

---

# Trabajo Práctico II

## Programación III

Universidad Nacional General Sarmiento  
*“Clustering”*

**Profesores:** Patricia Bagnes y Javier Marengo.

**Integrantes:** Matías Leandro Avila y Mendez Agustina

**Emails:** leandroavmmo@gmail.com agustina.996@hotmail.com

---

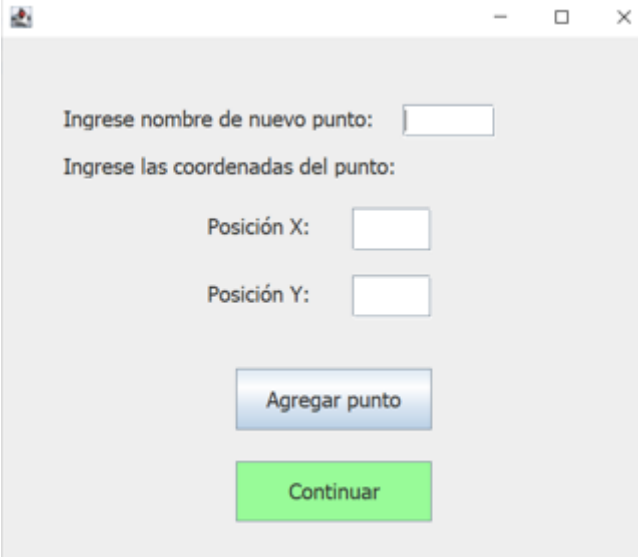
**13 de octubre del 2022**

El objetivo del trabajo práctico es implementar una aplicación para realizar clustering utilizando el algoritmo propuesto por Charles Zahn en el artículo “Graph-theoretical methods for detecting and describing clusters” (IEEE Transactions on Computers 20-1, 1971). Dado un conjunto de puntos en el plano y un grafo completo con estos puntos, con pesos en las aristas iguales a la distancia euclídea entre los vértices, el algoritmo es el siguiente:

1. *Calcular un árbol generador mínimo del grafo.*
2. *Eliminar del árbol las aristas cuyos pesos sean mayores que los pesos de sus aristas vecinas.*

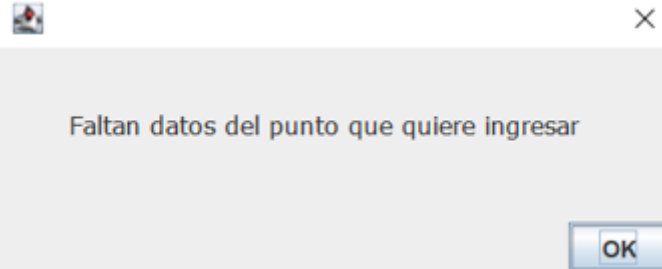
01. En cuanto a la interfaz:

El sistema cuenta con una interfaz visual donde el usuario ingresa de manera manual: el nombre del vértice y sus coordenadas (x, y).



A screenshot of a graphical user interface window titled "Agregar punto". The window has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. It contains two labels: "Ingrese nombre de nuevo punto:" followed by a text input field, and "Ingrese las coordenadas del punto:" followed by two text input fields labeled "Posición X:" and "Posición Y:". Below these fields are two buttons: a blue button labeled "Agregar punto" and a green button labeled "Continuar".

Se tiene en cuenta que si falta algún dato del punto ingresado, se muestra una ventana de diálogo para advertir al usuario la falta de esto.

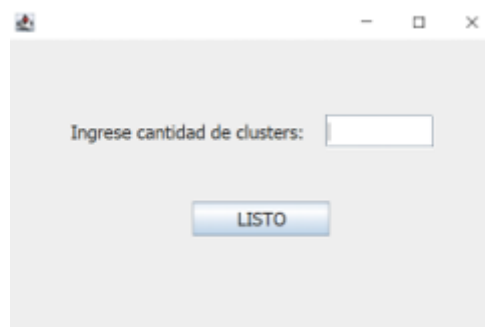


A screenshot of a small error dialog box. It has a light gray background and a close button (X) in the top right corner. The text inside reads "Faltan datos del punto que quiere ingresar". In the bottom right corner, there is a blue button labeled "OK".

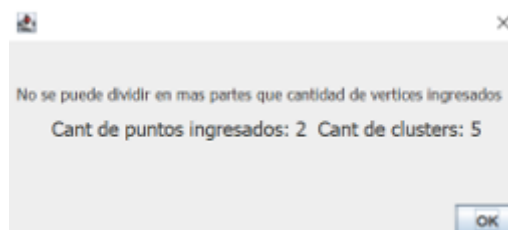
Una vez agregados los puntos, el usuario debe oprimir el botón **continuar**. Este botón llama al método del sistema *crearGrafoAGM()* que a su vez, llama al método *grafo.CrearGrafo()* que crea el grafo completo. A partir de ese grafo completo, se lo transforma en un AGM con la llamada al método *grafo.transformarArbolGeneradorMinimo()*.

```
 JButton btnCrearGrafo = new JButton("Continuar");
 btnCrearGrafo.setBackground(new Color(152, 251, 152));
 btnCrearGrafo.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
         if (!sistema.getGrafo().getVerticesGrafo().isEmpty()) {
             sistema.crearGrafoAGM();
             Interfaz interfaz = new Interfaz(sistema);
             interfaz.getFrame().setVisible(true);
         } else {
             ModalSinVerticesAgregados dialog = new ModalSinVerticesAgregados();
             dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
             dialog.setVisible(true);
         }
     }
 });
 btnCrearGrafo.setFont(new Font("Tahoma", Font.PLAIN, 16));
 btnCrearGrafo.setBounds(178, 322, 148, 46);
 frame.getContentPane().add(btnCrearGrafo);
```

El botón **continuar** también crea una nueva interfaz donde el usuario ingresará la cantidad de **clusters** en los que quiere dividir el grafo.



Al oprimir el botón **listo**, si la cantidad de divisiones que se quieren hacer es mayor a la cantidad de puntos ingresados, se mostrará una ventana de aviso para que se cambie el valor.



Si el valor ingresado es correcto, se hace la llamada al método *sistema.generarClusters()*.

```
JButton btnNewButton = new JButton("LISTO");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if (sistema.getGrafo().getVerticesGrafo().size() >= Integer.parseInt(textFieldCantClusters.getText())) {
            sistema.generarClusters(Integer.parseInt(textFieldCantClusters.getText()));
            InterfazResultado interfaz = new InterfazResultado(sistema);
            interfaz.getFrame().setVisible(true);
        } else {
            ModalCantidadDeClusters dialog = new ModalCantidadDeClusters(
                sistema.getGrafo().getVerticesGrafo().size(),
                Integer.parseInt(textFieldCantClusters.getText()));
            dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
            dialog.setVisible(true);
            textFieldCantClusters.setText("");
        }
    }
});
btnNewButton.setFont(new Font("Tahoma", Font.PLAIN, 16));
btnNewButton.setBounds(164, 144, 121, 30);
getFrame().getContentPane().add(btnNewButton);
```

*sistema.generarClusters()* llama a *grafo.generarClusters(cantidad)*. Este método es el encargado de buscar las *cantidad - 1* aristas (siendo cantidad el número ingresado por el usuario) de mayor peso y eliminarlas hasta obtener la cantidad de componentes necesarias.

```
public void generarClusters(Integer cantidad) {
    int contador = 0;
    while(cantidad - 1 > contador) {
        int indiceAristaMax = 0;
        for (int i = 0; i < listaAristas.size(); i++) {
            if (listaAristas.get(i).getPeso() > listaAristas.get(indiceAristaMax).getPeso()) {
                indiceAristaMax = i;
            }
        }
        eliminarArista(indiceAristaMax);
        contador++;
    }
}
```

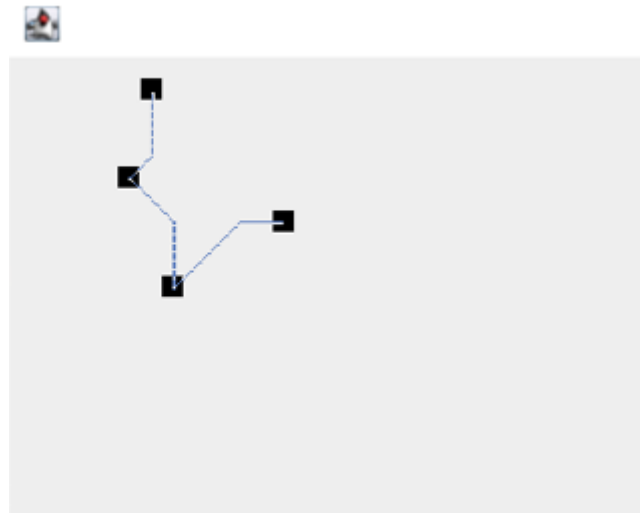
El botón **listo**, también abre la ventana donde se muestra el resultado final. Acá se puede observar gráficamente los puntos y las aristas de los clusters. Para dibujar los vértices, utilizamos Panel de 10x10 y agregamos el nombre de cada vértice.

```
private void dibujarVertices() {
    for(Vertex vertice: sistema.getGrafo().getVerticesGrafo()) {
        dibujarVertice(vertice);
    }
}

private void dibujarVertice(Vertex vertice) {
    Panel panel = new Panel();
    panel.setBackground(Color.BLACK);
    panel.setBounds(vertice.getcoordenadaVertice().x, vertice.getcoordenadaVertice().y, 10, 10);
    frmResultado.getContentPane().add(panel);

    JLabel nombreVertice = new JLabel(vertice.getNombre());
    nombreVertice.setFont(new Font("Tahoma", Font.PLAIN, 10));
    nombreVertice.setBounds(vertice.getcoordenadaVertice().x + 15, vertice.getcoordenadaVertice().y + 5, 10, 10);
    frmResultado.getContentPane().add(nombreVertice);
}
```

Para dibujar las aristas utilizamos JSeparator para formar una línea a partir de puntos. En un primer momento, para dibujar cada punto, nos movimos una posición en **x** y otra en **y**. Esto generaba que una vez alcanzado el valor de una de las coordenadas, se siguiera moviendo solo en **x** o en **y** hasta alcanzar la posición del segundo vértice.



Para que se vea mejor visualmente, se consideró obtener un porcentaje de movimiento para **x** y un porcentaje para **y**.

```
private void dibujarArista(Arista arista) {  
    // Dejamos en coorX la coordenada que está mas a la izquierda  
    int coorX = 5;  
    int coorXDestino = 5;  
    int coorY = 5;  
    int coorYDestino = 5;  
    if (arista.getCoordenada1().x < arista.getCoordenada2().x) {  
        coorX += arista.getCoordenada1().x;  
        coorY += arista.getCoordenada1().y;  
        coorXDestino += arista.getCoordenada2().x;  
        coorYDestino += arista.getCoordenada2().y;  
    } else {  
        coorX += arista.getCoordenada2().x;  
        coorY += arista.getCoordenada2().y;  
        coorXDestino += arista.getCoordenada1().x;  
        coorYDestino += arista.getCoordenada1().y;  
    }  
}
```

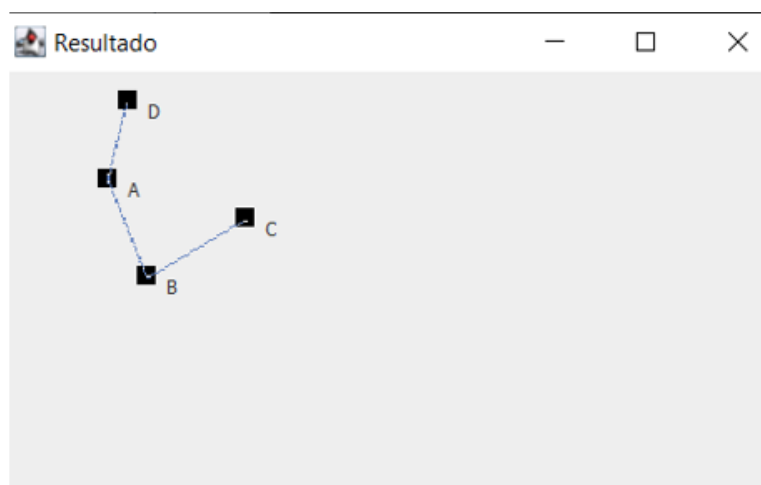
```

double difX = coorXDestino - coorX;
double difY = (coorY < coorYDestino) ? coorYDestino - coorY : coorY - coorYDestino;
double porcentajeDeAumentoParaX = difX / difY;
double porcentajeDeAumentoParaY = difY / difX;
double contadorAumentoX = porcentajeDeAumentoParaX;
double contadorAumentoY = porcentajeDeAumentoParaY;

while(coorX != coorXDestino || coorY != coorYDestino) {
    JSeparator separator = new JSeparator();
    separator.setBounds(coorX, coorY, 1, 1);
    separator.setBackground(Color.BLUE);
    frmResultado.getContentPane().add(separator);
    if (porcentajeDeAumentoParaX < porcentajeDeAumentoParaY) {
        if (contadorAumentoX >= 1){
            coorX++;
            contadorAumentoX--;
        }
        contadorAumentoX += porcentajeDeAumentoParaX;
        if (coorY < coorYDestino) {
            coorY++;
        } else if (coorY > coorYDestino) {
            coorY--;
        }
    } else if (porcentajeDeAumentoParaX > porcentajeDeAumentoParaY) {
        if (contadorAumentoY >= 1){
            if (coorY < coorYDestino) {
                coorY++;
            } else if (coorY > coorYDestino) {
                coorY--;
            }
            contadorAumentoY--;
        }
        contadorAumentoY += porcentajeDeAumentoParaY;
        if (coorX != coorXDestino) {
            coorX++;
        }
    } else {
        coorX++;
        if (coorY < coorYDestino) {
            coorY++;
        } else if (coorY > coorYDestino) {
            coorY--;
        }
    }
}
}
}

```

De esta manera logramos que las líneas se vean más parejas y que no haya “esquinas” que se puedan confundir con otros vértices.



## 02. En cuanto a la lógica de negocio:

La parte de la lógica de negocio cuenta con 4 clases (2 clases de test):

Arista
AristaTest
Grafo
GrafoTest
Sistema
Vertice

En cuanto a la clase **Vértice**, representa a los vértices del grafo. Su constructor:

Constructor	Description
<code>Vertice(String nombreDeVertice)</code>	<b>Constructor:</b> <u>Constructor de nodoGrafo.</u>

Alguno de sus métodos son:

<code>void</code>	<code>insertarCoordenadas(double coordenadaX, double coordenadaY)</code>	<b>insertarCoordenadas:</b> <u>Setter de las coordenadas del vertice.</u>
-------------------	--	--

En cuanto a la clase **Arista**, representa las aristas del grafo. Su constructor:

Constructor	Description
<code>Arista(Vertice vertice1, Vertice vertice2)</code>	<b>Constructor:</b> <u>Constructor de Arista.</u>

Alguno de sus métodos son:

Modifier and Type	Method	Description
<code>void</code>	<code>calcularDistancia()</code>	<b>calcularDistancia():</b> <u>Metodo que calcula el peso de una arista.</u>

En cuanto a la clase **Grafo**, es la encargada de todo lo referido al grafo. Su constructor:

Constructor	Description
<code>Grafo()</code>	<b>Constructor:</b> <u>Constructor de grafo.</u>

Alguno de sus métodos son:

Modifier and Type	Method	Description
void	<code>CrearGrafo()</code>	<b>CrearGrafo():</b> <u>Metodo que une el grafo, creando las aristas de este.</u>
void	<code>crearNuevaArista(Vertice vertice1, Vertice vertice2)</code>	<b>crearNuevaArista():</b> <u>Metodo que une dos vertice en el grafo, y crea la arista correspondiente.</u>
void	<code>eliminarArista(int indiceArista)</code>	<b>eliminarArista():</b> <u>Metodo que elimina la arista dada por el indice pasado.</u>
boolean	<code>existeAristaEnGrafo(Vertice vertice1, Vertice vertice2)</code>	<b>existeAristaEnGrafo():</b> <u>Metodo que devuelve si existe o no una arista en el grafo.</u>
boolean	<code>existeVertice(String nombreDeVertice)</code>	<b>existeVertice():</b> <u>Metodo que devuelve si existe o no un vertice en el grafo.</u>
void	<code>generarClusters(Integer cantidad)</code>	<b>generarClusters():</b> <u>Metodo que crea los clusters solicitados por el usuario.</u>
int	<code>getCantidadDeVertices()</code>	<b>getCantidadDeVertices():</b> Devuelve la cantidad de vertices del grafo.
<code>LinkedList&lt;Arista&gt;</code>	<code>getListaAristas()</code>	<b>getListaAristas():</b> Devuelve la lista de aristas del grafo.
<code>LinkedList&lt;Vertice&gt;</code>	<code>getVerticesGrafo()</code>	<b>getVerticesGrafo():</b> Devuelve los vertices del grafo.
void	<code>insertarVertice(String nombreDelVertice, double coordenadaX, double coordenadaY)</code>	<b>insertarVertice():</b> <u>Metodo que inserta un nuevo vertice en el grafo con su posicion correspondiente.</u>
void	<code>transformarArbolGeneradorMinimo()</code>	<b>transformarArbolGeneradorMinimo():</b> <u>Metodo que transforma el grafo en un grafo con aristas minimas.</u>

En cuanto a la clase **Sistema**, utilizado en la interfaz de la aplicación. Su constructor:

Constructor	Description
<code>Sistema()</code>	<b>Constructor:</b> <u>Constructor de Sistema.</u>

Alguno de sus métodos son:

Modifier and Type	Method	Description
void	<code>agregarVertice(String nombreDelVertice, double posicionX, double posicionY)</code>	<b>agregarVertice():</b> <u>Metodo que inserta un nuevo vertice en el grafo con su posicion correspondiente.</u>
void	<code>crearGrafoAGM()</code>	<b>crearGrafoAGM():</b> <u>Metodo que crea el grafo y lo transforma en un grafo con aristas minimas.</u>
void	<code>generarClusters(Integer cantidad)</code>	<b>generarClusters():</b> <u>Metodo que crea los clusters solicitados.</u>
Grafo	<code>getGrafo()</code>	<b>getGrafo():</b> Devuelve el grafo.

Además de estos quedan las clases encargadas de testear los métodos de otras clases. Detallado todo esto, la implementación y demás está dada en el código.