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 equivalencia 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 equivalenciaautomatas;

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

/**
 *
 * @author jacob
 */
public class EquivalenciaAutomatas {
   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;

      //automatas finitos deterministas
      AutomataFinitoDeterminista afd[] = new AutomataFinitoDeterminista[2];

      System.out.println("Equivalencia de automatas:");
      //ingreso de los datos del automata
      try{
         for(int i = 0; i < 2; i++){
            System.out.println("Automata " + i);
            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[i] = new AutomataFinitoDeterminista(estados, alfabeto,
                  funcionDeTransicion, estadoInicial, estadosFinales);
         }
      }catch(IOException e){
         e.printStackTrace();
      }

      if(afd[0].equivalenteA(afd[1])){
         System.out.println("\n\nSon equivalentes");
      }else{
         System.out.println("\n\nNo son equivalentes");
      }

   }

   /**
    *
    * @param estados -> estados del AFD
    * @param alfab -> simbolos en el alfabeto del AFD
    * @return -> funcion de trancision en hashMap
    */
   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;
   }

}
```

- Clase AutomataFinitoDeterminista

```
package equivalenciaautomatas;

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
```

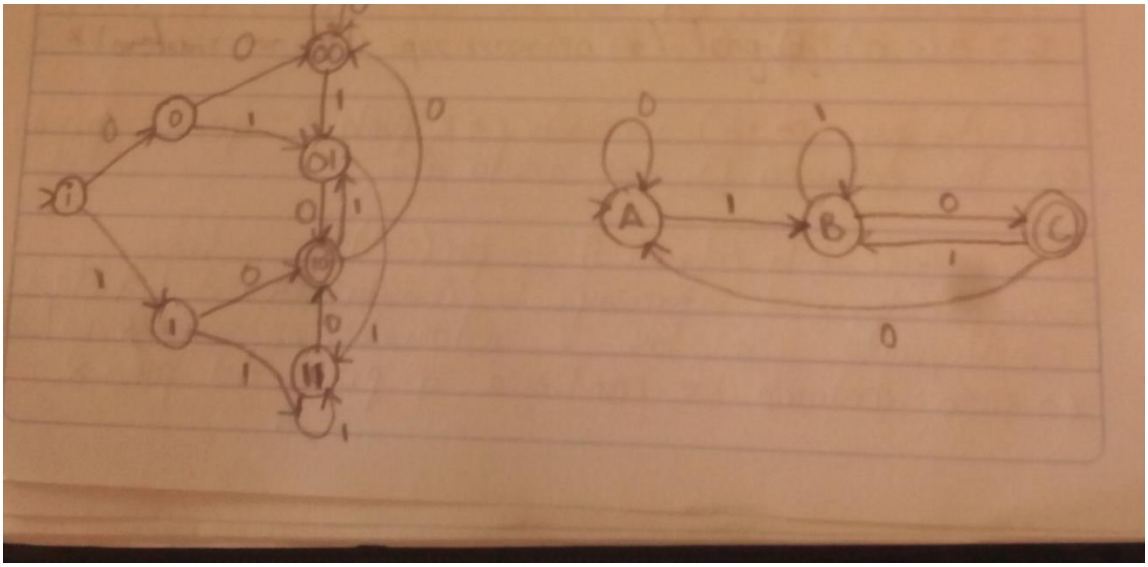
```
Stack<String> pila = new Stack();

//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;
    }

    public boolean esEstadoFinal(String estado){
        for(int i = 0; i < mEstadosFinales.length; i++ ){
            if(estado.equals(mEstadosFinales[i]))
                return true;
        }
        return false;
    }

}
```

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

Corrida

Verificar si los siguientes autómatas son equivalentes



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 primer autómata es:

M = (

Q = {i,0,1,00,01,10,11},

∑ = {0,1}

δ = { δ (i,0) = 0, δ(i,1) = 1, δ(0,0) = 00, δ(0,1) = 01, δ(1,0) = 10, δ(1,1) = 11, δ(00,0) = 00, δ(00,1) = 01, δ(01,0) = 10, δ(01,1) = 11, δ(10,0) = 00, δ(10,1) = 01, δ(11,0) = 10, δ(11,1) = 11 },

i,

F = {10}

)

El segundo autómata es:

M’ = (

Q = {A,B,C},

∑ = {0,1}

δ = { δ (A,0) = A, δ(A,1) = B, δ(B,0) = C, δ(B,1) = B, δ(C,0) = A, δ(C,1) = B},

A,

F = {C}

)

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

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

```
Equivalencia de automatas:
Automata 0
Ingrese el conjunto de estados separados por ',' :
i,0,1,00,01,10,11
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 (i,0) :
0
Ingresa el estado de trancision para (i,1) :
1
Ingresa el estado de trancision para (0,0) :
00
Ingresa el estado de trancision para (0,1) :
01
Ingresa el estado de trancision para (1,0) :
10
Ingresa el estado de trancision para (1,1) :
11
Ingresa el estado de trancision para (00,0) :
00
Ingresa el estado de trancision para (00,1) :
01
Ingresa el estado de trancision para (01,0) :
10
Ingresa el estado de trancision para (01,1) :
11
Ingresa el estado de trancision para (10,0) :
00
Ingresa el estado de trancision para (10,1) :
01
Ingresa el estado de trancision para (11,0) :
11
Ingrese el conjunto de Estados Finales separados por '
10
Ingrese el estado inicial:
i
```

Procedemos a ingresar los datos del segundo autómata:

```
Automata 1
Ingrese el conjunto de estados separados por ',' :
A,B,C
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) :
A
Ingresa el estado de trancision para (A,1) :
B
```

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

```
Ingresa el estado de trancision para (B,1) :
B
Ingresa el estado de trancision para (C,0) :
A
Ingresa el estado de trancision para (C,1) :
B
Ingrese el conjunto de Estados Finales separados por ','
C
Ingrese el estado inicial:
A
```

El resultado es:

```
Son equivalentes
BUILD SUCCESSFUL (total time: 3 minutes 27 seconds)
```