Universidad Nacional de General Sarmiento

Licenciatura en Sistemas
Tecnicatura Superior en Informática

Trabajo Práctico N°2
Clustering Humano

Programación 3
(2º semestre 2020)

INTEGRANTES:
Sanchez Juan Ignacio - 39160465 - juani.scz95@gmail.com
Schmidt Maximiliano - 40664272 - maxischmidt2011@gmail.com
Sosa Martín Leonel - 36289472 - martinleonelarce@gmail.com

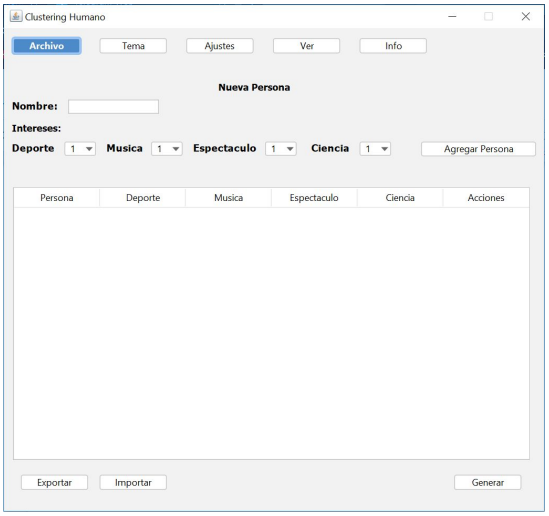
DOCENTES:
Alejandro Nelis
Fernando Torres

1.Introducción y Consignas

El objetivo del trabajo práctico es implementar una aplicación para identificar automáticamente grupos de personas sobre la base de sus características. Tenemos una lista de personas, y para cada persona tenemos su nombre y los siguientes datos:

- **di = Interés por los deportes**
- **mi = Interés por la música**
- **ei = Interés por las noticias del espectáculo**
- **ci = Interés por la ciencia**

Cada uno de estos datos se expresa como un entero entre 1 y 5, siendo 1 el menor interés y 5 el máximo interés en el tema. Buscamos dividir a esta población en dos grupos con intereses similares. Para esto, utilizamos el siguiente algoritmo:



1. Construir un grafo completo G con un vértice por cada persona, una arista entre cada par de personas, y de modo tal que cada arista tiene peso igual al ´ındice de similaridad entre las dos personas.
2. Construir un árbol generador mínimo T de G.
3. Eliminar la arista de mayor peso del árbol T.
4. Las dos componentes conexas del grafo resultante son los dos grupos que estamos buscando.

El índice de similaridad entre dos personas i y j se define como:
similaridad(i, j) = |di − dj | + |mi − mj | + |ei − ej | + |ci − cj |.
Este valor es cero cuando i y j tienen exactamente los mismos valores de interés en los cuatro temas, y es más alto para las personas con intereses distintos.
La aplicación implementada debe contar con una interfaz para cargar y visualizar los datos de las personas, y una interfaz para ver los dos grupos que se formaron luego del algoritmo anterior. Se puede contar con un botón para lanzar la ejecución del algoritmo.

Como objetivos opcionales no obligatorios, se pueden contemplar los siguientes elementos:

1. Mostrar estadísticas de los dos grupos generados, como por ejemplo similaridad promedio, promedio de interés en cada tema, etc.
2. Ejecutar el algoritmo para recalcular los grupos cada vez que se agrega una nueva persona a la lista.
3. Permitir que se generen más de dos grupos, preguntando al usuario cuantos grupos quiere generar.

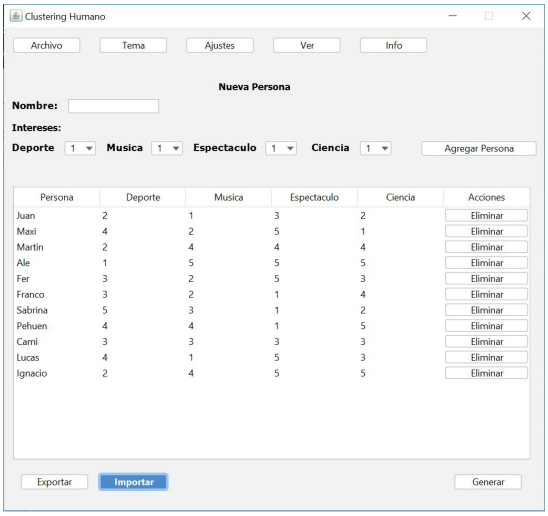
Condiciones de entrega: El trabajo práctico se debe entregar por mail a los docentes de la materia. Además del código, se debe incluir un documento en el que se describa la implementación y se detallen las decisiones tomadas durante el desarrollo. El código

responsable del funcionamiento de la aplicación debe estar claramente separado del código responsable de la interfaz. Todas las clases de negocio deben tener tests unitarios.

2. Separación de Capas y Denominación de Clases

Nuestra aplicación será capaz de:

1. Agregar personas, siempre y cuando no sea la misma.
2. Mostrar todas las personas en una tabla, donde se indicará los atributos de cada persona.
3. Importar y exportar listas de personas que se almacenarán en un archivo.
4. A partir de una lista de personas se podrá generar un grafo completo y un árbol generador mínimo Kruskal.
5. Cuando se generen los dos grupos de personas se mostrará la información en otra ventana, pudiendo seleccionar cada grupo para ver su tabla respectivamente



Para la implementación de la aplicación se decidió separar en cinco capas :

I. **Interfaces**
II. **Negocio**
III. **Modelo**

Cada capa es responsable de un aspecto de la aplicación, la capa de negocio se encarga de la lógica y las interfaces se encargan de toda la interfaz gráfica. Como clase principal de la capa de negocio se implementa la clase Negocio y como principales de la capa de Interfaces Panel, como ventana principal. En el caso del Modelo solo se encuentra Persistencia que se encarga de guardar y cargar en archivos los conjuntos de personas.

INTERFACES:

Package InterfazDialog

PanelDialog: Es el panel donde se muestran los grupos de personas con sus respectivos promedios. Contiene TablaCabecera y TablaDetalle.

TablaCabecera: Es la tabla que muestra los diferentes grupos que se generan.

TablaDetalle: Es la tabla que muestra a las personas contenidas en cada grupo con sus intereses y el nombre.

Package InterfazGeneral

BottonColumn: Representa los botones de “Eliminar” en la columna de “Acciones”, al utilizar este botón se elimina a la persona de la clase Negocio y modifica la Tabla de la interfaz principal.

Temas: Posee los temas para la aplicación.

Package InterfazPrincipal

Botonera: Son los botones que se encuentran dentro de la ventana del programa principal: Archivo, Tema, Ajustes, Ver, etc.

Panel: Es la ventana principal de la aplicación. Su función principal es recibir los datos de las personas y poseer los botones para generar los grafos correspondientes.

PersonaView: Son todos los elementos correspondientes para recibir los datos de una persona. El textField para ingresar el nombre de la persona y los comboBox para ingresar el valor de los intereses.

Tabla: Es la encargada de almacenar y mostrar a cada persona del conjunto de personas.

NEGOCIO:

Package Negocio

Arista: Representa la relación entre dos personas, posee la similaridad entre ambos.

ConjuntoDePersonas: Representa a un grupo de personas en una lista.

Grafo: Representa a un grafo que posee un conjunto de personas y las aristas entre ellas.

Negocio: Es el encargado de administrar y de llevar a cabo las tareas más importantes y nucleares del programa.

Persona: Representa a una persona con su nombre e intereses.

MODELO:

Package Modelo

Persistencia: Se encarga de guardar y cargar el conjunto de personas en un archivo.

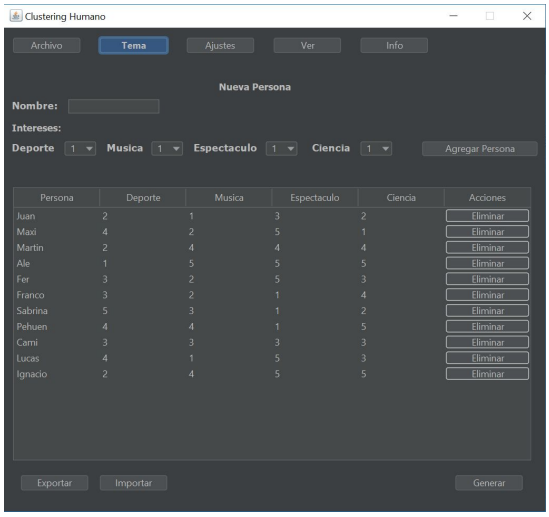
Desarrollo y decisiones de implementación:

La implementación inicial solo contaba con las clases: Negocio, Aristas, Grafo, PRIM y Kruskal. Pero nos dimos cuenta que las clases PRIM y Kruskal estaban muy

estrechamente vinculadas a la clase Grafo, así que decidimos integrar estas dos clases a Grafo.

Otra decisión tomada fue la de “graficar” los grupos generados al eliminar la arista más pesada con tablas en una ventana aparte, en vez de utilizar grafos, nos pareció que esta opción le daba un enfoque empresarial a la forma de mostrar los grupos de personas.

Originalmente no existía la clase ConjuntoDePersonas, en vez de eso teníamos una ArrayList<Persona>.



Al igual que en el trabajo práctico anterior decidimos “desmontar” toda la parte de la interfaz gráfica en pequeñas partes las cuales están controladas por la clase Panel.

Algoritmos:

Algoritmo encargado de crear el árbol generador mínimo en el método de Kruskal.

```
public Grafo generarAGMKuskal() {
        // ordenar aristas por peso de menor a mayor
        ArrayList<Arista> aristasOrdenadas = this.getAristas();
        aristasOrdenadas.sort((a1, a2) -> a1.getPeso().compareTo(a2.getPeso()));
        // creo un grafo con los mismos vertices que el que entra por parametro
        Grafo arbolGM = new Grafo(this.getVertices());
        // agrego la primer arista (es la de menor peso)
        arbolGM.agregarArista(aristasOrdenadas.get(0));
        aristasOrdenadas.remove(0);
        // for each de aristas
        for (Arista arista : aristasOrdenadas) // omito la primer arista porque ya la agregue
        {
            //// verificar si al agregar una arista se genera un ciclo (bfs)
            // si a partir de un arbol selecciono un vertice de la arista que quiero
            //// agregar, y llego al otro
            // vertice de la arista, significa que ya hay un camino y agregar la arista
            //// forma un ciclo
            if (!arbolGM.alcanzables(arista.getPersona1()).contiene(arista.getPersona2()))
                //// si no se genera, agregar arista
                arbolGM.agregarArista(arista);
            //// si se genera, no agregao la arista
        }
        // al finalizar tengo un AGM de kruskal
        return arbolGM;
    }
```

Algoritmo encargado de: dado un grafo de árbol generador mínimo de Kruskal, elimina la arista más pesada que posea, y devuelve un ArrayList<Grafo> con los grafos conexos que se obtienen al eliminar dicha arista más pesada.

```
public ArrayList<Grafo> separarGrafoEnDos() {
		ArrayList<Grafo> ret = new ArrayList<Grafo>();

		Arista aristaMasPesada = this.eliminarAristaMasPesada();

		ConjuntoDePersonas personas1 =
alcanzables(aristaMasPesada.getPersona1());
		ConjuntoDePersonas personas2 =
alcanzables(aristaMasPesada.getPersona2());
		ArrayList<Arista> aristas1 = restaurarAristas(personas1);
		ArrayList<Arista> aristas2 = restaurarAristas(personas2);

		Grafo grafo1 = new Grafo(personas1, aristas1);
		Grafo grafo2 = new Grafo(personas2, aristas2);

		ret.add(grafo1);
		ret.add(grafo2);

		return ret;
	}
```

Algoritmo encargado de generar un grafo completo solamente a través de sus vértices (ConjuntoDePersonas)

```
public Grafo generarGrafoCompleto() {
		Grafo ret = new Grafo(getVertices());
		for (int i = 0; i <= ret.getVertices().tamanio(); i++)
			for (int j = i + 1; j < ret.getVertices().tamanio(); j++)
				ret.agregarArista(new Arista(ret.getVertices().obtener(i),
ret.getVertices().obtener(j)));
		return ret;
	}
```

Algoritmo encargado de verificar si un grafo es conexo.

```
public boolean esConexo() {
		if (getAristas() == null || getVertices() == null)
			throw new IllegalArgumentException("El grafo es null");
		if (getVertices().tamanio() == 0)
			return true;
		return alcanzables(getVertices().obtener(0)).tamanio() ==
getVertices().tamanio();
	}
```

CLASES de Negocio

class Persona implements Serializable

```
	private String nombre;
```

```
    private int deportes;
    private int musica;
    private int espectaculos;
    private int ciencia;
    private static final long serialVersionUID = 1L;
```

class Arista

```
    private Persona persona1;
    private Persona persona2;
    private Integer peso;
```

class ConjuntoDePersonas implements Serializable

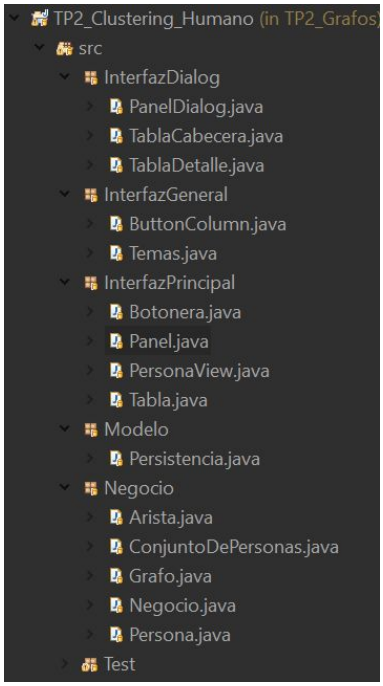
```
    private static final long serialVersionUID = 1L;
    private ArrayList<Persona> personas;
```

public class Grafo

```
    protected ConjuntoDePersonas vertices;
    protected ArrayList<Arista> aristas;
```

public class Negocio {

```
    Persona p;
    ConjuntoDePersonas personas;
```



CLASES de las Interfaces

public class Panel

```
    private JFrame frame;
    private Botonera botonera;
    private PersonaView personaView;
    public Tabla tabla;
    private Negocio negocio;
```

class Tabla extends JPanel

```
    private static final long serialVersionUID = 1L;
    private JTable table;
    private DefaultTableModel model;
```

public class PersonaView extends JPanel

```
    private static final long serialVersionUID = 1L;
    private JTextField textField;
    private JComboBox<Integer> cB_Deporte;
    private JComboBox<Integer> cB_Musica;
    private JComboBox<Integer> cB_Espectaculos;
    private JComboBox<Integer> cB_Ciencia;
```

class Botonera extends JPanel

```
        private static final long serialVersionUID = 1L;
        private Temas temas = new Temas();

public class Temas
        int temaActual;
        private List<LookAndFeel> temasDisponibles;

public class ButtonColumn extends AbstractCellEditor implements
TableCellRenderer, TableCellEditor, ActionListener, MouseListener
        private static final long serialVersionUID = 1L;
        private JTable table;
        private Action action;
        private int mnemonic;
        private Border originalBorder;
        private Border focusBorder;
        private JButton renderButton;
        private JButton editButton;
        private Object editorValue;
        private boolean isButtonColumnEditor;

class TablaDetalle extends JPanel
        private static final long serialVersionUID = 1L;
        private JTable table;
        private DefaultTableModel model;

class TablaCabecera extends JPanel
        private static final long serialVersionUID = 1L;
        private JTable table;
        private DefaultTableModel model;

class PanelDialog extends JFrame
        ArrayList<Grafo> grupos;
```