Asignatura:
Teoría de la Computación

Catedrático:
Ing. Cesar Orellana

Tema:
Evaluador de autómatas (DFA, NFA, ER)

Alumno:
David Edgardo Núñez Molina

No. de Carné:
21521086

San Pedro Sula, 31 de agosto del 2020

Indicé

Introducción

En el siguiente informe se detalla el desarrollo del proyecto de la clase Teoría de la Computación. Este proyecto consistió en el desarrollo de una aplicación que sea capaz de evaluar autómata, graficar y poder llevarlos a un estado de evaluación (DFA). Todo esto en base al lenguaje de Python con ayuda de algunas librerías grafica para así poder graficar el autómata anterior mente definido.

Definición del problema

Para tener un mejor contexto a que se quiere lograr con el proyecto primero hay que definir que son los autómatas DFA, NFA y las expresiones regular y como se puede evaluar.

Un autómata es un modelo matemático para una máquina de estado finito, en el que, dada una entrada de símbolos, «salta» mediante una serie de estados de acuerdo con una función de transición (que puede ser expresada como una tabla). Esta función de transición indica a qué estado cambiar dados el estado actual y el símbolo leído y los autómatas solo cuenta con un "estado" de evaluación el cual es DFA, sin embargo, los autómatas sin importar en que "estados" se encuentra se pueden graficar.

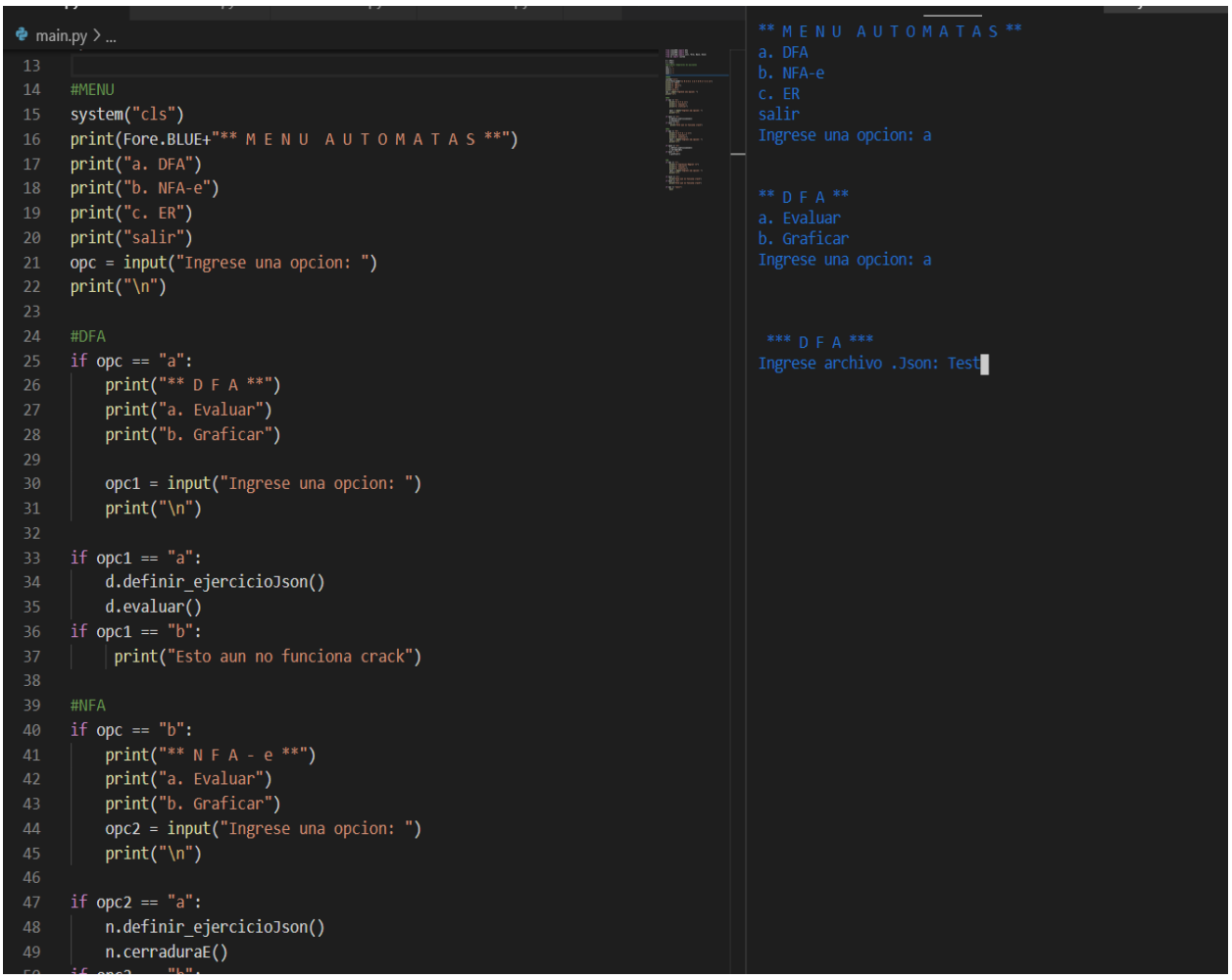
Otra parte del problema brindando por el proyecto es la creación de un algoritmo para convertir una expresión regular en el autómata finito no determinístico correspondiente. El algoritmo construye a partir de la expresión regular un autómata con transiciones vacías, es decir un autómata que contiene arcos rotulados con ε. Luego este autómata con transiciones vacías se puede convertir en un autómata finito sin transiciones vacías que reconoce el mismo lenguaje.

- Dada una expresión regular existe un autómata finito capaz de reconocer el lenguaje que ésta define.
- Recíprocamente, dado un autómata finito, se puede expresar mediante una expresión regular del lenguaje que reconoce.

Definición del Algoritmo

La estructura del proyecto se basó en un proyecto en consola, con un menú donde el usuario pueda interactuar con el proyecto y pueda definir qué tipo de autómata desea evaluar o graficar, en el cual solo necesita colocar el nombre de archivo .Json, donde contenga el autómata inicial o expresión regular a evaluar/graficar.

También se codifico que el ingreso del autómata fuera de manera manual desde la consola, pero por fines de realizar las pruebas del proyecto más rápido se decidió dejar que el algoritmo tomara el ejercicio de un Json donde ya se encontraría el ejercicio previamente definido.



Autómata DFA

Algoritmo definición del autómata:

Anteriormente mencionado que se codifico 2 tipos de algoritmos para definir el autómata, pero se optó por dejar funciona el algoritmo de definición de dinámica ya que ahorra mucho tiempo a la hora de las pruebas.

En el cual se extraer el autómata de un archivo .Json donde contiene previamente definido, igualmente el algoritmo se encarga que el autómata extraído no contenga Épsilon en su alfabeto o transiciones ya que el DFA no acepta Épsilon.

```
96      def definir_ejercicioJson(self):
97          dir = 'C:\\Users\\David\\Desktop\\ProyectoTeoria'
98
99          print("\n *** D F A ***")
100         file_name = input("Ingrese archivo .Json: ")
101         file_name=(file_name+ '.json')
102         with open(os.path.join(dir, file_name), "r") as f:
103             contenido = f.read()
104             jsondecoded = json.loads(contenido)
105
106         for x in jsondecoded["alphabet"]:
107             if(x=="e"):
108                 print("\n EPSILON no esta permitodo en DFA \n")
109                 break
110             self.alphabet = np.append(self.alphabet , x)
111
112         for x in jsondecoded["states"]:
113             self.states = np.append(self.states , x)
114
115         for x in jsondecoded["initial_state"]:
116             self.initial_state += x
117
118         for x in jsondecoded["accepting_states"]:
119             self.accepting_states = np.append(self.accepting_states , x)
120
121         for x in jsondecoded["transitions"]:
122             if(x[1]=="e"):
123                 print("\n EPSILON no esta permitodo en DFA \n")
124                 self.valid=False
125                 break
126             self.transitions = np.append(self.transitions , [x], axis = 0)
127         self.transitions = np.delete(self.transitions , 0 , axis = 0) #Eliminar los datos de la instancia [0,0,0]
128
```

Algoritmo de Evaluación:

Este algoritmo se basó a un algoritmo realizado en clase con ayuda del Ing. Cesar Orellana donde nos mostró como evaluar un autómata DFA.
Algoritmo el cual anote en Google colab:
https://colab.research.google.com/drive/1GYIN2BmabUzoeS2kwkiaeFS61dl0cTy

```
129     def evaluar(self):
130         #self.definir_ejercicio() De manera manual
131         #self.definir_ejercicioJson() Manera dinamica
132 
133         #Definicion del Automata
134         print("Evaluando DFA \n")
135 
136         #Ingreso de la expresion a evaluar con el automata definido
137         self.str_test = input("Ingrese test: ")
138 
139         system("cls")
140         print("*** E V A L U A N D O *** \n")
141         print("TEST: ", self.str_test)
142         print("Alfabeto: ",self.alphabet)
143         print("Estados: " , self.states)
144         print("Estado inicial: ", self.initial_state)
145         print("Estado final: ", self.accepting_states)
146         print("Transiciones: \n",self.transitions )
147 
```

En el siguiente segmento de algoritmo se toma el estado inicial para empezar a evaluar mi autómata. Se crea un ciclo for con el tamaño del str_test ingresado y se validan que las transiciones existan en el autómata y así por cada un elemento del str_test.

```
148
149         current_state = self.initial_state
150         trans_exists = True
151
152         for index in range(len(self.str_test)):
153             #Obtengo un caracter del str_test
154             current_char = self.str_test[index]
155             for x in self.transitions:
156                 if(x[0] == current_state and x[1] == current_char):
157                     trans_exists = True
158                     break
159                 else:
160                     trans_exists = False
161             next_state=""
162             for x in self.transitions:
163                 if((x[0]==current_state) and (x[1]==current_char)):
164                     next_state=x[2]
165             current_state = next_state
166
167         #If para evaluar si la transicion menciona que contiene el estado actual, esta dentro de la lista de
168         #mis estados de aceptacion, de no ser lo se manda en consola el que el test que se ingreso no pertenece al L(M)
169         if trans_exists and current_state in self.accepting_states:
170             print(Fore.GREEN+"Pertenece a L(M) \n\n")
171             self.graficar()
172         else:
173             print(Fore.RED+"No pertenece a L(M) \n\n")
```

Algoritmo de Graficar:

Para el algoritmo de graficar, me apoye en la librería de NetworkX la cual fue recomendada por compañeros en la clase, la cual solo le mando con ciclos mis estados y transiciones y la librería ya se encarga de graficar el autómata.

```
176  import networkx as nx
177  import matplotlib.pyplot as plt
178
179      def graficar(self):
180          Graph = nx.MultiDiGraph()
181          Graph.add_node(self.initial_state)
182
183          for x in self.states:
184              if(x != self.initial_state):
185                  Graph.add_node(x)
186
187          for x in self.transitions:
188              Graph.add_edge(x[0], x[2], element=x[1])
189
190          nx.draw(Graph , with_labels=True)
191          plt.tight_layout()
192          plt.savefig("GrafoDFA.png", format="PNG")
193          plt.show()
```

Autómata NFA

La clase NFA con tiene 2 funciones de graficar y una de convertir a diferencia de la clase DFA ya que para evaluar un “NFA” hay que convertirlo a un estado que se pueda evaluar ósea a un DFA, en el cual hay que sacar una tabla con las Cerradura dura épsilon de las transiciones y de ahí partimos a formar mi DFA.

Algoritmo de Convertir NFA-e a DFA:

En esta parte de algoritmo básicamente se forma la tabla principal que se necesitar para formar mi DFA en la línea 112 a 116 de código le digo que saque la cerradura épsilon de cada estado de mi NFA y lo vaya almacenando temporalmente un arreglo.

```
104     def NFAtoDFA(self):
105         self.alphabet = np.delete(self.alphabet, 0)
106         cE = []
107         cD = []
108         cEF = []
109         #SACAR CERRADURRAS
110         for al in self.alphabet:
111             if(al!="e"):
112                 for es in self.states:
113                     cE = np.append(cE , [es])
114                     for tra in self.transitions:
115                         if(es == tra[0] and tra[1] == "e"):
116                             cE = np.append(cE , tra[2])
117
```

En este ciclo for lo que le digo que cada estado que se metió en mi arreglo tmp de cerraduras épsilon se busque a que estado llega con el carácter de alfabeto en ese momento del ciclo y que luego almacene las transiciones de las cerraduras épsilon en mi arreglo tmp de cD.

```
117
118                     for est in cE:
119                         for trs in self.transitions:
120                             if(est == trs[0] and trs[1]==al):
121                                 cD = np.append(cD , [trs[2]] , axis=0)
122
```

En esta última parte de la “tabla” busco la cerradura épsilon de mis transiciones y las voy almacenando en mi array tmp cEF y después pregunto si en mi Lista de Estados esta la nueva cEF si no está la guardo en mi lista, y agrego la transición en mi Tabla de transiciones y con esto terminaría la tabla general.

```
123                         for estad in cD:
124                             cEF = np.append(cEF , [estad])
125                             for tran in self.transitions:
126                                 if(estad == tran[0] and tran[1] == "e"):
127                                     cEF = np.append(cEF , [tran[2]])
128                         if(cEF!=[]):
129                             if not self.ListaE.buscar(cEF):
130                                 self.ListaE.agregar(cEF)
131                             self.TablaTransiciones.agregar([[es],[al],list(cEF)])
132                         cEF=[]
133                     cE=[]
134                     cD=[]
```

En esta parte básicamente solo agrego mis nuevos estados de aceptación a mi lista que los contendrá, hago lo mismo para las transiciones agrego las transiciones que tengo en mi tabla transiciones y la meto en mi lista.

```
138
139         self.ListaE.agregar([self.initial_state])
140         ListaTMP = ListaNoOrdenada()
141         #Agregar nuevos estados de aceptacion
142         for x in self.accepting_states:
143             ListaTMP = self.ListaE.buscarEF(x)
144             iterador = ListaTMP.tamano()
145             ListaTMP.imprimir()
146             while(iterador > 0):
147                 tmp = ListaTMP.Sacar()
148                 self.ListaEstadoAcep.agregar(tmp)
149                 iterador = iterador - 1
150
151         #Agregar trans que encuentro en mi tabla general
152         tmp = self.TablaTransiciones.tamano()
153         while(tmp > 0):
154             aux = self.TablaTransiciones.Sacar()
155             aux2 = np.array(aux)
156             if not(aux[2] == []) and not(self.ListaTransicion.buscar(aux2)):
157                 aux2 = np.array(aux[0])
158                 if self.ListaE.buscar(aux2):
159                     self.ListaTransicion.agregar(aux)
160             tmp = tmp - 1
```

En esta última parte formo mi tabla final, en la cual encuentro mis ultimas transiciones que necesito esto está dividido en 2 partes ya que mi estado actual puede tener un tamaño 1 o mayor 1 y si recibo uno que es mayor a 1 tengo que convertir el estado recibido, por ejemplo:

Si recibo [q0][q1][q2] tengo que pasarlo a algo así

[[q0][q1][q2]] para poder agregarlo a mi lista de transiciones.

```
161
162         #Agregar trans que voy encontrando en mi nueva tabla dfa
163         tmp = self.ListaTransicion.tamano()
164         ListaAux = self.ListaTransicion.devoLista()
165         while(tmp > 0):
166             aux = ListaAux.Sacar()
167             aux1 = np.array(aux)
168             aux1 = np.array(aux1[2])
169             if not(self.ListaTransicion.buscarUnoElemento(0,aux1)):
170         #Este es cuando el estado actual tenga un tam mayor que 1
171                 if (len(aux1)>1):
172                     arrayTMP = []
173                     for e in self.alphabet:
174                         if(e!="e"):
175                             for x in aux1:
176                                 for y in self.transitions:
177                                     if y[0]==x and y[1] == e:
178                                         arrayTMP = np.append(arrayTMP, y[2])
179                             auxTMP = [list(aux1),e,list(arrayTMP)]
180                             self.ListaTransicion.agregar(auxTMP)
181                             arrayTMP = []
182         #Este es cuando estadoactual sea solo de tamano 1
183                 else:
184                     for e in self.alphabet:
185                         if(e!="e"):
186                             for x in self.transitions:
187                                 if x[0]==aux1 and x[1] == e:
188                                     auxTMP = [aux1,e,x[2]]
189                                     self.ListaTransicion.agregar(auxTMP)
190             tmp = tmp - 1
```

Algoritmo de Evaluar:

Este algoritmo es muy similar al evaluar de DFA normal, a diferencia de el otro es que mis transiciones están dentro una lista, saco el actual de la transición y el peso de la transición y cumple con el if le digo que el siguiente estado va a ser mi nuevo actual. En otras palabras, mi destino se convierte en mi siguiente actual.

```
    def evaluar(self):
        print("Evaluando NFA-e to DFA ")
        self.str_test = input("Ingrese test: ")
        for index in range(len(self.str_test)):
            if self.str_test[index] not in self.alphabet:
                print(Fore.RED+"No existe en el alfabeto: ", self.str_test[index])
                os._exit(0)

        current_state = np.array(self.initial_state)
        self.transition_exists = True

        for char_index in range(len(self.str_test)):
            current_char = self.str_test[char_index]
            #Evular si existen mis transiciones
            AuxTransicion = self.ListaTransicion.devoLista()
            tam = self.ListaTransicion.tamano()
            while(tam > 0):
                tmp = list(AuxTransicion.Sacar())
                tmp3 = np.array(current_char)
                tmp1 = np.array(tmp[0])
                if((tmp[0] == current_state).all() and tmp[1] == tmp3):
                    self.transition_exists = True
                    break
                else:
                    self.transition_exists = False
                tam = tam - 1
            next_state=[]
```

```
            #Evaluo en las transciones con el caracter del test
            AuxTransicion = self.ListaTransicion.devoLista()
            tam = self.ListaTransicion.tamano()
            while(tam > 0):
                tmp = list(AuxTransicion.Sacar())
                tmp1 = np.array(tmp[0])
                tmp2 = np.array(tmp[2])
                tmp3 = np.array(current_char)
                if((tmp1 == current_state).all() and (tmp[1]==tmp3)):
                    next_state=np.array(tmp2)
                    break
                tam = tam - 1
            current_state = next_state
        current_state = np.array(current_state)

        if self.ListaEstadoAcep.buscar(current_state) and self.transition_exists:
            print(Fore.GREEN+"Pertenece a L(M)")
        else:
            print(Fore.RED+"No pertenece a L(M)")
            os._exit(0)
```

Algoritmo de Graficar:

Cuento con 2 funciones de graficar en mi classNFA, ya que en el menú brindo la opción de solamente graficar el NFA ingresado o evaluar, convertir y graficar el DFA.

En graficar se hace 2 while uno para los estados y otro para las transiciones, en el primer while solo saco el estado mi lista y pregunto si no el mismo a mi estado inicial y luego solo lo asigno a una variable estados y lo mando de parámetro a add_node() que es una función de la librería networkx, y para las transiciones algo similar guardo el valor de la transición x en la posición de mi estado actual igualmente para mi peso y destino, se lo mando a add_edge().

```
    def graficar(self):
        AuxEstados  = self.ListaE.devoLista()
        tamEstados = self.ListaE.tamano()

        AuxTransicion = self.ListaTransicion.devoLista()
        tamTransiciones = self.ListaTransicion.tamano()

        Graph = nx.MultiDiGraph()
        Graph.add_node(self.initial_state)

        while(tamEstados > 0):
            AuxEst = AuxEstados.Sacar()
            AuxEst1 = np.array(AuxEst)
            if not (AuxEst1 == [self.initial_state]).all():
                Estados = ''.join(AuxEst)
                Graph.add_node(Estados)
            tamEstados = tamEstados - 1

        while(tamTransiciones > 0):
            AuxTrans = AuxTransicion.Sacar()
            actual = ''.join(AuxTrans[0])
            destino =  ''.join(AuxTrans[2])
            peso = ''.join(AuxTrans[1])
            Graph.add_edge(actual, destino, element=peso)
            tamTransiciones = tamTransiciones - 1

        nx.draw_circular(Graph , with_labels=True, node_color='red', node_size=2000)
        #plt.tight_layout()
        plt.savefig("GrafoDFAConvert.png", format="PNG")
        plt.show()

    def graficarNFA(self):    ···
```

Expresiones Regulares

Algoritmo ER to NFA:

Dentro de esta def hago la definición de la expresión mediante un input, no miraba innecesario en este caso tomarlo de un Json.

Cada if representan la evaluación de cómo actuar si encuentra los diferentes operadores (*, |, +) en la expresión regular, por ejemplo, si encuentra un * que se cree una transición del mismo estado a él con el carácter que esta antes del *.

```
def ERtoNFA(self):
    tmpFinales=[]
    print("\n\t*** E X P R E S I O N  R E G U L A R ***")
    flatI = 0
    self.exp_regular = input("Ingrese Expresion Regular: ")
    for index in range(len(self.exp_regular)):
        if self.exp_regular[index] == '|':
           estado1 = (self.est+str(self.cont))
           tmpFinales = np.append(tmpFinales, estado1)
           self.cont = self.cont + 1
           flatI = 0
           self.pi = True
        elif self.exp_regular[index]=='+':
            xtrans = [estado1, 'e', estado,flatI]
            self.transitions = np.append(self.transitions, [xtrans], axis=0)
        elif self.exp_regular[index]=='*':
            estado1 = (self.est+str(self.cont))
            self.cont = self.cont - 1
            aux = len(self.transitions)
            if(self.transitions[aux-1][3]=='0'):
                flatI = 0
            self.transitions = np.delete(self.transitions, (aux-1), axis=0)
            xtrans = [estado, ultimo, estado, flatI]
            self.transitions = np.append(self.transitions, [xtrans], axis=0)
            self.states = np.delete(self.states, (len(self.states)-1), axis=0)
            flatI = 1
        else:
            if self.exp_regular[index] not in self.alphabet :
               self.alphabet = np.append(self.alphabet, self.exp_regular[index])
            estado = (self.est+str(self.cont))
            self.cont = self.cont + 1
            estado1 = (self.est+str(self.cont))
            xtrans = [estado,self.exp_regular[index],estado1,flatI]
            ultimo= self.exp_regular[index]
            self.transitions = np.append(self.transitions,[xtrans],axis=0)
            flatI = 1
```

Y en el transcurso de el ciclo for del tamaño de mi string de la expresión regular, voy agregando mis estados a mi arreglo states.

Con el siguiente for agrego a mis transacciones iniciales, ósea que agrego las transiciones del nuevo estado a inicial a mis anteriores estados iniciales y los if es el caso contrario a hago las transiciones de mi anterior estado finales a mi nuevo estado final. Y al final agrego mi estado inicial q0, mi estado final y también lo agrego en mis estados de aceptación.

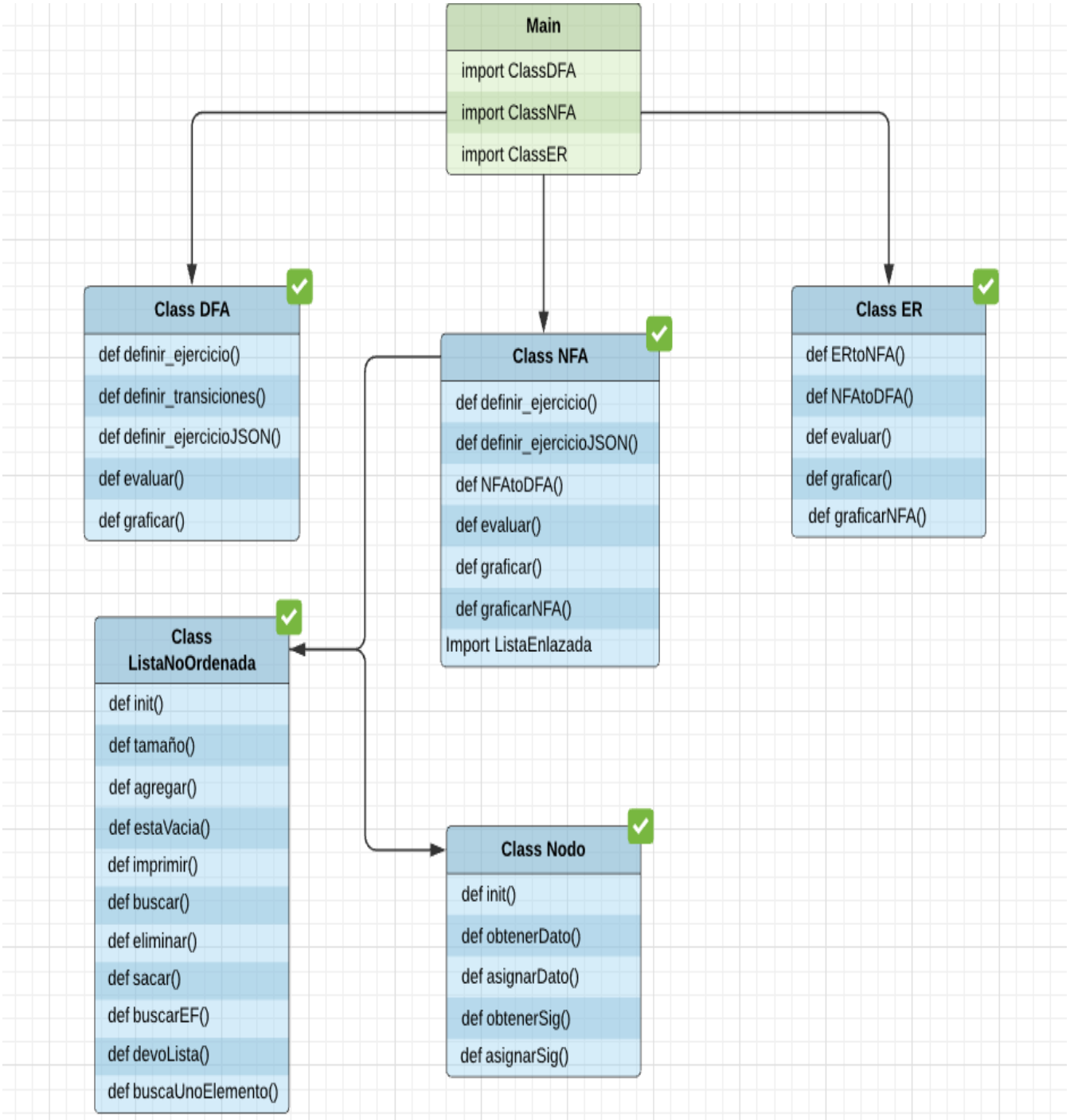
```
            if estado not in self.states:
                self.states = np.append(self.states, [estado], axis=0)
            if estado1 not in self.states:
                self.states = np.append(self.states, [estado1],axis=0)

    estadoPI = np.append(estadoPI, estado1)
    estado = "q0"
    estado1 = (self.est+str(self.cont))
    self.transitions = np.delete(self.transitions , 0 , axis = 0)
    #Transiciones iniciales
    for transicion in self.transitions:
        aux = list(transicion)
        if aux[3]=='0':
            transicion = [estado, 'e', aux[0], 1]
            self.transitions = np.insert(self.transitions,0,[transicion],axis=0)

    if self.pi==True:
        self.cont = self.cont + 1
        estado1 = (self.est+str(self.cont))
        for estad in estadoPI:
            xtrans = [estad, 'e', estado1, 1]
            self.transitions = np.append(self.transitions, [xtrans], axis=0)
    else:
        self.cont = self.cont + 1
        estado1 = (self.est+str(self.cont))
        tmp = [estadoPI[0],'e',estado1, 1]
        self.transitions = np.append(self.transitions, [tmp], axis=0)

    self.alphabet = np.insert(self.alphabet, 0, ['e'])
    self.states = np.insert(self.states, 0 , ['q0'])
    self.transitions = np.delete(self.transitions, [3], axis=1)
    self.accepting_states = np.append(self.accepting_states, [estado1])
```

Diagrama de Clases



Tiempo de ejecución

Algoritmo de Tiempo de ejecución:

Con este algoritmo calculo el tiempo de ejecución de las funciones dentro del cronometro.

```
from time import time

    start_time = time()

    d.definir_ejercicioJson()
    d.evaluar()

    elapsed_time = time() - start_time
    print("Tiempo de Ejecucion: %.10f segundos." % elapsed_time)
```

Autómata DFA:

Promedio

Tiempo de ejecución: 7.3900940418 segundos.

Autómata NFA-e a DFA:

Promedio

Tiempo de ejecución: 11.6001222134 segundos.

Expresión Regular a DFA:

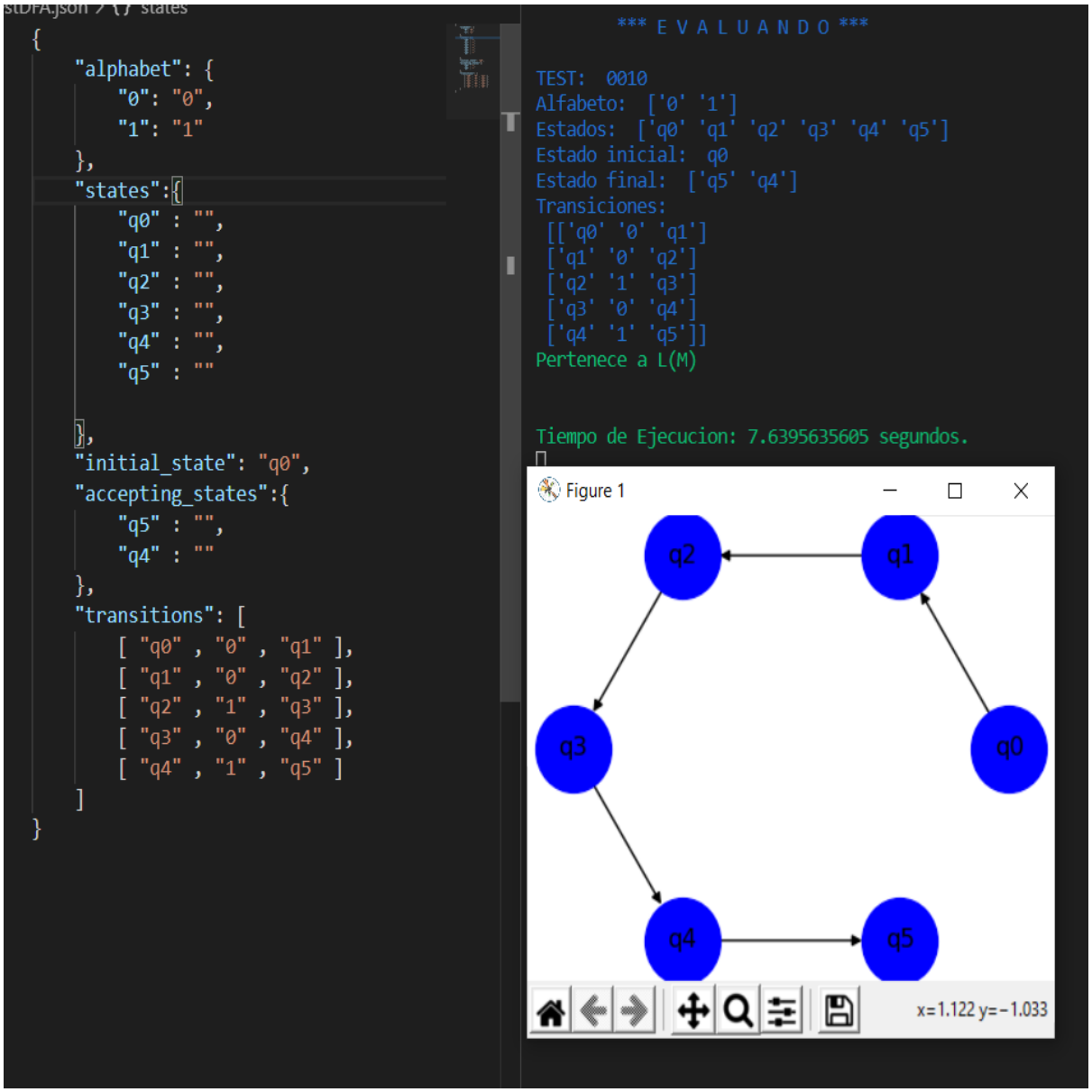
Promedio

Tiempo de ejecución: 12.8630924225 segundos.

Pruebas

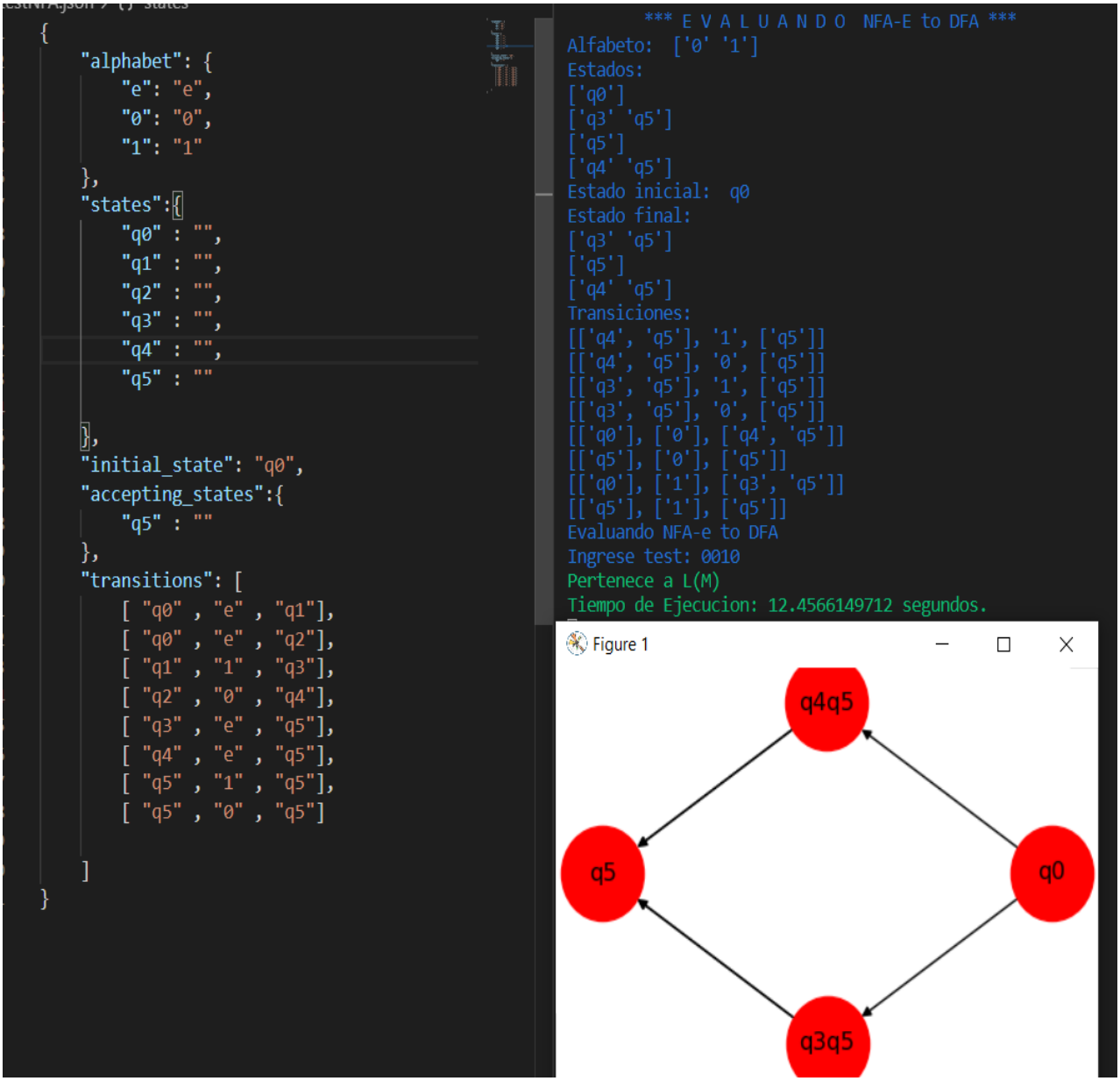
Prueba DFA:

Autómata por probar - Resultado



Prueba NFA-e:

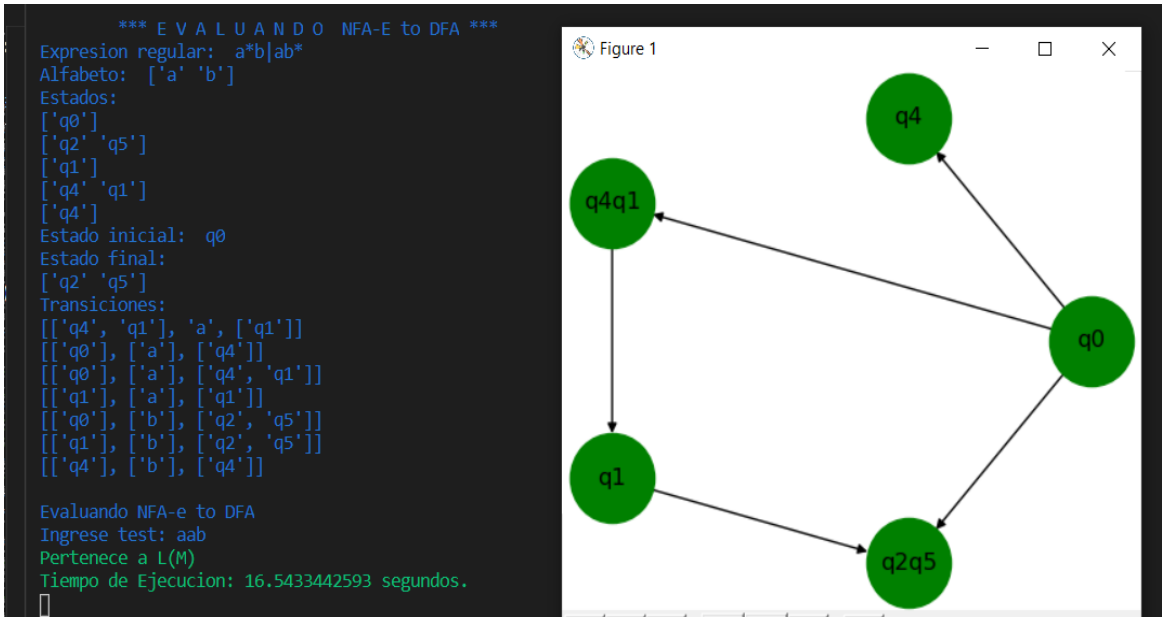
Autómata por probar - Resultado



Prueba Expresion Regular:

Expresion Regular: a*b | ab*

Resultado



Expresion Regular: a*b | b+a

Resultado