SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE LEÓN

MATERIA

Lenguajes y Autómatas 1

CARRERA

Ingeniería en sistemas computacionales

PRESENTA:

Programa para minimización de autómatas.

NOMBRE DEL ALUMNO

Emmanuel Jacob Maldonado López

NOMBRE DEL MAESTRA:

Ing. Sáez de Nanclares Rodríguez Ruth

LEÓN, GUANAJUATO **Periodo: Agosto-Diciembre 2017**

Emmanuel Jacob Maldonado López 16240950 1 de octubre del 2018
Grupo 12:15 martes, jueves y viernes

Codigo

- **Clase prueba**

```
package minimizacion;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

/**
 *
 * @author jacob
 */
public class MinimizacionAutomatasDeterministas {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
      //datos a pedir de cada automata
      String []estados;
      String []estadosFinales;
      String []alfabeto;
      Map<String,String> funcionDeTransicion;
      String estadoInicial;
      //cadena auxiliar
      String auxiliar;

      //automata finito determinista
      AutomataFinitoDeterminista afd = null;

      System.out.println("Equivalencia de automatas:");
      //ingreso de los datos del automata
      try{
            System.out.println("Automata Finito Determinista");
            System.out.println("Ingrese el conjunto de estados separados por ',' :");
            auxiliar = reader.readLine();
            estados = auxiliar.split(",");
            System.out.println("Ingrese el conjunto de simbolos del alfabeto separados por ',' ");
```

```
            auxiliar = reader.readLine();
            alfabeto = auxiliar.split(",");
            System.out.println("Ingrese la funcion de transición para cada uno de los siguientes:");
            funcionDeTransicion = pedirFuncionTransicion(estados, alfabeto);
            System.out.println("Ingrese el conjunto de Estados Finales separados por ',' ");
            auxiliar = reader.readLine();
            estadosFinales = auxiliar.split(",");
            System.out.println("Ingrese el estado inicial:  ");
            estadoInicial = reader.readLine();
            afd = new AutomataFinitoDeterminista(estados, alfabeto,
                  funcionDeTransicion, estadoInicial, estadosFinales);

        }catch(IOException e){
           e.printStackTrace();
        }

        //si se puede reducir se imprime el nuevo automata
        if(afd.reducirAutomata()){
           imprimir(afd);
        }else{
           System.out.println("El automata no se pudo reducir");
        }

    }

    /**
     *
     * @param estados -> estados del AFD
     * @param alfab -> simbolos en el alfabeto del AFD
     * @return -> funcion de trancision en hashMap
     * @throws java.io.IOException
     */
    public static Map pedirFuncionTransicion(String []estados, String []alfab) throws IOException{
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        Map <String,String> mapa = new HashMap();
        String transicion, valor;
        for (int i = 0; i < (estados.length * alfab.length); i++) {
           transicion = estados[i / alfab.length] + ","
                 + alfab[i % alfab.length];
           System.out.println("Ingresa el estado de trancision para ("
                 + transicion + ") : ");
           valor = reader.readLine();
```

```
            mapa.put(transicion, valor);
        }
        return mapa;
    }

    public static void imprimir(AutomataFinitoDeterminista afd){
        System.out.println("Estados: ");
        System.out.println(Arrays.toString(afd.getEstados()));
        System.out.println("Alfabeto: ");
        System.out.println(Arrays.toString(afd.getAlfabeto()));
        System.out.println("Funcion Trancision");
        System.out.println(afd.getFuncionTransicion().toString());
        System.out.println("Estado inicial: ");
        System.out.println(afd.getEstadoInicial());
        System.out.println("Estados Finales: ");
        System.out.println(Arrays.toString(afd.getEstadosFinales()));
    }

}
```

- **Clase AutomataFinitoDeterminista**

```
package minimizacion;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Stack;

/**
 *
 * @author jacob
 */
public class AutomataFinitoDeterminista {

    //Equivalente a Q, conjunto de estados
    private String mEstados[];
    //Equivalente a Sigma mayuscula, que es el alfabeto
    private String mAlfabeto[];
    //Equivalente a delta minuscula, que es la funcion de transición
    private Map<String, String> mFuncionTransicion;
    //Equivalente a q0, que es el estado inicial
    private String mEstadoInicial;
    //Equivalente a F, que es el conjunto de estados finales
```

```
  private String mEstadosFinales[];

  public AutomataFinitoDeterminista(String[] estados, String[] alfabeto,
        Map<String, String> funcionTransicion, String estadoInicial, String[] estadosFinales) {
    mEstados = estados;
    mAlfabeto = alfabeto;
    mFuncionTransicion = funcionTransicion;
    mEstadoInicial = estadoInicial;
    mEstadosFinales = estadosFinales;
  }

  public String[] getEstados() {
    return mEstados;
  }

  public void setEstados(String[] estados) {
    mEstados = estados;
  }

  public String[] getAlfabeto() {
    return mAlfabeto;
  }

  public void setAlfabeto(String[] alfabeto) {
    mAlfabeto = alfabeto;
  }

  public Map<String, String> getFuncionTransicion() {
    return mFuncionTransicion;
  }

  public void setFuncionTransicion(Map<String, String> funcionTransicion) {
    mFuncionTransicion = funcionTransicion;
  }

  public String getEstadoInicial() {
    return mEstadoInicial;
  }

  public void setEstadoInicial(String estadoInicial) {
    mEstadoInicial = estadoInicial;
  }
```

```
  public String[] getEstadosFinales() {
    return mEstadosFinales;
  }

  public void setEstadosFinales(String[] estadosFinales) {
    mEstadosFinales = estadosFinales;
  }

  //metodo de equivalencia de Automatas Finitos Deterministas
  //se usa guion bajo para diferenciar los parametros del afd al que se compara
  public boolean equivalenteA(AutomataFinitoDeterminista _afd) {
    //variable para saber si faltan estados o ya todos se analizaron
    boolean faltanEstadosPorAnalizar = true;
    //lista para guardar los estados analizados
    List<String> estadosAnalizados = new ArrayList();
    //cadena para estados devueltos al hacer la transicion por los simbolos del alfabeto
    String estados[];
    //pila para guardar estados pendientes
    Stack<String> pila = new Stack();

    //obtener estados iniciales
    String estadoActual = mEstadoInicial;
    String _estadoActual = _afd.getEstadoInicial();

    //mientras haya estados no analizados
    while (faltanEstadosPorAnalizar) {
      //agregar estados a la lista para evitar repeticion
      estadosAnalizados.add(estadoActual + "," + _estadoActual);
      //obtener la trancision de estados por cada simbolo del alfabeto
      estados = transicion(_afd, estadoActual, _estadoActual);
      //iterar por los estados dados por las transiciones
      for (int i = 0; i < estados.length; i++) {
        //si los estados no son compatibles
        if (esEstadoFinal(estados[i].split(",")[0])
            != _afd.esEstadoFinal(estados[i].split(",")[1])) {
          //devolver falso
          return false;
        } //si no
        else {
          //ingresar a una pila
          pila.push(estados[i]);
```

```
            }
        }
        //hacer mientras el estado se encuentre en la lista de analizados
        String estadoSiguiente = "";
        do {
            if (!pila.isEmpty()) {
                //sacar el siguiente elemento de la pila
                estadoSiguiente = pila.pop();
            } else {
                faltanEstadosPorAnalizar = false;
                break;
            }
        } while (estadosAnalizados.contains(estadoSiguiente));
        //si la cadena no esta vacia
        if (!estadoSiguiente.isEmpty()) {
            //dividir la cadena de la pila por "," y asignar a los estados actuales
            estadoActual = estadoSiguiente.split(",")[0];
            _estadoActual = estadoSiguiente.split(",")[1];
        }
    }

    return true;
}

/**
 * Regresa todos los estados de transicion de los simbolos del alfabeto
 *
 * Se usa guion bajo para diferenciar al automata primo al que se compara
 *
 * @param _afd -> automata finito determinista que se esta comparando
 * @param ea -> estado actual del automata comparado
 * @param _ea -> estado actual del automata al que se compara
 * @return -> conjunto de estados igual al numero de simbolos del alfabeto
 */
private String[] transicion(AutomataFinitoDeterminista _afd, String ea, String _ea) {
    //los estados son iguales al numero de simbolos en el alfabeto
    String[] resultado = new String[mAlfabeto.length];
    String nuevoEstado, _nuevoEstado;
    //iteramos por cada simbolo del alfabeto
    for (int i = 0; i < mAlfabeto.length; i++) {//TODO: crear funcion para obtener la trancision
        //obtenemos el estado al que se hace la transicion de dicho simbolo
        nuevoEstado = mFuncionTransicion.get(ea + "," + mAlfabeto[i]);
```

```
        _nuevoEstado = _afd.getFuncionTransicion().get(_ea + "," + _afd.getAlfabeto()[i]);
        resultado[i] = nuevoEstado + "," + _nuevoEstado;
      }

      return resultado;
    }

    //compara el estado para saber si es final
    public boolean esEstadoFinal(String estado) {
      for (String mEstadosFinale : mEstadosFinales) {
        if (estado.equals(mEstadosFinale)) {
          return true;
        }
      }
      return false;
    }

    /**
     * Metodo para reducir el automata finito determinista a su minima forma
     * equivalente.
     *
     * @return -> regresa true si se pudo reducir o false si no.
     */
    public boolean reducirAutomata() {
      AutomataFinitoDeterminista automataEquivalente;
      List<String> lista = new ArrayList();
      Stack<String> pila = new Stack();
      String[] transiciones;
      String trancisionActual, qActual, _qActual;

      //iterar por los estados del automata, llamados q
      for (String q : mEstados) {
        //iterar por los estados del automata, llamados q'(_q) para comparar
        for (String _q : mEstados) {

          //limpiamos la lista
          lista.clear();
          //si q es equivalente a q' (son o no finales los 2) y q no es q'(_q)
          if ((esEstadoFinal(q) == esEstadoFinal(_q)) && !q.equals(_q)) {
            //se agregan a la lista y pila q y q'(_q)
            lista.add(q + "," + _q);
            pila.push(q + "," + _q);
```

```
//hacer mientras la pila no este vacia
do {
  //sacamos la trancision de la pila y la asignamos
  trancisionActual = pila.pop();
  qActual = trancisionActual.split(",")[0];
  _qActual = trancisionActual.split(",")[1];
  //obtener las transiciónes posibles de cada simbolo
  transiciones = transicion(this, qActual, _qActual);
  //iterar por el par de estados de cada transicion
  for (String transicion : transiciones) {
    //separamos la trancision a cada estado q y q'(_q)
    qActual = transicion.split(",")[0];
    _qActual = transicion.split(",")[1];
    //si los estados actuales no estan en la lista
    if (!lista.contains(transicion)) {
      //si los estados actuales son equivalentes
      if (esEstadoFinal(qActual) == esEstadoFinal(_qActual)) {
        //agregar a la pila y lista las transiciónes
        pila.push(qActual + "," + _qActual);
        lista.add(qActual + "," + _qActual);
        //si no
      } else {
        //vaciar la pila
        pila.clear();
        //limpiar lista
        lista.clear();
        //salir del ciclo
        break;
      }
    }
  }
} while (!pila.isEmpty());
//si la lista no esta vacia
if (!lista.isEmpty()) {
  //obtenemos los estados que comparamos para equivalencia
  qActual = lista.get(0).split(",")[0];
  _qActual = lista.get(0).split(",")[1];
  System.out.println(qActual + " y " + _qActual + " son equivalentes");
  //creamos el automata equivalente
  automataEquivalente = crearAutomataEquivalente(qActual, _qActual);
  //si el nuevo automata se reduce
  automataEquivalente.reducirAutomata();
```

```
                    //asignamos los nuevos valores al automata local
                    setValuesFromAutomata(automataEquivalente);

                  //regresamos verdadero
                  return true;
                }
              }
            }
          }
          return false;
        }

        /**
         * Elimina el estado equivalente del conjunto de estados del Automata Finito
         * Determinista
         *
         * @param q -> estado a comparar
         * @param _q -> estado primo al comparado
         * @return arrelgo de estados sin el estado primo
         */
        private String[] eliminarEstadoEquivalente(String q, String _q) {
          String[] nuevosEstados = new String[mEstados.length - 1];
          int contador = 0;
          //si son iguales, retornamos los estados actuales
          if (q.equals(_q)) {
            return mEstados;
          }
          //iterar por los estados
          for(String estado: mEstados){
            //si el estado actual no es igual a q'
            if(!estado.equals(_q)){
              //agregamos al nuevo arreglo el estado
              nuevosEstados[contador] = estado;
              contador++;
            }
          }
          //regresamos el nuevo arreglo
          return nuevosEstados;
        }

        /**Elimina las trancisiones redundantes, quitando el estado equivalente
         * primo(q') y redirigiendo las trancisiones al estado equivalente (q)
```

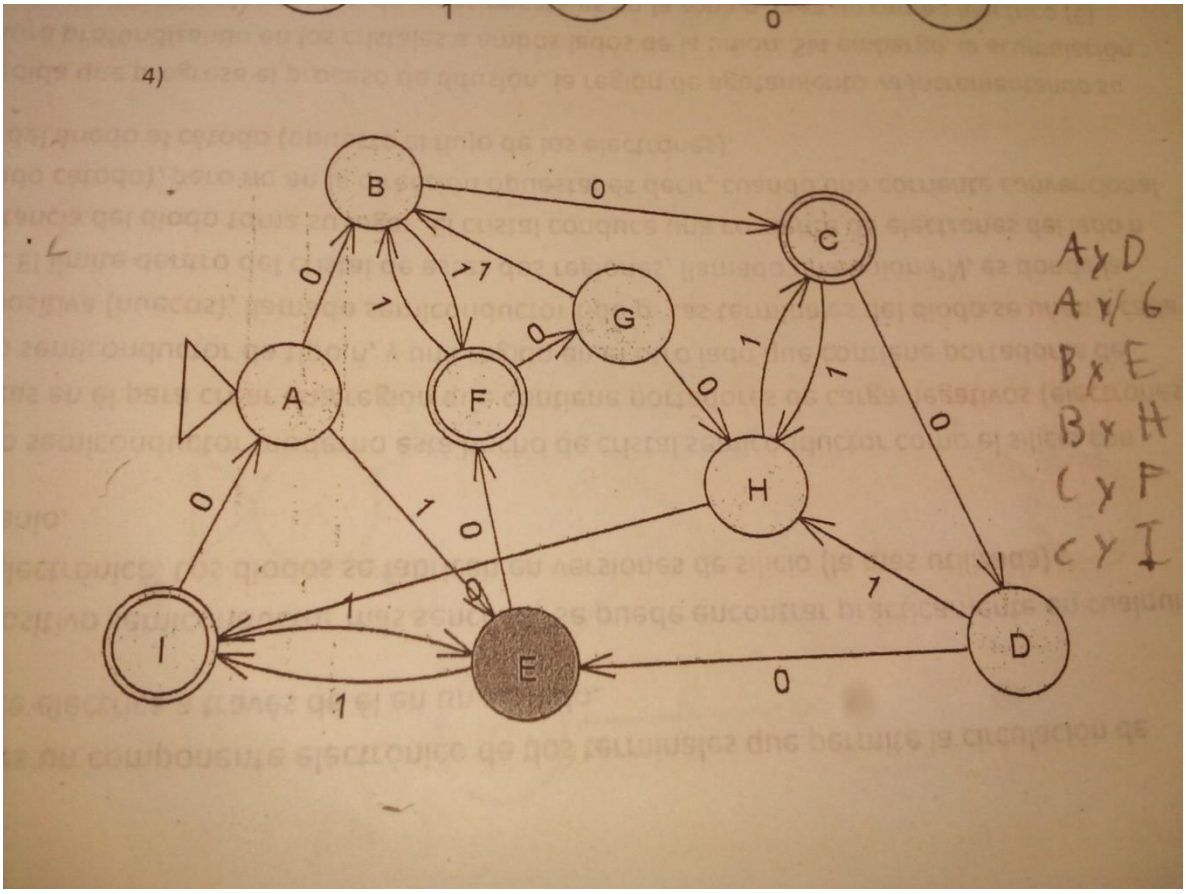
```
     *
     * @param q -> el estado comparado
     * @param _q -> el estado Equivalente al comparado
     * @return -> funcion de transicion sin trancisiones a el estaod primo (q')
     */
    private Map <String,String> eliminarTransicionEquivalente(String q, String _q){
        Map <String,String> funcionTrancision = mFuncionTransicion;
        //buscar transiciones que apuntent a el estado q' y apuntarlas a q
        for(Map.Entry<String, String> entry :funcionTrancision.entrySet()){
            if(entry.getValue().equals(_q)){
                entry.setValue(q);
            }
        }
        //eliminar las trancisiones de q'
        for(String simbolo: mAlfabeto){
            funcionTrancision.remove(_q + "," + simbolo);
        }
        //regresamos la funcion de trancision modificada
        return funcionTrancision;
    }

    /**Elimina el estado final equivalente y devuelve los nuevos estados finales
     *
     * @param q -> el estado comparado
     * @param _q -> el estado equivalente al comparado
     * @return los estados finales sin el equivalente, si los estados no son finales
     * devuelve los mismos
     */
    private String[] eliminarEstadoFinalEquivalente(String q, String _q){
        String[] nuevosEdosFinales = new String[mEstadosFinales.length - 1];
        int contador = 0;
        //si son finales, quitamos el estado eqivalente q' (_q)
        if(esEstadoFinal(q) && esEstadoFinal(_q)){
            for(String edoFinal : mEstadosFinales){
                if(!edoFinal.equals(_q)){
                    nuevosEdosFinales[contador] = edoFinal;
                    contador++;
                }
            }
        //si no devolver los mismos estados finales
        }else{
            return mEstadosFinales;
```

```
      }
      return nuevosEdosFinales;
    }


    /**Crea un nuevo automata sin uno de los estados equivalentes
     *
     * @param q estado equivalente
     * @param _q estado primo equivalente
     * @return Automata Finito determinista equivalente sin el estado q'
     */
    private AutomataFinitoDeterminista crearAutomataEquivalente(String q, String _q) {
      //Creamos un nuevo Automata
      return new AutomataFinitoDeterminista(  //con los mismos estados menos q'
                                eliminarEstadoEquivalente(q, _q),
                                //con el mismo alfabeto
                                mAlfabeto,
                                //las trancisiones que apuntan a q' ahora apuntan a q, y las que salian
de q' se eliminan
                                eliminarTransicionEquivalente(q, _q),
                                //con el mismo estado inicial
                                mEstadoInicial,
                                //con los nuevos estados finales (si q y q' no eran finales, quedaran los
mismso edos. finales)
                                eliminarEstadoFinalEquivalente(q, _q));
    }

    /**Asigna los valores de los atributos del automata parametro a
     * este automata
     *
     * @param afd -> automata del que se quieren asignar los valores
     * al automata actual
     */
    public void setValuesFromAutomata(AutomataFinitoDeterminista afd){
      setEstados(afd.getEstados());
      setAlfabeto(afd.getAlfabeto());
      setFuncionTransicion(afd.getFuncionTransicion());
      setEstadoInicial(afd.getEstadoInicial());
      setEstadosFinales(afd.getEstadosFinales());
    }

}
```

Emmanuel Jacob Maldonado López 16240950 1 de octubre del 2018
Grupo 12:15 martes, jueves y viernes

Corrida

Verificar si el siguiente autómata es reducible:



Sabiendo que un autómata se define por M= (Q, ∑ , δ , q0, F) donde Q es el conjunto de estados, ∑ es el alfabeto, δ es la función de transición, q0 es el estado inicial y F es el conjunto de estados finales.

Tenemos el autómata es:

M = (

Q = {A,B,C,D,E,F,G,H,I},

∑ = {0,1}

δ = { δ (A,0) = B, δ(A,1) = E, δ(B,0) = C, δ(B,1) = F, δ(C,0) = 10, δ(C,1) = 11, δ(D,0) = 00, δ(D,1) = 01, δ(E,0) = 10, δ(E,1) = 11, δ(F,0) = 00, δ(F,1) = 01, δ(G,0) = 10, δ(G,1) = 11, δ(H,0) = 11, δ(H,1) = 11, δ(I,0) = 11, δ(I,1) = 11 },

A,

F = {I,F,C}

)

Emmanuel Jacob Maldonado López 16240950 1 de octubre del 2018
Grupo 12:15 martes, jueves y viernes

Ingresamos los datos del autómata al programa para saber si son equivalentes

```
run:
Equivalencia de automatas:
Automata Finito Determinista
Ingrese el conjunto de estados separados por ',' :
A,B,C,D,E,F,G,H,I
Ingrese el conjunto de simbolos del alfabeto separados por ','
0,1
Ingrese la funcion de transición para cada uno de los siguientes:
Ingresa el estado de trancision para (A,0) :
B
Ingresa el estado de trancision para (A,1) :
E
Ingresa el estado de trancision para (B,0) :
C
Ingresa el estado de trancision para (B,1) :
F
Ingresa el estado de trancision para (C,0) :
D
Ingresa el estado de trancision para (C,1) :
H
Ingresa el estado de trancision para (D,0) :
E
Ingresa el estado de trancision para (D,1) :
H
Ingresa el estado de trancision para (E,0) :
F
Ingresa el estado de trancision para (E,1) :
I
Ingresa el estado de trancision para (F,0) :
G
Ingresa el estado de trancision para (F,1) :
B
Ingresa el estado de trancision para (G,0) :
```

```
H
Ingresa el estado de trancision para (G,1) :
B
Ingresa el estado de trancision para (H,0) :
I
Ingresa el estado de trancision para (H,1) :
C
Ingresa el estado de trancision para (I,0) :
A
Ingresa el estado de trancision para (I,1) :
E
Ingrese el conjunto de Estados Finales separados por ','
C,F,I
Ingrese el estado inicial:
A
A y D son equivalentes:
[A,D, B,E, E,H, F,I, I,C, G,A, H,B, C,F, D,G]
A y G son equivalentes:
[A,G, B,H, E,B, F,C, I,F, G,A, H,B, B,E, C,F, F,I, I,C, A,A, E,H, B,B, E,E, F,F, I,I, G,G, H,H, C,C, C,I, H,E]
B y E son equivalentes:
[B,E, C,F, F,I, A,A, B,B, E,E, F,F, I,I, C,C, H,H, H,B, I,C, E,H]
B y H son equivalentes:
[B,H, C,I, F,C, A,A, B,B, C,C, F,F, H,H, I,I, H,B, I,C, C,F]
C y F son equivalentes:
[C,F, A,A, B,B, C,C, F,F]
C y I son equivalentes:
[C,I, A,A, B,B, C,C]
```

El Automata resultante

```
Estados:
[A, B, C]
Alfabeto:
[0, 1]
Funcion Trancision
{A,0=B, A,1=B, B,0=C, C,0=A, B,1=C, C,1=B}
Estado inicial:
A
Estados Finales:
[C]
BUILD SUCCESSFUL (total time: 2 minutes 55 seconds)
```