# Trabajo Práctico 3

El Trabajo Práctico número 3 es de elaboración grupal, y tiene fecha de entrega para el **20/12**.

## Contenido



## Introducción

Luego de una ardua investigación por parte del FBI sobre una red de delincuentes, lograron encontrar una serie de mensajes enviados que los pueden llegar a comprometer. Lograron obtener los datos de más de 420.000 mensajes enviados por distintos medios: algunos son mensajes de texto, otros llamadas telefónicas, otros mensajitos por diversas redes sociales, mails, etc. Todos estos conectan a 260.000 sospechosos a nivel mundial. El problema que ahora enfrentan es que no saben bien cómo trabajar con estos datos, por lo que le pidieron a alumnes de la FIUBA si pueden ayudar con esta tarea.

## Análisis de la situación

Los agentes del FBI se contactaron pidiendo que se implementen las siguientes funcionalidades (o *features*). Cada uno de estos ítems se corresponde a una de las funcionalidades pedida en la segunda mitad de la consigna, y quizá sea conveniente una lectura a la par:

1. Tienen a algunos de los delincuentes menores bien vigilados, y saben que se van a contactar con los demás. Eventualmente, si siguen el rastro pueden acabar encontrando y atrapando a algún *pez gordo* de la lista de más buscados del mundo. Por esto, quieren poder obtener la mínima cantidad de seguimientos de personas que debe realizar un agente para dar con dicho *pez gordo*. Por ejemplo, ese tal *Toni, el Gordo* es un delincuente muy buscado en una ciudad de Estados Unidos.
2. Con tantos datos obtenidos, no tienen forma de definir cuáles son los delincuentes más importantes, para saber contra quiénes conviene apuntar en esta operación. En función de esto, les interesa poder tener una lista de cuáles pueden ser esos delincuentes. El problema es que son tantos, que saben que un algoritmo exacto puede tomar demasiado tiempo en calcular esto, y temen que los delincuentes se enteren que el FBI ha obtenido todos estos datos, y escapen. Por lo tanto, aceptan una buena aproximación.
3. Otra cosa que desean es, dada una lista de delincuentes a los que están siguiendo en estos momentos, determinar cuál es el mínimo esfuerzo para encontrar a alguno de los delincuentes más importantes.
4. Algo más que les permitiría entender como está conformada esta red de delincuentes sería poder determinar comunidades dentro de esta red.
5. Además de esto, cuentan con agentes infiltrados. Les interesa saber, si alguno de estos agentes comienza una noticia/rumor, hasta dónde pueda llegar.
6. En conjunto con lo anterior, les interesa poder saber si existe algún camino de un determinado largo que empiece en un delincuente en particular (que en realidad será un agente encubierto), y que termine en dicho agente otra vez.
7. Les interesaría poder obtener los conjuntos de delincuentes entre los cuales todos se conectan con todos. Pueden ser más de uno estos conjuntos.

## Datos disponibles

Se cuenta con un set de datos[1] de los mensajes, comunicaciones, encuentros, etc. entre los mencionados delincuentes (y agentes encubiertos). Por motivos tanto legales como de seguridad para les alumnes, solo se han dejado los números de identificación de dichos delincuentes, sin ningún dato personal. El formato del archivo es un tsv (los campos se separan por tabuladores):

```
id_vertice1     id_vertice2
id_vertice1     id_vertice3
...
id_vertice2     id_vertice1
id_vertice2     id_vertice3
...
id_vertice3     id_vertice5
```

Adicionalmente, se cuenta con el siguiente set mínimo de pruebas inventado, por lo que no representa un escenario real donde valga la pena realizar todas las operaciones pedidas, pero puede servir de prueba, y se utilizará este set de datos para los ejemplos. Además se deja un script generador para que se puedan realizar más sets de prueba (aleatorios).

[1]El set de datos corresponde a un set de Emails enviados entre investigadores de la Unión Europea, obtenidos por la Universidad de Stanford. Fuente real aquí.

# Aclaraciones

- Que un delincuente se comunique con otro no implica que también suceda la recíproca.
- Se ha visto también que hay delincuentes que se han enviado mensajes **a sí mismos**. Téngase esto en cuenta a la hora de procesar los datos.
- Aunque dos delincuentes se comuniquen 1, 2 o *n* veces, la comunicación sólo aparece una vez en el archivo (que quiere decir que se hace esa comunicación *al menos una vez*).

# Implementación

El trabajo puede realizarse en lenguaje a elección, siendo aceptados Python y C, y cualquier otro a ser discutido con le correctore asignade.

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 que luego se utilicen particularmente para el caso de la red de delincuentes. Esta biblioteca no debería tener referencias a delincuentes, sino simplemente aplicar algoritmos sobre grafos.
3. El programa `AlgoPoli` 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 (`$ ./algopoli mensajes.tsv`) y luego esperar la recepción de comandos por entrada estándar, del estilo `<comando> parametro1,parametro2`. Notar que esto permite tener un archivo de instrucciones a ser ejecutadas (i.e. `$ ./algopoli mensajes.tsv < comandos.txt`).

A continuación, se explica cada comando, con ejemplos de salida utilizando el set de datos reducido (para no adelantar los resultados para el set grande).

# Mínimos Seguimientos

- Comando: `min_seguimientos`.
- Parámetros: `origen` y `destino`.
- Utilidad: nos imprime una lista con los **delincuentes** (su código identificador) con los cuáles vamos del **delincuente** `origen` al **delincuente** `destino` de la forma más rápida. En caso de no poder hacer el seguimiento (i.e. no existe camino), imprimir `Seguimiento imposible`.
- Ejemplo:
  Entrada:

  ```
  min_seguimientos 10 4
  min_seguimientos 30 12
  ```

  Salida:

  ```
  10 -> 57 -> 4
  30 -> 36 -> 38 -> 20 -> 45 -> 12
  ```

# Delincuentes más importantes

Usualmente nos gustaría determinar cuáles son los vértices más importantes en un grafo en función de su centralidad exacta. Teniendo en cuenta que se cuenta con demasiados delincuentes, el cálculo exacto de la centralidad puede consumir una cantidad excesiva de tiempo. Por lo tanto, se pide realizar una aproximación para determinar los delincuentes más importantes. La forma sugerida para realizar esto es utilizando el algoritmo PageRank.

Por lo tanto, el comando pedido debe ser:

- Comando: `mas_imp`.
- Parámetros: `cant`.
- Utilidad: Imprime, de mayor a menor importancia, los `cant` delincuentes más importantes.
- Ejemplo:
  Entrada:

  ```
  mas_imp 10
  ```

  Salida:

  ```
  20, 89, 42, 3, 49, 47, 56, 28, 22, 8
  ```

Considerar que el "score" de pagerank no va a cambiar por más que se ejecute muchas veces diferentes, por lo que lo mejor será calcular únicamente la primera vez que se pidan, manteniendo guardado los scores para reutilizarlos si se vuelven a necesitar.

## Persecución rápida

- Comando: `persecucion`.
- Parámetros: `delincuente1,delincuente2,...,delincuenteN` y `K`.
- Utilidad: Dado cada uno de los delincuentes pasados (agentes encubiertos), obtener cuál es el camino más corto para llegar desde alguno de los delincuentes pasados por parámetro, a alguno de los `K` delincuentes más importantes. En caso de tener caminos de igual largo, priorizar los que vayan a un delincuente más importante.
- Ejemplo:
  Entrada:

  ```
  persecucion 10,14,17 5
  persecucion 19,11,7,12 3
  ```

  Salida:

  ```
  17 -> 35 -> 20
  19 -> 42
  ```

## Comunidades

Para implementar esto, utilizaremos el algoritmo de Label Propagation para detectar comunidades.

- Comando: `comunidades`.
- Parámetros: `n`.
- Utilidad: Imprime un listado de comunidades de al menos `n` integrantes.
- Ejemplo:
  Entrada:

  ```
  comunidades 10
  ```

  Salida:

  ```
  Comunidad 1: 0, 39, 59, 1, 47, 62, 2, 20, 3, 37, 31, 96, 16, 32, 80, 14, 40, 13, 89, 64, 72, 21, 15, 50, 97,
  ```

Tener en cuenta que siendo un archivo generado de forma aleatoria, los resultados obtenibles para este punto tienen muy poco sentido con la realidad.

## Divulgación de rumor

- Comando: `divulgar`.
- Parámetros: `delincuente` y `n`.
- Utilidad: Imprime una lista con todos los delincuentes a los cuales les termina llegando un rumor que comienza en el delincuente pasado por parámetro, y a lo sumo realiza `n` saltos (luego, se empieza a tergiversar el mensaje), teniendo en cuenta que *todos* los delincuentes transmitirán el rumor a sus allegados.
- Ejemplo:
  Entrada

  ```
  divulgar 30 4
  divulgar 30 1
  ```

  Salida:

```
36, 79, 84, 38, 71, 48, 13, 76, 77, 20, 64, 72, 57, 23, 7, 24, 85, 61, 47, 19, 25, 40, 37, 52, 56, 74, 66, 1,
36, 79, 84
```

## Ciclo de largo n

- Comando: `divulgar_ciclo`
- Parámetros: `delincuente` y `n`.
- Utilidad: Permite encontrar un camino simple que empiece y termine en el delincuente pasado por parámetro, de largo `n`. En caso de no encontrarse un ciclo de ese largo y dicho comienzo, imprimir `No se encontro recorrido`.
- Ejemplo:
  Entrada:

  ```
  divulgar_ciclo 74 5
  divulgar_ciclo 19 11
  ```

  Salida:

  ```
  74 -> 21 -> 81 -> 18 -> 42 -> 74
  19 -> 34 -> 12 -> 33 -> 54 -> 28 -> 79 -> 71 -> 57 -> 41 -> 56 -> 19
  ```

## Componentes Fuertemente Conexas

- Comando: `cfc`
- Parámetros: ninguno.
- Utilidad: Imprime cada conjunto de vértices entre los cuales todos están conectados con todos.
- Ejemplo:
  Entrada:

  ```
  cfc
  ```

  Salida:

  ```
  CFC 1: 10
  CFC 2: 77, 18, 73, 47, 91, 57, 30, 64, 82, 60, 85, 58, 22, 87, 50, 89, 14, 70, 32, 96, 37, 3, 29, 7, 40, 17,
  ```

# Entrega

Adicionalmente a los archivos propios del trabajo práctico debe agregarse un archivo `entrega.mk` que contenga la regla `algopoli` 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:

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

**Importante**: En caso de recibir un error `FileNotFoundError: [Errno 2] No such file or directory: './algopoli': './algopoli'`, 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 `algopoli.py`) la línea: `#!/usr/bin/python3`.

# 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.