**UNIVERSIDAD TECNOLÓGICA CENTROAMERICANA**

**FACULTAD DE INGENIERÍA Y ARQUITECTURA**

**PROYECTO DE CLASE TEORÍA DE LA COMPUTACIÓN**

# INFORME DE PROYECTO AUTÓMATAS


**PRESENTADO POR:**

**21541162 HEDMON ROBERTO LOPEZ SABILLON**

**ASESOR: ING. CESAR ORELLANA**


**CAMPUS SAN PEDRO SULA**

**AGOSTO, 2020**

# Indice

# Introducción

El siguiente informe se presenta el proyecto de autómatas desarrollado en el lenguaje de Python, en cual se implementa el concepto de un autómata aplicados a ciertos mecanismos. En donde se desarrolló código para poder imitar el comportamiento de estos mismos. Se implementan objetos y clases de la expresión regular o mejor conocida como REGEX autómata NFA épsilon, autómata NFA, autómata DFA y de la misma manera un evaluar de DFA con una expresión REGEX con el fin de averiguar si esta pertenece al autómata. Se muestran las distintas fases del proyecto, así como la descomposición del código para demostrar de cómo y para que se utilizan ciertas funciones, una sección de pruebas con imágenes representando los distintos autómatas con sus respuestas y tiempos de ejecución al momento de realizar cada proceso.

# Glosario

**Framework**: (Traducido como Marco de Trabajo) es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar

**GUI**: *Graphical User Interface* (cuya traducción es interfaz gráfica de usuario) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

**JSON**: *JavaScript Objeto Notación* (cuya traducción es Notación de Objetos JavaScript) Es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

**Objeto**: Es una unidad dentro de un programa de computadora que consta de un estado y de un comportamiento, que a su vez constan respectivamente de datos almacenados y de tareas realizables durante el tiempo de ejecución.

**Automata**: Autómata del griego automatos (αὐτόματος) que significa espontáneo o con movimiento propio, puede referirse a: Autómata: máquina que imita la figura y los movimientos de un ser animado.

**Atómata Finito:** Un autómata finito (AF) o máquina de estado finito es un modelo computacional que realiza cómputos en forma automática sobre una entrada para producir una salida. Este modelo está conformado por un alfabeto, un conjunto de estados finito, una función de transición, un estado inicial y un conjunto de estados finales.

**Autómata finito determinista:** Un autómata finito determinista es un autómata finito que además es un sistema determinista; es decir, para cada estado en que se encuentre el autómata, y con cualquier símbolo del alfabeto leído, existe siempre no más de una transición posible desde ese estado y con ese símbolo.

**Alfabetos:** (representa con Ɛ) son todos aquellos símbolos que conforman el autómata.

**Estados:** (representa con Q) son los vértices, etiquetados con su nombre en el interior.

**Estado Inicial:** (representa con ->) como lo indica su nombre es el primer estado del autómata.

**Estados aceptados:** estos se representan con un círculo y un doble circulo si es un estado final.

**Transiciones:** está compuesta por los distintos estados con flechas entre estados y en medio.

**Testing**: Las pruebas de software (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.

**Debugging**: (Traducido como Depuración de programas) es el proceso de identificar y corregir errores de programación.

**Graphviz**: Graphviz es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT. Fue desarrollado por AT&T Labs y liberado como software libre con licencie tipo Eclipse.

**Python**: Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

**Expersion Regular:** En cómputo teórico y teoría de lenguajes formales una **expresión regular**, o **expresión racional**, también son conocidas como **regex** o **regexp**,[3] por su contracción de las palabras inglesas ***reg****ular* ***ex****pression*, es una secuencia de caracteres que conforma un patrón de búsqueda. Se utilizan principalmentes para la búsqueda de patrones de cadenas de caracteres u operaciones de sustituciones.

# Composición de tecnologías utilizadas

El proyecto fue desarrollado en el lenguaje de programación Python. Python es un lenguaje de programación interpretado que es reconocido por la legibilidad de su código, corta curva de aprendizaje de sintaxis y por su gran poder y eficiencia en distintos contextos y procesos. Aparte se usa el lenguaje DOT para poder crear gráficos y crearlos con graphiz y poderlos mostrar.

- **DOT:** es un lenguaje descriptivo en texto plano. Proporciona una forma simple de describir grafos entendibles por humanos y computadoras. Los ficheros de DOT suelen usar la extensión. gv o *.dot* (aunque esta última está en desuso debido a que puede confundirse con la usada por Microsoft Word en sus plantillas.

- **JSON:** (acrónimo de **JavaScript Object Notation**, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje.

- **Graphviz:** (*Graph Visualization*) es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.[1] Fue desarrollado por *AT&T Labs*[2] y liberado como software libre con licencie tipo Eclipse.[3]

# Clases para la implementación de Autómatas

Para poder brindar un esquema o un fácil uso y manejo de los Autómatas, se utilizan clases y funciones utilizadas para poder tener métodos de evaluación y de equivalencia de los distintos tipos de Autómatas. A continuación, se enlistará el listado de clases utilizadas:

Lista de Clases:

- Class_Draw_Automata
- Class_NFA_DFA
- Class_NFA_Ɛ_NFA
- Class_REGEX
- CLass_TEST_DFA

Lista de Funciones:

- Func_NFA_Ɛ_TO_NFA
- Func_NFA_To_DFA
- Func_RE_To_NFA_Ɛ
- Render_Automata
- Main

# Composición de cada clase

- Class_Draw_Automata

Esta clase me crea un autómata dibujando en modo libre para el usuario que recibe como import una función **Render_Automata** que le ayuda a dibujarla en modo libre.

```
class DFA:

    # DFA automata properties M =[Q,E,F,S,&]
    # Q = states
    # E = simbols
    # F = Final State
    # S = Inicial State
    # & = Table

    def __init__(self, Q, E, F, S):
        self.Q = Q.split(",")

        self.E = E.split(",")
        self.F = F.split(",")
        self.S = S

    def stat_list(self):
        print(self.Q)
```

- Class_NFA_DFA

Esta clase recibe de parámetros un archivo json y manda llamar a la función **Func_NFA_DFA que** hace la conversión entre un autómata NFA a DFA.

```python
class NFATODFA:

    def __init__(self, path):
        self.path = path

    def NFA_To_DFA(self):
        nfa_to_dfa(self.path)
```

- Class_NFA_Ɛ_NFA

Esta clase recibe de parámetros un archivo json y manda llamar a la función **Func_ NFA_Ɛ_NFA** que hace la conversión entre un autómata NFA_e a NFA.

```python
class NFAETONFA:

    def __init__(self, path):
        self.path = path

    def NFAE_TO_NFA(self):
        nfa_Ɛ_to_nfa(self.path)
```

- Class_REGEX

Esta clase recibe de parámetros una expresión regular y hace la conversión para crear un autómata, renderizarlo y crear el json a un nfa_e usando la función **Func_ Reg_NFA_Ɛ** .

```python
class Regex:

    def __init__(self, regex):

        self.regex = regex

    def regex_to_nfa_E(self):
        re_to_nfa_E(self.regex)
```

## CLass_TEST_DFA

Esta clase recibe de parámetro un path del Json dfa,muestra el dfa y puede insertar una expresión para revisar si es parte del autómata o no.

```python
class Test_DFA:

    def readJSONEvaluate(self, expresion, automata):
        print('\n\nEVALUATE JSON')
        with open("/media/hedmon/disk/Unitec/Q3_2020/Teoria_de_Computacion/project/project/Json&Images/"+automata) as file:
            data = json.load(file)
            alphabet = data['alphabet']
            states = data['states']
            initalState = data['initial_state']
            acceptingStates = data['accepting_states']
            transitions = data['transitions']
```

# Composición de cada Función

## - Func_NFA_Ɛ_TO_NFA

La función recibe un autómata como objeto y hace la conversión de un **NFA_Ɛ_TO_NFA** creando un json como el nuevo NFA y Renderizando el autómata.

```
def nfa_Ɛ_to_nfa(file_name: str):

    with open(file_name) as file:
        data = json.load(file)
        alphabet = data['alphabet']
        states = data['states']
        initalState = data['initial_state']
        acceptingStates = data['accepting_states']
        transitions = data['transitions']
```

## - Func_NFA_TO_DFA

La función recibe un autómata como objeto y hace la conversión de un **NFA_To_DFA** creando un json como el nuevo DFA y Renderizando el autómata.

```
def nfa_to_dfa(file_name):

    with open(file_name) as file:
        data = json.load(file)
        alphabet = data['alphabet']
        states = data['states']
        initalState = data['initial_state']
        acceptingStates = data['accepting_states']
        transitions = data['transitions']
```

## - Func_RE_TO_NFAE

La función recibe de parámetro un objeto RE y genera el autómata con la expresión regular

```python
def re_to_nfa_Ɛ(input: str) -> str:

    initial_state = ""
    final_states = []
    multiple_final_states = []
    count_finals = 0
    previews_state = ""
    state = "q"
    position = 0
    count = 0
    is_first = False
    open_par = False
    next_inpar = False
    new_or_final = False
    count_states = 0
    is_last_concat = False
    is_or = False
    comesfrom_par = False
    is_final = True
    nfa_Ɛ_table = []
```

- Write_json

La función recibe de parámetro el autómata y genera un json con los parámetros autómata properties M =[Q,E,F,S,&]  y genera el json del nuevo autómata.

```python
def write_jason(path, alphabet: list, states: list, initial_state: str, accepting_states: list, transitions: list):

    # alphabet = ["a", "b", "c"]
    # states = ["q0", "q1", "q3"]
    # initial_state = "q0"
    # accepting_states = ["q3", "q2"]
    # transitions = [['q0', '0', 'q0'], ['q0', '0', 'q1'], ['q0', '0', 'q2']]

    dictionary = {
        "alphabet": alphabet,
        "states": states,
        "initial_state": initial_state,
        "accepting_states": accepting_states,
        "transitions": transitions
    }

    # Serializing json
    json_object = json.dumps(dictionary, indent=4)

    # Writing to sample.json
    with open("Json&Images/"+path, "w") as outfile:
        outfile.write(json_object)
```

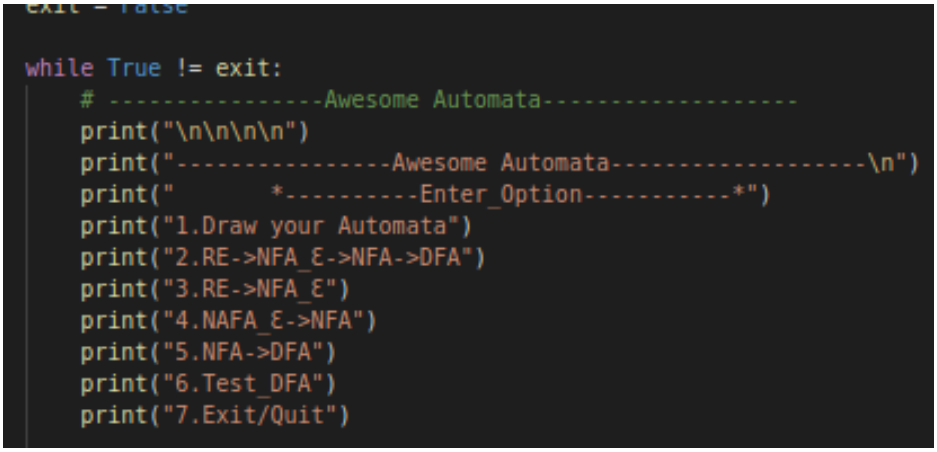
- Render_Automata

La función render autómata recibe de parámetros la tabla de transiciones sus estados iniciales y finales, renderiza el autómata a un formato pdf.

```
def render(path, Initicial_state: str, Final_state: list, Table_lists: list):
    # -------------------start rendering automata---------------
    f = Digraph(
        'Graph', filename='Json&Images/'+path)
    f.attr(rankdir='LR', size='8')
    # ----------------------Inicial state-----------------------------
    f.node('fake', style='invisible')
    f.edge('fake', Initicial_state, style='bold')
    if Initicial_state in Final_state:
        f.node(Initicial_state, root='true', shape='doublecircle')
    else:
        f.node(Initicial_state, root='true')
    # ----------------------final state-----------------------------
    f.attr('node', shape='doublecircle')
    for final_state in Final_state:
        f.node(final_state)
    # ----------------------Edge Linker-----------------------------
    print("------------Automata_Linker---------------------")
    f.attr('node', shape='circle')
    for x in range(0, len(Table_lists)):
        if Table_lists[x] == ["", "", ""]:
            print(" ")
        else:
            f.edge(Table_lists[x][0], Table_lists[x]
                   [2], label=Table_lists[x][1])
    f.view()
```

- Main

El main función tiene la opción de elegir la conversión que desea o si desea pintar su autómata en un modo libre.



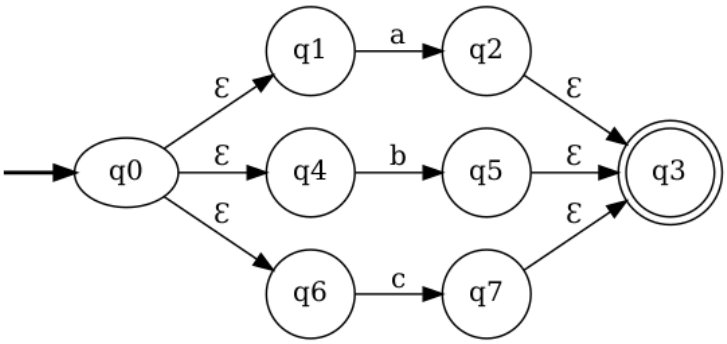
```
exit = False

while True != exit:
    # ----------------Awesome Automata-------------------
    print("\n\n\n\n")
    print("----------------Awesome Automata------------------\n")
    print("       *----------Enter Option----------*")
    print("1.Draw your Automata")
    print("2.RE->NFA ε->NFA->DFA")
    print("3.RE->NFA ε")
    print("4.NAFA ε->NFA")
    print("5.NFA->DFA")
    print("6.Test DFA")
    print("7.Exit/Quit")
```

## Test

La sección de prueba se mostrarán pruebas usando un json y el autómata que se renderiza usando la clase.

## Test1: Regular_Expresion

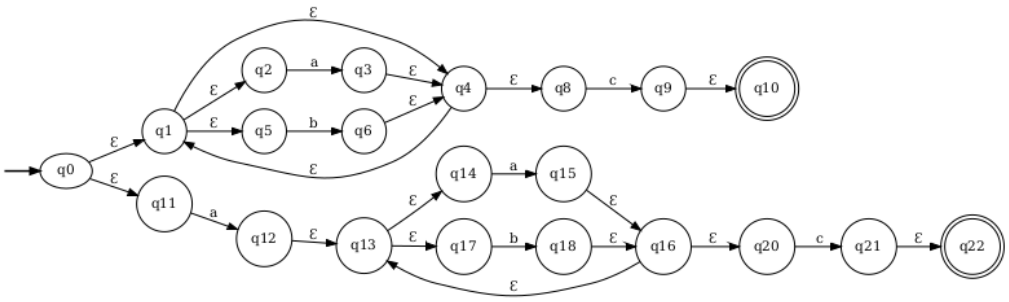
Usando la función que usa la clase regex  re_to_nfa_Ɛ("a | b | c")




Execution time: 0.378

## Test2: Regular_Expresion

Usando la función que usa la clase regex  re_to_nfa_Ɛ("(a | b)* c | a (a|b)+c")

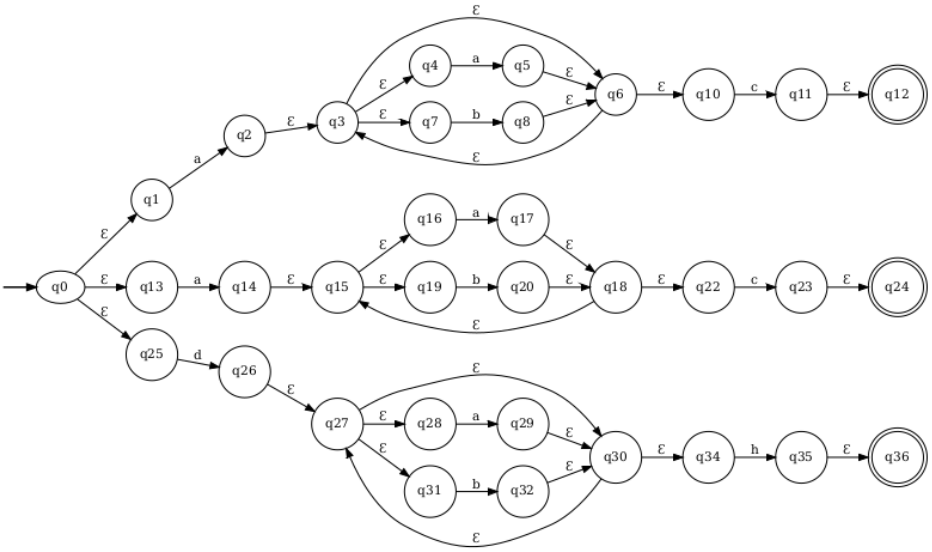



Execution time: 0.432

## Test3: Regular_Expresion

Usando la función que usa la clase regex

re_to_nfa_Ɛ(" a ( a | b )* c | a ( a | b )+ c | d ( a | )*h")




Execution time: 0.425

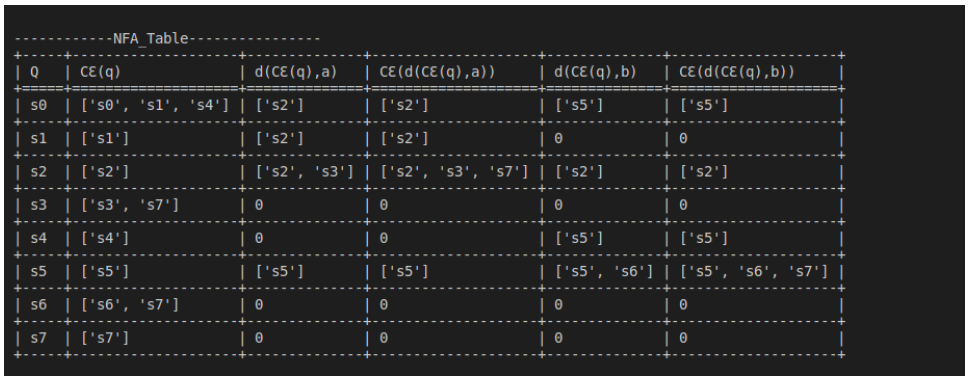
## Test1: NFA_Ɛ_TO_NFA

Usando la clase **NFA_Ɛ_TO_NFA** generamos el json, la tabla NFA y el autómata nfa.

Json autómata nfa_E

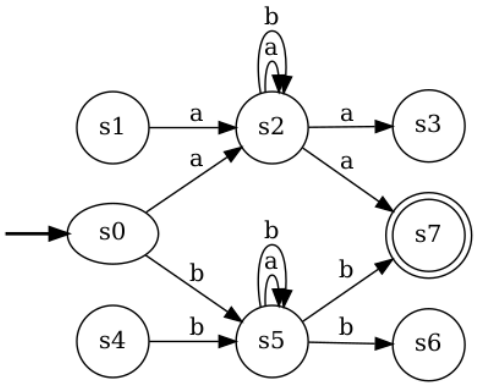
```
{
  "alphabet": ["a", "b", "Ɛ"],
  "states": ["s0", "s1", "s2", "s3", "s4", "s5", "s6", "s7"],
  "initial_state": "s0",
  "accepting_states": ["s7"],
  "transitions": [
    ["s0", "Ɛ", "s1"],
    ["s1", "a", "s2"],
    ["s2", "a", "s2"],
    ["s2", "b", "s2"],
    ["s2", "a", "s3"],
    ["s3", "Ɛ", "s7"],
    ["s0", "Ɛ", "s4"],
    ["s4", "b", "s5"],
    ["s5", "a", "s5"],
    ["s5", "b", "s5"],
    ["s5", "b", "s6"],
    ["s6", "Ɛ", "s7"]
  ]
}
```


Table NFA_E_to_NFA



```
------------NFA_Table----------------
+-----+------------------+-------------+--------------------+-------------+--------------------+
| Q   | CE(q)            | d(CE(q),a)  | CE(d(CE(q),a))     | d(CE(q),b)  | CE(d(CE(q),b))     |
+=====+==================+=============+====================+=============+====================+
| s0  | ['s0', 's1', 's4'] | ['s2']      | ['s2']             | ['s5']      | ['s5']             |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s1  | ['s1']           | ['s2']      | ['s2']             | 0           | 0                  |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s2  | ['s2']           | ['s2', 's3'] | ['s2', 's3', 's7'] | ['s2']      | ['s2']             |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s3  | ['s3', 's7']     | 0           | 0                  | 0           | 0                  |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s4  | ['s4']           | 0           | 0                  | ['s5']      | ['s5']             |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s5  | ['s5']           | ['s5']      | ['s5']             | ['s5', 's6'] | ['s5', 's6', 's7'] |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s6  | ['s6', 's7']     | 0           | 0                  | 0           | 0                  |
+-----+------------------+-------------+--------------------+-------------+--------------------+
| s7  | ['s7']           | 0           | 0                  | 0           | 0                  |
+-----+------------------+-------------+--------------------+-------------+--------------------+
```

Render NFA




## Test2: NFA_Ɛ_TO_NFA

Usando la clase **NFA_Ɛ_TO_NFA** y generamos el json, la tabla NFA y el autómata nfa.

Json automata nfa_E

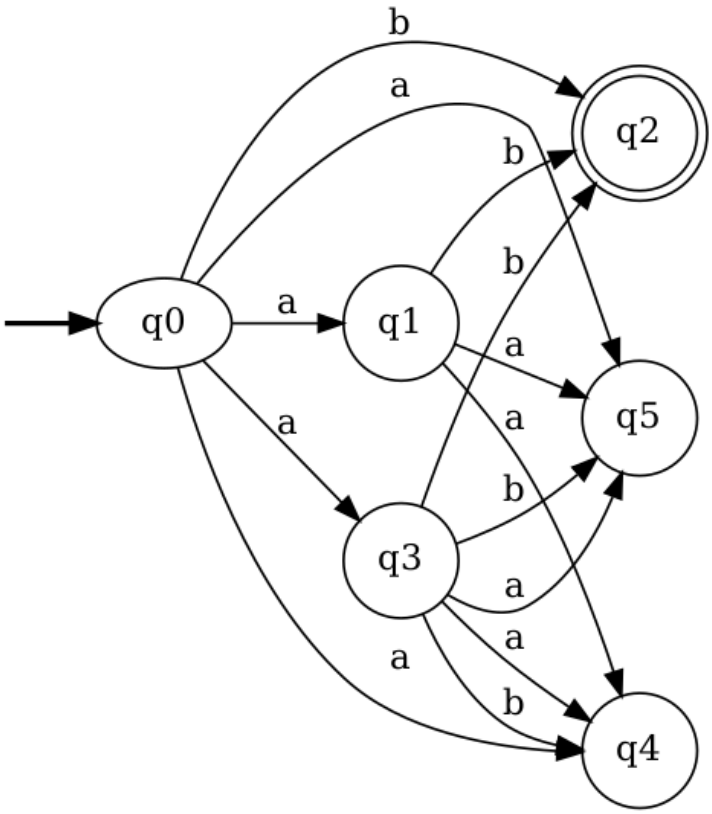

```
{
  "alphabet": ["a", "b", "Ɛ"],
  "states": ["q0", "q1", "q2", "q3", "q4", "q5"],
  "initial_state": "q0",
  "accepting_states": ["q2"],
  "transitions": [
    ["q0", "a", "q3"],
    ["q0", "Ɛ", "q1"],
    ["q1", "a", "q4"],
    ["q1", "b", "q2"],
    ["q3", "b", "q4"],
    ["q3", "Ɛ", "q1"],
    ["q4", "Ɛ", "q5"]
  ]
}
```


Table NFA_E_to_NFA

```
------------NFA_Table---------------
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| Q   | CE(q)       | d(CE(q),a)  | CE(d(CE(q),a))          | d(CE(q),b)  | CE(d(CE(q),b))     |
+=====+=============+=============+=========================+=============+====================+
| q0  | ['q0', 'q1'] | ['q3', 'q4'] | ['q3', 'q1', 'q4', 'q5'] | ['q2']      | ['q2']             |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| q1  | ['q1']      | ['q4']      | ['q4', 'q5']            | ['q2']      | ['q2']             |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| q2  | ['q2']      | 0           | 0                       | 0           | 0                  |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| q3  | ['q3', 'q1'] | ['q4']      | ['q4', 'q5']            | ['q4', 'q2'] | ['q4', 'q5', 'q2'] |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| q4  | ['q4', 'q5'] | 0           | 0                       | 0           | 0                  |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
| q5  | ['q5']      | 0           | 0                       | 0           | 0                  |
+-----+-------------+-------------+-------------------------+-------------+--------------------+
```

Execution time: 0.482

Render NFA

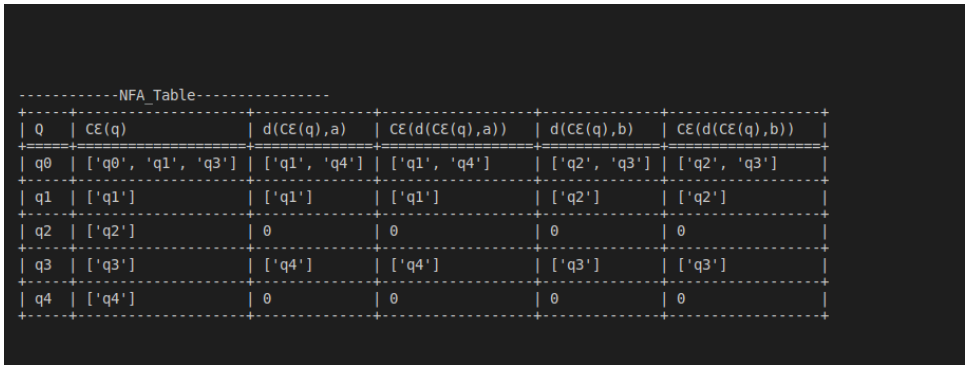



## Test3: NFA_Ɛ_TO_NFA

Usando la clase **NFA_Ɛ_TO_NFA** y generamos el json, la tabla NFA y el autómata NFA.

Json autómata nfa_E


Table NFA_E_to_NFA

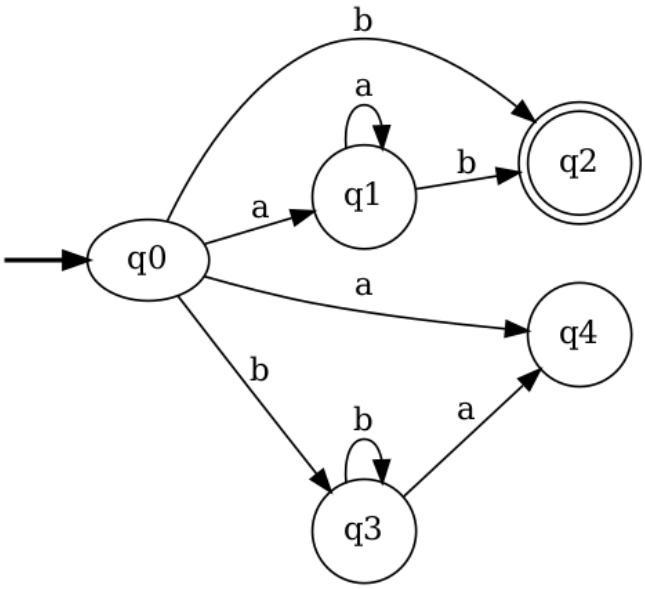


----------NFA Table----------

| Q | CƐ(q) | d(CƐ(q),a) | CƐ(d(CƐ(q),a)) | d(CƐ(q),b) | CƐ(d(CƐ(q),b)) |
|---|---|---|---|---|---|
| q0 | ['q0', 'q1', 'q3'] | ['q1', 'q4'] | ['q1', 'q4'] | ['q2', 'q3'] | ['q2', 'q3'] |
| q1 | ['q1'] | ['q1'] | ['q1'] | ['q2'] | ['q2'] |
| q2 | ['q2'] | 0 | 0 | 0 | 0 |
| q3 | ['q3'] | ['q4'] | ['q4'] | ['q3'] | ['q3'] |
| q4 | ['q4'] | 0 | 0 | 0 | 0 |

Execution time: 0.482

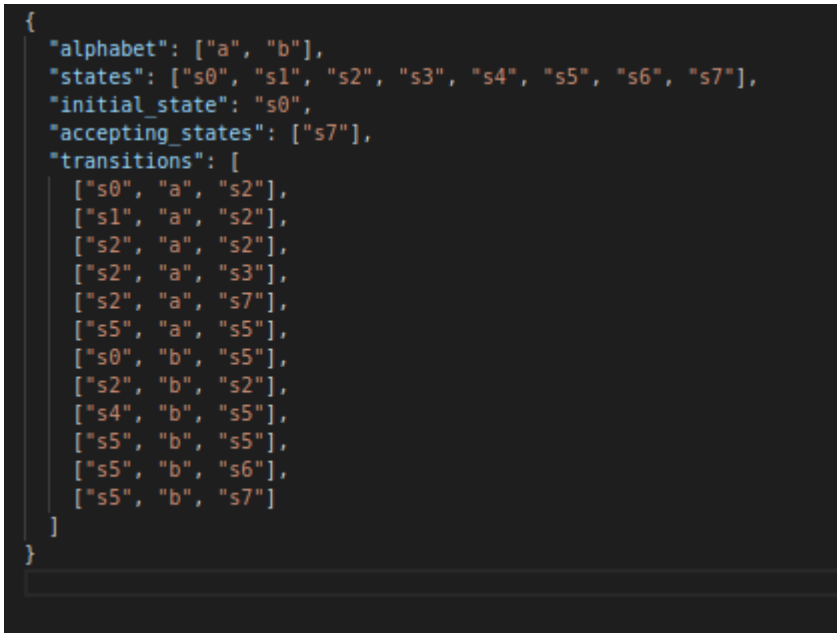
Render NFA




## Test1: NFA_To_DFA

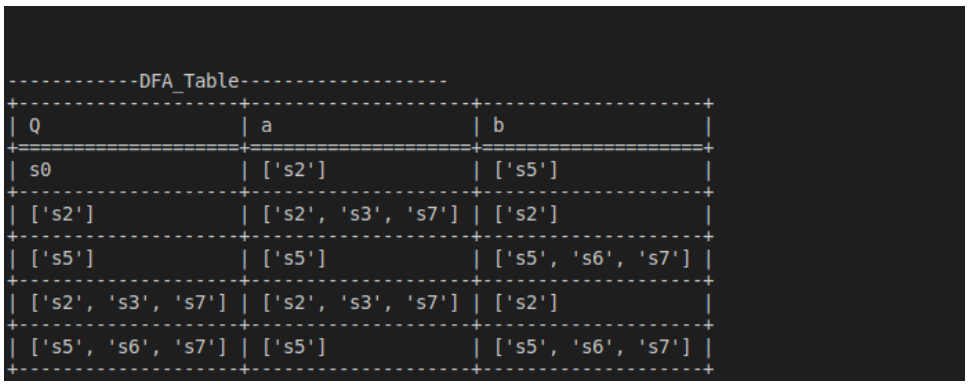
Usando la función **NFA_To_DFA** generamos el json, la tabla DFA , renderizamos el DFA.

## Json automata nfa



```
{
  "alphabet": ["a", "b"],
  "states": ["s0", "s1", "s2", "s3", "s4", "s5", "s6", "s7"],
  "initial_state": "s0",
  "accepting_states": ["s7"],
  "transitions": [
    ["s0", "a", "s2"],
    ["s1", "a", "s2"],
    ["s2", "a", "s2"],
    ["s2", "a", "s3"],
    ["s2", "a", "s7"],
    ["s5", "a", "s5"],
    ["s0", "b", "s5"],
    ["s2", "b", "s2"],
    ["s4", "b", "s5"],
    ["s5", "b", "s5"],
    ["s5", "b", "s6"],
    ["s5", "b", "s7"]
  ]
}
```

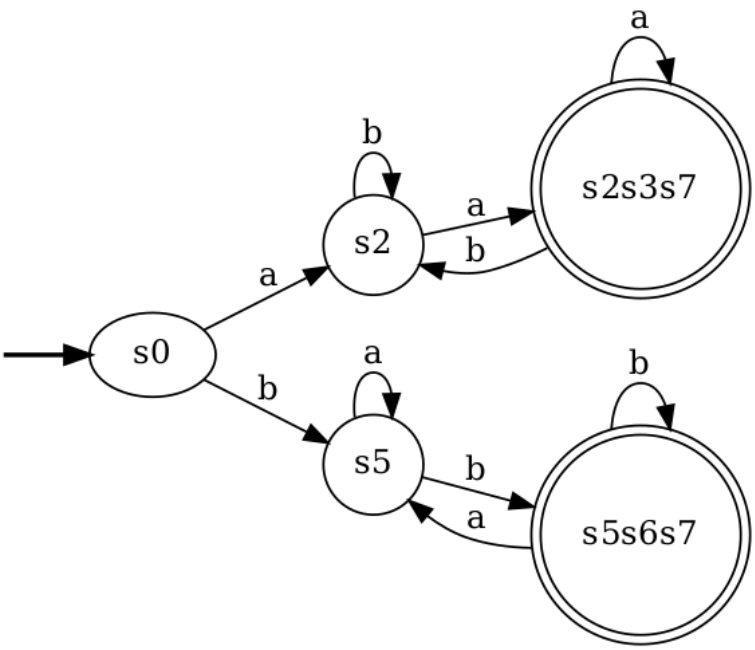
Table NFA_To_DFA



```
------------DFA_Table-------------------
+--------------------+--------------------+--------------------+
| Q                  | a                  | b                  |
+====================+====================+====================+
| s0                 | ['s2']             | ['s5']             |
+--------------------+--------------------+--------------------+
| ['s2']             | ['s2', 's3', 's7'] | ['s2']             |
+--------------------+--------------------+--------------------+
| ['s5']             | ['s5']             | ['s5', 's6', 's7'] |
+--------------------+--------------------+--------------------+
| ['s2', 's3', 's7'] | ['s2', 's3', 's7'] | ['s2']             |
+--------------------+--------------------+--------------------+
| ['s5', 's6', 's7'] | ['s5']             | ['s5', 's6', 's7'] |
+--------------------+--------------------+--------------------+
```

Execution time: 0.378

## Render DFA_Table




## Test2:NFA_To_DFA

Usando la función **NFA_To_DFA** generamos el json, la tabla DFA, renderizamos el DFA.

Json autómata NFA

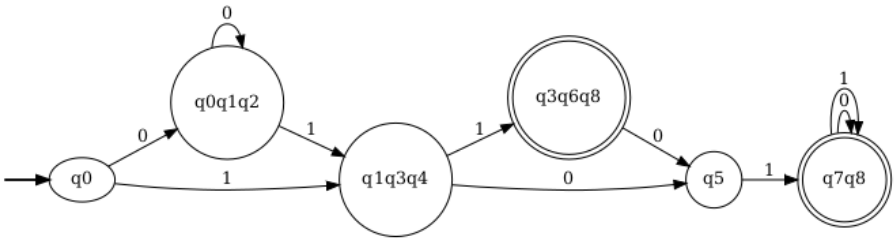
```
{
  "alphabet": ["0", "1"],
  "states": ["q0", "q1", "q2", "q3", "q4", "q5", "q6", "q7", "q8"],
  "initial_state": "q0",
  "accepting_states": ["q8"],
  "transitions": [
    ["q0", "0", "q0"],
    ["q0", "0", "q1"],
    ["q0", "0", "q2"],
    ["q3", "0", "q5"],
    ["q7", "0", "q7"],
    ["q7", "0", "q8"],
    ["q0", "1", "q1"],
    ["q0", "1", "q3"],
    ["q0", "1", "q4"],
    ["q1", "1", "q3"],
    ["q2", "1", "q4"],
    ["q4", "1", "q6"],
    ["q4", "1", "q8"],
    ["q5", "1", "q7"],
    ["q5", "1", "q8"],
    ["q7", "1", "q7"],
    ["q7", "1", "q8"]
  ]
}
```


Table NFA_To_DFA

```
------------DFA_Table-------------------
+--------------------+--------------------+--------------------+
| Q                  | 0                  | 1                  |
+====================+====================+====================+
| q0                 | ['q0', 'q1', 'q2'] | ['q1', 'q3', 'q4'] |
+--------------------+--------------------+--------------------+
| ['q0', 'q1', 'q2'] | ['q0', 'q1', 'q2'] | ['q1', 'q3', 'q4'] |
+--------------------+--------------------+--------------------+
| ['q1', 'q3', 'q4'] | ['q5']             | ['q3', 'q6', 'q8'] |
+--------------------+--------------------+--------------------+
| ['q5']             | 0                  | ['q7', 'q8']       |
+--------------------+--------------------+--------------------+
| ['q3', 'q6', 'q8'] | ['q5']             | 0                  |
+--------------------+--------------------+--------------------+
| ['q7', 'q8']       | ['q7', 'q8']       | ['q7', 'q8']       |
+--------------------+--------------------+--------------------+
```

Execution time: 0.341

## Render DFA_Table




## Test3:NFA_To_DFA

Usando la función **NFA_To_DFA** generamos el json, la tabla DFA, renderizamos el DFA.

Json autómata nfa

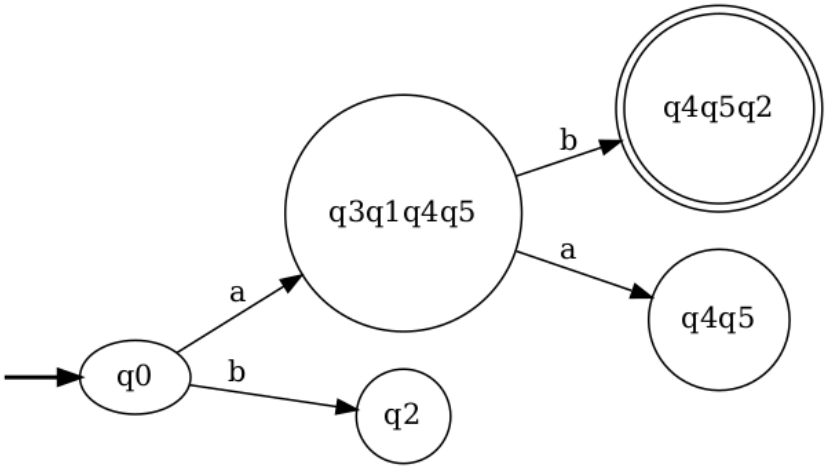

```
{
  "alphabet": ["a", "b"],
  "states": ["q0", "q1", "q2", "q3", "q4", "q5"],
  "initial_state": "q0",
  "accepting_states": ["q2"],
  "transitions": [
    ["q0", "a", "q3"],
    ["q0", "a", "q1"],
    ["q0", "a", "q4"],
    ["q0", "a", "q5"],
    ["q1", "a", "q4"],
    ["q1", "a", "q5"],
    ["q3", "a", "q4"],
    ["q3", "a", "q5"],
    ["q0", "b", "q2"],
    ["q1", "b", "q2"],
    ["q3", "b", "q4"],
    ["q3", "b", "q5"],
    ["q3", "b", "q2"]
  ]
}
```


Table NFA_To_DFA

```
------------DFA_Table------------------
+--------------------------+--------------------------+--------------------+
| Q                        | a                        | b                  |
+==========================+==========================+====================+
| q0                       | ['q3', 'q1', 'q4', 'q5'] | ['q2']             |
+--------------------------+--------------------------+--------------------+
| ['q3', 'q1', 'q4', 'q5'] | ['q4', 'q5']             | ['q4', 'q5', 'q2'] |
+--------------------------+--------------------------+--------------------+
| ['q2']                   | 0                        | 0                  |
+--------------------------+--------------------------+--------------------+
| ['q4', 'q5']             | 0                        | 0                  |
+--------------------------+--------------------------+--------------------+
| ['q4', 'q5', 'q2']       | 0                        | 0                  |
+--------------------------+--------------------------+--------------------+
```

Execution time: 0.693

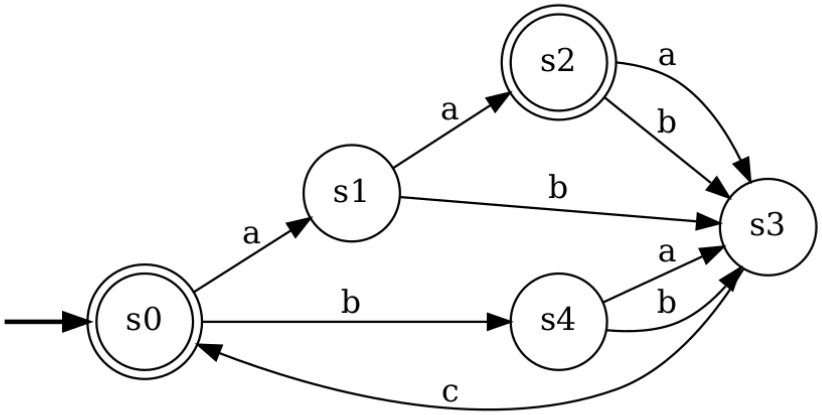
Render DFA_Table




### Test1: Test_DFA1:

El test_DFA Evalua si la expresion es parte del automata.

Automata_DFA:




Regex1:abc

Result:

```
EVALUATE JSON
-------------Automata_Linker---------------------
Enter value to evaluate: abc
current-> s0 - a
enter s0 a s1
current-> s1 - b
enter s1 b s3
current-> s3 - c
enter s3 c s0
--> s s0
--> s0 s0

The expresion is part of the automata languaje
--Press Enter to continue evaluating or 0 to exit-
```

Execution time: 0.340

Regex2:abc

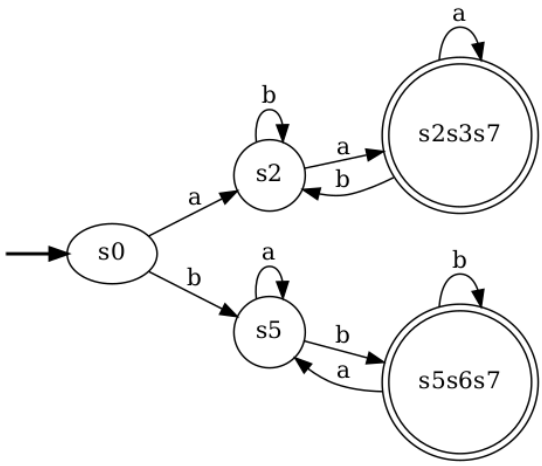
```
EVALUATE JSON
-------------Automata_Linker---------------------
Enter value to evaluate: ab
current-> s0 - a
enter s0 a s1
current-> s1 - b
enter s1 b s3
--> s s3
--> s0 s3
--> s s3
--> s2 s3

The expresion is not part of the automata languaje
--Press Enter to continue evaluating or 0 to exit-
```

Execution time: 0.336

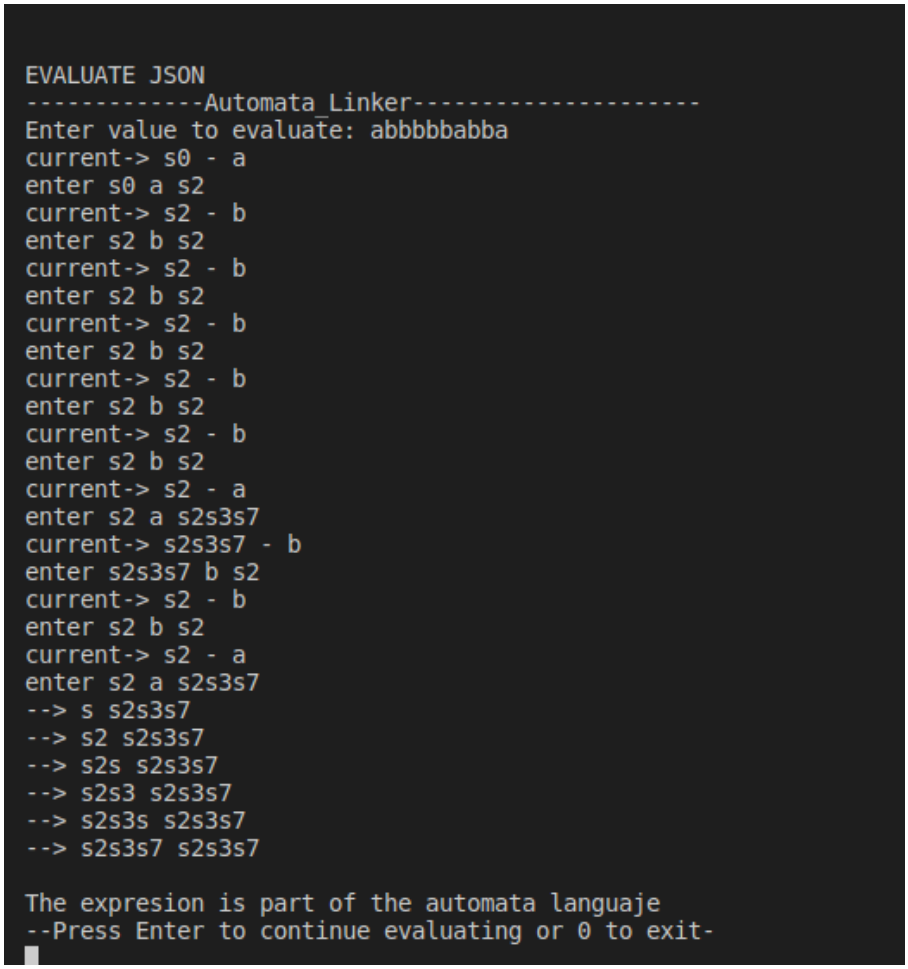
### Test2: Test_DFA2:

El test_DFA Evalua si la expresion es parte del automata.

Automata_DFA:

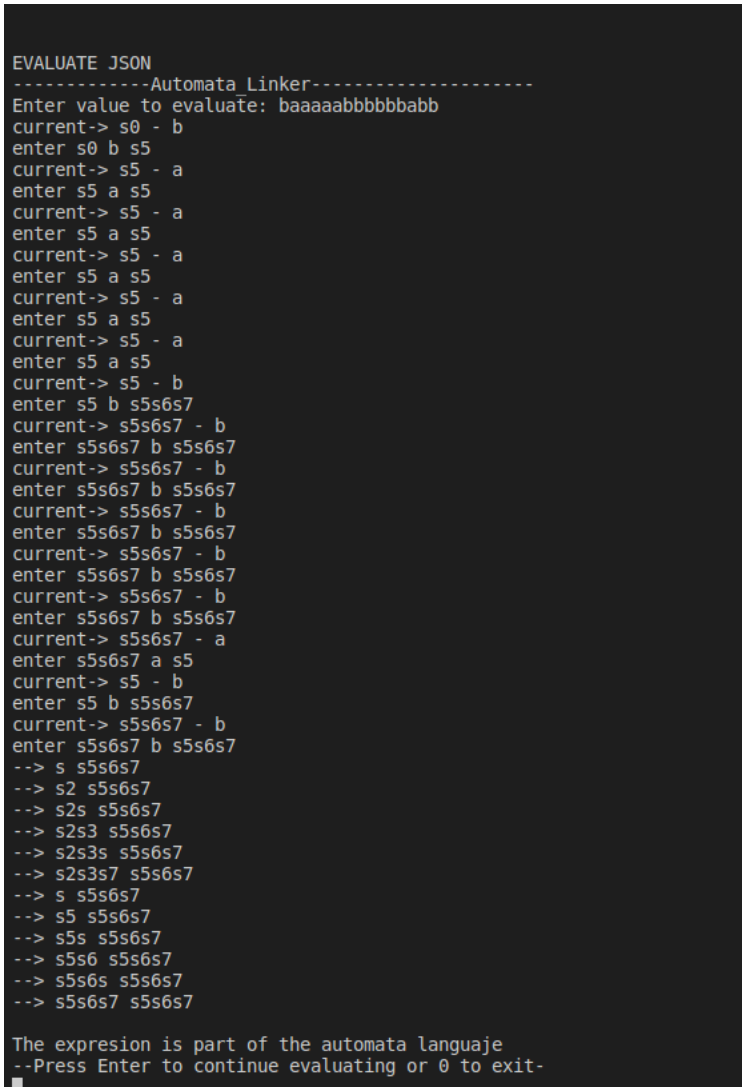



Regex1:abbbbbabba




Execution time: 0.453

Regex2:baaaaabbbbbbabb




Execution time: 0.521

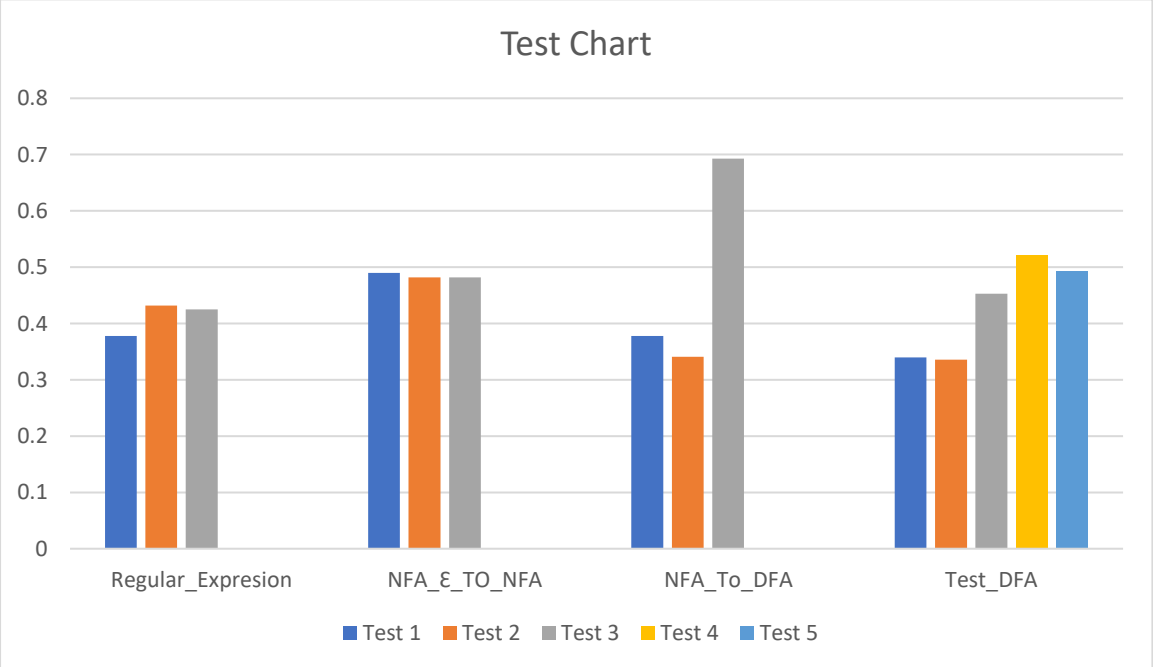
Regex3: babaaabaa

```
EVALUATE JSON
------------Automata_Linker--------------------
Enter value to evaluate: babaaabaa
current-> s0 - b
enter s0 b s5
current-> s5 - a
enter s5 a s5
current-> s5 - b
enter s5 b s5s6s7
current-> s5s6s7 - a
enter s5s6s7 a s5
current-> s5 - a
enter s5 a s5
current-> s5 - a
enter s5 a s5
current-> s5 - b
enter s5 b s5s6s7
current-> s5s6s7 - a
enter s5s6s7 a s5
current-> s5 - a
enter s5 a s5
--> s s5
--> s2 s5
--> s2s s5
--> s2s3 s5
--> s2s3s s5
--> s2s3s7 s5
--> s s5
--> s5 s5
--> s5s s5
--> s5s6 s5
--> s5s6s s5
--> s5s6s7 s5

The expresion is not part of the automata languaje
--Press Enter to continue evaluating or 0 to exit-
```

Execution time: 0.492

## Tiempo promedio

En la siguiente grafica se muestra el promedio de ejecución de que tuve cada prueba.



# Conclusión

Este Trabajo me llevo a ampliar mis conceptos sobre el tema señalado de la creación de autómatas la conversión de cada tipo todo lo que podemos hacer con los autónomos crear nuevos lenguajes con la misma y también aprender nuevos frameworks y softwares que hay para poder crear estos autómatas desde cero a su vez estos conocimientos y el proyecto realizado ayudara a muchas personas que quieran tener un conocimiento de autómatas y también como poder dibujarlos con el programa creado ya que tiene las óptimas funcionalidades para crear, convertir y testear los autómatas con la explicación de muchos conceptos nuevos con fácil entendimiento espero que estos conocimientos se puedan expandir a las demás personas y poder aportar a la comunidad mi informe con dicho programa.