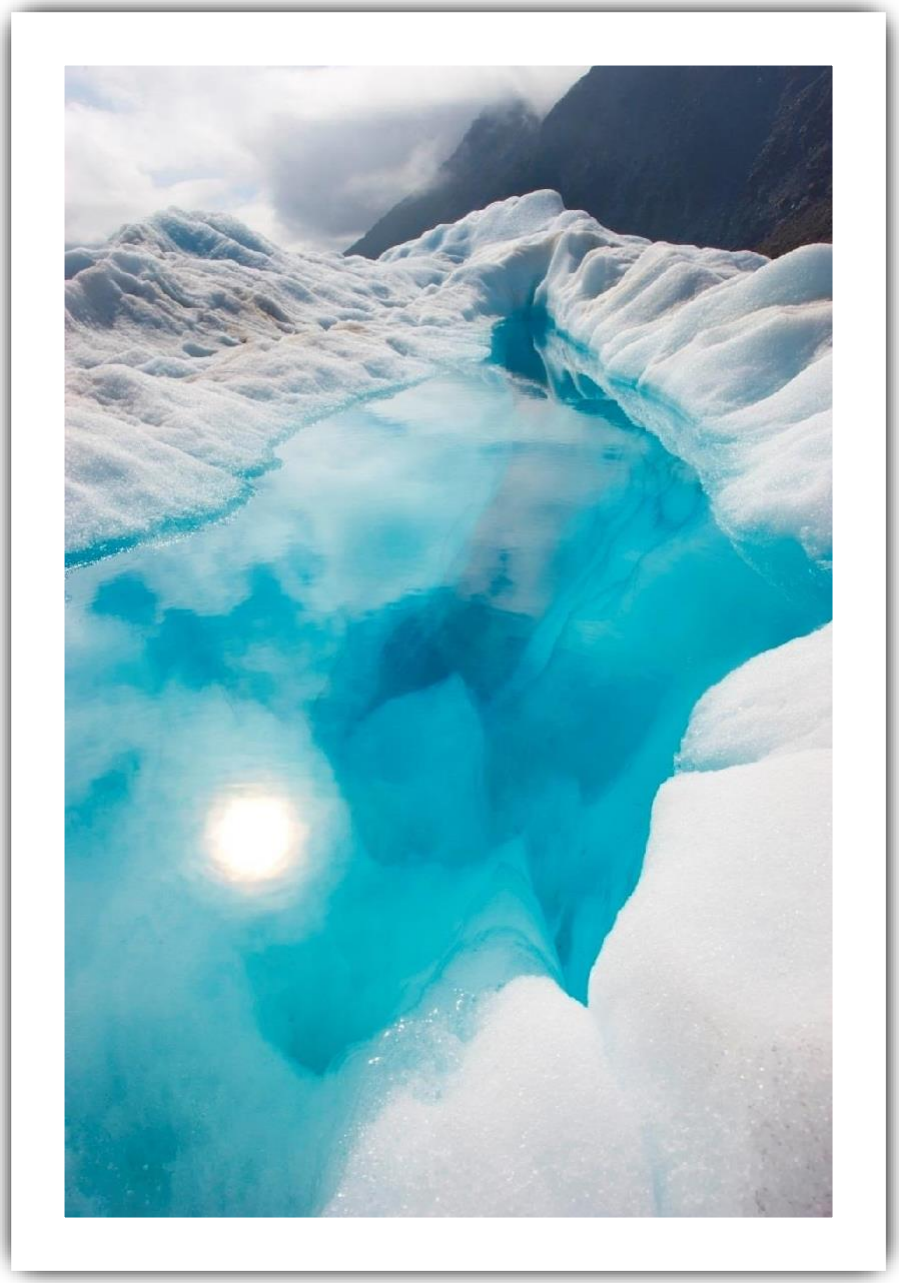


Práctica 2 AMC

AFD y AFND.

Uadad Sidelgaum Limam y Jesús Tejón Carillo. Algoritmos y Modelos de Computación.

INDICE

IMPLEMENTACIÓN DE LA PRÁCTICA....3
CODIGO FUENTE....4
AFND: 4
AFD: 7
Transición: 9
ControladorFich: 10
Aplicación: 11
CONJUNTO DE PRUEBAS 13

IMPLEMENTACIÓN DE LA PRÁCTICA

Para la realización de la practica hemos partido de las cabeceras ofrecidas en el enunciado de la práctica, de las cuales hemos realizado su debida implementación de métodos.

El primer problema surge a la hora de leer y escribir en los ficheros. Para ello hemos implementado una clase especifica en eso y a la que recurrimos cuando necesitamos realizar cualquier acción de lectura o escritura de un fichero.

Por ultimo hemos realizado una interfaz gráfica con JFrame para el menú principal y los submenús, los mensajes se lanzan a través de un JOptionPanel y los grafos son graficados con la librería “Jung”.

CODIGO FUENTE

AFND:

```java
import edu.uci.ics.jung.algorithms.layout.CircleLayout;
import edu.uci.ics.jung.graph.util.EdgeType;
import edu.uci.ics.jung.visualization.VisualizationViewer;
import edu.uci.ics.jung.visualization.decorators.ToStringLabeller;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Paint;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;
import javax.swing.JFrame;
import edu.uci.ics.jung.graph.DirectedSparseMultigraph;
import javax.swing.JOptionPane;

/**
 *
 * @author wadad
 */
public class AFND implements Proceso {

    private int[] estadosFinales; //indica cuales son los estados Finales
    private List<Transicion> transiciones; //indica la lista de transiciones del AFND
    private List<Transicion> transicionesλ; //indica la lista de transiciones λ del AFND
    private List<Integer> estados;
    private JFrame frame;

    public AFND() {
        transiciones = new ArrayList<>();
        transicionesλ = new ArrayList<>();
        estados = new ArrayList<>();
        frame = new JFrame(title: "Grafo AFND");
    }
```

```java
/**
 * metodo que añade una transición la grafo
 *
 * @param e1 numero del nodo inicial de la transición
 * @param simbolo simbolo de la transición
 * @param e2 numeros de los nodos finales de la transición
 */
public void agregarTransicion(int e1, char simbolo, int[] e2) {
    Transicion tran = new Transicion(inicio:e1, destinos: e2, simbolo);
    transiciones.add(e: tran);
    añadeEstado(e1, e2);
}

/**
 * metodo que añade una transición lamda
 *
 * @param e1 numero del nodo inicial de la transición
 * @param e2 numeros de los nodos finales de la transición
 */
public void agregarTransicionλ(int e1, int[] e2) {
    Transicion tran = new Transicion(inicio:e1, destinos: e2);
    transicionesλ.add(e: tran);
    añadeEstado(e1, e2);
}

/**
 * metodo que devuelve los estados finales de una transición pasandole su
 * estado inicial y su simbolo
 *
 * @param estado estado incicial de la transicion buscada
 * @param simbolo simbolo de la transición buscada
 * @return Devuelve el array de los nodos finales de la transición, si no
 * existe devuelve null
 */
```

```java
private int[] transicion(int estado, char simbolo) {

    int i = 0;
    while (i < transiciones.size()) {
        if (transiciones.get(index: i).getInicio() == estado && transiciones.get(index: i).getsimbolo() == simbolo) {
            return transiciones.get(index: i).getDestinos();
        }

        i++;
    }

    return null;
}

/**
 * metodo que devuelve los estados finales de una transición pasandole su
 * estado inicial y su simbolo
 *
 * @param macroestado estado incicial de la transicion buscada
 * @param simbolo simbolo de la transición buscada
 * @return Devuelve el array de los nodos finales de la transición, si no
 * existe devuelve null
 */
public int[] transicion(int[] macroestado, char simbolo) {
    ArrayList<Integer> aux = new ArrayList<>();
    for (int i = 0; i < macroestado.length; i++) {
        int[] destinos = transicion(macroestado[i], simbolo);
        if (destinos != null) {
            for (int j = 0; j < destinos.length; j++) {
                if (!aux.contains(destinos[j])) {
                    aux.add(destinos[j]);
                }
            }
        }
    }
```

```
        }
        int[] devolver = new int[aux.size()];
        for (int i = 0; i < aux.size(); i++) {
            devolver[i] = aux.get(index: i);
        }
        return devolver;
    }

    public int[] transicionλ(int estado) {
        int i = 0;
        while (i < transicionesλ.size()) {
            if (transicionesλ.get(index: i).getInicio() == estado) {
                return transicionesλ.get(index: i).getDestinos();
            }
            i++;
        }

        return null;
    }

    /**
     * metodo que comprueba si un estado es nodo final
     *
     * @param estado estado del que se desea saber si es final o inicial
     * @return Devuelve verdadero si es final y falso si no lo es
     */
    @Override
    public boolean esFinal(int estado) {
        int i = 0;
        if (estadosFinales != null) {
            while (i < estadosFinales.length) {
                if (estadosFinales[i] == estado) {
                    return true;
                }
                i++;
```

```
                }
            }
        }
        return false;
    }

    /**
     * metodo que comprueba si un macroestado es nodo final
     *
     * @param macroestado macroestado del que se desea saber si es final o
     * inicial
     * @return Devuelve verdadero si es final y falso si no lo es
     */
    public boolean esFinal(int[] macroestado) {
        boolean aux = false;
        for (int j = 0; j < macroestado.length; j++) {
            aux = aux || esFinal(macroestado[j]);
        }
        return aux;
    }

    public int[] λ clausura(ArrayList<Integer> macroestado) {
        ArrayList<Integer> aux = new ArrayList<>();
        for (int i = 0; i < macroestado.size(); i++) {
            aux.add(e: macroestado.get(index: i));
        }

        for (int i = 0; i < macroestado.size(); i++) {
            int[] destinos = transicionλ(estado:macroestado.get(index: i));
            if (destinos != null) {
                for (int j = 0; j < destinos.length; j++) {
                    if (!aux.contains(destinos[j])) {
                        aux.add(destinos[j]);
                    }
                    if (!macroestado.contains(destinos[j])) {
                        macroestado.add(destinos[j]);
```

```
                    }
                }
            }
        }

        int[] devolver = new int[aux.size()];
        for (int i = 0; i < aux.size(); i++) {
            devolver[i] = aux.get(index: i);
        }
        return devolver;
    }

    /**
     * metodo que comprueba si una cadena se encuentra dentro del grafo
     *
     * @param cadena cadena que se desea comprobar
     * @return Devuelve verdadero si existe en el grafo y falso si no existe
     */
    @Override
    @SuppressWarnings("empty-statement")
    public boolean reconocer(String cadena) {
        char[] simbolo = cadena.toCharArray();
        ArrayList<Integer> estado = new ArrayList<>(); //El estado inicial es el 0
        estado.add(e: 0);
        int[] macroestado = λ_clausura(macroestado: estado);
        for (int i = 0; i < simbolo.length; i++) {
            macroestado = transicion(macroestado, simbolo[i]);
        }
        ArrayList<Integer> devolver = new ArrayList<>(initialCapacity: macroestado.length);
        for (int i = 0; i < macroestado.length; i++) {
            devolver.add(macroestado[i]);
        }
        return esFinal( macroestado: λ_clausura(macroestado: devolver));
    }
```

```
/**
 * metodo que pinta el grafico
 *
 * @throws java.lang.InterruptedException
 */
public void dibujar() throws InterruptedException {
    DirectedSparseMultigraph<String, String> graph = new DirectedSparseMultigraph<>();

    // Add vertices
    for (int i = 0; i < transiciones.size(); i++) {
        String v1 = "q" + transiciones.get(index: i).getInicio();
        graph.addVertex(vertex:v1);

        int[] s = transiciones.get(index: i).getDestinos();
        for (int j = 0; j < s.length; j++) {
            String v2 = "q" + s[j];
            graph.addVertex(vertex:v2);
            graph.addEdge("[ " + v1 + "-" + v2 + "] " + transiciones.get(index: i).getsimbolo(), v1, v2, edge_type:EdgeType.DIRECTED);
        }
    }

    for (int i = 0; i < transicionesλ.size(); i++) {
        String v1 = "q" + transicionesλ.get(index: i).getInicio();
        graph.addVertex(vertex:v1);

        int[] s = transicionesλ.get(index: i).getDestinos();
        for (int j = 0; j < s.length; j++) {
            String v2 = "q" + s[j];
            graph.addVertex(vertex:v2);
            graph.addEdge("[ " + v1 + "-" + v2 + "] " + "λ", v1, v2, edge_type:EdgeType.DIRECTED);
        }
    }

    dibujarGrafo(graph);
}
```

```java
/**
 * metodo que dibuja paso a paso el grafo
 *
 * @param Contador
 */
public void dibujarPasoaPaso(int Contador) {
    DirectedSparseMultigraph<String, String> graph = new DirectedSparseMultigraph<>();

    // Add vertices
    if (Contador <= transiciones.size()) {
        for (int i = 0; i < Contador; i++) {
            String v1 = "q" + transiciones.get(index: i).getInicio();
            graph.addVertex(vertex:v1);

            int[] s = transiciones.get(index: i).getDestinos();
            for (int j = 0; j < s.length; j++) {
                String v2 = "q" + s[j];
                graph.addVertex(vertex:v2);
                graph.addEdge("[ " + v1 + "-" + v2 + "] " + transiciones.get(index: i).getsimbolo(), v1, v2, edge_type:EdgeType.DIRECTED);
            }
        }
        if (Contador <= transicionesλ.size()) {
            for (int i = 0; i < Contador; i++) {
                String v1 = "q" + transicionesλ.get(index: i).getInicio();
                graph.addVertex(vertex:v1);

                int[] s = transicionesλ.get(index: i).getDestinos();
                for (int j = 0; j < s.length; j++) {
                    String v2 = "q" + s[j];
                    graph.addVertex(vertex:v2);
                    graph.addEdge("[ " + v1 + "-" + v2 + "] " + "λ", v1, v2, edge_type:EdgeType.DIRECTED);
                }
            }
        }
        dibujarGrafo(graph);
```

```java
private void dibujarGrafo(DirectedSparseMultigraph<String, String> graph) {
    frame.dispose();
    // Create a visualization viewer with a layout
    VisualizationViewer<String, String> vv = new VisualizationViewer<>(new CircleLayout(g: graph));

    vv.getRenderContext().setVertexLabelTransformer(new ToStringLabeller());
    vv.getRenderContext().setEdgeLabelTransformer(new ToStringLabeller());

    // Set up the vertex shape and paint transformer using Java 8 Function
    Function<String, Paint> vertexPaint = vertex -> Color.yellow;

    vv.getRenderContext().setVertexFillPaintTransformer(vertexPaint::apply);

    // Create a JFrame to display the graph
    frame.setDefaultCloseOperation(operation:JFrame.DISPOSE_ON_CLOSE);
    frame.getContentPane().add(comp: vv, constraints: BorderLayout.CENTER);
    frame.pack();  //frame.setSize(x, y);
    frame.setVisible(b: true);
}

/**
 * metodo que devuelve los ids de los estados finales del grafo
 *
 * @return Devuelve un ArrayList con los ids de los estados finales del
 * grafo
 */
public int[] getEstadosFinales() {
    return estadosFinales;
}

/**
 * metodo que devuelve las transiciones del grafo
 *
 * @return Devuelve un List con las transiciones del grafo
 */
```

```java
public List<Transicion> getTransiciones() {
    return transiciones;
}

/**
 * metodo que devuelve las transiciones lamda del grafo
 *
 * @return Devuelve un List con las transiciones lamda del grafo
 */
public List<Transicion> getTransicionesλ() {
    return transicionesλ;
}

/**
 * metodo que añade estados al array de estados. Compruba si existe y si no
 * existe los añade
 *
 * @param e1 id del estado que se añade
 * @param e2 ids de los estados que se añade
 */
private void añadeEstado(int e1, int[] e2) {
    if (!estados.contains(o: e1)) {
        estados.add(e: e1);
    }
    for (int i = 0; i < e2.length; i++) {
        if (!estados.contains(e2[i])) {
            estados.add(e2[i]);
        }
    }

}
```

AFD:

```java
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Paint;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JFrame;
import edu.uci.ics.jung.algorithms.layout.CircleLayout;
import edu.uci.ics.jung.graph.util.EdgeType;
import edu.uci.ics.jung.visualization.VisualizationViewer;
import edu.uci.ics.jung.visualization.decorators.ToStringLabeller;
import java.util.function.Function;
import edu.uci.ics.jung.graph.DirectedSparseMultigraph;
import javax.swing.JOptionPane;

/**
 *
 * @author wadad
 */
public class AFD implements Proceso {

    private int estadoinicial;
    private ArrayList<Integer> estadosFinales; //indica cuales son los estados Finales
    private List<Transicion> transiciones; //indica la lista de transiciones del AFD
    private List<Integer> estados;
    private JFrame frame;

    public AFD() {
        estadosFinales = new ArrayList<>();
        estadoinicial = 0;
        transiciones = new ArrayList<>();
        estados = new ArrayList<>();
        frame = new JFrame(title: "Grafo AFD");
    }
```

```java
/**
 * metodo que añade una transición la grafo
 *
 * @param e1 numero del nodo inicial de la transición
 * @param simbolo simbolo de la transición
 * @param e2 numero del nodo final de la transición
 */
public void agregarTransicion(int e1, char simbolo, int e2) {
    Transicion tran = new Transicion(inicio: e1, destino: e2, simbolo);
    boolean encontrado = false;
    int i = 0;
    while (i < transiciones.size() && encontrado == false) {
        if (transiciones.get(index: i).getInicio() == e1 && transiciones.get(index: i).getDestino() == e2 && transiciones.get(index: i).getsimbolo() == simbolo)
            encontrado = true;
        }

        i++;
    }
    if (!encontrado) {
        añadeEstado(e1, e2);
        transiciones.add(e: tran);
    } else {
        System.out.println(x: "Dicha transicion existe.");
    }
}

/**
 * metodo que devuelve el estado final de una transición pasandole su estado
 * inicial y su simbolo
 *
 * @param estado estado incicial de la transicion buscada
 * @param simbolo simbolo de la transición buscada
 * @return Devuelve el id del nodo final de la transición, si no existe
 * devuelve -1
 */
```

```java
/**
 * metodo que devuelve el estado final de una transición pasandole su estado
 * inicial y su simbolo
 *
 * @param estado estado incicial de la transicion buscada
 * @param simbolo simbolo de la transición buscada
 * @return Devuelve el id del nodo final de la transición, si no existe
 * devuelve -1
 */
public int transicion(int estado, char simbolo) {
    int estadof = -1;
    int i = 0;
    while (i < transiciones.size() && estadof == -1) {
        if (transiciones.get(index: i).getInicio() == estado && transiciones.get(index: i).getsimbolo() == simbolo) {
            estadof = transiciones.get(index: i).getDestino();
        }

        i++;
    }

    return estadof;
}

/**
 * metodo que comprueba si un estado es nodo final
 *
 * @param estado estado del que se desea saber si es final o inicial
 * @return Devuelve verdadero si es final y falso si no lo es
 */
@Override
public boolean esFinal(int estado) {
    return estadosFinales.contains(o: estado);
}
```

```java
/**
 * metodo que comprueba si una cadena se encuentra dentro del grafo
 *
 * @param cadena cadena que se desea comprobar
 * @return Devuelve verdadero si existe en el grafo y falso si no existe
 */
@Override
public boolean reconocer(String cadena) {
    char[] simbolo = cadena.toCharArray();
    int estado = estadoinicial;
    for (int i = 0; i < simbolo.length; i++) {
        estado = transicion(estado, simbolo[i]);
    }
    return esFinal(estado);
}

/**
 * metodo que añade un estado final
 *
 * @param parseInt estado final que se añade
 */
public void añadirEstadoFinal(int parseInt) {
    estadosFinales.add(e: parseInt);
}

/**
 * metodo que cambia el estado final del grafo
 *
 * @param ei estado que se desea poner como inicial
 */
public void setEstadoInicial(int ei) {
    estadoinicial = ei;
}
```

```java
public void dibujar() throws InterruptedException {
    DirectedSparseMultigraph<String, String> graph = new DirectedSparseMultigraph<>();

    for (int i = 0; i < transiciones.size(); i++) {
        String v1 = "q" + transiciones.get(index: i).getInicio();
        String v2 = "q" + transiciones.get(index: i).getDestino();
        graph.addVertex(vertex:v1);
        graph.addVertex(vertex:v2);
        graph.addEdge("[ " + v1 + "-" + v2 + " ] " + transiciones.get(index: i).getsimbolo(), v1, v2, edge_type:EdgeType.DIRECTED);
    }

    dibujarGrafo(graph);

}

/**
 * metodo que dibuja paso a paso el grafo
 *
 * @param Contador
 */
public void dibujarPasoaPaso(int Contador) {
    DirectedSparseMultigraph<String, String> graph = new DirectedSparseMultigraph<>();

    // Add vertices
    if (Contador <= transiciones.size()) {
        for (int i = 0; i < Contador; i++) {
            String v1 = "q" + transiciones.get(index: i).getInicio();
            graph.addVertex(vertex:v1);
            String v2 = "q" + transiciones.get(index: i).getDestino();
            graph.addVertex(vertex:v2);
            graph.addEdge("[ " + v1 + "-" + v2 + "] " + transiciones.get(index: i).getsimbolo(), v1, v2, edge_type:EdgeType.DIRECTED);

        }

        dibujarGrafo(graph);
```

```java
private void dibujarGrafo(DirectedSparseMultigraph<String, String> graph) {
    frame.dispose();
    // Create a visualization viewer with a layout
    VisualizationViewer<String, String> vv = new VisualizationViewer<>(new CircleLayout(g: graph));

    vv.getRenderContext().setVertexLabelTransformer(new ToStringLabeller());
    vv.getRenderContext().setEdgeLabelTransformer(new ToStringLabeller());

    // Set up the vertex shape and paint transformer using Java 8 Function
    Function<String, Paint> vertexPaint = vertex -> Color.yellow;

    vv.getRenderContext().setVertexFillPaintTransformer(vertexPaint::apply);

    frame.setDefaultCloseOperation(operation:JFrame.DISPOSE_ON_CLOSE);
    frame.getContentPane().add(comp: vv, constraints: BorderLayout.CENTER);
    frame.pack();  //frame.setSize(x, y);
    frame.setVisible(b: true);
}

/**
 * metodo que devuelve el id del estado inicial del grafo
 *
 * @return Devuelve el id del estado inicial del grafo
 */
public int getEstadoinicial() {
    return estadoinicial;
}

/**
 * metodo que devuelve los estados finales del grafo
 *
 * @return devuelve la lista de estados finales del grafo
 */
```

```java
public ArrayList<Integer> getEstadosFinales() {
    return estadosFinales;
}

/**
 * metodo que devuelve las transiciones del grafo
 *
 * @return Devuelve un ArrayList con las transiciones del gra
 */
public List<Transicion> getTransiciones() {
    return transiciones;
}

/**
 * metodo que añade estados al array de estados. Compruba si
 * existe los añade
 *
 * @param e1 id del estado que se añade
 * @param e2 id del estado que se añade
 */
private void añadeEstado(int e1, int e2) {
    if (!estados.contains(o: e1)) {
        estados.add(e: e1);
    }
    if (!estados.contains(o: e2)) {
        estados.add(e: e2);
    }
}

/**
 * metodo que devuelve los estados del grafo
 *
 * @return devuelve la lista de estados del grafo
 */
```

```java
private void añadeEstado(int e1, int e2) {
    if (!estados.contains(o: e1)) {
        estados.add(e: e1);
    }
    if (!estados.contains(o: e2)) {
        estados.add(e: e2);
    }
}

/**
 * metodo que devuelve los estados del grafo
 *
 * @return devuelve la lista de estados del grafo
 */
public List<Integer> getEstados() {
    return estados;
}

public void setEstadoinicial(int estadoinicial) {
    this.estadoinicial = estadoinicial;
}

public void setEstadosFinales(ArrayList<Integer> estadosFinales) {
    this.estadosFinales = estadosFinales;
}
```

Transición:

```java
public class Transicion {

    private int inicio;
    private int destino = -1;
    private int[] destinos;
    private final char simbolo;

    /**
     * Constructor transicion determinista
     *
     * @param inicio estado inicial de la transicion
     * @param destino estado final de la transicion
     * @param simbolo simbolo de la transición
     */
    public Transicion(int inicio, int destino, char simbolo) {
        this.inicio = inicio;
        this.destino = destino;
        this.simbolo = simbolo;
    }

    /**
     * constructor transicion no determinista
     *
     * @param inicio estado inicial de la transicion
     * @param destinos estados finales de la transicion
     * @param simbolo simbolo de la transición
     */
    public Transicion(int inicio, int[] destinos, char simbolo) {
        this.inicio = inicio;
        this.destinos = destinos;
        this.simbolo = simbolo;
    }

    /**
     * constructor transicion lambda
     */
    public Transicion(int inicio, int[] destinos) {
        this.inicio = inicio;
        this.destinos = destinos;
        this.simbolo = '#';

    }

    /**
     * devuelve el id del estado inicial de la transicion
     *
     * @return devuelve el id del estado inicial
     */
    public int getInicio() {
        return inicio;
    }

    /**
     * devuelve el id del estado final de la transicion
     *
     * @return devuelve el id del estado final
     */
    public int getDestino() {
        return destino;
    }

    /**
     * devuelve el id del estado inicial de la transicion
     *
     * @return devuelve los ides de los estados finales
     */
    public int[] getDestinos() {
        return destinos;
    }
```

ControladorFich:

```java
import Modelo.AFD;
import Modelo.AFND;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import javax.swing.JOptionPane;

/**
 *
 * @author wadad
 */
public class ControladorFich {

    private static AFD automataD;
    private static AFND automataND;
    private Scanner scan;

    public Object leerFichero(File fichero) {
        Object devolver = null;
        try {
            if (fichero.exists()) {
                scan = new Scanner(source:fichero);
                String linea = scan.nextLine();
                String[] partes = linea.split(regex: " ");
                if (partes[1].equals(anObject: "AFD")) {
                    devolver = leerAFD();

                } else {

                    devolver = leerAFND();

                }
```

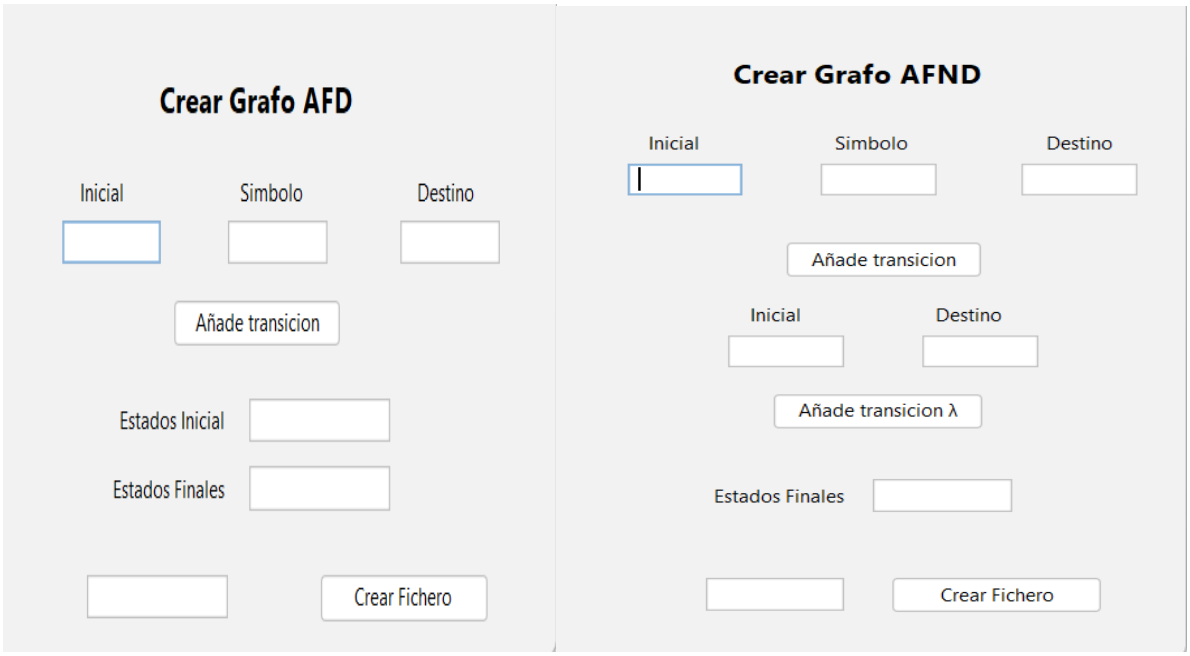
```java
public AFD leerAFD() {
    automataD = new AFD();
    for (int i = 0; i < 4; i++) {
        String linea = scan.nextLine();
        String[] partes = linea.split(regex: " ");
        switch (partes[0]) {
            case "INICIAL:":
                String[] ei = partes[1].split(regex: "q");
                automataD.setEstadoInicial(ei: Integer.parseInt(ei[1]));
                break;

            case "FINALES:":
                for (int j = 1; j < partes.length; j++) {
                    String[] ef = partes[j].split(regex: "q");
                    automataD.añadirEstadoFinal(parseInt: Integer.parseInt(ef[1]));
                }
                break;

            case "TRANSICIONES:":
                linea = scan.nextLine();
                while (!linea.equals(anObject: "FIN")) {

                    partes = linea.split(regex: " ");
                    System.out.println(partes[0] + " " + partes[1] + " " + partes[2]);
                    String[] eo = partes[0].split(regex: "q");
                    String[] simb = partes[1].split(regex: "'");
                    String[] ed = partes[2].split(regex: "q");
                    System.out.println(eo[1] + " " + simb[1] + " " + ed[1]);
                    automataD.agregarTransicion(e1: Integer.parseInt(eo[1]), simbolo: simb[1].charAt(index: 0),
                            e2: Integer.parseInt(ed[1]));

                    linea = scan.nextLine();
                }
                break;
```

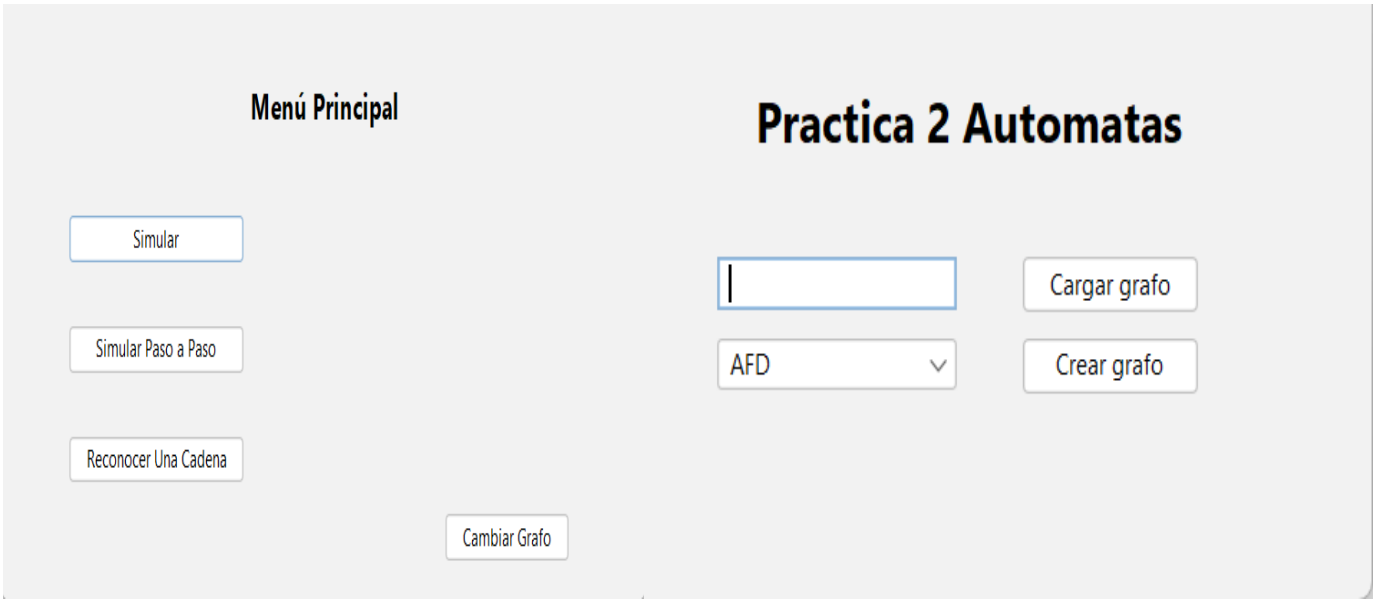
```java
    /**
     * metodo que lee del fichero un grafo de tipo AFND
     *
     * @return Devuelve el grafo AFND del fichero
     */
    private AFND leerAFND() {
        automataND = new AFND();
        for (int i = 0; i < 4; i++) {
            String linea = scan.nextLine();
            String[] partes = linea.split(regex: " ");

            switch (partes[0]) {
                case "INICIAL:":
                    break;

                case "FINALES:":
                    int[] fin = new int[partes.length];
                    for (int j = 1; j < partes.length; j++) {
                        String[] ef = partes[j].split(regex: "q");
                        fin[j - 1] = Integer.parseInt(ef[1]);
                    }
                    automataND.setEstadosFinales(estadosFinales: fin);
                    break;

                case "TRANSICIONES:":
                    linea = scan.nextLine();
                    while (!linea.equals(anObject: "TRANSICIONES LAMBDA:")) {

                        partes = linea.split(regex: " ");

                        String[] eo = partes[0].split(regex: "q");
                        String[] simb = partes[1].split(regex: "'");
                        int[] ed = new int[partes.length - 2];
                        int k = 0;
                        for (int j = 2; j < partes.length; j++) {

                            String[] a = partes[j].split(regex: "q");
                            ed[k] = Integer.parseInt(a[1]);
                            k++;
                        }
                        automataND.agregarTransicion(e1: Integer.parseInt(eo[1]), simbolo: simb[1].charAt(index: 0),
                                e2: ed);

                        linea = scan.nextLine();
                    }
                    linea = scan.nextLine();
                    while (!linea.equals(anObject: "FIN")) {
                        partes = linea.split(regex: " ");

                        String[] eo = partes[0].split(regex: "q");
                        int[] ed = new int[partes.length - 1];
                        int k = 0;
                        for (int j = 1; j < partes.length; j++) {

                            String[] a = partes[j].split(regex: "q");
                            ed[k] = Integer.parseInt(a[1]);
                            k++;
                        }
                        automataND.agregarTransicionλ(e1: Integer.parseInt(eo[1]), e2: ed);

                        linea = scan.nextLine();
                    }
                    break;
            }
        }
```

```java
/**
 * metodo que crea un fichero a partir de un grafo en formato object y un nombre con el que se guarda
 *
 * @param devolver grafo en formato Object que va a ser guardado
 * @param nom cadena de caracteres que contiene el nombre del fichero donde se van a guardar los datos
 * @throws java.io.IOException
 */
public void CreaFich(Object devolver, String nom) throws IOException {
    FileWriter fich;

    fich = new FileWriter("Ficheros\\" + nom + ".txt");
    if (devolver instanceof AFD) {

        AFD afd = (AFD) devolver;
        fich.write(str:"TIPO: AFD");
        fich.write(str:"\nESTADOS:");
        for (int i = 0; i < afd.getEstados().size(); i++) {
            fich.write(" q" + afd.getEstados().get(index: i));
        }
        fich.write("\nINICIAL: q" + afd.getEstadoinicial());
        fich.write(str:"\nFINALES:");
        for (int i = 0; i < afd.getEstadosFinales().size(); i++) {
            fich.write(" q" + afd.getEstadosFinales().get(index: i));
        }
        fich.write(str:"\nTRANSICIONES:");
        for (int i = 0; i < afd.getTransiciones().size(); i++) {
            fich.write("\nq" + afd.getTransiciones().get(index: i).getInicio() + " '" +
                    afd.getTransiciones().get(index: i).getsimbolo() + "' q" +
                    afd.getTransiciones().get(index: i).getDestino());
        }
        fich.write(str:"\nFIN");
    } else {
        AFND afnd = (AFND) devolver;
        fich.write(str:"TIPO: AFND");
        fich.write(str:"\nESTADOS:");
```

```java
    } else {
        AFND afnd = (AFND) devolver;
        fich.write(str:"TIPO: AFND");
        fich.write(str:"\nESTADOS:");
        for (int i = 0; i < afnd.getEstados().size(); i++) {
            fich.write(" q" + afnd.getEstados().get(index: i));
        }
        fich.write("\nINICIAL: q" + afnd.getTransiciones().get(index: 0).getInicio());
        fich.write(str:"\nFINALES:");
        for (int i = 0; i < afnd.getEstadosFinales().length; i++) {
            fich.write(" q" + afnd.getEstadosFinales()[i]);
        }
        fich.write(str:"\nTRANSICIONES:");
        for (int i = 0; i < afnd.getTransiciones().size(); i++) {
            fich.write("\nq" + afnd.getTransiciones().get(index: i).getInicio() + " '" +
                    afnd.getTransiciones().get(index: i).getsimbolo() + "'");
            for (int j = 0; j < afnd.getTransiciones().get(index: i).getDestinos().length; j++) {
                fich.write(" q" + afnd.getTransiciones().get(index: i).getDestinos()[j]);
            }
        }
        fich.write(str:"\nTRANSICIONES LAMBDA:");
        for (int i = 0; i < afnd.getTransicionesλ().size(); i++) {
            fich.write("\nq" + afnd.getTransicionesλ().get(index: i).getInicio());
            for (int j = 0; j < afnd.getTransicionesλ().get(index: i).getDestinos().length; j++) {
                fich.write(" q" + afnd.getTransicionesλ().get(index: i).getDestinos()[j]);
            }
        }
        fich.write(str:"\nFIN");
    }
    fich.close();
}
```

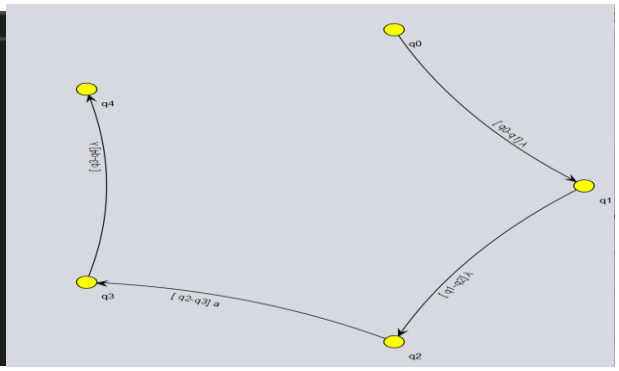
Aplicación:

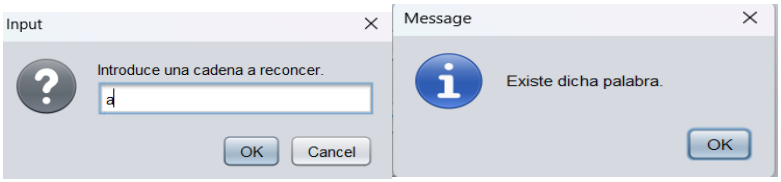



Menú Principal
Practica 2 Automatas
Simular
Simular Paso a Paso
Reconocer Una Cadena
Cambiar Grafo
Cargar grafo
AFD
Crear grafo

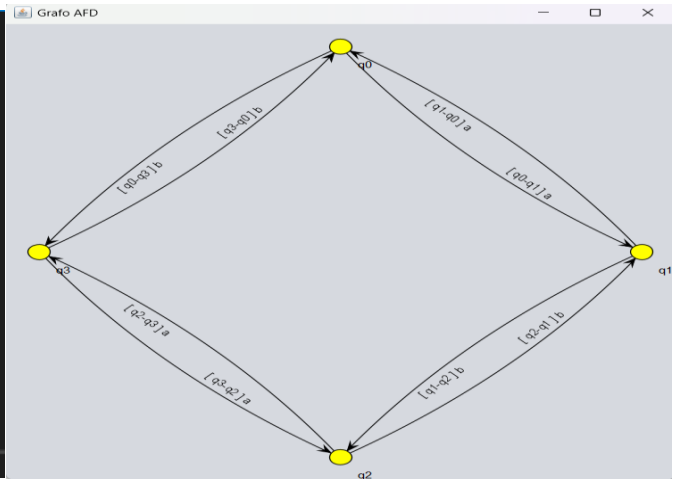
CONJUNTO DE PRUEBAS

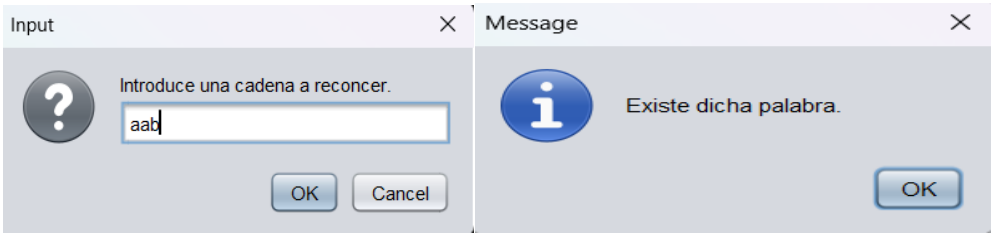
```
1   TIPO: AFND
2   ESTADOS: q0 q1 q2 q3 q4
3   INICIAL: q0
4   FINALES: q4
5   TRANSICIONES:
6   q2 'a' q3
7   TRANSICIONES LAMBDA:
8   q0 q1
9   q1 q2
10  q3 q4
11  FIN
```

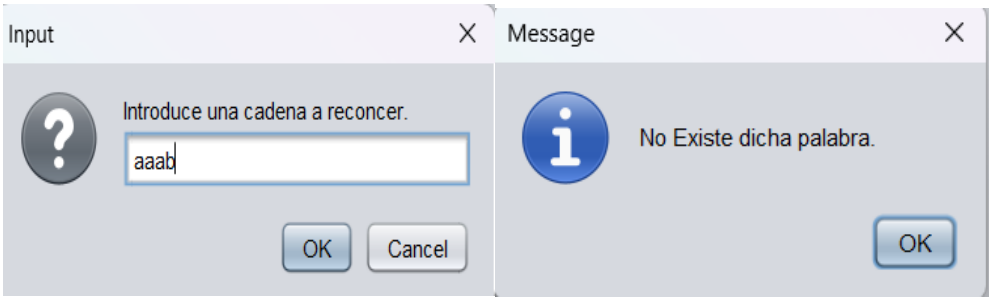




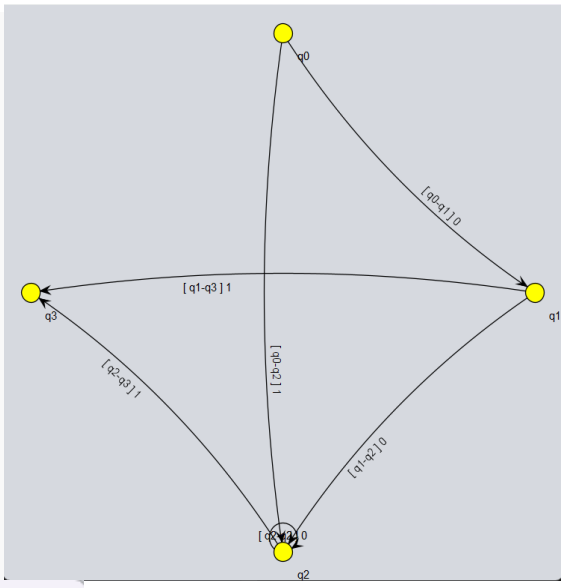

```
1   TIPO: AFD
2   ESTADOS: q0 q1 q2 q3
3   INICIAL: q0
4   FINALES: q3
5   TRANSICIONES:
6   q0 'a' q1
7   q0 'b' q3
8   q1 'a' q0
9   q1 'b' q2
10  q2 'b' q1
11  q2 'a' q3
12  q3 'a' q2
13  q3 'b' q0
14  FIN
```

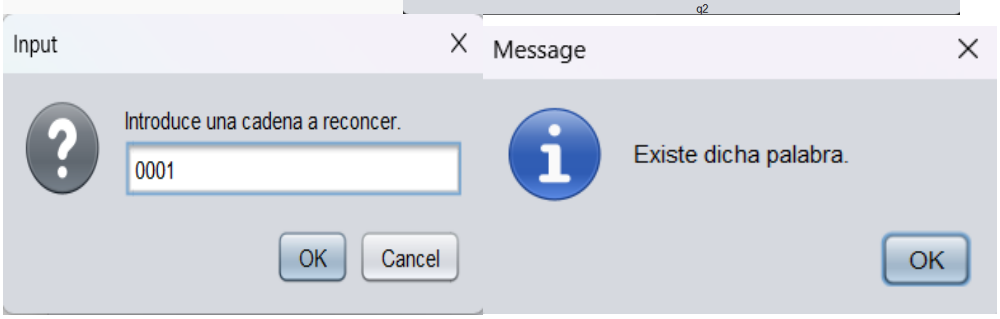


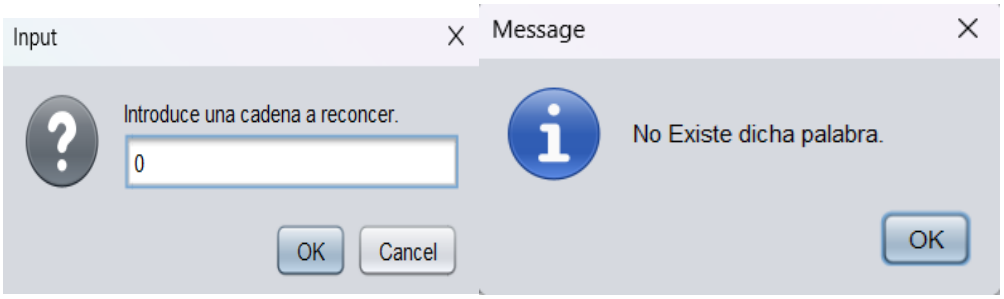





```
TIPO: AFD
ESTADOS: q0 q2 q1 q3
INICIAL: q0
FINALES: q3 q2
TRANSICIONES:
q0 '1' q2
q0 '0' q1
q1 '1' q3
q1 '0' q2
q2 '0' q2
q2 '1' q3
FIN
```







```
TIPO: AFND
ESTADOS: q0 q1 q2 q3
INICIAL: q0
FINALES: q2 q3
TRANSICIONES:
q0 '0' q1
q0 '1' q2 q3
q1 '1' q0 q2 q3
q2 '0' q1
TRANSICIONES LAMBDA:
q0 q1
q3 q0 q2
FIN
```