Universidad Tecnológica Centroamericana

CLASE CURSADA:

Teoría de la Computación

CATEDRATICO:

Ing. Orellana Pineda Cesar Dario

PRESENTADO POR:

Vincent Barrios #21651085

CAMPUS

SAN PEDRO SULA, CORTES, HONDURAS

02 DE SEPTIEMBRE DEL 2020

Indice

Glosario 4

Introducción 6

1 Estructura del Proyecto 7

1.1 Autómatas 7

1.2 Librerías 8

1.3 Clases 9

1.3.1 RegularExpresion.py 9

1.3.2 NFAƐ_Automata.py 10

1.3.3 NFA_Automata.py 11

2 Pruebas y resultados 12

2.1 Expresión regular Regex 12

Prueba 1 12

Prueba 2 12

Prueba 3 13

Prueba 4 13

Prueba 5 13

2.2 NFA Épsilon 14

Prueba 1 14

Prueba 2 15

Prueba 3 16

Prueba 4 17

Prueba 5 18

2.3 NFA 19

Prueba 1 19

Prueba 2 20

Prueba 3 21

Prueba 4 22

Prueba 5 23

2.3 Evaluador DFA 24

Prueba 1 24

Prueba 2 26

Prueba 3 28

Prueba 4 29
Prueba 5 30
2.3.1 Tiempo promedio 32
Conclusion 33

Glosario

1. **Automata:** Máquina automática programable capaz de realizar determinadas operaciones de manera autónoma y sustituir a los seres humanos en algunas tareas, en especial las pesadas, repetitivas o peligrosas; puede estar dotada de sensores, que le permiten adaptarse a nuevas situaciones.

2. **Automata finito no determinista:** es un autómata finito que, a diferencia de los autómatas finitos deterministas, posee al menos un estado $q \in Q$, tal que para un símbolo $a \in \Sigma$ del alfabeto, existe más de una transición posible.

3. **Clase:** es un modelo que define un conjunto de variables (el estado), y métodos apropiados para operar con dichos datos (el comportamiento). Cada objeto creado a partir de la clase se denomina instancia de la clase.

4. **Graphviz:** es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT**.**

5. **Lenguaje del autómata:** Conjunto no vacio y finito de símbolos (puedo o no contenar la cadena vacio)

6. **Metodo:** es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase, como es el caso de los métodos de clase o estáticos, como a un objeto, como es el caso de los métodos de instancia.

7. **Orientado a objetos:** se define como un paradigma de la programación, una manera de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones.

8. **Regex:** denotan expresiones regulares que se utilizan en la informática teórica, programación, desarrollo de software, procesamiento de textos y optimización de motores de búsqueda.

9. **Renderizar:** es un anglicismo para representación gráfica, usado en la jerga informática para referirse al proceso de generar imagen fotorrealista o no fotorrealista a partir de un modelo 2D o 3D por medio de programas informáticos.

10. **JSON:** 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.

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 implementó la expresión regular o mejor conocida como regex, autómata nfa epsilon, 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.

1 Estructura del Proyecto

Se utiliza la estructura orientadas a objectos en donde se divide en clases, métodos, atributos, eventos, objectos, herencia, etc. Se definió una estructura basa para todos los autómatas con la cual se trabaja y se utiliza también para los archivos json. Las gráficas llevan un formato ya determinado con el cual se trabaja para poder renderizar ese formato y desplegarlo en una imagen.

1.1 Autómatas

Un autómata se refiero a todo dispositivo tecnológico capaz de realizar ciertas tareas de manera autómata o determinados movimientos. Como se mencionó previamente cada autómata sigue la misma estructura (figura 1 se muestra la estructura) la cual está compuesta por un alfabeto, estados, estado inicial, estados aceptados y transiciones.

```
{
    "alphabet": ["A", "B"],
    "states": ["q0", "q1", "q2", "q3", "q4", "q5"],
    "initial_state": "q0", "accepting_states": ["q5"],
    "transitions": [
        ["q0", "ε", "q1"],
        ["q1", "A", "q2"],
        ["q2", "ε", "q3"],
        ["q3", "B", "q4"],
        ["q4", "ε", "q5"]]
}
```

Figura 1

- 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 indican el símbolo que pasa atreves de esa transición. (figura 1.1 muestra la simbología)

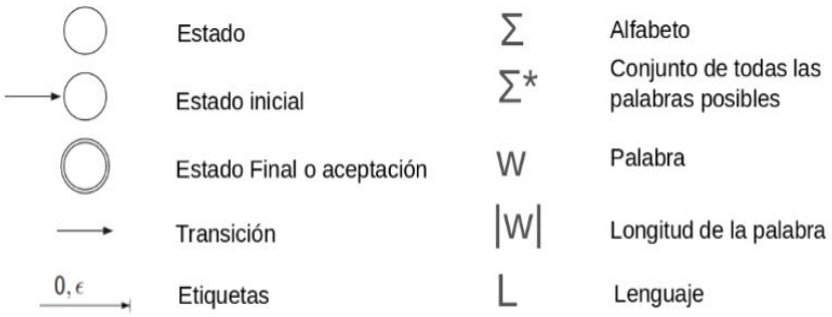


Figura 1.1

1.2 Librerías

El proyecto este desarrollo en el lenguaje de Python. En donde se utilizaron las librerías:

- **JSON** se utilizó para poder hacer posible la importación y exportación de archivos json en el proyecto los cuales se utilizan para leer y escribir el resultado de los autómatas.
- **Graphviz** se encarga de renderizar el resultado de cada autómata, pero de manera visual en una gráfica con cada una de sus tradiciones y componentes.

1.3 Clases

Este proyecto está compuesto de 3 clases principales que se encargar de realizar todo el trabajo pesado. (la figura 1.2 se pueden observar estas clases)

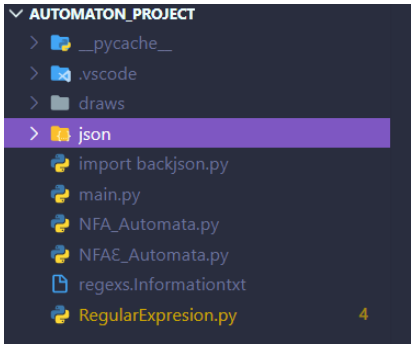


Figura 1.2

1.3.1 RegularExpresion.py

Como lo detona su nombre realiza la tarea de convertir una expresión regular o regex a un autómata de tipo nfa epsilon. Esto los hace mediante una serie de métodos cuales son los siguientes:

- **AddNode** (agregar nodo) crea un nuevo nodo o estado para cada transición realizada.
- **Transition** (transición) crea una nueva transición enviado le dé parámetro el estado de inicio y final al igual que el símbolo de transición.
- **ConvertToNFA Ɛ** (convertir a nfa epsilon) se encarga del trabajo pensado dentro de este método se encuentra todo el código que hace posible la conversión de una simple expresión regular a crear un autómata épsilon completo.
- **CreateView** utiliza la librería graphviz para poder renderizar las tradiciones del autómata a un archivo de formato gv el cual se puede visualizar después y observar el resultado de esa expresión en una gráfica.

1.3.2 NFAƐ_Automata.py

Realiza la conversión de un autómata nfa epsilon a un autómata nfa. Para este autómata se lee desde un archivo json el cual primero realiza una tabla base y a partir de esa tabla base los métodos acceden a ella realizan la conversión hasta obtener el resultado final. Se llevo a cabo con los siguientes métodos:

- **BuildTable** (construir la tabla) se encarga de crear la tabla base que sirve para las demas funciones.
- **EpsilonState** (estado epsilon) realiza la primera table de nfa epsilon la cual es cerradura epsilon y las realiza de manera simultánea.
- **EpsilonAlpha** (alfa epsilon) realiza el siguiente set de tablas que son cerradura epsilon con alfabeto las cuales reciben de parámetro las tablas de cerradura epsilon. Pero de la misma manera esta contiene otras dos funciones (epsilonAlphaHelper y epsilonState) que le ayudan a completar su trabajo.
- **CreateNfa** (crear nfa) como lo indica su nombre se encarga de crear el nfa equivalente. Esta función recibe de parametros el alfabeto, estado incial, los estados, los estados aceptados y por último la tabla de la funcion EpsilonAlpha con todos estos datos crea el nfa y lo dibujo con la ayuda de la funcion createView.
- **CreateView** utiliza la librería graphviz para poder renderizar las tradiciones del autómata a un archivo de formato gv el cual se puede visualizar después y observar el resultado de esa expresión en una gráfica.

1.3.3 NFA_Automata.py

En esta clase se realizan dos distintas tareas la primera es leer el autómata nfa que se escoja y lo convierte en un autómata dfa de la misma manera se puede ingresar una expresión regular y evaluar esa expresión con el autómata que se convirtió con el fin de constatar si pertenece al lenguaje del mismo.

- **NfaFistEvaluation** (nfa primera evaluación) lee el autómata nfa y a partir de ese autómata produce una tabla base que utiliza la función nfa_convertion (conversión nfa) y no recibe ningún parámetro.
- **Nfa_Convertion** (conversión de nfa) ejecute un bloque de código el cual recibe de parámetro la tabla base menciona anteriormente y a partir de esa tabla comienza a construir el dfa. Todo esto lo realiza de manera recursiva cuando se evalúa cada nueva combinación de estado que puede surgir en el proceso.
- **EvaluateSymbol** (evaluar simbolo) esta funcion se encarga de la segunta tarea la cual es evaluar si la expresion ingresa por el usuario es parte del lenguaje del automata previamente creado o de cierto archivo json igualmente.
- **CreateView** utiliza la librería graphviz para poder renderizar las tradiciones del autómata a un archivo de formato gv el cual se puede visualizar después y observar el resultado de esa expresión en una gráfica.

2 Pruebas y resultados

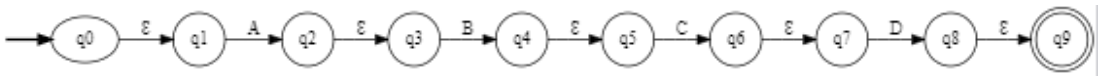
En este proyecto se ejecutaron múltiples pruebas para determinar qué tan consistentes eran los resultados. Por cada tipo de autómata se hicieron pruebas con distintos datos de esta manera se podía asegurar que la integridad de cada método no era afectada por otra y los resultados no fluctuaran.

2.1 Expresión regular Regex

Para las siguientes pruebas la palabra “regex” se referirá a la expresión ingresada del usuario. Debajo de cada expresión regex se adjuntará una imagen la cual es la equivalencia a un autómata epsilon.

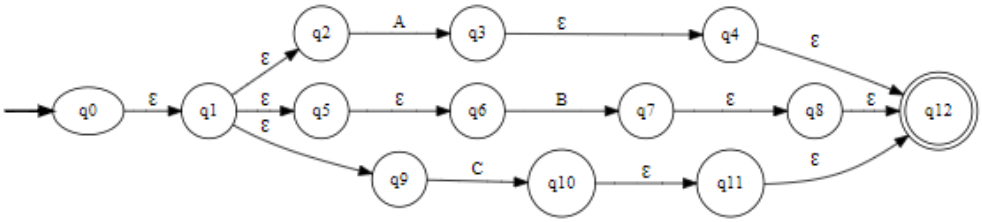
Prueba 1

Regex = ABCD



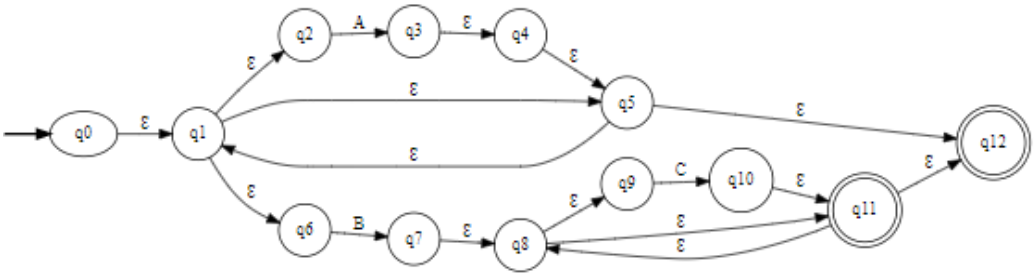
Prueba 2

Regex = A|B|C



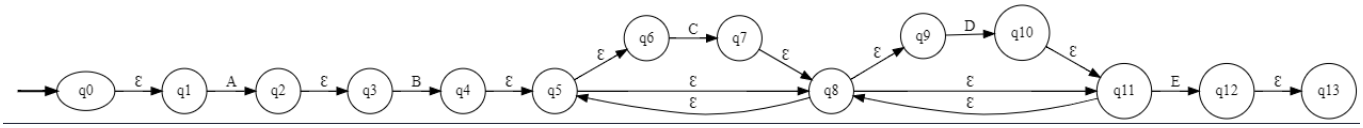
Prueba 3

Regex = A*|BC*



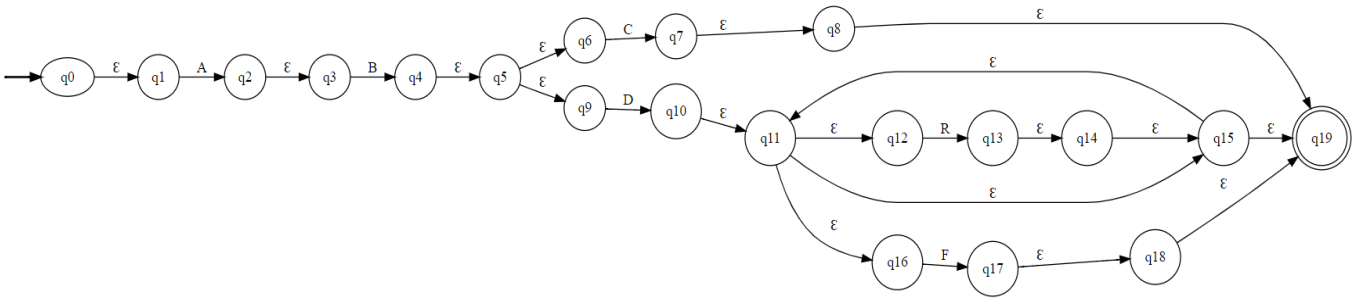
Prueba 4

Regex = AB(C*D*)E



Prueba 5

REGEX = AB(C|D(R*|F))



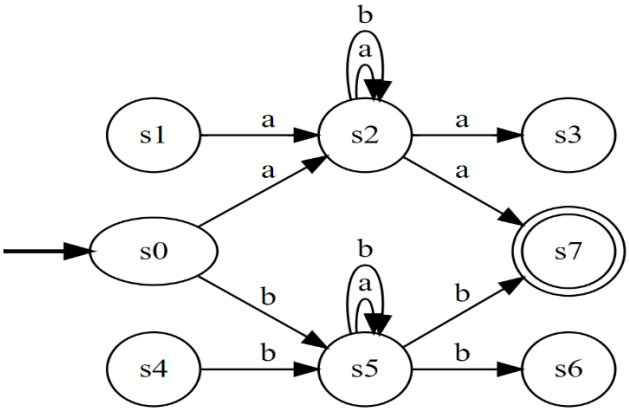
2.2 NFA Épsilon

En esta fase de pruebas se adjuntará una imagen de un autómata nfa épsilon en formato json y en la parte inferior de la misma se mostrará una imagen con el resultado de la gráfica. Como resultado entrega un autómata nfa equivalente. En los formatos json se encuentra un texto “\u0190” ese texto equivale al símbolo “Ɛ”.

Prueba 1

```json
{
    "alphabet": ["a", "b",   "\u0190"],
    "states": ["s0", "s1", "s2", "s3", "s4", "s5", "s6", "s7"],
    "initial_state": "s0",
    "accepting_states": ["s7"],
    "transitions": [
        ["s0", "\u0190", "s1"],
        ["s1", "a", "s2"],
        ["s2", "a", "s2"],
        ["s2", "b", "s2"],
        ["s2", "a", "s3"],
        ["s3", "\u0190", "s7"],
        ["s0", "\u0190", "s4"],
        ["s4", "b", "s5"],
        ["s5", "a", "s5"],
        ["s5", "b", "s5"],
        ["s5", "b", "s6"],
        ["s6", "\u0190", "s7"]
    ]
}
```

Formato JSON



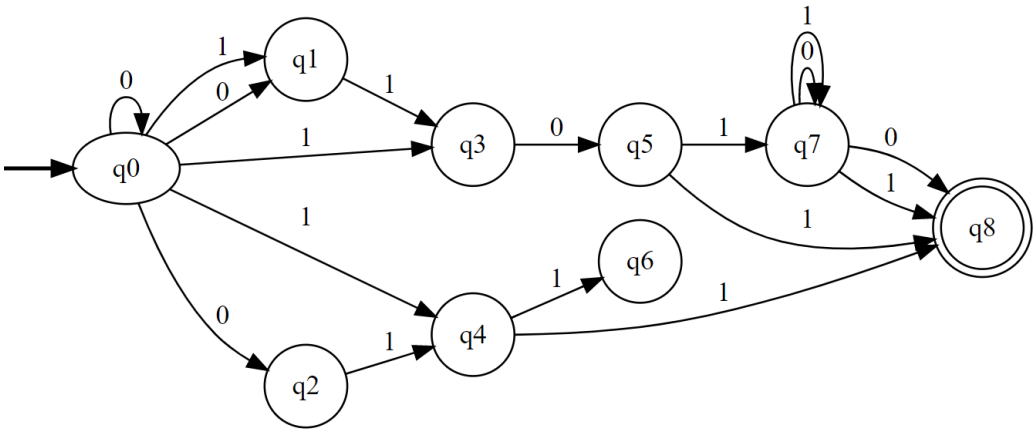
Respuesta Grafica

Prueba 2


```
{
    "alphabet": ["0", "1","\u0190"],
    "states": ["q0", "q1","q2","q3","q4","q5","q6","q7","q8"],
    "initial_state": "q0",
    "accepting_states": ["q8"],
    "transitions": [

        ["q1", "1","q3"],
        ["q1", "\u0190","q1"],
        ["q2", "1","q4"],
        ["q2", "\u0190","q2"],
        ["q3", "0","q5"],
        ["q3", "\u0190","q3"],
        ["q0", "0","q0"],
        ["q0", "1","q1"],
        ["q0", "\u0190","q0"],
        ["q0", "\u0190","q1"],
        ["q0", "\u0190","q2"],
        ["q4", "1","q6"],
        ["q4", "\u0190","q4"],
        ["q5", "1","q7"],
        ["q5", "\u0190","q5"],
        ["q6", "\u0190","q6"],
        ["q6", "\u0190","q8"],
        ["q7", "0","q7"],
        ["q7", "1","q7"],
        ["q7", "\u0190","q8"],
        ["q7", "\u0190","q7"],
        ["q8", "\u0190","q8"]
    ]
}
```


Formato JSON

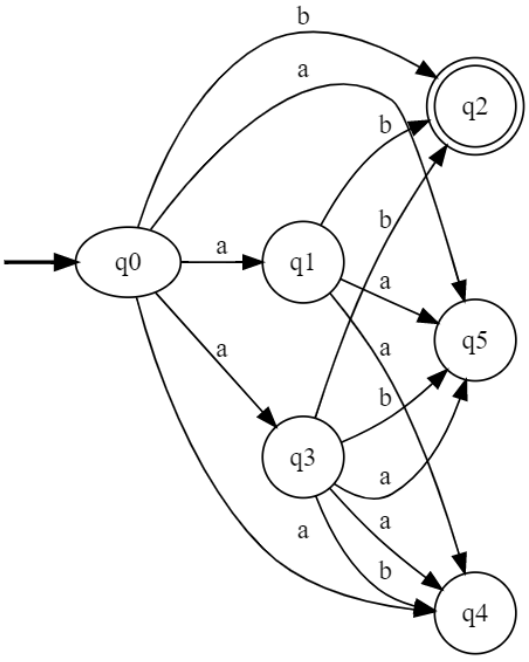


Respuesta Grafica

Prueba 3


```
{
    "alphabet": ["a", "b","\u0190"],
    "states": ["q0", "q1","q2","q3","q4","q5"],
    "initial_state": "q0",
    "accepting_states": ["q2"],
    "transitions": [
        ["q0", "a","q3"],
        ["q0", "\u0190","q1"],
        ["q1", "a","q4"],
        ["q1", "b","q2"],
        ["q3", "b","q4"],
        ["q3", "\u0190","q1"],
        ["q4", "\u0190","q5"]
    ]
}
```


Formato JSON

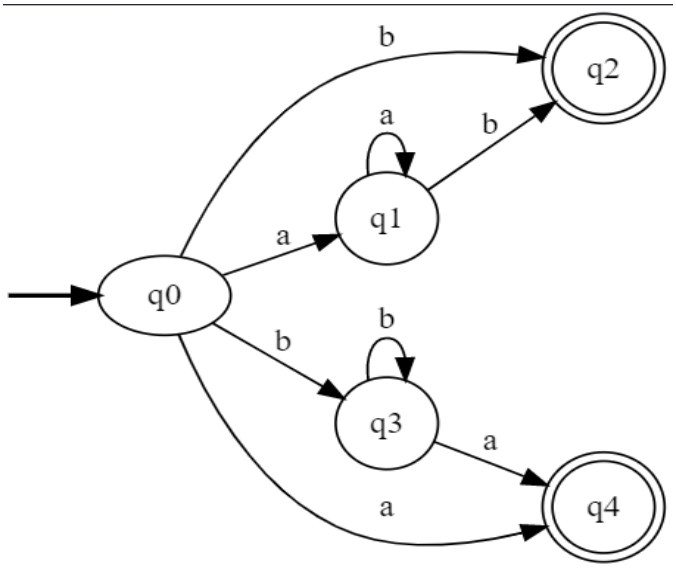


Respuesta Grafica

Prueba 4


```
{
    "alphabet": ["a", "b","\u0190"],
    "states": ["q0", "q1","q2","q3","q4"],
    "initial_state": "q0",
    "accepting_states": ["q2"],
    "transitions": [
        ["q0", "\u0190","q1"],
        ["q0", "\u0190","q3"],
        ["q1", "a","q1"],
        ["q1", "b","q2"],
        ["q3", "a","q4"],
        ["q3", "b","q3"]
    ]
}
```


Formato JSON



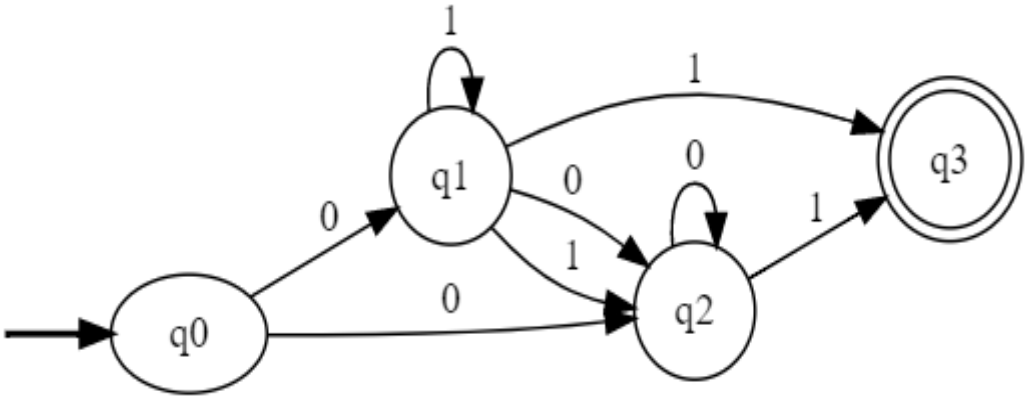
Respuesta Grafica

Prueba 5


```
{
    "alphabet": ["0", "1","\u0190"],
    "states": ["q0", "q1","q2","q3"],
    "initial_state": "q0",
    "accepting_states": ["q3"],
    "transitions": [

        ["q0", "0","q1"],
        ["q1", "1","q1"],
        ["q1", "\u0190","q2"],
        ["q2", "0","q2"],
        ["q2", "1","q3"]
    ]
}
```


Formato JSON



Respuesta Grafica

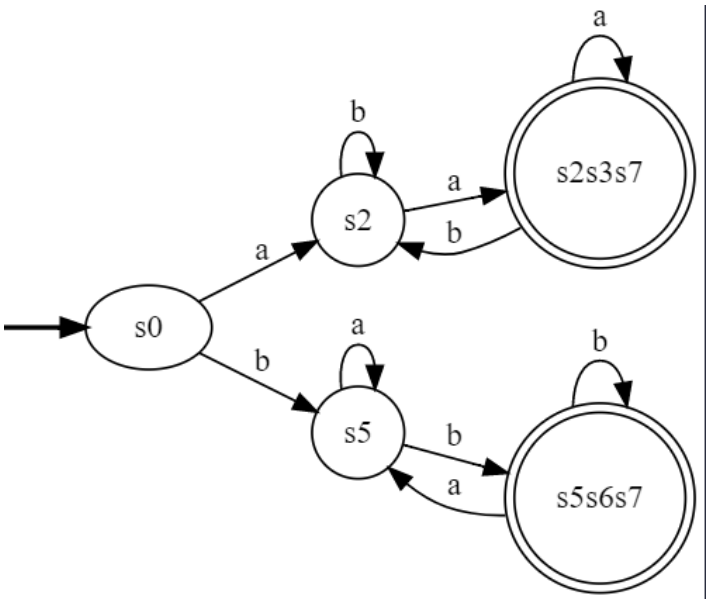
2.3 NFA

En esta fase de pruebas se adjuntará una imagen de un autómata nfa en formato json y en la parte inferior de la misma se mostrará una imagen con el resultado de la gráfica. Como resultado entrega un autómata dfa equivalente.

Prueba 1


```
{"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"]]
}
```


Formato JSON

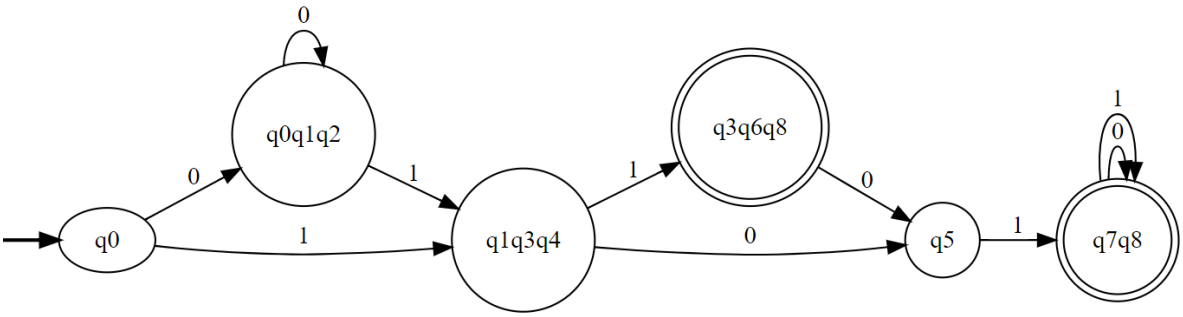


Respuesta Grafica

Prueba 2


```
{"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"]]}
```


Formato JSON

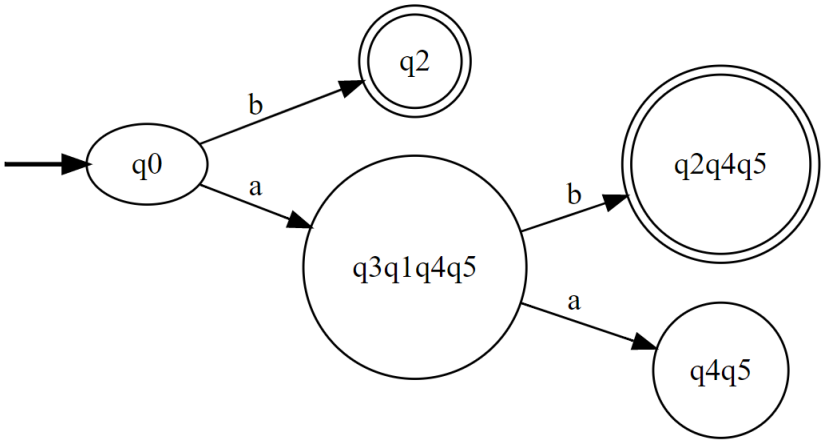


Respuesta Grafica

Prueba 3


```
{"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"]]}
```


Formato JSON

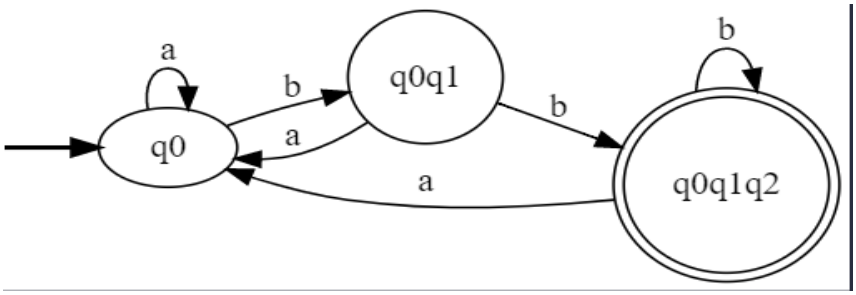


Respuesta Grafica

Prueba 4

```
{
'alphabet': ['a', 'b'],
'states': ['q0', 'q1', 'q2'],
'initial_state': 'q0',
'accepting_states': ['q2'],
'transitions': [
    ['q0', 'a', 'q0'],
    ['q0', 'b', 'q0'],
    ['q0', 'b', 'q1'],
    ['q1', 'b', 'q2'],
    ]
}
```

Formato JSON

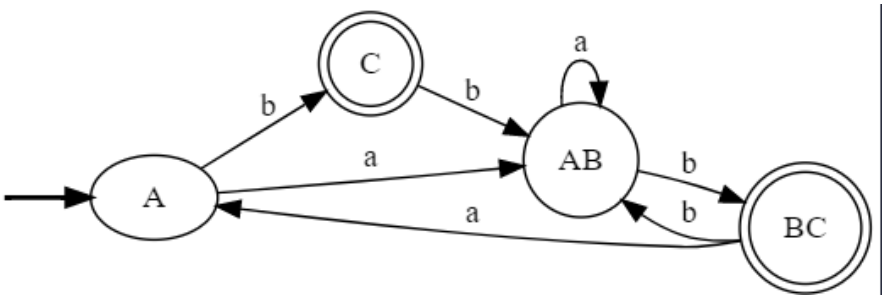


Respuesta Grafica

Prueba 5

```
{
    'alphabet': ['a', 'b'],
    'states': ['A', "B", "C"],
    'initial_state': 'A',
    'accepting_states': ['C'],
    'transitions': [
        ['A', 'a', 'A'],
        ['A', 'a', 'B'],
        ['A', 'b', 'C'],
        ['B', 'a', 'A'],
        ['B', 'b', 'B'],
        ['C', 'b', 'A'],
        ['C', 'b', 'B'],
    ]
}
```

Formato JSON

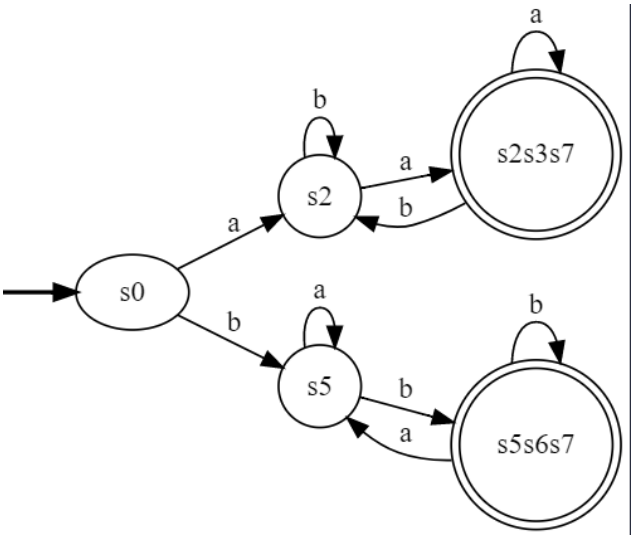


Respuesta Grafica

2.3 Evaluador DFA

En esta fase de pruebas se adjuntará una imagen de un autómata dfa en una gráfica y en la parte inferior de la misma se mostrará una gráfica con la imagen del resultado de la expresión ingresada es decir si es parte o no del lenguaje del autómata. La expresión ingresada por el usuario se denotará con la palabra “regex”.

Prueba 1

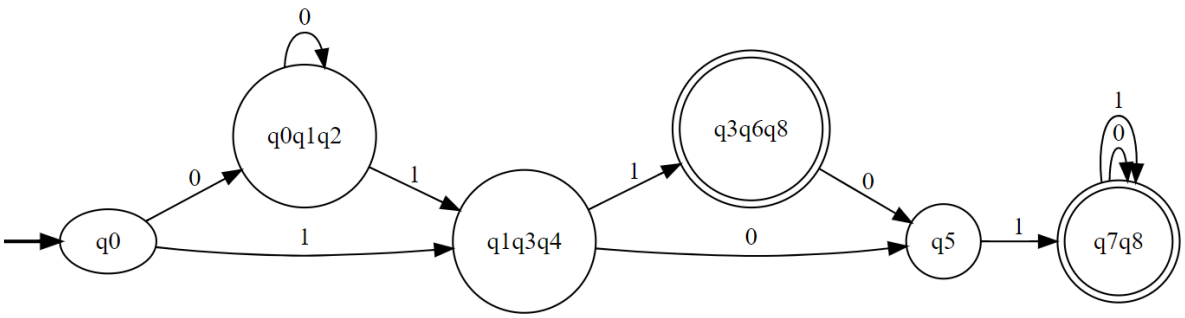


Grafica

Regex	Resultado	Tiempo (segundos)
abbaaaa	EVALUATE JSON --> s0 a s2 --> s2 b s2 --> s2 b s2 --> s2 a s2s3s7 --> s2s3s7 a s2s3s7 --> s2s3s7 a s2s3s7 --> s2s3s7 a s2s3s7 La expresion es parte del automata	0.369
ab	EVALUATE JSON --> s0 a s2 --> s2 b s2 La expresion no es parte del lenguaje del automata.	0.367
baaaaaaaaa	EVALUATE JSON --> s0 b s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 --> s5 a s5 La expresion no es parte del lenguaje del automata.	0.351
bbb	EVALUATE JSON --> s0 b s5 --> s5 b s5s6s7 --> s5s6s7 b s5s6s7 La expresion es parte del automata	0.341

babbbbba	EVALUATE JSON --> s0 b s5 --> s5 a s5 --> s5 b s5s6s7 --> s5s6s7 b s5s6s7 --> s5s6s7 b s5s6s7 --> s5s6s7 b s5s6s7 --> s5s6s7 a s5 La expresion no es parte del lenguaje del automata.	0.369

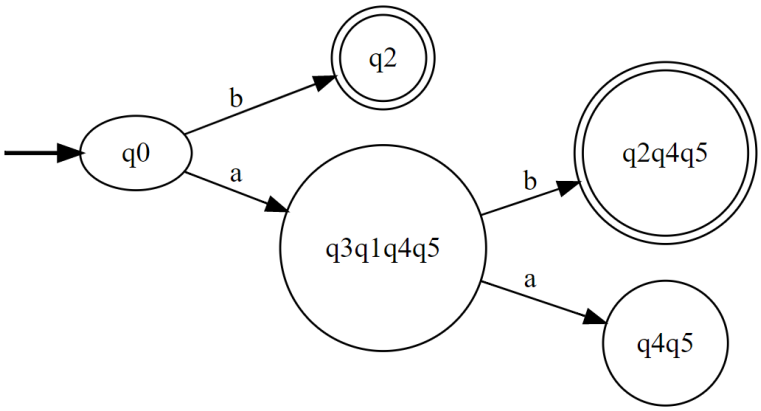
Prueba 2



Grafica

Regex	Resultado	Tiempo (segundos)
101	EVALUATE JSON --> q0 1 q1q3q4 --> q1q3q4 0 q5 --> q5 1 q7q8 La expresion es parte del automata	0.371
0000000	EVALUATE JSON --> q0 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 La expresion no es parte del lenguaje del automata.	0.366
111	EVALUATE JSON --> q0 1 q1q3q4 --> q1q3q4 1 q3q6q8 --> q3q6q8 1 Ø La expresion no es parte del lenguaje del automata.	0.362
00011010	EVALUATE JSON --> q0 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 0 q0q1q2 --> q0q1q2 1 q1q3q4 --> q1q3q4 1 q3q6q8 --> q3q6q8 0 q5 --> q5 1 q7q8 --> q7q8 0 q7q8 La expresion es parte del automata	0.366
110	EVALUATE JSON --> q0 1 q1q3q4 --> q1q3q4 1 q3q6q8 --> q3q6q8 0 q5 La expresion no es parte del lenguaje del automata.	0.374

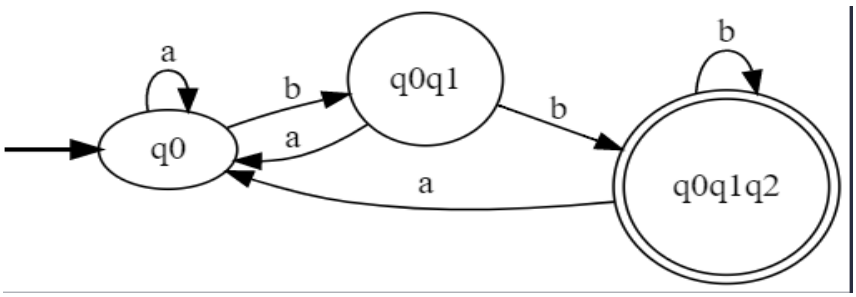
Prueba 3



Grafica

Regex	Resultado	Tiempo (segundos)
b	EVALUATE JSON --> q0 b q2 La expresion es parte del automata	0.330
aa	EVALUATE JSON --> q0 a q3q1q4q5 --> q3q1q4q5 a q4q5 La expresion no es parte del lenguaje del automata.	0.332
ab	EVALUATE JSON --> q0 a q3q1q4q5 --> q3q1q4q5 b q2q4q5 La expresion es parte del automata	0.331
001	EVALUATE JSON La expresion no es parte del lenguaje del automata.	0.265
aba	EVALUATE JSON --> q0 a q3q1q4q5 --> q3q1q4q5 b q2q4q5 --> q2q4q5 a Ø La expresion no es parte del lenguaje del automata.	0.333

Prueba 4

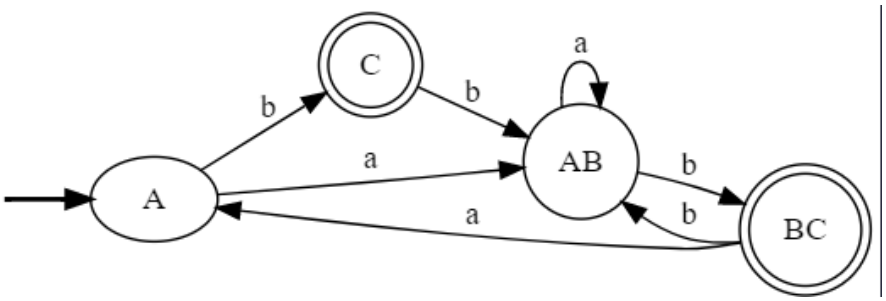


Grafica

Regex	Resultado	Tiempo (segundos)
abbbbbb	EVALUATE JSON --> q0 a q0 --> q0 b q0q1 --> q0q1 b q0q1q2 --> q0q1q2 b q0q1q2 --> q0q1q2 b q0q1q2 --> q0q1q2 b q0q1q2 --> q0q1q2 b q0q1q2 La expresion es parte del automata	0.366
ababab	EVALUATE JSON --> q0 a q0 --> q0 b q0q1 --> q0q1 a q0 --> q0 b q0q1 --> q0q1 a q0 --> q0 b q0q1 La expresion no es parte del lenguaje del automata.	0.364
abbaa	EVALUATE JSON --> q0 a q0 --> q0 b q0q1 --> q0q1 b q0q1q2 --> q0q1q2 a q0 --> q0 a q0 La expresion no es parte del lenguaje del automata.	0.367
b	EVALUATE JSON --> q0 b q0q1 La expresion no es parte del lenguaje del automata.	0.261

ab	EVALUATE JSON --> q0 a q0 --> q0 b q0q1 La expresion no es parte del lenguaje del automata.	0.267

Prueba 5



Grafica

Regex	Resultado	Tiempo (segundos)
b	EVALUATE JSON --> A b C La expresion es parte del automata	0.363
bbb	EVALUATE JSON --> A b C --> C b AB --> AB b BC La expresion es parte del automata	0.369

aaaaaaba	EVALUATE JSON --> A a AB --> AB a AB --> AB a AB --> AB a AB --> AB a AB --> AB a AB --> AB b BC --> BC a A La expresion no es parte del lenguaje del automata.	0.372
bbaaaa	EVALUATE JSON --> A b C --> C b AB --> AB a AB --> AB a AB --> AB a AB --> AB a AB La expresion no es parte del lenguaje del automata.	0.367
ab	EVALUATE JSON --> A a AB --> AB b BC La expresion es parte del automata	0.360

2.3.1 Tiempo promedio

En la siguiente grafica se muestra el tiempo promedio que se obtuvo de cada prueba al momento de evaluar una expresión regex.

Conclusion

En cuanto a lo abordado con anterioridad, el proyecto tenía como objetivo cumplir los requerimientos previamente mencionados en la introducción. Y atreves de una serie de clases y métodos desarrollados en el lenguaje de Python se fue posible cumplir tales requerimientos es posible decir que el proyecto ha cumplido su meta basado en los resultados observados previamente. En donde se realice una serie de pruebas a cada una de las clases y sus funciones para verificar que sus resultados fueran consistentes al momento de probar creando distintos autómatas y expresiones. En cuanto al tiempo al momento de evaluar las expresiones los resultados demostraron que el tiempo depende del alfabeto (es el símbolo forma parte del alfabeto del autómata) y la cantidad de caracteres a evaluar dentro del autómata.