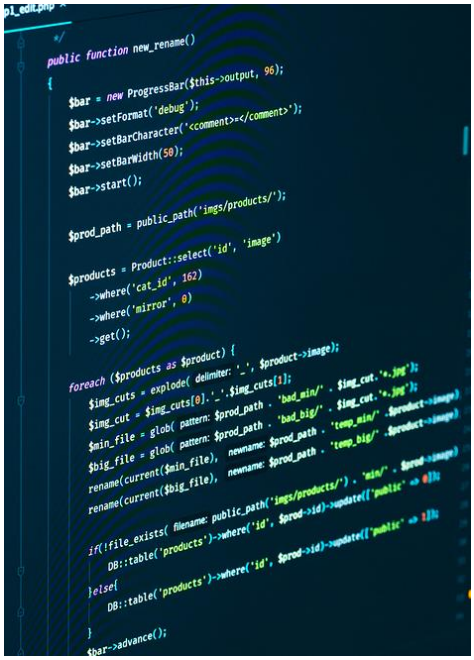
Instituto Politécnico Nacional

Escuela Superior de Cómputo



PRÁCTICA 1

Generador de Autómatas Finitos

COMPILADORES – 3CV7

El estudiante pondrá en práctica sus conocimientos de los temas vistos en clase de autómatas finitos.

Oscar Eduardo López Cabagné

2015070715

Índice

Introducción

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. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

La finalidad de los autómatas finitos es la de reconocer lenguajes regulares, que corresponden a los lenguajes formales más simples según la Jerarquía de Chomsky.

Formalmente, un autómata finito es una 5-tupla (Q, Σ, q0, δ, F) donde:

- Q, es un conjunto finito de estados;
- Σ, es un alfabeto finito;
- q0 in Q es el estado inicial;
- δ: Q x Σ -> Q, es una función de transición;
- F ⊆ o Q es un conjunto de estados finales o de aceptación.

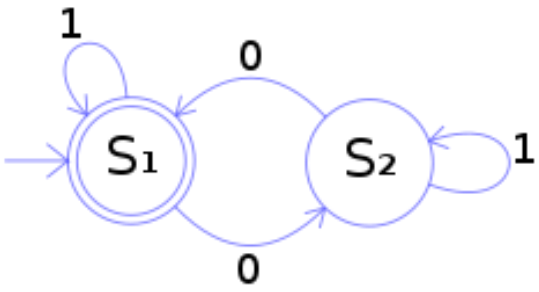
La Jerarquía de Chomsky consta de cuatro niveles:

- **Gramáticas de tipo 0 (sin restricciones),** que incluye a todas las gramáticas formales. Estas gramáticas generan todos los lenguajes capaces de ser reconocidos por una máquina de Turing. Los lenguajes son conocidos como lenguajes recursivamente enumerables. Nótese que esta categoría es diferente de la de los lenguajes recursivos, cuya decisión puede ser realizada por una máquina de Turing que se detenga.

- **Gramáticas de tipo 1 (gramáticas sensibles al contexto)**, generan los lenguajes sensibles al contexto. Los lenguajes descritos por estas gramáticas son exactamente todos aquellos lenguajes reconocidos por una máquina de Turing determinista cuya cinta de memoria está acotada por un cierto número entero de veces sobre la longitud de entrada, también conocidas como autómatas linealmente acotados.

- **Gramáticas de tipo 2 (gramáticas libres del contexto),** generan los lenguajes independientes del contexto. Estos lenguajes son aquellos que pueden ser reconocidos por un autómata con pila.

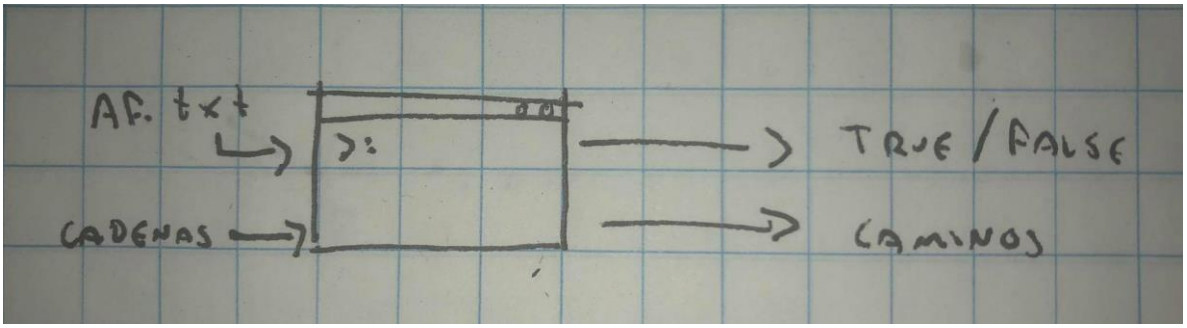
- **Gramáticas de tipo 3 (gramáticas regulares),** generan los lenguajes regulares. Estos lenguajes son aquellos que pueden ser aceptados por un autómata finito. También esta familia de lenguajes puede ser obtenidas por medio de expresiones regulares.



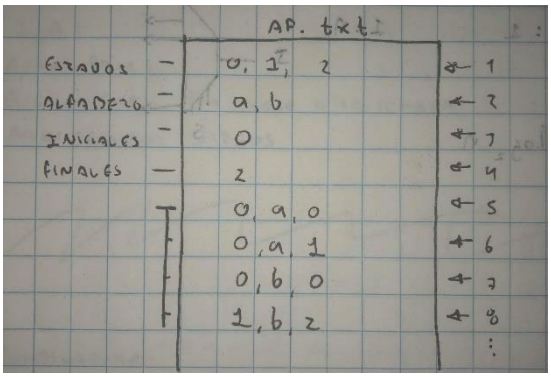
Ejemplo de un Autómata Finito

Desarrollo

Para desarrollar la práctica, comencé realizando un pequeño dibujo representativo del funcionamiento general final para no perder de vista el objetivo durante el desarrollo.



En este simplemente destaqué las dos entradas principales: el archivo de texto que contiene las especificaciones y la cadena que se desea analizar, a su vez, se pueden ver las posibles salidas: un mensaje de error (En caso de que la cadena no corresponda con el autómata) y un mensaje de análisis exitoso junto con los caminos posibles encontrados.



Adicional a este dibujo, realicé un pequeño esquema sobre el formato que debe contener el archivo de texto, de esta forma me resultaría más fácil consultarlo durante el desarrollo. Además, este me ayudaría a analizarlo y encontrar patrones que me ayuden a programar la fase del análisis de este.

Antes de programar, hice una lista seriada de los pasos que se deberían cumplir para realizar cada una de las tareas por separado. Encontré que algunos se podían englobar en tareas separadas, como el proceso encargado de abrir, leer y guardar los datos del archivo, el encargado de obtener la cadena y verificar que pertenezca al autómata descrito en el archivo y el proceso del análisis final. Podemos ver estas anotaciones en la siguiente imagen:

P.I

- ABRIR ARCHIVO *.txt
  - VERIFICAR QUE CONTENGA AL MENOS 5 RENGLONES (!)
  - VERIFICAR QUE CONTENGA AL MENOS 2 COLUMNAS
  - LEER Y ALMACENAR RENGLÓN 1, ESTADOS
  - LEER Y ALMACENAR RENGLÓN 2, ALFABETO
  - 3, E. INICIALES
  - ~~A PARTIR DEL RE~~ 4, E. FINALES
  - // A PARTIR DEL RENGLÓN 5 SE ENCUENTRAN LAS
  - // RELACIONES ENTRE ESTADOS Y TRANSICIONES
  - LEER Y ALMACENAR COMO MATRIZ LAS RELACIONES

RELACIONES =

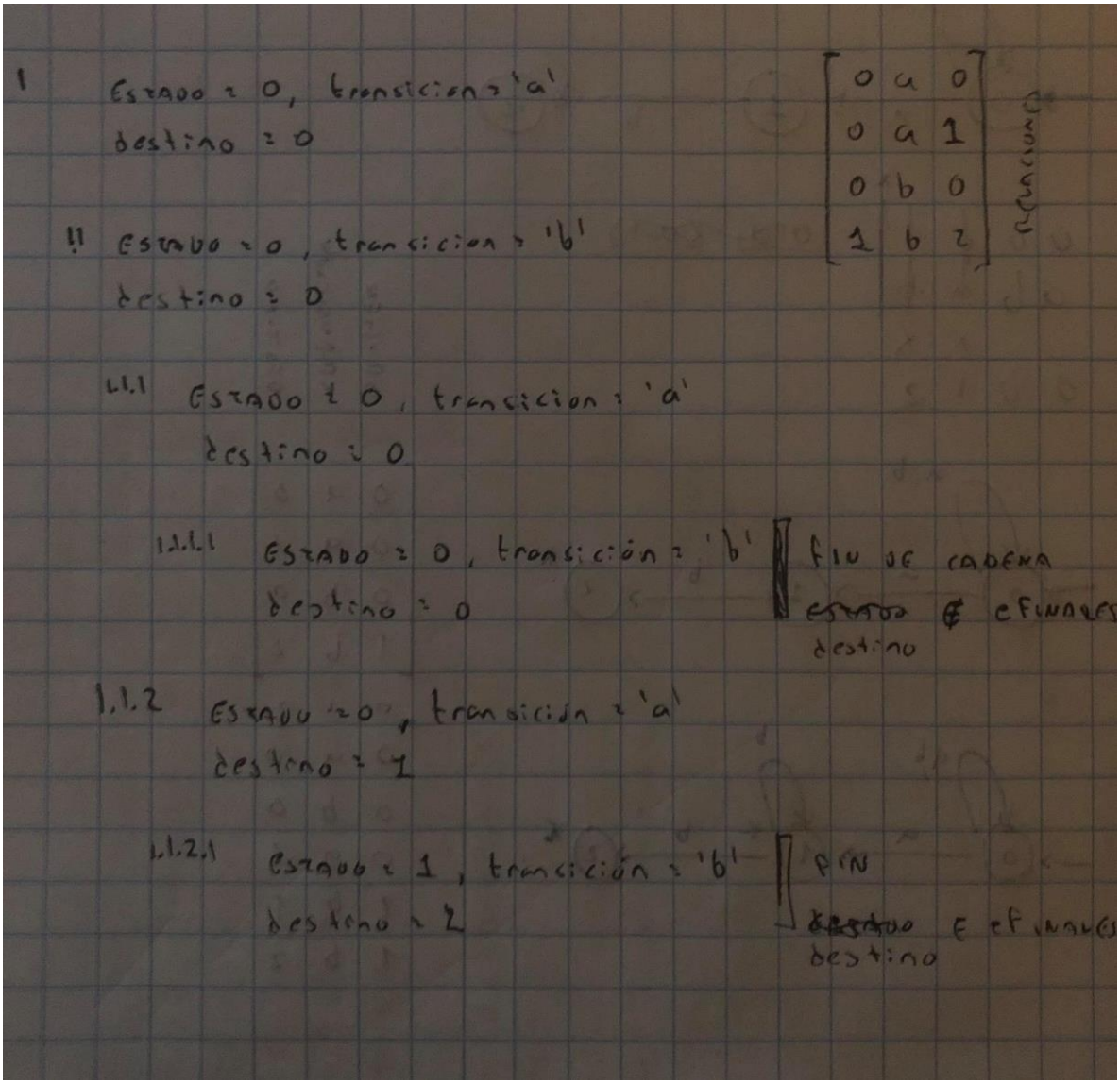
ESTADO ORIGEN	CARÁCTER DE TRANSICIÓN	ESTADO DESTINO

- OBTENER CADENA
  - VERIFICAR Q'S LOS CARACTERES DE LA CADENA PERTENECEN AL ALFABETO. (!)

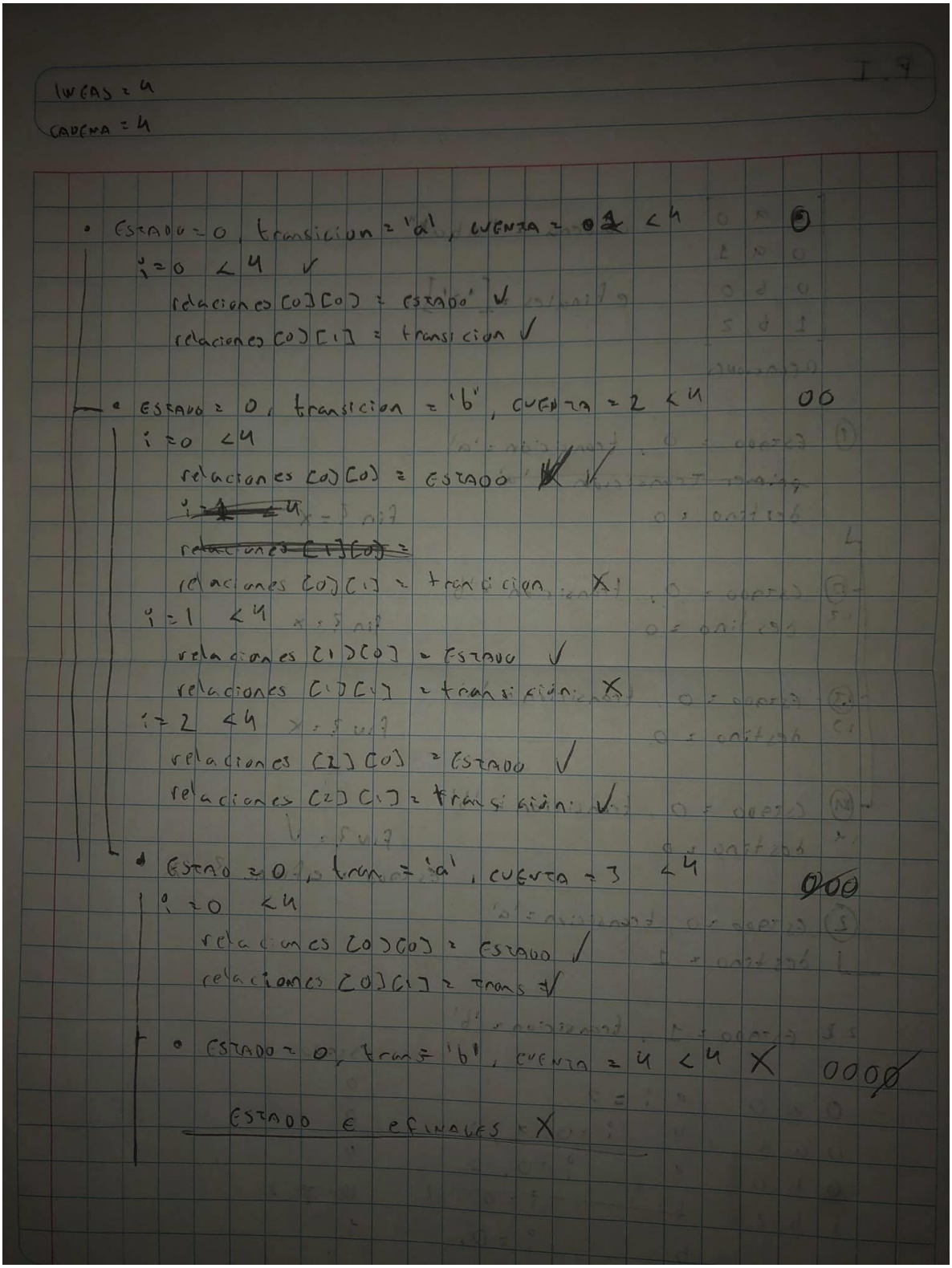
- ANÁLISIS
  - BUSCAR UNA TRANSICIÓN EN EL ESTADO ACTUAL QUE CUMPLA CON EL CARÁCTER DE LA CADENA.
  - ¿ SE ACABÓ LA CADENA ?
    - ✓ VERIFICAR QUE EL ESTADO ACTUAL SEA FINAL
    - X CAMINO INCORRECTO

Destaqué con signo de admiración a aquellos pasos en los que el programa podría terminar por algún motivo externo.

Finalmente, comencé a escribir y dibujar ideas sobre el algoritmo que deberá realizar el análisis. Fue en este momento que decidí realizarlo de manera recursiva, pues, el programa realizará el mismo análisis exacto para cada parte de la cadena, además que deberá revisar todas las combinaciones posibles.



Una vez obtenido un algoritmo, hice un seguimiento de este a mano sobre una cadena sencilla para verificar su funcionamiento antes de siquiera tocar el teclado.



Llegado a este punto, con el problema resuelto sobre papel, comencé la programación de la solución. Opté por utilizar Python, pues me permitirá concentrarme mejor en las tareas complicadas que en otras tareas, como el manejo de archivos.

- **Inicio**

Comencé programando una función que servirá de pantalla inicial, donde se solicita el path al archivo de texto que queramos usar. Este intentará abrir el archivo en el path indicado, de lograrlo, revisará que el archivo contenga información suficiente, ejecutará la función que obtendrá los datos de este y luego nos solicitará la cadena a analizar. Finalmente iniciará la función de análisis.

En caso de ocurrir un error en alguno de los pasos, se desplegará el mensaje de error correspondiente y el programa finalizará.

```
94
95  def inicio():
96      print(">: Introduzca el path: ")
97      path1 = input()
98      camino = ""
99      try:
100         file = open(path1, "r")
101         print(">: El path es correcto")
102         file.close()
103         rows = renglones(path1)
104         if rows < 5:
105             print(">: Hay un problema con el archivo. Informacion Insuficiente.")
106         else:
107             print(">: Cargando archivo.")
108             leer(path1)
109             print(">: Archivo cargado correctamente.")
110             obtenerCadena()
111             analisis(eIniciales[0], 0, 1, camino)
112     except:
113         print(">: Ha ocurrido un error. Verifique el path. ")
114
115 inicio()
```

Código de la función inicio

- **Renglones**

Esta función únicamente está destinada a obtener el número de renglones que contiene el archivo especificado. Es importante revisar esto, pues, de acuerdo a la estructura del archivo, las relaciones de estado-transición-estado se encuentran a partir del renglón número 5, por lo que, si el archivo cuenta con menos de 5 renglones, tendremos información insuficiente para realizar el análisis.

```
10
11  def renglones(path):
12      # Obtener número de renglones.
13      file = open(path, "r")
14      Counter = 0
15
16      # Leyendo Archivo
17      Content = file.read()
18      CoList = Content.split("\n")
19
20      for i in CoList:
21          if i:
22              Counter += 1
23
24      return Counter # Número de renglones leídos.
25
```

Código de la función renglones

- **Leer**

Aquí es donde se realiza la lectura del archivo y se obtienen los datos de este.

Para realizar esto, declaré una serie de variables globales para almacenar estos datos y darle acceso a estos a las demás funciones sin necesidad de pasarlos como parámetros.

En este punto, el path ya ha sido verifiifcado, por lo que comenzamos abriendo el archivo en modo de lectura y lo leemos renglón por renglón. De acuerdo al formato que el archivo debe contener, cada renglón pertenece a una única categoría de acuerdo a la siguiente relación:

Renglón 1	Estados
Renglón 2	Alfabeto
Renglón 3	Estados Iniciales
Renglón 4	Estados Finales
Renglón 5 y en adelante	Relación de transiciones

Por cada renglón leído, lo separamos por cada “,” y lo guardamos en forma de array en su variable correspondiente. En el caso de las líneas, además de guardarlo en formato de matriz, se llevará la cuenta de líneas escritas en este para ser utilizado más adelante.

```
def leer(path):
    global estados
    global alfabeto
    global eIniciales
    global eFinales
    global relaciones
    global lineas
    i = 0
    count = 0
    file = open(path, "r")
    while True:
        count += 1
        line = file.readline().rstrip()
        if count < 5:
            if count == 1:
                estados = line.split(",")
            if count == 2:
                alfabeto = line.split(",")
            if count == 3:
                eIniciales = line.split(",")
            if count == 4:
                eFinales = line.split(",")
        else:
            relaciones[i] = line.split(",")
            i += 1
            lineas += 1
        if not line:
            lineas -= 1
            break
```

Código de la función leer

- **Obtener Cadena**

Esta función, además de solicitar y obtener la cadena que se desea analizar, verificará que la misma pertenezca al autómata especificado por el archivo mucho antes de comenzar el análisis final.

Esto lo realiza verificando que la cadena esté completamente compuesta por caracteres que pertenezcan al alfabeto especificado en el archivo (Correspondiente al renglón 2). En el primer carácter que no pertenezca al alfabeto, se notificará el error y se detendrá.

Haciendo este análisis previo, podemos ahorrarnos algunos posibles errores futuros y agilizamos el funcionamiento de nuestro programa.

```
56  def obtenerCadena():
57      global cadena
58      i = 0
59      print("Introduzca la cadena: ")
60      cadena = input()
61      print(f">: {cadena}")
62          # Verificar que cada caracter de la cadena se encuentra dentro del alfabeto
63      for letra in cadena:
64          try:
65              alfabeto.index(letra)
66              i += 1
67              if i == len(cadena):
68                  print("Cadena valida.")
69          except:
70              print(f"{letra} NO se encuentra en el alfabeto")
71              break
72
```

Código de la función obtenerCadena

- **Análisis**

Es en esta función en la que finalmente ocurre el análisis de la cadena respecto al autómata especificado.

Aquí es donde se determinará si la cadena introducida puede ser producida por el autómata y, de ser así, cuáles son los caminos por los que puede ser producida.

Esta función es la más compleja, por lo que se explica directamente en los comentarios del programa para facilitar la visualización respecto al código.

```
72
73  def analisis(estado, posicion, cuenta, camino):
74      camino += estado                                        #Actualizamos la cadena del camino con el estado actual
75      if cuenta <= len(cadena):                               # Si la cuenta de caracteres es menor al total de caracteres en la cadena...
76          transicion = cadena[posicion]                       # Obtenemos el caracter siguiente de la cadena
77          i = 0
78          while i <= lineas:                                  # Mientras estemos dentro de la relación Estado-Transición-Estado...
79              if relaciones[i][0] == estado:                  # Si el elemento en la posición [i][0] (Primer columna) es igual al estado actual...
80                  if relaciones[i][1] == transicion:              # Si el elemento [i][1] (Segunda columna) es igual al siguiente caracter en la cadena...
81                      analisis(relaciones[i][2], posicion + 1, cuenta + 1, camino)    # Realizamos de nuevo este análisis, ahora con el estado destino (Tercera columna),
82                                                                  # incrementamos la posición y la cuenta, y enviamos la cadena del camino actualizada.
83              i += 1                                      # Incrementamos el contador del arreglo.
84
85      else:                                               # Si nuestra cuenta de caracteres es mayor al total de la cadena...
86          try:                                            # Verificamos que el último estado obtenido pertenezca a los estados finales (Renglón 4).
87              eFinales.index(estado)
88              print(f">: Camino exitoso: {camino}")       # Imprimimos el camino obtenido
89
90          except:                                         # Si el último estado no es final, entonces el camino es incorrecto.
91              print("")
92              #print(f"Camino fallido: {camino}")         # Podemos imprimir también los caminos fallidos que se encuentren.
93
94      return 0
```

Código de la función análisis

A continuación, se encuentran únicamente los comentarios de la función para facilitar su visualización:

```
#Actualizamos la cadena del camino con el estado actual
# Si la cuenta de caracteres es menor al total de caracteres en la cadena...
# Obtenemos el caracter siguiente de la cadena

# Mientras estemos dentro de la relación Estado-Transición-Estado...
# Si el elemento en la posición [i][0] (Primer columna) es igual al estado actual...
    # Si el elemento [i][1] (Segunda columna) es igual al siguiente caracter en la cadena...
    # Realizamos de nuevo este análisis, ahora con el estado destino (Tercera columna),
    # incrementamos la posición y la cuenta, y enviamos la cadena del camino actualizada.
# Incrementamos el contador del arreglo.

# Si nuestra cuenta de caracteres es mayor al total de la cadena...
# Verificamos que el último estado obtenido pertenezca a los estados finales (Renglón 4).

# Imprimimos el camino obtenido

# Si el último estado no es final, entonces el camino es incorrecto.

# Podemos imprimir también los caminos fallidos que se encuentren.
```

Pruebas

Una vez terminado, llegó el momento de realizar las pruebas.

Debemos recordar que el programa únicamente puede mostrar confirmaciones, mensajes de error y los caminos obtenidos. También hay que aclarar que los caminos por defecto comienzan en el estado inicial.

AF1.txt

- aab

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF1.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
aaab
>: aaab
Cadena valida.

>: Camino exitoso: 00012


PS C:\Users\Oscar>
```

- ababab

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF1.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
ababab
>: ababab
Cadena valida.

>: Camino exitoso: 0000012


PS C:\Users\Oscar>
```

AF2.txt

- ab

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF2.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
ab
>: ab
Cadena valida.
>: Camino exitoso: 123
>: Camino exitoso: 155
PS C:\Users\Oscar>
```

- aaab

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF2.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
aaab
>: aaab
Cadena valida.
>: Camino exitoso: 12223

PS C:\Users\Oscar>
```

- abab

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF2.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
abab
>: abab
Cadena valida.


PS C:\Users\Oscar>
```

En este caso no devolvió nada, lo que significa que la cadena no pudo ser reproducida por el autómata.

AF3.txt

- a.aa

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF3.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
a.aa
>: a.aa
Cadena valida.
>: Camino exitoso: 04333
PS C:\Users\Oscar>
```

- +a.a

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF3.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
+a.a
>: +a.a
Cadena valida.
>: Camino exitoso: 01123
>: Camino exitoso: 01433
PS C:\Users\Oscar>
```

- +aa.aa

```
PS C:\Users\Oscar> & C:/Users/Oscar/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Osc
ar/Desktop/Pruebas/vscode/prueba.py
>: Introduzca el path:
AF3.txt
>: El path es correcto
>: Cargando archivo.
>: Archivo cargado correctamente.
Introduzca la cadena:
+aa.aa
>: +aa.aa
Cadena valida.
>: Camino exitoso: 0111233
>: Camino exitoso: 0114333
PS C:\Users\Oscar>
```

Conclusión

Un autómata es un modelo que representa las relaciones que existen entre un estado y otro, el tipo de estado (normal, inicial o final) y las transiciones que hay entre ellos. Este conjunto de normas podrá determinar si una cadena pertenece al autómata, verificando si cumple con las características necesarias, tales como que sus caracteres pertenezcan al alfabeto o que finalice con una transición a un estado final.

Esta práctica me ayudó a recordar el concepto y funcionamiento de un autómata, además de practicar mi programación en Python.

Bibliografía

[1]J. Hopcroft, J. Ullman, R. Motwani and S. Vuelapluma, *Introducción a la teoría de autómatas, lenguajes y computación*. Madrid: Pearson Educación, 2010.

[2]J. Giró, *Lenguajes formales y teoría de autómatas*. [Barcelona]: Marcombo, 2015.

[3]E. Alfonseca Cubero, M. Alfonseca Moreno and R. Moriyón Salomón, *Teoría de autómatas y lenguajes formales*. Madrid: McGraw-Hill/Interamericana de España, 2007.