LABORATORIO #2
SIMULADOR DE CONTAGIOS COVID-19

MANUAL DEL SISTEMA

UNIVERSIDAD DEL NORTE

INGENIERÍA DE SISTEMAS
06 de Noviembre 2020

DYLAN CANTILLO ARRIETA
FREDDY FAJARDO HERNANDEZ
LAURA GONZÁLEZ SOLANO

UNIVERSIDAD DEL NORTE

Contenido

Librería ArrayList

Esta es una librería propia, implementada para solucionar la necesidad de guardar los diferentes nodos hijos que puede tener un nodo, ya que se implementa un árbol n-ario, lo cual indica que la cantidad de hijos no es definida.

En esta librería creamos dos clases, una Elemento y otra ArrayList. Estas clases trabajan como una lista enlazada simple. Y simula los métodos del ArrayList original, propio de Java, es decir, cuenta con el método ”add” el cual agrega un elemento al final de la lista, método “size”, el cual retorna el número de elementos que tiene la lista, y finalmente el método “get”, el cual recorre la lista y retorna el elemento ubicado la posición indicada por el parámetro.

Método add

```
private Elemento addElemento(Elemento ptr,
        Object info) {

    Elemento p = ptr;
    Elemento q = new Elemento();

    q.info = info;
    if (ptr == null) {
        ptr = q;

    } else {
        while (p.Link != null) {
            p = p.Link;
        }
        p.Link = q;
    }
    return ptr;
}
```

Método get

```
public Object get(int i) {
    int cont = 0;
    Elemento p = ptr;

    while (p != null) {
        if (i == cont) {
            return p.info;
        } else {
            p = p.Link;
            cont++;
        }
    }

    return null;
}
```

Método size

```
public int size() {
    int tam = 0;
    Elemento p = ptr;

    while (p != null) {
        p = p.Link;
        tam++;
    }

    return tam;
}
```

Clase Elemento

```
public class Elemento {

    Object info;
    Elemento Link;

}
```

Visual

El paquete visual contiene todo aquello que se muestra en pantalla para el usuario, maneja la ventana principal y la ventana que muestra el grafo. Todo el paquete hace uso de JFrame y JPanel, junto con la clase Graphics que nos otorga java.

Ventana principal

Esta ventana cuenta con el main que ejecuta el programa y abre al usuario una ventana con varios botones. Los botones principales son los botones de “Generar” y “Salir”. El botón generar toma los valores de el JTextField y el grupo de botones, para próximamente, ya hecho el proceso de lectura y validación de las entradas, ejecutar la clase Grafo y generar el grafo, creando una ventana con el Panel.

```
JFrame ventana = new JFrame("Simulador");
Panel panel = new Panel(ventana);
Grafo grafo = new Grafo(num, panel, opcion);

ventana.add(panel);
ventana.setSize(ANCHO, ALTO);
ventana.setUndecorated(true);
ventana.setLocationRelativeTo(null);
ventana.setVisible(true);
this.dispose();
```

(Creación ventana con el Panel)

Panel

Luego de tener el grafo con el conjunto de nodos y arcos, se prepara su ejecución visual, primeramente ubicando sus JButton y JLabel. Posteriormente, espera a la clase Grafo que le mande la indicación de comenzar.

```
public void start(ArrayList nodos, ArrayList arcos) {
    this.nodos = nodos;
    rellenarObjetos();
    this.arcos = arcos;
    repaint();
}
```

(Método Start)

Después, la clase procede a crear unos rectángulos para utilizarlos luego como botones.

```
private void rellenarObjetos() {

    for (int i = 0; i < nodos.size(); i++) {
        Nodo nodo = (Nodo) nodos.get(i);
        Rectangle r = new Rectangle(nodo.getPosicionX() - 25,
                nodo.getPosicionY() - 25, nodo.ANCHONODO, nodo.ANCHONODO);
        objetos.add(r);
    }
}
```

(Método rellenarObjetos)

La clase también cuenta con la clase de MouseListener, para detectar los clic del ratón y proceder a hacer los procesos dependiendo de donde se haga clic.

```
@Override
public void mousePressed(MouseEvent me) {

    if (objetos.size() != 0) {
        ArrayList rects = objetos;

        Point punto = MouseInfo.getPointerInfo().getLocation();
        Rectangle r = new Rectangle(punto.x, punto.y, 1, 1);

        seleccion = false;
        dijkstra.setVisible(false);
        for (int i = 0; i < rects.size(); i++) {
            Rectangle rect = (Rectangle) rects.get(i);
            ((Nodo) nodos.get(i)).setSeleccionFalse();
            if (r.intersects(rect)) {
                ((Nodo) nodos.get(i)).setSeleccionTrue();
                nodoSeleccionado = (Nodo) nodos.get(i);
                seleccion = true;
                if (numIteraciones > 0) {
                    dijkstra.setVisible(true);
                }
            }
        }

        for (int i = 0; i < arcos.size(); i++) {
            Arco arco = (Arco) arcos.get(i);
            arco.setSeleccionFalse();
            arco.setRutaContagioFalse();
        }

        repaint();
    }
}
```

(Método mousePressed)

Lista

El paquete Lista es muy simple, contiene la clase Lista y Elemento, la cual simula una lista enlazada simple donde posteriormente se guardaran los Nodos formando la lista de adyacencia.

Lista de adyacencia

La clase Lista hace uso de la clase Elemento y simula una lista enlazada simple.

```java
public class Lista {

    private Elemento ptr;

    public Lista() {
        ptr = null;
    }

    private Elemento addElemento(Elemento ptr, Nodo info, int peso) {

        Elemento p = ptr;
        Elemento q = new Elemento();
        q.nodo = info;
        q.peso = peso;

        if (ptr == null) {
            ptr = q;

        } else {
            while (p.link != null) {
                p = p.link;
            }
            p.link = q;
        }
        return ptr;
    }
}
```

(Clase Lista)

Elementos

Clase que que contiene el Nodo, peso o distancia y el link para el siguiente Elemento.

```java
public class Elemento {

    public Nodo nodo;
    public Elemento link;
    public int peso;

}
```

(Clase Elemento)

Grafo

El paquete grafo contiene todo lo que respecta al grafo, los métodos (dijkstra), nodos, arcos.

Grafo

La clase Grafo cuando es instanciada en la ventana principal, determina las distancias necesarias, que son a la distancia que un nodo estará de otro, que este además de crear el conjunto de nodos, les da una posición, que será sobre el plano de manera circular, siguiendo la posición de una circunferencia creada por este, con respecto a un radio.

```
private void puntosdeCircunferencia() {
    for (int i = MEDIONODO; i <= RADIO * 2 + MEDIONODO; i++) {
        for (int j = MEDIONODO; j <= RADIO * 2 + MEDIONODO; j++) {
            if (distanciaNodos(i, j, CENTRO.x, CENTRO.y) == RADIO) {
                circunferencia.add(new Point(i, j));
            }
        }
    }
}
```

(Creación circunferencia)

```
private void getDistacia() {
    if (numNodos < 19 && numNodos > 0) {
        int angulo = 360 / numNodos;
        distancia = (int) Math.ceil(2 * RADIO * Math.sin(Math.toRadians(angulo / 2)));
        inicializarNodos();
    }
}
```

(Determinar distancia entre nodos)

En este proceso la clase Grafo obtiene un primer nodo con posición al azar, y luego va creando los demás a partir de este, con la distancia obtenida anteriormente.

```
private void inicializarNodos() {

    int x, y;
    do {
        x = (int) (Math.random() * (ANCHO - MEDIONODO) + MEDIONODO);
        y = (int) (Math.random() * (ANCHO - MEDIONODO) + MEDIONODO);
    } while (distanciaNodos(x, y, CENTRO.x, CENTRO.y) != RADIO);
    nodos.add(new Nodo(x, y, usoMascarilla()));
    numNodos--;

    while (numNodos != 0) {

        int i = 0;
        while (i < circunferencia.size()) {

            int puntoX = ((Point) circunferencia.get(i)).x;
            int puntoY = ((Point) circunferencia.get(i)).y;
            Nodo nodo = (Nodo) nodos.get(nodos.size() - 1);;

            if (isPosible(puntoX, puntoY, nodo) && isDiferente(puntoX, puntoY)) {

                nodos.add(new Nodo(puntoX, puntoY, usoMascarilla()));
                numNodos--;
                i = circunferencia.size();
            }
            i++;
        }
    }
    inicializarAristas();
}
```

(Posición nodos)

Con las aristas realiza un proceso parecido al de los nodos, solo que recorre los nodos y obtiene un número al azar para saber cuantas conexiones tendrá ese nodos, luego obtiene los nodos con que se conectará también de forma al azar.

```
private void inicializarAristas() {

    for (int i = 0; i < nodos.size(); i++) {
        int conexiones = (int) (Math.random() * (nodos.size() - 2) + 1);

        ArrayList numConexiones = new ArrayList();
        int l = 0;
        while (l < conexiones) {
            int nuevoNum;
            do {
                nuevoNum = (int) (Math.random() * nodos.size());
            } while (nuevoNum == i);

            boolean sw = true;
            for (int k = 0; k < numConexiones.size(); k++) {
                if ((int) numConexiones.get(k) == nuevoNum) {
                    sw = false;
                }
            }
            if (sw) {
                numConexiones.add(nuevoNum);
                l++;
            }
        }

        Nodo nodo = (Nodo) nodos.get(i);
        for (int k = 0; k < numConexiones.size(); k++) {
            int peso = generarDistancia();
            nodo.addNodo((Nodo) nodos.get((int) numConexiones.get(k)), peso);
            arcos.add(new Arco(peso, nodo, (Nodo) nodos.get((int) numConexiones.get(k))));
        }
    }
    panel.start(nodos, arcos);
}
```

(Posición aristas)

Vértices

Nodo

Esta clase Nodo guarda la posición donde se dibujara, además de contener la información de la persona, el nombre, si está contagiado o no, la lista de adyacencia de este, ya que cada nodo tiene su lista de adyacencia, si tiene mascarilla o no, tambien sabe si es el nodo seleccionado del grafo. Esta clase también se dibuja a sí misma con el método paint.

```java
public void paint(Graphics g) {

    if (contagio) {
        g.setColor(Color.RED);
    } else {
        if (!seleccionado) {
            g.setColor(Color.WHITE);
        } else {
            g.setColor(Color.YELLOW);
        }
    }
    g.fillOval(posicionX - ANCHONODO / 2, posicionY - ANCHONODO / 2,
            ANCHONODO, ANCHONODO);
}
```

(Método paint)

También cuenta con un método que usa cuando está contagiado, para contagiar a los demás nodos dependiendo del uso de mascarilla y a la distancia a la que estén.

```java
private void metodoContagiar(Nodo contagiado,
        Nodo noContagado, int distancia) {
    int probabilidad = (int) (Math.random() * 100);

    if (!contagiado.isMascarilla()) {
        if (!noContagado.isMascarilla()) {
            if (distancia > 2) {
                if (probabilidad <= 80) {
                    noContagado.setContagio();
                }
            } else if (distancia <= 2) {
                if (probabilidad <= 90) {
                    noContagado.setContagio();
                }
            }
        } else {
            if (distancia > 2) {
                if (probabilidad <= 40) {
                    noContagado.setContagio();
                }
            } else if (distancia <= 2) {
                if (probabilidad <= 60) {
                    noContagado.setContagio();
                }
            }
        }
    } else {
        if (!noContagado.isMascarilla()) {
            if (distancia > 2) {
                if (probabilidad <= 30) {
                    noContagado.setContagio();
                }
            } else if (distancia <= 2) {
                if (probabilidad <= 40) {
                    noContagado.setContagio();
                }
            }
        } else {
            if (distancia > 2) {
                if (probabilidad <= 20) {
                    noContagado.setContagio();
                }
            } else if (distancia <= 2) {
                if (probabilidad <= 30) {
                    noContagado.setContagio();
                }
            }
        }
    }
```

Nombre nodos

La clase NombresNodo lee el archivo Lista Nombres.txt que se encuentra en la carpeta del proyecto y le asigna a cada nodo un nombre.

```java
public static String escogerNombre(int num) {
    if (num < NOMBRES.size()) {
        return (String) NOMBRES.get(num);
    }
    return "";
}
```

Aristas

Arco

Esta clase Arco guarda las coordenadas de la arista para ser dibujada por sí misma en el Panel, también guarda el nodo origen y el de llegada, ya que es una arista dirigida y también guarda su peso o distancia. La clase cuenta con un método llamado reducirArco, que su función es hacer que la punta de la arista no llegue al centro del nodo, sino que llegue a su borde para poder dibujar luego una flecha.

```
private void reducirArcos() {

    double distancia = Math.sqrt(Math.pow(X1 - X2, 2) + Math.pow(Y1 - Y2, 2));
    double factor = (nodoLlegada.ANCHONODO / 2) * Math.pow(distancia, -1);
    X2 = (int) (X2 + factor * (X1 - X2));
    Y2 = (int) (Y2 + factor * (Y1 - Y2));

}
```

(Método reducirArco)

Su proceso es buscar un factor de disminución, basado en los métodos de triángulos, luego resta el exceso de línea que hay dentro del nodo.

```
public void paint(Graphics g) {

    if (rutaContagio) {
        escribirPeso(g);
        g.setColor(Color.RED);
    } else {
        if (seleccionado) {
            escribirPeso(g);
            g.setColor(Color.YELLOW);
        } else {
            g.setColor(Color.PINK);
        }
    }
    g.drawLine(flecha.getLineaP1().x, flecha.getLineaP1().y,
            flecha.getAristaIP().x, flecha.getAristaIP().y);
    g.drawLine(flecha.getLineaP1().x, flecha.getLineaP1().y,
            flecha.getArista2P().x, flecha.getArista2P().y);
    g.drawLine(X1, Y1, X2, Y2);
}
```

(Método paint)

Flecha

Esta clase busca realizar dos líneas con un ángulo de 45° con la línea entre los dos nodos, para poder simular la forma de una flecha.

```
private Point trasladarFigura(double x, double y) {
    double xp = x - lineaP1.x;
    double yp = lineaP1.y - y;
    return new Point((int) xp, (int) yp);
}
```

(Traslación)

```
private Point calcularCateto(double x, double y) {
    double angulo = Math.atan(y / x);
    double a = Math.sqrt(Math.pow(LONGITUD, 2)
            / (1 + Math.pow(Math.tan(angulo), 2)));
    double b = Math.sqrt(Math.pow(LONGITUD, 2) - Math.pow(a, 2));
    return new Point((int) a, (int) b);
}
```

(Catetos opuesto y adyacente)

```
private Point calcularFactor(double x, double y) {
    double fx = x / Math.abs(x);
    double fy = y / (Math.abs(y) * -1);
    return new Point((int) fx, (int) fy);
}
```

(Factor de disminución)

```
private Point rotacionHoraria(double x, double y) {
    double xp = x * Math.cos(DELTA) + y * Math.sin(DELTA);
    double yp = (-1 * x * Math.sin(DELTA)) + y * Math.cos(DELTA);
    xp = lineaP1.x + xp;
    yp = lineaP1.y - yp;
    return new Point((int) xp, (int) yp);
}
```

(Rotación horaria)

```
private Point rotacionAntiHorario(double x, double y) {
    double xp = x * Math.cos(DELTA) + (-1 * y * Math.sin(DELTA));
    double yp = x * Math.sin(DELTA) + y * Math.cos(DELTA);
    xp = lineaP1.x + xp;
    yp = lineaP1.y - yp;
    return new Point((int) xp, (int) yp);
}
```

(Rotación anti-horaria)

Dijkstra

Esta clase se encarga de calcular el Dijkstra de los nodos contagiados, para luego usar estos resultados, para calcular la ruta de contagio más rápida hacia los otros nodos no contagiados.

```
public static ArrayList dijkstra(ArrayList nodos, Nodo nodoSeleccionado) {
    ArrayList ptr = new ArrayList();

    for (int i = 0; i < nodos.size(); i++) {
        if (ptr.size() == 0) {
            Lista p = new Lista();
            p.anterior = null;
            p.distancia = Integer.MAX_VALUE;
            ptr.add(p);
        } else {
            Lista q = new Lista();
            q.anterior = null;
            q.distancia = Integer.MAX_VALUE;
            ptr.add(q);
        }
    }
    int num = posicionNodo(nodoSeleccionado, nodos);
    Lista p = (Lista) ptr.get(num);
    p.anterior = nodoSeleccionado;
    p.distancia = 0;

    int repeticiones = 0;
    while (isEstablecido(ptr) && repeticiones <= nodos.size()) {
        Nodo nodo = getMejor(nodos, ptr);
        Elemento q = nodo.getListaADY().getPtr();
        while (q != null) {
            if (((Lista) ptr.get(posicionNodo(nodo, nodos))).distancia + q.peso
                    < ((Lista) ptr.get(posicionNodo(q.nodo, nodos))).distancia) {
                ((Lista) ptr.get(posicionNodo(q.nodo, nodos))).distancia
                        = ((Lista) ptr.get(posicionNodo(nodo, nodos))).distancia + q.peso;
                ((Lista) ptr.get(posicionNodo(q.nodo, nodos))).anterior = nodo;
            }
            q = q.link;
        }
        repeticiones++;
    }
    return ptr;
}
```

(Dijkstra)

El método asigna primeramente a cada nodo una distancia de infinito (aproximadamente) y un nodo anterior null, luego toma el nodo contagiado y le asigna nodo anterior a sí mismo y una distancia de 0. Posteriormente, calcula para cada uno de sus nodos su distancia y toma el menor, al menor lo coloca como visitado y realiza el mismo proceso de mirar sus adyacentes y junto con los del anterior nodos mira a ver cual es el menor, así hasta que todos los nodos hayan sido visitados.

Archivo

La clase Archivo, crea un archivo si este no existe dentro de la carpeta del programa, si ya existe la sobreescribe. En el archivo creado o editado, escribe la lista de adyacencia de todos los grafos que se han creado a lo largo que se ha ejecutado el programa.

```
private static String escribir(ArrayList nodos, boolean primera) {
    StringBuilder sb = new StringBuilder();
    if (!primera) {
        sb.append("\n");
        sb.append("\n");
        sb.append("_____________________________________________________________");
        sb.append("\n");
    }

    sb.append("Lista de adyacencia del grafo");
    sb.append("\n");
    sb.append("\n");
    for (int i = 0; i < nodos.size(); i++) {
        Nodo nodo = (Nodo) nodos.get(i);
        sb.append(nodo.getNombre()).append("_").append(i);
        Elemento p = nodo.getListaADY().getPtr();
        while (p != null) {
            sb.append("   --->   ").append(p.nodo.getNombre());
            p = p.link;
        }
        sb.append("   --->   ").append("/");
        sb.append("\n");
        sb.append(" |");
        sb.append("\n");
        sb.append(" |");
        sb.append("\n");
        sb.append(" V");
        sb.append("\n");
        if (i == nodos.size() - 1) {
            sb.append(" /");
        }
    }
    return sb.toString();
}
```