Aplicación de un AFD

Para la resolución de problemas en la vida real

Cuatrimestre	7mo
Grupo	A
Apellidos	SOLIS LOPEZ
Nombre	Marisol
Matricula	203411
Corte	1
Actividad	1
Repositorio GitHub	https://github.com/203411/lenguajes-y-automatas

Contenido

Tabla de ilustraciones

Introducción

Un autómata finito determinista (AFD) se puede definir como un autómata reconocedor, este se limita a aceptar o no una determinada cadena recibida en la entrada, por lo tanto, se puede decir que la salida de los mismos solo tendrá dos valores posibles: aceptar o no la cadena de entrada.

Estos autómatas transitarán en un conjunto finito de estados posibles, a medida que reciban sucesivamente los caracteres de entrada, en un instante determinado de tiempo el autómata solo podrá estar en uno y solo uno de los estados posibles. La principal característica de este tipo de autómatas es el determinismo, lo que significa que estando en un estado y recibiendo una entrada del exterior el autómata tendrá la posibilidad de transitar a uno y solo un estado del conjunto de estados posibles.

Los autómatas finitos deterministas quedan definidos formalmente mediante una quíntupla:

$$AFD = (\Sigma , Q, q_0, F, f)$$

Donde:

Σ	Alfabeto de símbolos de entrada.
Q	Conjunto finito de estados
q_0	$q_0 \in Q$ – estado inicial previsto
F	$F \subseteq Q$ - es el conjunto de estado finales de aceptación.
f	Función de transición de estados definida como $f: Q \times \Sigma \longrightarrow Q$

Este autómata recibirá una cadena en la cinta de entrada e irá procesando de uno a la vez los símbolos de entrada. Comenzará su funcionamiento posicionado en el estado inicial, y desde este estado comenzará su ejecución. En cada intervalo de tiempo discreto realizará dos acciones las cuales serán consideradas como acciones indivisibles en un determinado intervalo de tiempo.

Las acciones que realiza son:

- Leer el próximo símbolo, desde la cinta de entrada.
- Transitar, cambiar de estado

Una palabra será aceptada por un AFD si estando formada por símbolos pertenecientes al alfabeto de entrada, al finalizar de procesar la misma, el autómata queda posicionado en uno de los estados perteneciente al conjunto de estados finales de aceptación.

Un lenguaje reconocido por un AFD es el conjunto de todas las palabras reconocidas por el autómata finito determinista.

Problemática

La dirección MAC es un identificador que los fabricantes asignan a una tarjeta o dispositivo que se conecta a la red y está formada por 48 bits, representados por dígitos hexadecimales. Cada dirección MAC incluye seis pares de números, los primeros 3 sirven para identificar al fabricante y los siguientes 3 al modelo en concreto.

Basándose en la implementación en la carrera de Ingeniería en Desarrollo de Software de administrar el uso de internet mediante las direcciones MAC de los dispositivos de los alumnos de esta carrera, surgió la idea de realiza un autómata que se encargue de identificar si una dirección MAC es correcta o está dentro de los parámetros que se consideran correctos para posteriormente buscar en un archivo de datos XLSX si se encuentra asignada a un alumno, dicho archivo contará con matrículas, nombres, apellidos, direcciones MAC y consumo de los alumnos, en caso de encontrar la dirección MAC mostrará la matricula, nombre y consumo que tiene..

El autómata debe verificar que la dirección MAC ingresada cumpla con una estructura como el siguiente ejemplo: “00:FA:1F:32:BC”, es decir, solo aceptará 6 pares de dígitos hexadecimales cuyas letras deben estar en mayúsculas y separado por dos puntos.

La simbología utilizada para representar el modelado es la siguiente:

Modelado

A continuación, se presenta el modelado que se utilizó para darle solución a nuestra problemática, donde se puede observar que el autómata se limita a recibir dos entradas de caracteres hexadecimales y posteriormente dos puntos y finaliza recibiendo dos caracteres hexadecimales sin dos puntos al final. Presenta también el estado inicial, los estándares y el estado final.

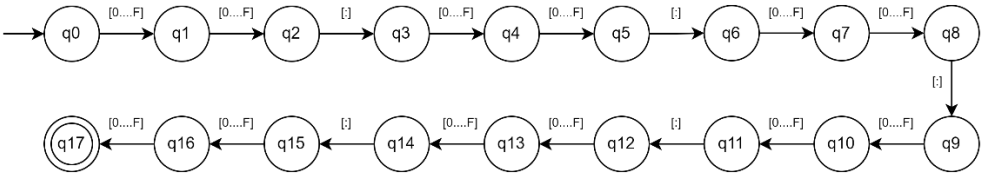

Ilustración 1 Modelado AFD

Descripción del autómata

$\sum: \{'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F', ':'\}$

$Q: \{'Q0', 'Q1', 'Q2', 'Q3', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13', 'Q14', 'Q15', 'Q16', 'Q17'\}$,

$q0: \{Q0\}$

$F: \{Q17\}$

δ :{'Q0','0'):'Q1', ('Q0','1'):'Q1', ('Q0','2'):'Q1', ('Q0','3'):'Q1', ('Q0','4'):'Q1', ('Q0','5'):'Q1', ('Q0','6'):'Q1', ('Q0','7'):'Q1',
('Q0','8'):'Q1',('Q0','9'):'Q1', ('Q0','A'):'Q1',('Q0','B'):'Q1', ('Q0','C'):'Q1', ('Q0','D'):'Q1', ('Q0','E'):'Q1', ('Q0','F'):'Q1',('Q1','0'):'Q2',
('Q1','1'):'Q2', ('Q1','2'):'Q2', ('Q1','4'):'Q2', ('Q1','3'):'Q2',('Q1','5'):'Q2',('Q1','6'):'Q2',('Q1','7'):'Q2', ('Q1','8'):'Q2', ('Q1','9'):'Q2', ('Q1','A'):'Q2',
('Q1','B'):'Q2', ('Q1','C'):'Q2', ('Q1','D'):'Q2',('Q1','E'):'Q2', ('Q1','F'):'Q2', ('Q2',':'):'Q3', ('Q3','0'):'Q4', ('Q3','1'):'Q4', ('Q3','2'):'Q4',
('Q3','3'):'Q4', ('Q3','4'):'Q4',('Q3','5'):'Q4', ('Q3','6'):'Q4', ('Q3','7'):'Q4', ('Q3','8'):'Q4', ('Q3','9'):'Q4', ('Q3','A'):'Q4', ('Q3','B'):'Q4',
('Q3','C'):'Q4', ('Q3','D'):'Q4', ('Q3','E'):'Q4', ('Q3','F'):'Q4', ('Q4','0'):'Q5', ('Q4','1'):'Q5', ('Q4','2'):'Q5', ('Q4','4'):'Q5', ('Q4','3'):'Q5',
('Q4','5'):'Q5', ('Q4','6'):'Q5', ('Q4','7'):'Q5', ('Q4','8'):'Q5', ('Q4','9'):'Q5', ('Q4','A'):'Q5', ('Q4','B'):'Q5', 'Q4','C'):'Q5', ('Q4','D'):'Q5',
('Q4','E'):'Q5', ('Q4','F'):'Q5',('Q5',':'):'Q6', ('Q6','0'):'Q7', ('Q6','1'):'Q7', ('Q6','2'):'Q7', ('Q6','3'):'Q7', ('Q6','4'):'Q7', ('Q6','5'):'Q7',
('Q6','6'):'Q7', ('Q6','7'):'Q7', ('Q6','8'):'Q7', ('Q6','9'):'Q7', ('Q6','A'):'Q7', ('Q6','B'):'Q7', ('Q6','C'):'Q7', 'Q6','D'):'Q7', ('Q6','E'):'Q7',
('Q6','F'):'Q7',('Q7','0'):'Q8', ('Q7','1'):'Q8', ('Q7','2'):'Q8', ('Q7','4'):'Q8', ('Q7','3'):'Q8', ('Q7','5'):'Q8', ('Q7','6'):'Q8', ('Q7','7'):'Q8',
('Q7','8'):'Q8', ('Q7','9'):'Q8', ('Q7','A'):'Q8', ('Q7','B'):'Q8', ('Q7','C'):'Q8', ('Q7','D'):'Q8', ('Q7','E'):'Q8', 'Q7','F'):'Q8', ('Q8',':'):'Q9',
('Q9','0'):'Q10', ('Q9','1'):'Q10', ('Q9','2'):'Q10', ('Q9','3'):'Q10', ('Q9','4'):'Q10', ('Q9','5'):'Q10', ('Q9','6'):'Q10', ('Q9','7'):'Q10',
('Q9','8'):'Q10', ('Q9','9'):'Q10', ('Q9','A'):'Q10', ('Q9','B'):'Q10', ('Q9','C'):'Q10', ('Q9','D'):'Q10', ('Q9','E'):'Q10', ('Q9','F'):'Q10',
('Q10','0'):'Q11', ('Q10','1'):'Q11', ('Q10','2'):'Q11', ('Q10','3'):'Q11', ('Q10','4'):'Q11', ('Q10','5'):'Q11', ('Q10','6'):'Q11', ('Q10','7'):'Q11',
('Q10','8'):'Q11', ('Q10','9'):'Q11', ('Q10','A'):'Q11', ('Q10','B'):'Q11', ('Q10','C'):'Q11', ('Q10','D'):'Q11', ('Q10','E'):'Q11', ('Q10','F'):'Q11',
('Q11',':'):'Q12',('Q12','0'):'Q13', ('Q12','1'):'Q13', ('Q12','2'):'Q13', ('Q12','3'):'Q13', ('Q12','4'):'Q13', ('Q12','5'):'Q13', ('Q12','6'):'Q13',
('Q12','7'):'Q13', ('Q12','8'):'Q13', ('Q12','9'):'Q13', ('Q12','A'):'Q13', ('Q12','B'):'Q13', ('Q12','C'):'Q13', ('Q12','D'):'Q13', ('Q12','E'):'Q13',
('Q12','F'):'Q13', ('Q13','0'):'Q14', ('Q13','1'):'Q14', ('Q13','2'):'Q14', ('Q13','4'):'Q14', ('Q13','3'):'Q14', ('Q13','5'):'Q14', ('Q13','6'):'Q14',
('Q13','7'):'Q14', ('Q13','8'):'Q14', ('Q13','9'):'Q14', ('Q13','A'):'Q14', ('Q13','B'):'Q14', ('Q13','C'):'Q14', ('Q13','D'):'Q14', ('Q13','E'):'Q14',
('Q13','F'):'Q14', ('Q14',':'):'Q15',('Q15','0'):'Q16', ('Q15','1'):'Q16', ('Q15','2'):'Q16', ('Q15','3'):'Q16', ('Q15','4'):'Q16', ('Q15','5'):'Q16',
('Q15','6'):'Q16', ('Q15','7'):'Q16', ('Q15','8'):'Q16', ('Q15','9'):'Q16', ('Q15','A'):'Q16', ('Q15','B'):'Q16', ('Q15','C'):'Q16', ('Q15','D'):'Q16',
('Q15','E'):'Q16', ('Q15','F'):'Q16', ('Q16','0'):'Q17', ('Q16','1'):'Q17', ('Q16','2'):'Q17', ('Q16','3'):'Q17', ('Q16','4'):'Q17', ('Q16','5'):'Q17',
('Q16','6'):'Q17', ('Q16','7'):'Q17', ('Q16','8'):'Q17', ('Q16','9'):'Q17', ('Q16','A'):'Q17', ('Q16','B'):'Q17', ('Q16','C'):'Q17', ('Q16','D'):'Q17',
('Q16','E'):'Q17', ('Q16','F'):'Q17'}

Implementación

Para el desarrollo de este proyecto se utilizó Python como lenguaje de programación en su versión 3.9.13 debido a la facilidad de uso de sus métodos y funciones, en un principio el programa mostraba los resultados de la implementación por consola, pero se modificó para que a través de una sencilla interfaz realizada en QT Designer en su versión 5.15 muestre los resultados de una manera más amigable a la vista.

El programa funciona con dos entradas diferentes en formato .txt, uno de ellos llamado "automata.txt" que contiene la estructura del AFD (todos los estados posibles, alfabeto, transiciones, estado inicial y estado final), este archivo es seleccionado mediante un botón llamado "seleccionar autómata" (el nombre del archivo puede variar) y leído por una función que lo convierte en un diccionario, y de esta forma el algoritmo puede usarlo de manera más sencilla. El otro archivo lleva por nombre "test.txt" (este debe llamarse obligatoriamente "test.txt" ya que existe una función que valida dicho nombre) es seleccionado mediante otro botón, este contiene las cadenas que deseamos evaluar con el autómata para validar si son aceptadas o no por el autómata, este archivo ya contiene alrededor de 200 cadenas (direcciones MAC) para ser evaluadas por el autómata, este archivo es leído de la misma forma que el anterior para su uso en el programa.

Respecto al algoritmo utilizado para evaluar las cadenas, se crearon diversas funciones que cumplen con una determinada tarea, como lo es el recorrido del AFD, encontrar si un carácter de la cadena forma parte de nuestro alfabeto o si cumple con la transición hacia un estado determinado y finalmente valida si las cadenas cumplen con esas condiciones.

A través de un ciclo determinado por la cantidad de cadenas en el archivo "test.txt" se realiza todo el proceso anteriormente mencionado, si la cadena es aceptada por el programa, es decir, cumple con la estructura de una dirección MAC, entonces el siguiente paso es buscar dicha cadena dentro del archivo de datos provisional que tenemos. Este archivo .xlsx es leído por una función diferente a la que lee el archivo .txt, debido a que son formatos diferentes, leemos el archivo antes de ingresar al ciclo. Se creó una función específica que se encarga de recorrer todos los datos que tenemos almacenados en el archivo .xlsx para compararlos con la dirección MAC que fue aceptada por el autómata, si es encontrada, muestra la matrícula del alumno a quien pertenece, el nombre y el consumo de este. En caso de que no sea encontrada muestra en consola un mensaje que notifica que la dirección MAC no se encuentra entre los datos. En caso de que la cadena no cumpla con la estructura de una dirección MAC informa que fue rechazada y omite todo el proceso anterior de búsqueda.

Los resultados de aceptación o rechazo de la dirección MAC se muestran en un Widget de Lista de QT Designer.

```python
import ast
import pylightxl as xl
import random

def read_file_xlsx(filename):
    data = []
    try:
        db = xl.readxl(fn=filename)
        for row in db.ws(ws='bd').rows:
            data.append([row[0], row[1], row[2], row[3], row[4]])
        return data
    except Exception as e:
        print(f"El archivo '{filename}' no existe.")

def read_file_txt(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as file:
            if "test" in filename:
                content = file.read().split('\n')
                return content
            else:
                content = file.read()
        DFA = ast.literal_eval(content)
        return DFA
    except Exception as e:
        print(f"El archivo '{filename}' no existe.")
```

Ilustración 2 Funciones de lectura

```python
def step_dfa(D,q,a):
    try:
        assert(a in D["Sigma"])
        assert(q in D["Q"])
        return D["Delta"][(q,a)]
    except:
        return False

def run_dfa(D,w):
    curstate = D["q0"]
    if w == "":
        return curstate
    else:
        return run_dfa_h(D,w[1:], step_dfa(D,curstate,w[0]))

def run_dfa_h(D,w,q):
    if w == "":
        return q
    else:
        return run_dfa_h(D,w[1:], step_dfa(D,q,w[0]))

def accepts_dfa(D,w):
    return run_dfa(D,w) in D["F"]

def search(mac, database):
    for i in range(0,len(database)):
        if str(mac) == str(database[i][3]):
            return (f"Encontrado -> Matricula: {database[i][0]} Nombre: {database[i][1]} Consumo: {database[i][4]}")
    return "MAC no encontrada en la base de datos"
```

Ilustración 3 Algoritmo de recorrido de AFD y búsqueda de MAC

```python
if __name__ == "__main__":
    D = read_file_txt('automata.txt')
    testing_macs = read_file_txt('test.txt')
    database = read_file_xlsx('archivo.xlsx')
    i = 1
    # print(database)
    for mac in testing_macs:
        if accepts_dfa(D,mac):
            consumo = search(mac, database)
            print(f"ACEPTADO {i}  ({mac}) {consumo}")
        else:
            print(f"RECHAZADO {i} ({mac})")
        i+=1
```

Ilustración 4 Llamada a las funciones anteriores

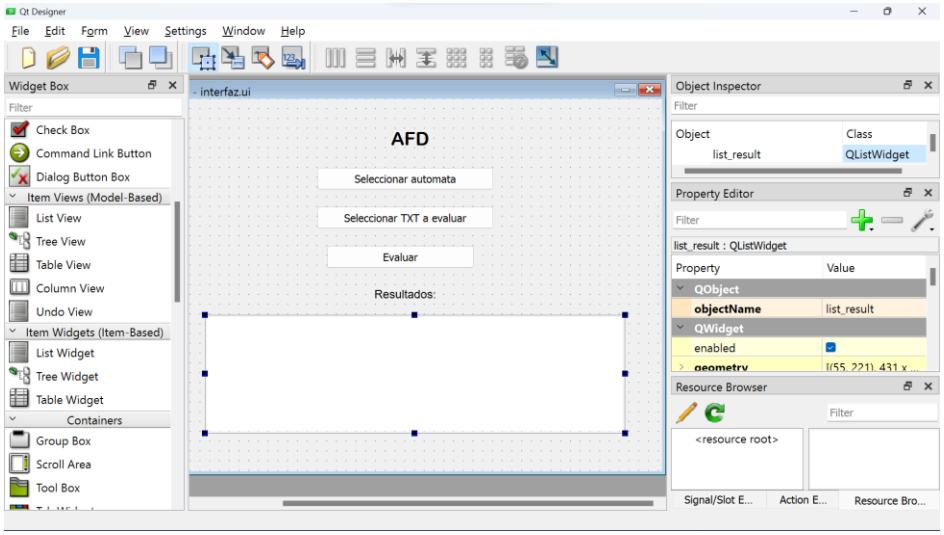

Ilustración 5 Diseño de la interfaz

Resultados

En un principio era el usuario quien ingresaba las direcciones MAC a probar, después de que se implementara la función para leer el archivo .txt, el programa lee las cadenas desde ese archivo y las valida según el autómata leído del archivo "automata.txt". Las primeras pruebas, que se realizaron mediante consola, arrojaron los siguientes resultados:

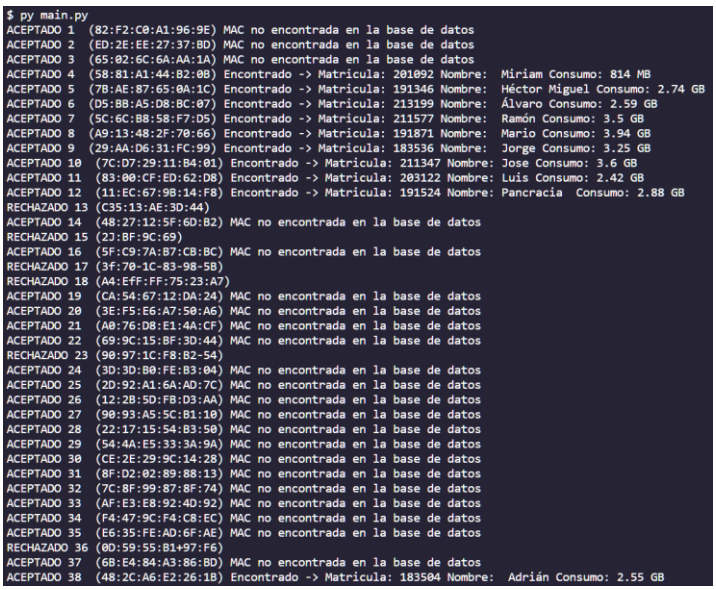

Ilustración 6 Resultados en consola

En consola obteniamos el estado de validación, ya sea RECHAZADO o ACEPTADO, si es aceptado entonces procede a buscar la dirección MAC en el archivo de datos. Posteriormente se implementó la interfaz y los resultados se muestran de la siguiente manera:

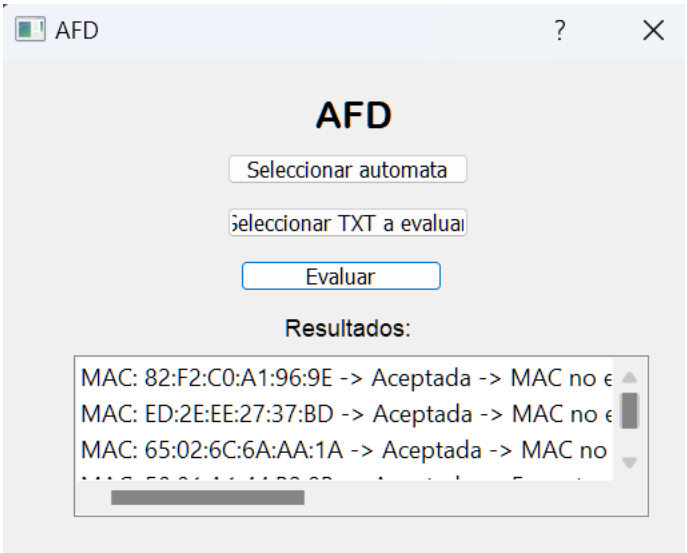

Ilustración 7 Resultado en interfaz

Discusión

Durante todo el proceso de desarrollo del programa y su ejecución se observo el comportamiento del autómata que se modeló, que consistía en verificar y validar que una cadena ingresada sea correcta según la estructura de una dirección MAC. Al principio se hicieron pruebas con datos ingresados de forma manual, por lo que el avance del programa se sentía lento, una vez implementada la función para leer los datos, fue más fluido todo el proceso.

Gracias a que se basó en el libro que referenciamos y al uso del lenguaje Python, fue más sencillo el desarrollo de nuestro autómata. También a los conceptos ya definidos y el apoyo de los distintos símbolos fue posible crear el grafo que sostendría el funcionamiento del autómata para este caso, lo cual hizo más simple el desarrollo del código y su implementación.

Debido a que no se conocía una herramienta para diseñar interfaces gráficas para Python, se dedicó mucho tiempo en investigación sobre el uso de estás, en un principio se usaría Tkinter para el desarrollo de la interfaz, pero a causa de complicaciones al combinarlo con lo que ya se tenia realizado se decidió usar QT Designer, siendo esta una herramienta más sencilla de usar y de agregar a lo ya funcional.

Conclusión

Como conclusión de esta actividad se obtiene una definición más clara de lo que es un autómata y su uso, el autómata es la primera máquina con lenguaje, es decir, un calculador lógico cuyo juego de instrucciones se orienta hacia los sistemas de evolución secuencial. Dada la información recaudad fue posible implementar un autómata para resolver un problema, en este caso, un sistema que reconozca la estructura de una dirección MAC y darle una aplicación en la vida real.

Además de lo anterior, se adquirieron nuevos conocimientos sobre herramientas que ayudaran en futuros proyectos, así como la experiencia adquirida en el uso e implementación de autómatas en código, el uso e importancia de las tablas de transición al crear y recorrer un autómata y la implementación de una interfaz gráfica con QT Designer.

Referencias Bibliográficas

Ganesh Gopalakrishnan, Automata and computability: programmer's perspective. Boca Raton, Fl: Crc Press, Taylor & Francis Group, 2019.

J. E. Hopcroft, Rajeev Motwani, and J. D. Ullman, Introducción a la teoría de autómatas, lenguajes y computación (3a. ed.). Madrid: Pearson Educación, 2007

Tablado, F. (2022, 26 mayo). Dirección MAC: Definición y funciones. Grupo Atico34. Recuperado 27 de septiembre de 2022, de https://protecciondatos-lopd.com/empresas/direccion-mac/#:%7E:text=La%20direcci%C3%B3n%20MAC%20es%20un,OUI%20(Identificador%20%C3%9Anico%20Organizacional).

Real Python. (2021, 22 enero). Qt Designer and Python: Build Your GUI Applications Faster. Recuperado 1 de octubre de 2022, de https://realpython.com/qt-designer-python/#installing-and-running-qt-designer