UNIVERSIDAD METROPOLITANA

Departamento de Gestión de Proyectos y Sistemas.
Estructuras de datos. BPTSP06.
Trimestre 2223-3.
Prof. Christian Guillén Drija.

Proyecto 1

Contexto preliminar

En el siguiente proyecto, usted tendrá que realizar una **investigación documental** que le permita obtener información sobre el contexto del problema. En tal sentido, se sugiere que comience por realizar la siguiente lectura relativa a grafos, pero tome en cuenta que es solo un recurso inicial que debe ser complementado con la búsqueda autónoma de información por parte de los integrantes del equipo de trabajo:

https://drive.google.com/file/d/1Q65Rh-Tx6qwJODUlsmWVfw_-bzqttwil/view?usp=sharing

Problema

En el ámbito de los grafos, puede usted conseguirse con el término “Isla”, el cual indica la presencia de nodos que están agrupados, es decir, enlazados entre sí, pero aislados de otros conglomerados de nodos. Pueden entonces existir varias islas en un conjunto de datos que representan un grafo, es decir, al ser construida una representación visual de los datos, se pueden identificar sub grafos que entre sí, no están conectados. La figura 1 es un ejemplo de lo antes mencionado. El concepto anterior puede ser utilizado para el análisis de datos en las redes sociales. Si cada usuario es representado por un nodo y la relación de amistad entre los usuarios es representado por un arco (camino de un nodo a otro), entonces un investigador puede identificar grupos de usuarios que forman islas (subgrupos o comunidades). Su equipo de trabajo deberá desarrollar una aplicación en Java que reciba del usuario un conjunto de datos relativos a una red social y generar, tras su procesamiento, información relativa a los distintos grupos existentes, si los hubiera.

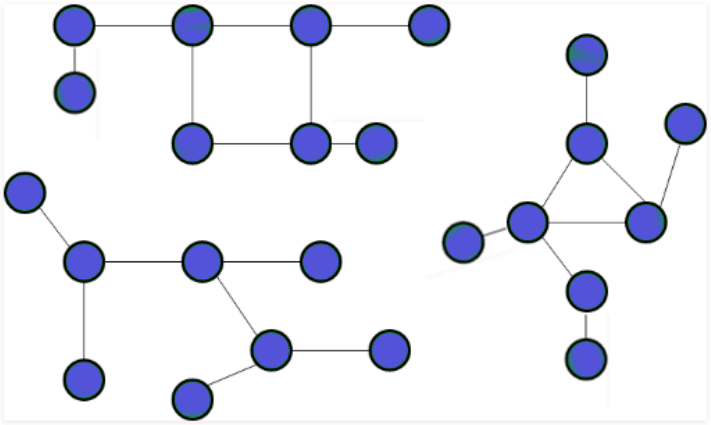

Figura 1.

A continuación, se describen con mayor detalle el conjunto inicial de requerimientos que deberán implementar.

Requerimientos funcionales

1. **General**

   A. **Cargar archivo:** El usuario puede seleccionar a través de **JFileChooser** un archivo **txt** (archivo de texto plano) para ser cargado en el sistema, el cual contará con la información necesaria para la creación del grafo, es decir: **usuarios** y **relaciones de amistad**. Cuando el usuario cargue un nuevo archivo, el sistema debe enviar un mensaje de alerta indicando al usuario la necesidad de guardar los datos actualmente cargados en memoria. La estructura del archivo de datos (*archivo de texto plano*) se indicará posteriormente.

   B. **Actualizar repositorio**: Esta función permitirá que la información cargada en memoria, referente a los usuarios y sus relaciones de amistad. Es decir, los cambios realizados a cualquiera de estos deben de actualizarse en el archivo texto de tal forma que cuando se vuelva a cargar ese archivo contenga todos los cambios realizados. Al iniciarse el programa por primera vez debe cargarse el archivo de texto dado al final del enunciado *(debe de mantener el mismo formato)*.

   C. **Mostrar grafo:** El sistema deberá mostrar una representación visual del grafo según la información contenida en la memoria, es decir, las relaciones entre los usuarios y su información. Si existen islas, entonces deberá mostrar tales islas de manera gráfica, mostrando los valores de los nodos y de los arcos.

   D. **Mostrar la cantidad de Islas:** que se obtendrá mediante el recorrido en anchura (BFS) o el recorrido de profundidad (DFS). El usuario podrá seleccionar entre los dos métodos.

   E. **Identificación de puentes**: un arco es un puente si al ser removido del grafo original, aumenta la cantidad de componentes (islas) desconectados. Esta función debe indicar en pantalla una lista de los puentes presentes en el grafo general. En el caso que nos atañe, Esta funcionalidad equivale a identificar a aquellos usuarios que son enlaces entre subgrupos o subcomunidades.

   F. **Modificar grafo**: el usuario podrá seleccionar usuarios para ser eliminados. De igual forma, el usuario podrá agregar un nuevo usuario, indicando la relación que tenga con otros usuarios dentro del grafo.

Requerimientos técnicos

1. La solución debe ser implementada con base en un grafo, que a su vez puede ser implementado mediante una **matriz de adyacencia** o una l**ista de adyacencia**.

2. Puede utilizar cualquier otra estructura auxiliar de ser necesario. Sin embargo, **NO podrá utilizar librerías para la implementación del tipo de dato**

abstracto, solo podrá utilizar librerías para lo relativo a la representación gráfica del grafo.

3. El programa debe poder representar el grafo correspondiente de manera gráfica.
4. La aplicación debe ofrecer una interfaz gráfica al usuario.
5. El programa debe poder cargar un archivo de texto para la lectura de datos. Para ello, es requerido el uso del componente **JFileChooser**.
6. Debe documentar el proyecto con **Javadoc**.
7. Junto al programa, cada equipo deberá presentar un **Diagrama de clases** *(arquitectura detallada)* que explique la solución obtenida.
8. La estructura de los archivos es la siguiente: para la sección de los usuarios, el primer campo es el ID, el segundo es el usuario. Para la sección de relaciones (de amistad), los campos son usuario1, usuario2, tiempo de amistad. El grafo no es dirigido.

Se muestra a continuación un ejemplo de la estructura del archivo de texto plano que deberá cargar el programa.

Archivo de texto

Usuarios
121, @Pepe_Gónzales
254, @StephaniaCominos
365, @AndreaStanislao
412, @Josefina_La_Sifrina
512, @RosaMosa
231, @EduardoPetardo
123, @EnriqueManrique
129, @casanova23
870, @venepositivo
758, @yosoylatorre
578, @pitiypo
909, @obiwan123
893, @caribu_sol
467, @trapos232
788, @bandido121
239, @justiciero11
443, @fuerza_bruta
907, @Presentesiempre
Relaciones
121, 254, 7
121, 909, 8
254,909,5
909, 893, 5
254, 893, 1
893, 129, 3
129, 512, 10
512, 412, 2

893, 412, 4
231, 870, 5
231, 123, 1
123, 870, 15
123, 467, 6
788, 239, 7
788, 443, 11
239, 443, 6
239, 907, 3
443, 907, 9
788, 412, 7
870, 578, 7
870, 758, 1
758, 365, 9
578, 365, 4

La figura 2 es una representación del grafo contenido en el ejemplo anterior.

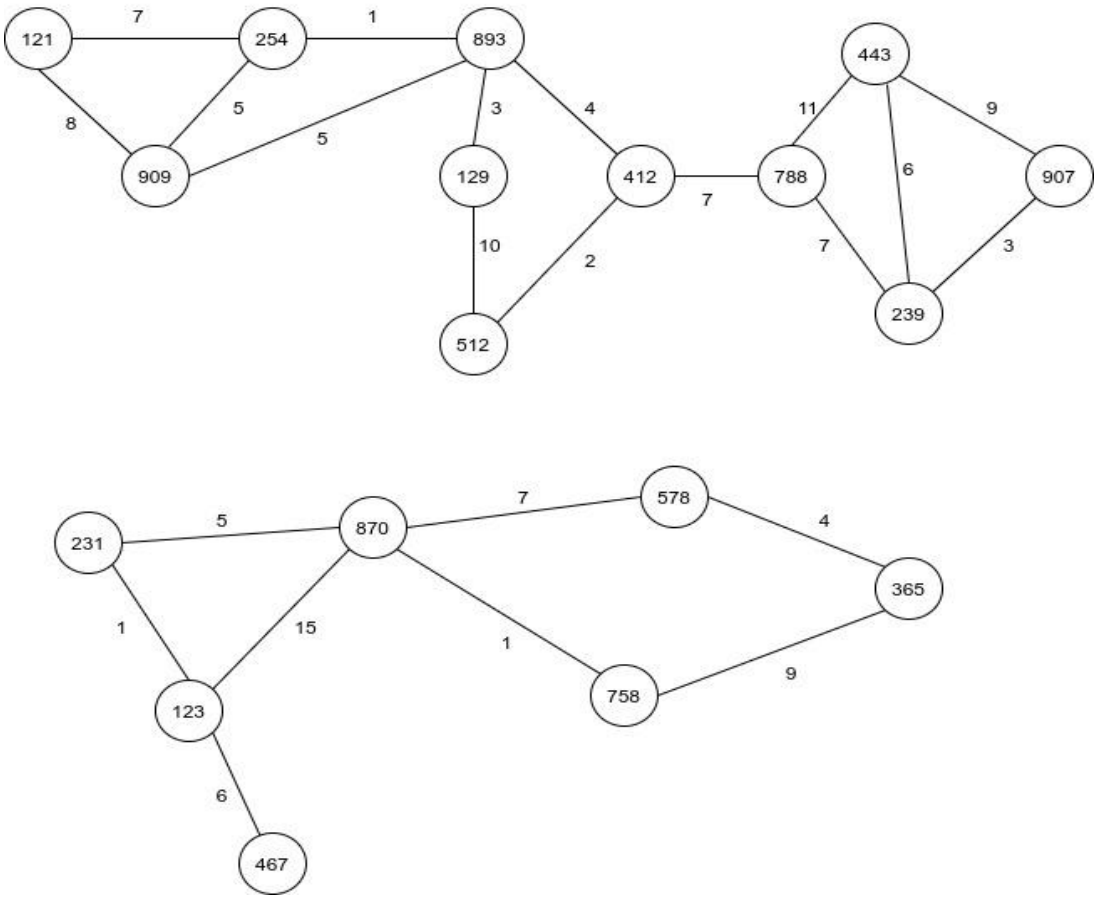

Figura 2

Consideraciones

- Los proyectos **podrán ser sometidos a defensa**, es decir, el facilitador convocará al equipo para una revisión.

- Los equipos de trabajo deberán utilizar **GitHub** para el control de versiones y facilitar el trabajo en equipo de manera remota. De esta forma, podrán comenzar a crear su portafolio de trabajos, elemento que puede ser importante a la hora de buscar trabajo. **En el registro se deberá reflejar la participación activa y significativa de los integrantes.**
- Los proyectos que no tengan interfaz gráfica, serán calificados con **0 (cero)**.
- Los proyectos que sean iguales o parecidos, serán calificados con **0 (cero)**.
- Los programas que “no corran”, serán calificados con **0 (cero).**
- Los equipos pueden tener como **máximo 3 personas.**

Criterios de evaluación

- *Funcionalidad:* Capacidad para proporcionar las funcionalidades que satisfacen las necesidades explícitas e implícitas bajo unas ciertas condiciones. **(60%)**
    - *Adecuación:* El programa ofrece todas funcionalidades que respondan a las necesidades, tanto explícitas (contenidas en el documento descriptivo del proyecto) como implícitas; entendiendo como necesidades implícitas, aquellas que no estando descritas en el documento, surgen como resultado de un concienzudo análisis del problema planteado y que aseguran el correcto funcionamiento del programa.
    - *Exactitud*: El programa genera los resultados o efectos correctos o acordados, con el grado necesario de precisión.
- *Fiabilidad*: Capacidad para mantener un nivel especificado de prestaciones cuando se usa bajo ciertas condiciones.
    - *Madurez*: El programa no presenta fallas originadas por errores de programación, análisis o diseño. **(10%)**
    - *Tolerancia a fallos*: El programa responde adecuadamente al manejo inadecuado del usuario; es decir, mantiene su correcto funcionamiento aun cuando el usuario introduzca datos erróneos o manipule inadecuadamente las interfaces de usuario. **(10%)**
- *Usabilidad*: Capacidad del proyecto para ser entendido, aprendido, usado y al mismo tiempo, ser atractivo para el usuario, cuando se usa bajo condiciones específicas.
    - *Comprensibilidad*: El programa ofrece una interfaz de fácil comprensión, facilitando su aprendizaje y correcta utilización. El programa emite mensajes de alerta cuando se introducen valores erróneos. Existen elementos informativos que indican al usuario como operar el programa. **(5%)**

    - *Capacidad de ser atractivo*: El diseño de la interfaz de usuario, esto es: disposición de controles, esquema de colores, utilización de cajas de diálogo y demás elementos gráficos; es atractivo para el usuario. **(5%)**

- *Eficiencia*: Capacidad para proporcionar prestaciones apropiadas, relativas a la cantidad de recursos utilizados, bajo condiciones determinadas.

    - *Estructuras de datos*: Utiliza eficientemente las estructuras de datos para la solución del problema. **(10%)**