



INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

Programa protocolo

Teoría de la computación

Alumno:

Reyes Garnelo Uziel Bruno

Profesor: Genaro Juárez Martinez

Grupo: 5BM1

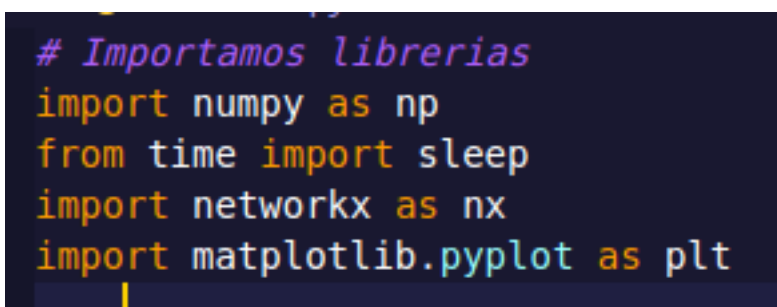
17 de abril de 2023

1. Introducción:

Dentro de la computación, una herramienta importante que se usa para implementar diferentes soluciones a diversos problemas son los autómatas. En este caso, se implementa un autómata capaz de determinar la imparidad de una cadena binaria sin importar su longitud.

2. Desarrollo:

Como primer paso, se utilizaron algunas librerías auxiliares que ayudaron al desarrollo, la optimización y la graficación del programa.



```
# Importamos librerías
import numpy as np
from time import sleep
import networkx as nx
import matplotlib.pyplot as plt
```

- **numpy:** Librería que proporciona estructuras de datos y funciones para el manejo de números optimizadas en lenguajes de bajo nivel.
- **time:** De la librería time, se usa solamente la función "sleep", que detiene la ejecución del programa en algunos segundos determinados.
- **networkx:** Librería que proporciona estructuras de datos para modelar grafos y en conjunto con matplotlib se puede crear una representación gráfica de estos mismos.
- **matplotlib:** Librería que proporciona herramientas de graficación.

Lo primero que se define en el programa es el autómata y su funcionamiento. En este caso, se usa una clase para modelar el autómata y poderlo usando usando creando un objeto de esta clase.

```
# Creamos el automata
class ADF:
    def __init__(self, cadena:list):
        self.cadena = cadena
        self.estado = 0

    def imparidad(self):
        for c in self.cadena:
            self.funcion_transicion(c)

        if self.estado == 0:
            return False
        else:
            return True

    def funcion_transicion(self, caracter):
        # Estado Q0
        if self.estado == 0:
            if caracter == 0:
                self.estado = 2
            else:
                self.estado = 1

        # Estado Q1
        elif self.estado == 1:
            if caracter == 0:
                self.estado = 3
            else:
                self.estado = 0

        # Estado Q2
        elif self.estado == 2:
            if caracter == 0:
                self.estado = 0
            else:
                self.estado = 3

        # Estado Q3
        elif self.estado == 3:
            if caracter == 0:
                self.estado = 1
            else:
                self.estado = 2

    def cadena_str(self):
        binaria = ''
        for c in self.cadena:
            binaria += str(c)
        return binaria
```

El nombre de la clase es ADF (Autómata Determinista Finito).

La primer función "`__init__`", es el constructor de la clase, los parámetros se recibe son: a si mismo y una lista que es la cadena a evaluar. El constructor define los atributos del objeto, en este caso, define el atributo «cadena» como la cadena recibida y el atributo «estado» directamente como "0".

La función «imparidad», recibe los atributos del mismo objeto para poder acceder a ellos. Esta función se encarga de recorrer la cadena a evaluar, llamando a la función de transición en cada iteración, enviando como parámetro el caracter actual. Posteriormente, se verifica el estado en el que terminó y se devuelve False si terminó en el estado "0", y True en cualquier otro caso.

La "función_transicion", se encarga de verificar y pasar al siguiente estado. Verificando el estado actual y posteriormente, el caracter recibido, así, se cubren todos los casos.

Finalmente, la función «cadena_str», se encarga de convertir la cadena de tipo de dato lista, a la misma cadena pero en tipo cadena, para poderla escribir en el archivo.

La siguiente función que se define es «ejecutarProtocolo».

```
def ejecutarProtocolo(generadas, aceptadas, rechazadas):
    # Generar 10^6 cadenas binarias aleatoriamente de longitud 64.
    no_cadenas = 1000000
    long_cadenas = 64
    cadenas = np.random.randint(2, size=(no_cadenas, long_cadenas))

    # Hacer que el programa espere 3 segundos.
    print('Iniciando protocolo. . .')
    sleep(3)
    print('Ejecutando protocolo. . .')

    for i in range(cadenas.shape[0]):
        automata = ADF(cadenas[i,:])
        cadena = automata.cadena_str()
        # Agregamos al archivo de las cadenas generadas
        generadas.write(cadena+'\n')

        # Agregamos cadena al archivo correspondiente
        if automata.imparidad():
            aceptadas.write(cadena+'\n')
        else:
            rechazadas.write(cadena+'\n')
```

La función «ejecutarProtocolo» funciona de la siguiente manera:

- Recibe como parámetros tres archivos, archivo que almacenará las cadenas generadas, las cadenas aceptadas y las cadenas rechazadas.
- Se declara el «no_cadenas» que es el número de cadenas a generar, «long_cadenas» que es la longitud de cada cadena y con la librería numpy, se genera aleatoriamente una matriz de nxm con números entre 0 y 1.
- Se informa al usuario cuando se inicia el protocolo, se crea un retraso en la ejecución del programa de 3 segundos, y se informa que comienza la ejecución del protocolo.
- Se recorren las filas de la matriz (cada fila representa una cadena), se crea el objeto de la clase «ADF» enviándole la fila como parámetro. Posteriormente pide la cadena enviada en formato «str» para almacenarla en el archivo de todas las cadenas generadas.
- Finalmente, se verifica la imparidad de la cadena, si devuelve «True» significa que es una cadena aceptada y se almacena en el archivo correspondiente, en otro caso se almacena en el archivo de cadenas rechazadas.

El siguiente fragmento de código, controla el manejo de archivos, y la ejecución del protocolo en base a un número aleatorio.

```
corrida = 1
encendido = True

#Abrimos los archivos.
generadas = open('CadenasGeneradas.txt', mode='w')
aceptadas = open('CadenasAceptadas.txt', mode='w')
rechazadas = open('CadenasRechazadas.txt', mode='w')

while encendido:
    generadas.write('EJECUCION '+str(corrida)+'\n')
    aceptadas.write('EJECUCION '+str(corrida)+'\n')
    rechazadas.write('EJECUCION '+str(corrida)+'\n')

    ejecutarProtocolo(generadas, aceptadas, rechazadas)
    corrida+= 1
    encendido = False if np.random.randint(2) == 0 else True

print('Ejecución de protocolos terminada. . .')
# Cerramos los archivos.
generadas.close()
aceptadas.close()
rechazadas.close()
```

El funcionamiento del código anterior es el siguiente:

- Se define la variable «corrida» y se le asigna el valor de 1, para identificar en los archivos a que corrida corresponden las cadenas. A demás, se define la variable «encendido» como True, para inicializar el protocolo como encendido.
- Se crean y/o abren los archivos: «CadenasGeneradas.txt» , «CadenasAceptadas.txt», «CadenasRechazadas.txt»
- Se crea un bucle donde se escribe sobre todos los archivos el numero de ejecucion actual.
- Se ejecuta el protocolo. enviando los archivos creados como parámetros.
- Se suma uno a la corrida para identificar la siguiente en caso de que haya. Se reasigna el valor de «encendido» generando un número aleatorio entre 0 y 1, si es 0 se apaga el protocolo, si es 1 se deja encendido. El bucle termina hasta que el protocolo se apague automáticamente.
- Posteriormente, se informa que la ejecución del protocolo ha terminado y se cierran todos los archivos.

Para la visualización del autómata se usa el siguiente fragmento de código.

```
# Graficando representacion visual de automata

# Agregar arista
def agregar_arista(G, u, v, w=1, di=True):
    G.add_edge(u, v, weight=w)

    # Si el grafo no es dirigido
    if not di:
        # Agrego otra arista en sentido contrario
        G.add_edge(v, u, weight=w)

G = nx.Graph()

# Agregar nodos y aristas
agregar_arista(G, "Ready", "End", "No")
agregar_arista(G, "Ready", "Generate", "Yes")
agregar_arista(G, "Generate", "q0", "Timeout")
agregar_arista(G, "q0", "q2", 0)
agregar_arista(G, "q2", "q0", 0)
agregar_arista(G, "q0", "q1", 1)
agregar_arista(G, "q1", "q0", 1)
agregar_arista(G, "q1", "q3", 0)
agregar_arista(G, "q3", "q1", 0)
agregar_arista(G, "q2", "q3", 1)
agregar_arista(G, "q3", "q2", 1)
agregar_arista(G, "q0", "Append", "pair")
agregar_arista(G, "q1", "Append", "odd")
agregar_arista(G, "q2", "Append", "odd")
agregar_arista(G, "q3", "Append", "odd")
agregar_arista(G, "Append", "Ready", False)

# Draw the networks
pos = nx.layout.planar_layout(G)
nx.draw_networkx(G, pos)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.title("Protocolo")
plt.show()
```

- Primero se crea la función «agregar_arista», recibe como parámetros el grafo, el estado actual, el estado final, y algún texto en la conexión. Se agrega un arista al grafo.
- Se crea el grafo "G", instanciando la clase «Graph» de la librería «networkx».
- Se usa la función «agregar_arista», para ir construyendo el grafo.
- Se dibuja el grafo, primero se define un lienzo, después se dibuja el grafo sobre el lienzo, se obtienen las etiquetas, se escriben las etiquetas sobre el lienzo, se le da un nombre al lienzo y se muestra con matplotlib.

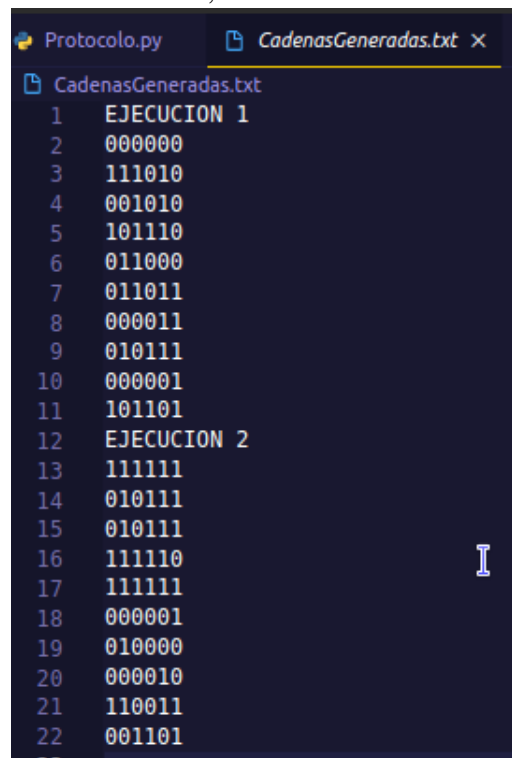
La ejecución del programa es el siguiente:

Se ejecuta el script.

```
(base) bruno-rg@bruno-rg > ~/Documents/6to Semestre/TC /bin/python3 "/home/bruno-rg/Documents/6to Semestre/TC/Practica 3/Protocolo.py"
Iniciando protocolo. . .
Ejecutando protocolo. . .
Iniciando protocolo. . .
Ejecutando protocolo. . .
Ejecución de protocolos terminada. . .
(base) bruno-rg@bruno-rg > ~/Documents/6to Semestre/TC
```

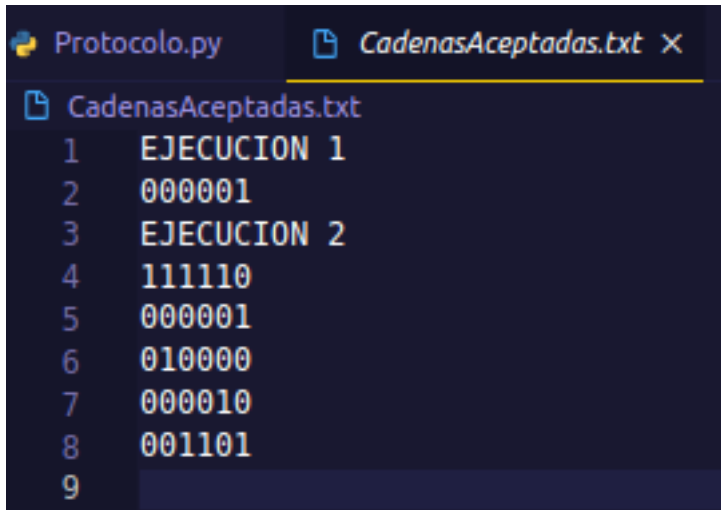
En este caso, el protocolo se ejecutó dos veces de forma automática.

A continuación, se muestran todas las cadenas generadas.



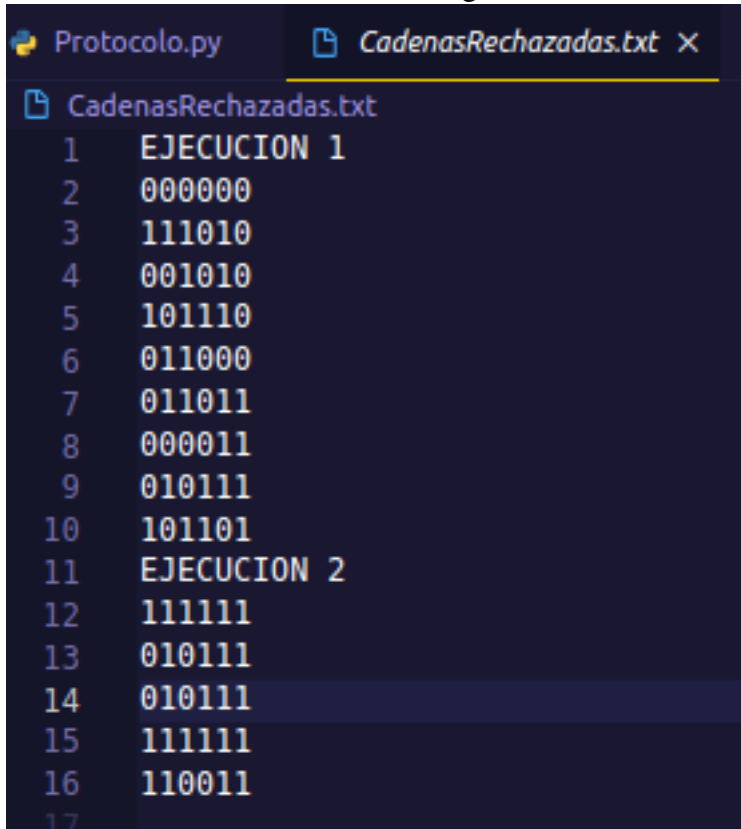
```
Protocolo.py CadenasGeneradas.txt x
CadenasGeneradas.txt
1 EJECUCION 1
2 000000
3 111010
4 001010
5 101110
6 011000
7 011011
8 000011
9 010111
10 000001
11 101101
12 EJECUCION 2
13 111111
14 010111
15 010111
16 111110
17 111111
18 000001
19 010000
20 000010
21 110011
22 001101
```

De todas las cadenas mostradas en la imagen anterior, se aceptaron solamente las siguientes cadenas.



```
Protocolo.py  CadenasAceptadas.txt X
CadenasAceptadas.txt
1  EJECUCION 1
2  000001
3  EJECUCION 2
4  111110
5  000001
6  010000
7  000010
8  001101
9
```

Y las cadenas rechazadas, son las siguientes.



```
Protocolo.py  CadenasRechazadas.txt X
CadenasRechazadas.txt
1  EJECUCION 1
2  000000
3  111010
4  001010
5  101110
6  011000
7  011011
8  000011
9  010111
10 101101
11 EJECUCION 2
12 111111
13 010111
14 010111
15 111111
16 110011
17
```

Finalmente, la visualización del autómata quedó de la siguiente manera.

