Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2233 — Programación Avanzada 2020-2

5 de Noviembre de 2020
Actividad Sumativa

Actividad Sumativa 03

Estructuras Nodales II: Grafos

Entrega

- **Lugar:** En su repositorio privado de GitHub, en la **carpeta** `Actividades/AS03/`
- **Hora del *push*:** 16:50

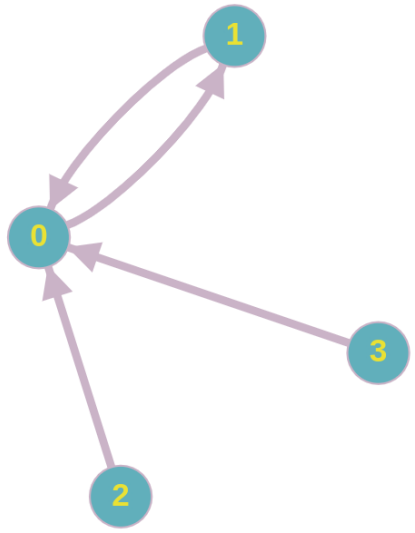
Importante: Antes de comenzar, comprueba que Git este funcionando correctamente en tu repositorio privado. Para esto, **sube los archivos base de la actividad de inmediato** (*add*, *commit*, *push*). Se espera que en esta actividad (así como en las demás actividades y tareas) utilices Git a lo largo de **todo tu desarrollo** como una herramienta, no sólo como un método de entrega. Es por esto que recomendamos enfáticamente que vayas subiendo tus cambios constantemente (***push***), ya que **problemas de último minuto** relacionados con la entrega y Git **no serán considerados**.

Introducción

¡El mundo está abriendo sus puertas nuevamente! Mejoras en las cifras de salud han hecho que, en varios países del mundo, los aeropuertos estén comenzado a funcionar nuevamente. Sin embargo, el traslado de un gran numero de personas puede generar contagios y, por ende, problemas de salud para un país. Se te ha encargado a ti, fundador de **DCConexiones**, la gran compañía en modelación de redes, administrar las distintas redes de aeropuertos que existen en el mundo para que estas cumplan con los protocolos de salud. Para cumplir esta tarea, crearás un simulación que te permita controlar las redes.

Flujo del programa

Antes de entender el flujo del programa, es importante considerar unas definiciones. En este grafo, los nodos representan aeropuertos y las aristas representan las conexiones (direccionales) entre ellos. Una **conexión** es una relación (asimétrica) entre 2 aeropuertos, indicando el sentido de la relación. Una **red** es una agrupación de aeropuertos, que se relacionan con al menos un aeropuerto en común. Por ejemplo, en la siguiente imagen, tenemos una **red de 4 aeropuertos**, con un total de **4 conexiones** (de 2 a 0, de 3 a 0, de 0 a 1, y de 1 a 0), representados de la siguiente manera:



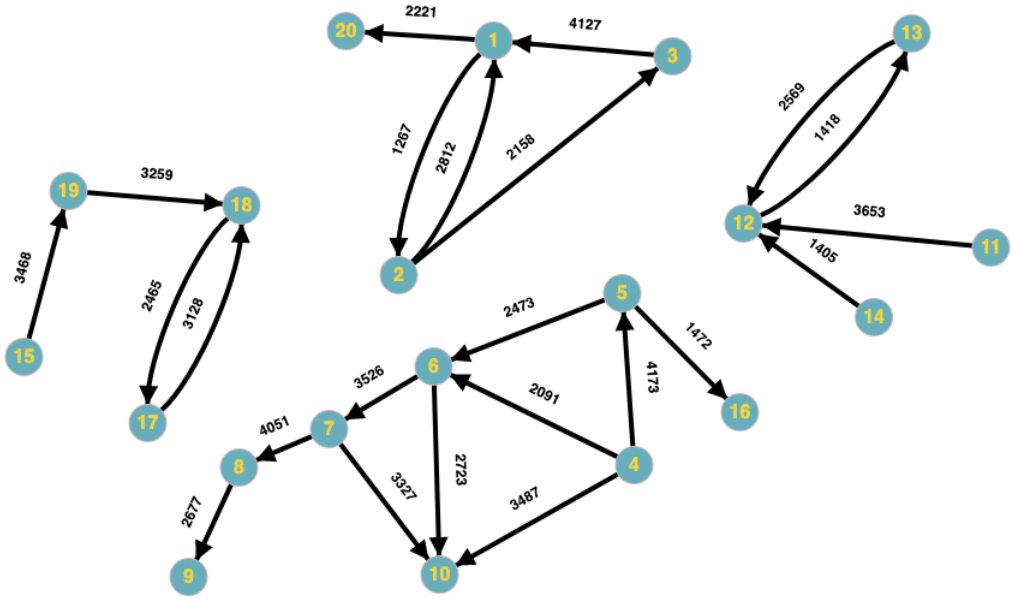
Ahora que tenemos estas definiciones, podemos pasar a explicar el flujo del programa.

El programa consiste representar las redes de aeropuertos del mundo como un grafo dirigido. Los aeropuertos tienen conexiones entre sí, y cada una de ellas tiene un cierto riesgo asociado, representado como un número de **infectados**. Este número equivale a una estimación (realizada por el departamento de Posibilidades y Estadísticas) sobre la cantidad de infectados que puede generar un vuelo desde un aeropuerto hacia otro.

Para el seguro funcionamiento de una red, todos sus aeropuertos deben tener una suma total de infectados estimados menor al `UMBRAL`. Esto significa que, al recorrer todos los Aeropuertos posibles desde un Aeropuerto específico en una red, la suma de los infectados presentes en cada `Conexion` debe ser menor al `UMBRAL` definido. Volviendo el ejemplo anterior, si partimos desde el nodo 2, se debe cumplir que la suma del valor de `infectados` de las conexiones $2 \to 0$, $0 \to 1$ y $1 \to 0$ sea menor al `UMBRAL`. Esto se debe cumplir para todos los nodos del grafo.

Al iniciar el programa, se poblará el grafo, representado en `RedesMundiales`, con los datos iniciales, los cuales ya cumplen con la condición del `UMBRAL`. Luego se deberá agregar los `Aeropuertos` candidatos y sus `Conexiones`. Por cada nueva `Conexion` generada por un nuevo `Aeropuerto` agregado, se deberá verificar que los infectados totales de cada nueva ruta generada por la adición de esta nueva `Conexion` se encuentren bajo el `UMBRAL`. En caso contrario, se elimina esta `Conexion` de la red. Toda `Conexion` nueva debe ser verificada (y eliminada si fuese necesario), **antes** de agregar la siguiente `Conexion`.

Para facilitar tu tarea, a continuación se deja una figura que representa el grafo con todas las redes que se encuentran en `RedesMundiales` y los infectados en cada conexión, a partir de los datos reales de los archivos entregados:



Archivos

Deberás trabajar dentro del siguiente archivo:

- `redes_mundiales.py`: Contiene la clase `RedesMundiales`, que **deberás completar**. Este será el archivo principal que debes ejecutar. Además en este archivo contiene el parámetro `UMBRAL`. Este **debe** ser utilizado durante el programa.

Los siguientes archivos **no deben ser modificados**:

- `entidades.py`: Este archivo contiene las clases `Aeropuerto` y `Conexion`, que representan a los vértices y aristas de las redes de aeropuertos.
- `cargar_archivos.py`: Este archivo contiene las funciones generadoras `cargar_aeropuertos` y `cargar_conexiones`, que reciben las rutas a los archivos `*.txt` y generan las tuplas con los valores para instanciar los aeropuertos y las conexiones. Deberás usar estas funciones en tu programa.

  La función `cargar_aeropuertos` genera tuplas de la forma `(aeropuerto_id, nombre)`, donde el primer valor es un identificador del aeropuerto (`int`) y el segundo el nombre del mismo (`str`). La función `cargar_conexiones` genera tuplas de la forma `(id_partida, id_llegada, infectados)`, con los identificadores de los aeropuertos conectados (ambos `int`, según el sentido de Conexion) y la cantidad de infectados posibles al agregar la Conexion (`int`).
- `datos/aeropuertos.txt`: Este archivo contiene los aeropuertos ya considerados en la simulación.
- `datos/aeropuertos_candidatos.txt`: Este archivo contiene los aeropuertos que serán evaluados en la simulación.
- `datos/conexiones.txt`: Este archivo contiene las conexiones asociadas a los aeropuertos ya considerados en la simulación.

- `datos/conexiones_candidatas.txt`: Este archivo contiene las conexiones asociadas a los aeropuertos que serán evaluados en la simulación.

Entidades

La clase `Aeropuerto` representa a los **nodos** del grafo `RedesMundiales`. Esta clase ya viene implementada, por lo que **no deberás modificarla**, e incluye los siguientes métodos:

- `def __init__(self, aeropuerto_id, nombre)`: define los siguientes atributos:
  - `self.id`: Un `int` que representa al `Aeropuerto`. Se recibe del argumento `aeropuerto_id`.
  - `self.nombre`: Un `str` que representa al nombre del `Aeropuerto`. Se recibe del argumento `nombre`.
  - `self.conexiones`: Una `list`, inicialmente vacía, que va a contener a todas las instancias de la clase `Conexion` las cuales su atributo `aeropuerto_inicio_id` sea igual al de este aeropuerto, o sea, aquellas conexiones que se inician en este aeropuerto.
- `def __repr__(self)`: retorna un `str` representativo de la instancia, que contiene su valor de `self.id`.

La clase `Conexion` representa a las **aristas** del grafo `RedesMundiales`. Recibe los identificadores de los nodos `aeropuerto_inicio_id` y `aeropuerto_llegada_id` con los `infectados` potenciales correspondientes al tramo. Esta clase ya viene implementada, por lo que **no deberás modificarla**, e incluye los siguientes métodos:

- `def __init__(self, aeropuerto_inicio_id, aeropuerto_llegada_id, infectados)`: recibe los argumentos indicados para definir los siguientes atributos:
  - `self.aeropuerto_inicio_id`: es un `int` que representa el id del aeropuerto de inicio
  - `self.aeropuerto_llegada_id`: es un `int` que representa el id del aeropuerto de llegada.
  - `self.infectados`: un `int` que representa los infectados potenciales de un vuelo entre los aeropuertos `aeropuerto_inicio` y `aeropuerto_llegada`. Cabe destacar que este valor es válido solo en una dirección, es decir, para un vuelo entre `self.aeropuerto_llegada_id` y `self.aeropuerto_inicio_id` podría ser distinto.
- `def __repr__`: retorna un `str` representativo de la conexión entre los aeropuertos.

Parte I - RedesMundiales

La clase `RedesMundiales`, definida en el archivo `redes_mundiales.py`, representa al grafo principal del programa. En ella deberás completar los siguientes métodos:

- `def __init__(self)`: Este método define el atributo `self.aeropuertos` como un diccionario vacío. Este diccionario tendrá el identificador de cada aeropuerto como llave y su aeropuerto correspondiente como valor (una instancia de la clase `Aeropuerto`). **No debes modificar este método**.
- `def agregar_aeropuerto(self, aeropuerto_id, nombre)`: Deberás completar este método, tal que se cree una instancia de la clase `Aeropuerto` con los argumentos `aeropuerto_id` y `nombre` recibidos, y esta sea agregada a `self.aeropuertos`.
- `def agregar_conexion(self, aeropuerto_id_partida, aeropuerto_id_llegada, infectados)`: Este método recibe `aeropuerto_id_partida`, que es el `id` del `Aeropuerto` de partida de la conexión

y `aeropuerto_id_llegada`, que es el `id` del de llegada. Deberás completar este método, tal que se verifique que ambos aeropuertos se encuentran en `self.aeropuertos` y que la conexión no existe. En caso de cumplirse estas condiciones, se deberá crear la conexión entre ambos `Aeropuertos`. Para esto basta actualizar las listas `conexiones` del `Aeropuerto` de partida.

- `def infectados_generados_desde_aeropuerto(self, aeropuerto_id)`: Debes completar este método tal que recorra todas las `Conexiones` posibles partiendo desde el aeropuerto con id `aeropuerto_id`. Debes encontrar el valor total de `infectados` en este recorrido, sumando el valor de `infectados` de cada `Conexion`. Finalmente, debes imprimir un mensaje que contenga el nombre del Aeropuerto de inicio y la cantidad total de infectados, y retornar dicha cantidad. Por ejemplo:

```
"La cantidad estimada de infectados generados por el aeropuerto Anor londo Airport es de 21768"
```

- `def verificar_candidatos(self, ruta_aeropuertos_candidatos, ruta_conexiones_candidatas)`: En este método se deberá agregar al grafo los aeropuertos candidatos con sus respectivas conexiones (cargar desde `aeropuertos_candidatos.txt` y `conexiones_candidatas.txt`, ambos en la carpeta `datos/`) si es que cumplen con la condición de no superar el `UMBRAL` definido. Cada vez que se agregue una nueva conexión, se debe verificar si la cantidad de infectados generados desde su nodo inicial (utiliza el método `infectados_generados_desde_aeropuerto`) es menor que el `UMBRAL` definido en el programa. Si la condición no se cumple, se debe eliminar la conexión e imprimir un mensaje informando que ésta no cumplió las reglas de seguridad, de la siguiente manera:

```
"La conexión 29->11 rompe las reglas de seguridad"
```

Esta clase también incluye los siguientes métodos que ya vienen implementados y **no deberás modificar**:

- `def cargar_red(self, ruta_aeropuertos, ruta_conexiones)`: Este método crea la red de aeropuertos. A partir de `cargar_aeropuertos` y `cargar_conexiones`, se agregan los aeropuertos y conexiones a la red utilizando `agregar_aeropuerto` y `agregar_conexión` respectivamente.
- `def eliminar_conexion(self, conexion)`: Este método recibe una instancia de `Conexion`, verifica que la conexión esté presente en el aeropuerto de partida y, en caso de cumplirse esto, se eliminará la conexión entre ambos `Aeropuertos`, actualizando el atributo `conexiones` del aeropuerto de partida. En caso de no cumplirse, no hace nada.
- `def eliminar_aeropuerto(self, aeropuerto_id)`: Este método verifica que el aeropuerto correspondiente al identificador recibido está presente en `self.aeropuertos` y, si existe, lo elimina. En caso de intentar eliminar un aeropuerto que todavía tiene conexiones, se levanta una excepción.

Parte II - Bonus

Para obtener el bonus deberás implementar correctamente el siguiente método en la clase `RedesMundiales`:

- `def escala_mas_corta(self, id_aeropuerto_1, id_aeropuerto_2)`: Este método recibe los identificadores para un aeropuerto de partida y un aeropuerto de llegada. Este método puede retornar o simplemente imprimir las partes de la ruta más corta posible entre estos dos aeropuertos (puedes retornar/imprimir los nodos o las conexiones). En caso, de que no se pueda llegar del aeropuerto de partida al aeropuerto de llegada, no deberás, retornar ni imprimir nada. **Puedes agregar la cantidad de argumentos que te parezcan necesarios para completar este método**.

Requerimientos

- (1.00 pts) Define correctamente el método `agregar_aeropuerto`.
- (1.25 pts) Define correctamente el método `agregar_conexion`.
- (1.75 pts) Define correctamente el método `infectados_generados_desde_aeropuerto`.
- (2.00 pts) Define correctamente el método `verificar_candidatos`.