Facultad de Ingeniería

Auxiliar**.** Byron Estuardo Caal Catún

Sección: A

Proyecto
Teoría de Grafos

Alison Pamela Vásquez Villagran

Carné: 202100125

Andrea Alejandra Pérez Sandoval

Carné: 202201136

Fecha: 28/04/2023

Estructura del programa

El programa este hecho en lenguaje Java, el IDE utilizado es Apache NetBeans. El Programa este hecho por clases, la primera clase se llama Ventana en donde se implementaron MouseListener, ActionListener y MouseMotionListener que son utilizadas para manejar eventos al dar click. Asimismo, la clase se definieron variables y componentes de la inerfaz gráfica, un JFrame, varios botones y label, un textbox, asimismo un objeto Lienzo de Grafo y un objeto Grafo, también se utilizaron ArrayLists.

Dos líneas dos ArrayLists; uno para almacenar objetos de la clase para dibujar los puntos y el otro es para dibujar las aristas. Las variables se utilizan para configurar la interfaz gráfica y almacenar información sobre el grafo que se está dibujando como JButton, JLabel, JTextArea, entre otros.

```
public class Ventana implements MouseListener, ActionListener, MouseMotionListener{
    int cantidad = 0;

    ArrayList<DibujarPunto> puntos;
    ArrayList<DibujarArista> aristas;
    static JFrame frame;
    JButton aplicar,nuevo;
    Container contenedor;
    JRadioButton CrearNodo, CrearArista;
    JLabel titulo;
    String larista = "0";
    ButtonGroup grupo;
    public JTextArea area;
    DibujarPunto pun[];
    public LienzodeGrafo lienzo;
    Grafo grafo;
    int j,i;
    int x,y;

private int contador = 0;
```

El constructor de la clase ventana, se utiliza para inicializar las variables y se configuró la interfaz gráfica, los objetos Font se utiliza para establecer el estilo y el tamaño de la interfaz gráfica, el new Grafo() se utilizó para almacenar la información sobre el grado que se estará dibujando. Las líneas de lienzo es un objeto de lienzo de grafo donde se establecerá la posición y el tamaño.

```
public Ventana(){
    frame = new JFrame();
    contenedor=frame.getContentPane();

    Font font=new Font("Tahoma",Font.BOLD,11);
    Font font1=new Font("Tahoma",Font.PLAIN,12);

    grafo=new Grafo();

    lienzo = new LienzodeGrafo();
    lienzo.setBounds(0, 0, 600, 600);
    lienzo.setBorder(BorderFactory.createBevelBorder(1));
    lienzo.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
```

El mouseClixked se ejecuta cada vez que el usuario dará un clic en el lienzo del grafo, El if es para la verificación si se hace un doble clic en el lienzo y aparecerá la un mensaje para crear el nodo y el usuario podrá ingresar el nombre del nodo, do- While se utilizó para que el usuario ingresara el nombre del nodo, de esta manera se verifica si el dato ingresado ya existe, si en dado caso ingresa un mismo nodo o si no ingreso el nombre del nodo aparecerá un mensaje de error, esto se repite y muestra el mensaje la veces que el usuario ingresa un nuevo nodo. El constructor Dibujar Punto toma en coordenadas del evento clic y el nombre del nodo ingresado, se agrega el nombre del nodo usando el método ingresarNodo(), luego se dibuja el punto en el lienzo y se agregarán a la lista de puntos.

```
public void mouseClicked(MouseEvent evento){
    if (evento.getClickCount() == 2&&CrearNodo.isSelected()==true){
        String nombre = "existen";

        do{
            try{
                nombre = JOptionPane.showInputDialog(null,"Ingrese nombre del Nodo");
                nombre.length();
            }
            catch(NullPointerException e){
                return;
            }if(grafo.getNombres().contains(nombre)||nombre==null){
                JOptionPane.showMessageDialog(null,"Los nombres de los nodos no pueden ser iguales.","Ingrese nuevamente el nodo",JO
                nombre="existen";
            }
        }
        while(nombre=="existen");

        DibujarPunto punto = new DibujarPunto((int) evento.getPoint().getX() - 5, (int) evento.getPoint().getY() - 5, nombre);
        grafo.ingresarNodo(nombre);
        punto.pintarPunto(lienzo.getGraphics());
        puntos.add(punto);
        lienzo.setPuntos(puntos);
    }
```

Para Crear la arista, se recorrerá un ciclo for que se ejecutara para cada punto dibujado en el lienzo y el nodo seleccionado se guarda en la posición 1 en el array cambiando de color del nodo para indicar que ha sido seleccionado, cuando ambos nodos han sido seleccionados se dibujara una línea entre los dos nodos.

```
if(CrearArista.isSelected()){
    for (int i = 0; i < puntos.size(); i++){
        if (puntos.get(i).ecuacionDeCirculo(e.getPoint())){
            pun[1] = puntos.get(i);
            pun[1].setColorPunto( Color.RED);
            break;
        }else
            if(pun[1]!=null&&pun[1]!=pun[0])
                    pun[1].setColorPunto(Color.BLUE);
    }

        if(pun[0]!=null){
                lienzo.setA(new Point(x,y));
                lienzo.setB(e.getPoint());

                lienzo.punto=true;
                lienzo.repaint();
        }
    contador=2;
    }
```

La clase Nodo se tiene un objeto y una lista llamadas nombre y enlaces que conectan a otros nodos, el contador enlacesExistentes se utiliza para llevar el registro de la cantidad de enlaces, .getEnlaces() retornara la lista de enlaces del nodo, y getNombre() retornara el nombre del nodo.

```
public class Nodo {

private String nombre;
private ArrayList<Enlace> enlaces;
private int enlacesExistentes;

public ArrayList<Enlace> getEnlaces(){
    return enlaces;
}

public Nodo(String newName){
    nombre = newName;
    enlacesExistentes = -1;
    enlaces = new ArrayList<Enlace>();
}

public int getEnlacesExistentes(){
    return enlacesExistentes;
}

public String getNombre(){
    return nombre;
}
```

El método enlacePosicion devolverá el peso del enlace que se encuentra en la posición posi de la lista de enlaces del nodo, y el NodoPosición devuelve el destino del enlace que se encuentra en la posición posi de la lista de enlaces del nodo.

```
public int existeEnlace(String enlazar){
    for (int i = 0; i < enlaces.size(); i++){
        Enlace miEnlace;
        miEnlace = enlaces.get(i);
        if (miEnlace.getDestino().equals(enlazar))
            return i;
    }
    return -1;
}

public double enlacePosicion(int posi){
    Enlace miEnlace;
    miEnlace = enlaces.get(posi);
    return miEnlace.getPeso();
}

public String NodoPosicion(int posi){
    Enlace miEnlace;
    miEnlace = enlaces.get(posi);
    return miEnlace.getDestino();
}

}
```

La clase Grafo contiene los nodos y las aristas que conforman el grafo, el constructor inicia con tres listas, la lista de nombres, una lista de nodos y una lista de aristas, la función ingresarNodo permitirá agregar un nuevo nodo al grafo, la función adicionarEnlace permite agregar una nueva arista al grafo y el peso de la arista, además se agrega el enlace correspondiente al nodo inicial y al nodo final utilizando la función agregarEnlace del objeto Nodo.

```
public Grafo(){
    nombres=new ArrayList<String>();
    nodos=new Hashtable <String,Nodo>();
    aristas=new ArrayList <Arco>();
}

public void ingresarNodo(String nombre){
    nombres.add(nombre);
    nodos.put(nombre,new Nodo(nombre));
}
public void adicionarEnlace(String nodoInicial,String nodoTerminal,float peso){
    Arco nuevo=new Arco(nodoInicial,nodoTerminal,peso);
    int i=buscarIndice(nuevo.getPeso());

    if(i==-1)
        aristas.add(nuevo);
    else
        aristas.add(i,nuevo);
        nodos.get(nodoInicial).agregarEnlace(nodoTerminal,peso);
        nodos.get(nodoTerminal).agregarEnlace(nodoInicial,peso);
}
public boolean busarArista(Arco arco){
    for(int i=0;i<aristas.size();i++){
        Arco otro=aristas.get(i);
        if(arco.getInicial().equals(otro.getInicial())&&arco.getTerminal().equals(otro.getTerminal())&&arco.g
            aristas.remove(otro);
            return true;
        }
    }
```

El primer constructor Enlace recibe dos parámetros desti que representará el destino del enlace y peso1 que representará el peso del enlace, este constructor inicializa los atributos destino y peso con los valores de los parámetros recibidos. El segundo constructor recibe los parámetros destii que representará el destino de enlace inicializa el atributo destino con el valor del parámetro recibido y el atributo peso.

```
public class Enlace {

private String destino;
private double peso;

public Enlace(String desti, double pesol){
    destino = desti;
    peso = pesol;
}

public Enlace(String desti){
    destino = desti;
    peso = -1;
}

public void modificar(double pesol){
    peso = pesol;
}
```

En esta clase Dibujar Punto contiene el constructor que recibe la coordenada x, la coordenada y, y el nombre del nodo, contiene una variable ubicación que es un objeto point que almacena las coordenadas de los nodos ingresadas por el usuario. El método pintarPunto recibe un objeto Graphics y dibuja el punto en la posición especificada por ubicación, asimismo esta clase contiene un método Pintar punto, que dará un color y pintará ese punto con el color especificado.

```
    private Point ubicacion;
    private String nombre;
    private Color colorPunto = Color.BLUE;
    private static final int RADIO = 10;

    public DibujarPunto(int x, int y, String nombre){
        ubicacion = new Point(x, y);
        this.nombre = nombre;
    }

public void pintarPunto(Graphics g){
    g.setColor(Color.BLACK);
    g.drawString(nombre, ubicacion.x - 3, ubicacion.y - 3);
    g.setColor(colorPunto);
    g.fillOval(ubicacion.x, ubicacion.y, 10, 10);
}

public void pintarPunto(Graphics g, Color color){
    g.setColor(Color.BLACK);
    g.drawString(nombre, ubicacion.x - 3, ubicacion.y - 3);
    g.setColor(color);
    g.fillOval(ubicacion.x, ubicacion.y, 10, 10);
}

public boolean ecuacionDeCirculo(Point punto){
    return (((punto.x - ubicacion.x) * (punto.x - ubicacion.x) + (punto.y - ubicacion.y) * (punto.
}
```

La clase dibujarArista se encargará de dibujar una arista conectada entres dos nodos, el constructo toma dos objetos Dibujar Punto que representan los puntos iniciales y finales de la arista, el peso de la arista. El método pintarRecta es la que se encarga de dibujar la arista, se obtiene la ubicación de los nodos, y termina en la función drwline para dibujar una línea recta que une ambos puntos.

```
public DibujarArista(DibujarPunto puntoA, DibujarPunto puntoB, int tipo, float peso){
    arista = new Arco(puntoA.getNombre(), puntoB.getNombre(), peso);

    a = puntoA;
    b = puntoB;
    this.tipo = tipo;
    this.peso = peso;
}

public void pintarRecta(Graphics ga){
    inicial = a.getUbicacion();
    terminal = b.getUbicacion();

    Graphics2D g = (Graphics2D) ga;
    double a = (inicial.y - terminal.y);
    double b = (inicial.x - terminal.x);
    double m = a / b;
    double grado = Math.atan(m);

    switch (tipo){
        case 0:
            g.setColor(color);
            g.drawLine(inicial.x + 5, inicial.y + 5, terminal.x + 5, terminal.y + 5);
            g.setColor(aux);
            g.drawString(peso + "", (inicial.x + terminal.x) / 2, (inicial.y + terminal.y) / 2);
            break;
        }
```

El método aplicarKruskal toma como entrada un objeto de tipo Grafo y devolverá el mismo objeto que representa el árbol de expansión mínima del grafo de entrada. En el código se inicia con un árbol vació que se construye cuando el usuario ingresa las aristas de menor peso. El método Haycliclo es para comprobar que si agrega una arista a un nodo especifico crearía un ciclo en el árbol, Se realizará una búsqueda recursiva desde el nodo terminal para comprobar si hay un camino a lo largo de la cual se pueda llegar de nuevo al nodo inicial, sin pasar por el nodo.

```
public Grafo aplicarKruskal(Grafo grafo){
    Grafo árbol=new Grafo();
    ArrayList<String> nodos=grafo.getNombres();

    for(int j=0;j<nodos.size();j++){
        árbol.ingresarNodo(nodos.get(j));
    }

    ArrayList<Arco> L=(ArrayList<Arco>)grafo.getAristas().clone();
    Arco pro=L.get(0);
    árbol.adicionarEnlace(pro.getInicial(), pro.getTerminal(), pro.getPeso());
    L.remove(pro);

    while(L.size()!=0){
        pro=L.get(0);

        if(HayCiclo(árbol, pro,árbol.getNodo(pro.getTerminal()) , pro.getTerminal())==false)
            árbol.adicionarEnlace(pro.getInicial(), pro.getTerminal(), pro.getPeso());
        L.remove(pro);
    }

return árbol;
}
```

```
public boolean HayCiclo(Grafo g,Arco aVerificar,Nodo terminal,String N){
    ArrayList<Enlace> aux=terminal.getEnlaces();

    if(aux.size()==0)
            return false;

    if(terminal.existeEnlace(aVerificar.getInicial())!=-1)
            return true;

    for(int i=0;i<aux.size();i++){
        Enlace nodo=aux.get(i);

        if(nodo.getDestino().equals(N)==false)

            if( HayCiclo(g,aVerificar,g.getNodo(nodo.getDestino()),terminal.getNombre()))
                return true;
    }

return false;
}

}
```