (aparte del archivo creado):

  ```
  OK
  ```

Itinerario cultural

- Comando: `itinerario`
- Parámetros: `ruta`, la ruta el archivo del itinerario.
- Utilidad: El archivo de `ruta` tiene el formato:

  ```
  ciudad_1,ciudad_2,ciudad_3, ...,ciudad_N
  ciudad_i,ciudad_j
  ```

  La primera línea indica las ciudades que se desean visitar. Las siguientes indican que la `ciudad_i` debe visitarse **sí o sí** antes que la `ciudad_j`. Se debe:

  1. Imprimir el orden en el que deben visitarse dichas ciudades.
  2. Imprimir el camino mínimo en tiempo o escalas (según lo que se haya implementado en ese caso) a realizar.
- Complejidad: El cálculo de la obtención del itinerario debe realizarse en $\mathcal{O}(I + R)$, siendo I la cantidad de ciudades a visitar, y R la cantidad de restricciones impuestas. Luego, el cálculo de los caminos debe realizarse en $\mathcal{O}\left(I \times F \log(A)\right)$ o bien $\mathcal{O}\left(I \times (A + F)\right)$, dependiendo de si se trata de un caso u otro.
- Ejemplo, trabajando con itinerario_ejemplo.csv, Entrada:

  ```
  itinerario itinerario_ejemplo.csv
  ```

  Salida:

```
Asheville, Newburgh, Oakland, Wilmington, Jackson, Garden City, Kalispell, Hobbs, Aberdeen, Traverse City,
AVL -> ATL -> DTW -> SWF
SWF -> FLL -> OAK
OAK -> ABQ -> MCO -> ILG
ILG -> MCO -> DEN -> JAC
JAC -> DFW -> GCK
GCK -> DFW -> DEN -> FCA
FCA -> ATL -> IAH -> HOB
HOB -> IAH -> MSP -> ABR
ABR -> MSP -> TVC
TVC -> MSP -> RHI
RHI -> MSP -> LAX -> CLD
CLD -> LAX -> DFW -> ACT
ACT -> DFW
DFW -> JFK
JFK -> SLC -> WYS
```

Si bien en este ejemplo no llega a notarse, no tiene por qué ser el mismo aeropuerto el de llegada que el de salida (si deben ser aeropuertos de la ciudad en la cual se está parando).

Exportar a archivo KML

- Comando: `exportar_kml`.
- Parámetros: `archivo`.
- Utilidad: exporta el archivo KML con la ruta del último comando ejecutado (que incluya algún camino, o rutas áereas). Esto aplica para los comandos de caminos mínimos. Contamos con un apunte sobre cómo crear, usar y visualizar archivos KML.
- Complejidad: Este comando debe ejecutar en $\mathcal{O}(A + F)$.
- Ejemplo: Entrada:

  ```
  camino_mas rapido,San Diego,New York
  exportar_kml kml-ejemplo.kml
  ```

  Salida:

  ```
  SAN -> JFK
  OK
  ```

Entrega

Adicionalmente a los archivos propios del trabajo práctico debe agregarse un archivo `entrega.mk` que contenga la regla `flycombi` para generar el ejecutable de dicho programa (sea compilando o los comandos que fueren necesarios). Por ejemplo, teniendo un TP elaborado en Python, podría ser:

```
flycombi: flycombi.py grafo.py biblioteca.py
        cp flycombi.py flycombi
        chmod +x flycombi
```

Importante: En caso de recibir un error `FileNotFoundError: [Errno 2] No such file or directory: './flycombi': './flycombi'`, tener en cuenta que para el caso de enviar código escrito en Python es necesario además indicar la ruta del intérprete. Esto puede hacerse agregando como primera línea del archivo principal (en el ejemplo, sería `flycombi.py`) la línea: `#!/usr/bin/python3`.

Si el TP fuera realizado en Go, un posible ejemplo del archivo podría ser:

```
flycombi:
    cd mi_implementacion_tp3; go build -o ../flycombi
```

Criterios de aprobación

El código entregado debe ser claro y legible y ajustarse a las especificaciones de la consigna. Debe compilar sin advertencias y correr sin errores de memoria.

La entrega incluye, obligatoriamente, los siguientes archivos de código:

- el código del TDA Grafo programado, y cualquier otro TDA que fuere necesario.
- el código de la solución del TP.

La entrega se realiza en forma digital a través del sistema de entregas, con todos los archivos mencionados en un único archivo ZIP.