

Ejercicios 3^{er} Parcial

Nevárez Tovar Juan Carlos - 2CM5

23 de junio de 2020

Índice

| | |
|---|-----------|
| 1. Expresión regular. | 2 |
| 1.1. Pruebas de ejecución. | 2 |
| 1.2. Código. | 3 |
| 2. Palíndromo con una Gramática Libre de Contexto. | 5 |
| 2.1. Pruebas de ejecución. | 5 |
| 2.2. Código. | 7 |
| 3. Autómata de Pila. | 10 |
| 3.1. Pruebas de ejecución. | 10 |
| 3.2. Código. | 11 |
| 4. Maquina de Turing. | 15 |
| 4.1. Pruebas de ejecución. | 16 |
| 4.2. Código. | 17 |

1. Expresión regular.

Se generan 10 cadenas a partir de la siguiente expresión regular: $(0 + 10)^* (e + 1)$. Es decir cadenas de 0's y 1's sin dos 1's consecutivos

Se generan automáticamente 10 cadenas, y de manera aleatoria se accede a cada parte de la ecuación.

Para el caso de la cerradura de Kleene, se llega como máximo a el valor de 1000.

Las cadenas que se generen deben pertenecer al lenguaje y se almacenan en un archivo de texto. Si se generan mas cadenas, éstas se anexan al mismo archivo.

1.1. Pruebas de ejecución.

Ejecución inicial, se generan 10 cadenas automáticamente.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\juann\Documents\ESCOM\Cuarto semestre\Teoria Computacional\Ejercicio 6 EXPRESION REGULAR>python ExpReg.py
Generando cadenas...
Cadenas generadas, revise el archivo.

Expresion regular, (Cadenas de 0's y 1's sin dos 1's consecutivos):
(1) Crear mas cadenas.
(2) Salir.
Ingresa una opcion: █
```

Salida de la primera ejecución.

```
salida.txt  X
Ejercicio 6 EXPRESION REGULAR > salida.txt
1  1
2  10100001000001000101000010000010100010100010010010100100100101000001010101000101000100000000101000100001010
3  10101010001010000000010100101000010100010101001010000010100101010101010000010100010001001000100000000
4  E
5  10101000001010001010001001010101001000000101000000100000101010100100001010100010010001010001010001001001010
6  0010101010101010101001000100010101010000000010001010000101010000100100010001001010100000101010010010001000001
7  101001001010100100100000101001010101010001000001001001010101010100100101000101000100000000100010010101001001010
8  E
9  1
10 1
11 █
```

Ejecución pidiendo que se generen otras 10 cadenas.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\juann\Documents\ESCOM\Cuarto semestre\Teoria Computacional\Ejercicio 6 EXPRESION REGULAR>python ExpReg.py
Generando cadenas...
Cadenas generadas, revise el archivo.

Expresion regular, (Cadenas de 0's y 1's sin dos 1's consecutivos):
(1) Crear mas cadenas.
(2) Salir.
Ingresa una opcion: 1
Generando cadenas...
Cadenas generadas, revise el archivo.

Expresion regular, (Cadenas de 0's y 1's sin dos 1's consecutivos):
(1) Crear mas cadenas.
(2) Salir.
Ingresa una opcion: █
```

Salida de la generación de 10 cadenas más.

```
salida.txt x
Ejercicio 6 EXPRESION REGULAR > salida.txt
1 1
2 10100001000001000101000010000010100010100100100100101010000010101000010000001010010100101000101000010000000101000100001010
3 10101000101000000001010010100001010100101000001010100101010101000000101000100010001000100010001000100010000000
4 E
5 101010000010100010100100101010100100000010100000010000010101010010000010100001001000101000101010001001000100101001010
6 0010101010101010101000100010101010000000100010100001010100010010100010001010101000001010010010001001000100100000100001
7 10100100101010010010000001010010101010001000001001001010101010100100010100001000000001000010010101010001001010101010
8 E
9 1
10 1
11 1
12 00010100101010100010101010101000001000100001010100100100101010100100010010101000000001010010000100101010010000101000001000101001
13 1
14 100101010101010001000101001000010010100010010001001001010000101001010101000000100101010100010010101000101010101010010
15 10010
16 E
17 001010101010010100100101000010000010001000101010100100101010100001000010010100100100010101000000101001001000010100100010000
18 00101010100000010100010001010010001001000100010010010101001010000000010101001010100010010000101001001010100100100101010
19 10010010101001010100000000100010100001010001000010101001000000100101001010100001010010101000010100000101000000101000001010
20 010100101001010100000001000101001000100001000100100001000001001001010001010010001000001010101010001000101010001010
21
```

1.2. Código.

ExpReg.py

Listing 1: Programa para generar las cadenas.

```
import random

def GenerarCadenas(archivo):
    for i in range(0, 10):
        cadena = ""
        hacerKleene = random.randint(0,1)
        if(hacerKleene):
            #print(" Se hace Kleene")
            rangoKleene = random.randint(1, 1000)
            for j in range(0, rangoKleene):
                # Hacer j veces (0+10) = {0, 10}
                cadena = hacerSuma(cadena)

            cadena = concatenar(cadena, True)
            escribirCadena(archivo, cadena)
        else:
            #print(" No hacer Kleene")
            cadena = concatenar(cadena, False)
            escribirCadena(archivo, cadena)

def escribirCadena(archivo, cadena):
    #print(cadena)
    archivo.write(cadena)
    archivo.write("\n")

def hacerSuma(cadena):
    unir = random.randint(0,1)
```

```

    if(unir):
        cadena += "0"
    else:
        cadena += "10"
    return cadena

def concatenar(cadena, kleene):
    hacerConc = random.randint(0,1)
    if(hacerConc):
        cadena += "1"
        return cadena
    elif(kleene):
        return cadena
    else:
        cadena += "E"
        return cadena

def menu():
    while(True):
        print("└ Expresion regular ,└ (Cadenas de 0's y 1's sin dos 1's consecutivos)")
        print("└ (1)└ Crear mas cadenas.")
        print("└ (2)└ Salir.")
        try:
            opcion = int(input("└ Ingresa una opcion:└"))
            if(opcion == 1):
                break
            elif(opcion == 2):
                break
            else:
                print("└ Elija una opcion valida.")
                continue
        except:
            print("└ Ingresa una opcion valida.")
    return opcion

def inicio():
    # Reescribir el archivo
    salida = open("salida.txt", "w")
    salida.write("")
    salida.close()
    while(True):
        print("└ Generando cadenas...")
        salida = open("salida.txt", "a+")
        GenerarCadenas(salida)
        salida.close()
        print("└ Cadenas generadas ,└ revise el archivo.\n")
        opc = menu()
        if(opc == 2):
            break

```

```
print(" Fin del programa.")

inicio()
```

2. Palíndromo con una Gramática Libre de Contexto.

El programa construye palíndromos de un lenguaje binario, se solicita únicamente la longitud del palíndromo a calcular o bien la longitud se genera de forma automática, de esta manera el programa construye el palíndromo de manera aleatoria.

La longitud máxima que podría alcanzar un palíndromo de 100,000 caracteres.

En la salida del programa va a un archivo de texto, en el se especifica qué regla se selecciona y la cadena resultante hasta llegar a la cadena final.

La reglas de producción de la gramática libre de contexto que se usan para construir el palíndromo son las siguientes:

1. $P \rightarrow e$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

2.1. Pruebas de ejecución.

Ingreso de una longitud de palíndromo de 10.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\juann\Documents\ESCOM\Cuarto semestre\Teoria Computacional\Ejercicio 7 GRAMATICAS>python Palindromo.py

Palindromo con GLC
(1) Ingresar una longitud
(2) Generar una longitud
(3) Salir
Ingresa una opcion: 1
Ingresa la longitud: 10

Palindromo con GLC
(1) Ingresar una longitud
(2) Generar una longitud
(3) Salir
Ingresa una opcion: █

```

Archivo de salida con un palíndromo de 10 caracteres.

```

procedimiento.txt  salida.txt  X
Ejercicio 7 GRAMATICAS > salida.txt
1 El resultado es: 1110110111

```

Archivo con el proceso que se va siguiendo para generar el palíndromo.

```
procedimiento.txt × salida.txt
Ejercicio 7 GRAMATICAS > procedimiento.txt
1 Se aplico regla 5 a "P".
2 Se aplico regla 5 a "1P1".
3 Se aplico regla 5 a "11P11".
4 Se aplico regla 4 a "111P111".
5 Se aplico regla 5 a "1110P0111".
6 Se aplico regla 1 a "11101P10111".
7
```

Ingreso de una longitud de palíndromo de 50.

C:\Users\juann\Documents\ESCOM\Cuarto semestre\Teoria Computacional\Ejercicio 7 GRAMATICAS>python Palindromo.py

```
Palindromo con GLC
(1) Ingresar una longitud
(2) Generar una longitud
(3) Salir
Ingresa una opcion: 1
Ingresa la longitud: 50
```

```
Palindromo con GLC
(1) Ingresar una longitud
(2) Generar una longitud
(3) Salir
Ingresa una opcion: 3
```

Archivo de salida con un palíndromo de 50 caracteres.

```
procedimiento.txt salida.txt ×
Ejercicio 7 GRAMATICAS > salida.txt
1 El resultado es: 00001001011000110110000011000001101100011010010000
```

Archivo con el proceso que se va siguiendo para generar el palíndromo.

procedimiento.txt × salida.txt

Ejercicio 7 GRAMATICAS > procedimiento.txt

```
1 Se aplico regla 4 a "P".
2 Se aplico regla 4 a "0P0".
3 Se aplico regla 4 a "00P00".
4 Se aplico regla 4 a "000P000".
5 Se aplico regla 5 a "0000P0000".
6 Se aplico regla 4 a "00001P10000".
7 Se aplico regla 4 a "000010P010000".
8 Se aplico regla 5 a "0000100P0010000".
9 Se aplico regla 4 a "00001001P10010000".
10 Se aplico regla 5 a "000010010P010010000".
11 Se aplico regla 5 a "0000100101P1010010000".
12 Se aplico regla 4 a "00001001011P11010010000".
13 Se aplico regla 4 a "000010010110P011010010000".
14 Se aplico regla 4 a "0000100101100P0011010010000".
15 Se aplico regla 5 a "00001001011000P00011010010000".
16 Se aplico regla 5 a "000010010110001P100011010010000".
17 Se aplico regla 4 a "0000100101100011P1100011010010000".
18 Se aplico regla 5 a "00001001011000110P01100011010010000".
19 Se aplico regla 5 a "000010010110001101P101100011010010000".
20 Se aplico regla 4 a "0000100101100011011P1101100011010010000".
21 Se aplico regla 4 a "00001001011000110110P01101100011010010000".
22 Se aplico regla 4 a "000010010110001101100P001101100011010010000".
23 Se aplico regla 4 a "0000100101100011011000P0001101100011010010000".
24 Se aplico regla 4 a "00001001011000110110000P00001101100011010010000".
25 Se aplico regla 5 a "000010010110001101100000P000001101100011010010000".
26 Se aplico regla 1 a "0000100101100011011000001P1000001101100011010010000".
27
```

2.2. Código.

Palindromo.py

Listing 2: Programa para generar palíndromos utilizando Gramatica Libre de Contexto.

```
import random

def r1(palindromo, archivo):
    archivo.write("Se aplico regla 1 a \n" + palindromo + "\n.\n")
    if(len(palindromo) == 1):
        return palindromo.replace("P", "e")
    else:
        return palindromo.replace("P", "")

def r2(palindromo, archivo):
    archivo.write("Se aplico regla 2 a \n" + palindromo + "\n.\n")
    return palindromo.replace("P", "0")
```

```

def r3(palindromo, archivo):
    archivo.write("Se_aplico_regla_3_a_\n" + palindromo + "\n")
    return palindromo.replace("P", "1")

def r4(palindromo, archivo):
    archivo.write("Se_aplico_regla_4_a_\n" + palindromo + "\n")
    return palindromo.replace("P", "0P0")

def r5(palindromo, archivo):
    archivo.write("Se_aplico_regla_5_a_\n" + palindromo + "\n")
    return palindromo.replace("P", "1P1")

def aplicar23(palindromo, archivo):
    regla = random.randint(2, 3)
    if(regla == 2):
        palindromo = r2(palindromo, archivo)
    else:
        palindromo = r3(palindromo, archivo)

    return palindromo

def aplicar45(palindromo, archivo):
    regla = random.randint(4, 5)
    if(regla == 4):
        palindromo = r4(palindromo, archivo)
    else:
        palindromo = r5(palindromo, archivo)

    return palindromo

def inicio():
    while(True):
        opcion = menu()
        if(opcion == 1):
            longitud = pedirLongitud()
            procesar(longitud)
        elif(opcion == 2):
            longitud = generarLongitud()
            procesar(longitud)
        elif(opcion == 3):
            break
        else:
            print("\nIngrese una opcion valida.")

def menu():
    while(True):
        print("\n_Palindromo_con_GLC")
        print("\n(1)_Ingresar_una_longitud")
        print("\n(2)_Generar_una_longitud")
        print("\n(3)_Salir")

```



```

    try:
        opcion = int(input("Ingresa una opcion: "))
        break
    except:
        print("Ingresa una opcion valida.")

return opcion

def generarLongitud():
    return random.randint(0,10000)

def pedirLongitud():
    while(True):
        try:
            longitud = int(input("Ingresa la longitud: "))
            if(longitud >= 0 and longitud <= 100000):
                break
            else:
                print("La longitud debe ser entre 0 y 100000")
        except:
            print("Ingresa una longitud valida.")

    return longitud

def procesar(longitud):
    procedimiento = open("procedimiento.txt", "w")
    salida = open("salida.txt", "w")
    palindromo = "P"

    if(longitud == 0):
        palindromo = r1(palindromo, procedimiento)
    elif(longitud == 1):
        palindromo = aplicar23(palindromo, procedimiento)
    else:
        palindromo = aplicar45(palindromo, procedimiento)
        if(esPar(longitud)):
            while(True):
                if(longitud+1 == len(palindromo)):
                    palindromo = r1(palindromo, procedimiento)
                    break
                else:
                    palindromo = aplicar45(palindromo, procedimiento)
        else:
            while(True):
                if(longitud == len(palindromo)):
                    palindromo = aplicar23(palindromo, procedimiento)
                    break
                else:
                    palindromo = aplicar45(palindromo, procedimiento)

```

```

salida.write("El resultado es: " + palindromo)

salida.close()
procedimiento.close()

def esPar(longitud):
    return longitud % 2 == 0

inicio()

```

3. Autómata de Pila.

El programa reconoce el lenguaje libre de contexto $0^n 1^n | n \geq 1$. Como entrada recibe una cadena escrita por el usuario, o una generada por el programa, esta cadena tiene a lo máximo 1000 caracteres. Como salida el programa pone en un archivo de salida las descripciones instantáneas de cada paso de la computación.

3.1. Pruebas de ejecución.

Ejecución con una cadena valida.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Versión 10.0.18362.900]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\juann\Documents\ESCOM\Cuarto semestre\Teoria Computacional\Ejercicio 8 AUTOMATA DE PILA>python AutomataPila.py

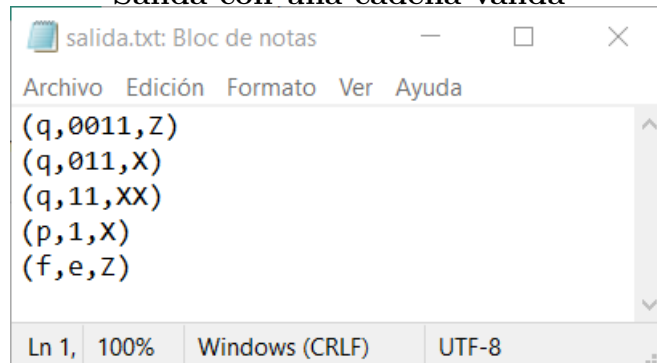
Atomata de Pila.
1) Ingresar una cadena.
2) Generar una cadena al azar.
3) Generar una cadena valida.
4) Salir
Ingrese una opcion: 1
Ingresa una cadena para evaluar, debe tener menos de 1000 caracteres: 0011

La cadena 0011 pertenece al lenguaje, revisa el archivo de salida.

Atomata de Pila.
1) Ingresar una cadena.
2) Generar una cadena al azar.
3) Generar una cadena valida.
4) Salir
Ingrese una opcion: █

```

Salida con una cadena valida



Ejecución con una cadena invalida.

Atomata de Pila.

- 1) Ingresar una cadena.
- 2) Generar una cadena al azar.
- 3) Generar una cadena valida.
- 4) Salir

Ingrese una opcion: 1

Ingresar una cadena para evaluar, debe tener menos de 1000 caracteres: 000100

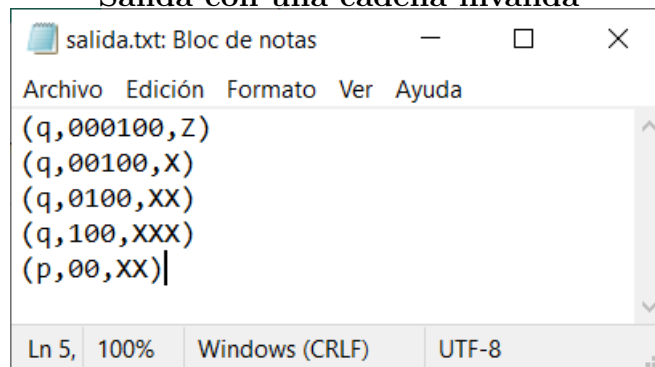
La cadena 000100 no pertenece al lenguaje, revisa el archivo de salida.

Atomata de Pila.

- 1) Ingresar una cadena.
- 2) Generar una cadena al azar.
- 3) Generar una cadena valida.
- 4) Salir

Ingrese una opcion: █

Salida con una cadena invalida



3.2. Código.

AutomataPila.py

Listing 3: Programa para reconocer el lenguaje.

```
import random

# Funciones para la estructura Pila
def esVacia(Pila):
    return Pila == []

def top(Pila):
    return Pila[len(Pila) - 1]

def push(Pila, elem):
    Pila.append(elem)
    return Pila

def tam(Pila):
    return len(Pila)

# Funciones generales
```

```

def menu():
    print("\nAutomata de Pila.")
    print("\t1) Ingresar una cadena.")
    print("\t2) Generar una cadena al azar.")
    print("\t3) Generar una cadena valida.")
    print("\t4) Salir")
    while(True):
        try:
            opcion = input("Ingrese una opcion: ")
            break
        except:
            print("Ingrese una opcion valida")
    return opcion

def PedirCadena():
    cadena = input("Ingrese una cadena para evaluar, debe tener menos de 100")
    return cadena

def PedirTam():
    while(True):
        try:
            tam = int(input("Ingrese el tamaño de la cadena a generar, debe"))
            break
        except:
            print("Ingrese una opcion valida.")
    return tam

def GenerarCadena(tam):
    cadena = ""
    tam = int(tam)

    for i in range(0,tam):
        car = random.randint(0,1)
        if(car == 1):
            cadena = cadena + "1"
        else:
            cadena = cadena + "0"

    return cadena

def GenValida(tam):
    cadena = ""

    for i in range(0,tam):
        if(i < tam/2):
            cadena = cadena + "0"
        else:
            cadena = cadena + "1"

    return cadena

```

```

def esCadValida(cad):
    if(len(cad) <= 1000):
        return True
    else:
        return False

def esTamValido(tam):
    if(tam <= 1000 and tam > 0):
        return True
    else:
        return False

def hacerTamValido(tam):
    if(esPar(tam)):
        return tam
    else:
        print("└tam└" + str(tam) + ")└necesita└ser└par,└se└le└sumara└1.")
        return tam+1

def inicio():
    while(True):
        opc = int(menu())

        if(opc == 1): # El usuario ingresa la cadena
            cadena = PedirCadena()
            if(esCadValida(cadena)):
                procesar(cadena)
            else:
                print("La└cadena└" + cadena + "└no└es└valida.")
        elif(opc == 2): # El sistema genera la cadena
            tam = PedirTam()
            if(esTamValido(tam)):
                cadena = GenerarCadena(tam)
                procesar(cadena)
            else:
                print("└" + str(tam) + "└no└es└valido.")
        elif(opc == 3):
            tam = PedirTam()
            if(esTamValido(tam)):
                tam = hacerTamValido(tam)
                cadena = GenValida(tam)
                procesar(cadena)
            else:
                print("└" + str(tam) + "└no└es└valido.")
        elif(opc == 4):
            print("└Fin└del└programa")
            break
        else:
            print("└Ingresa└una└opcion└valida.")

```

```

def esPar(n):
    if ((n%2) == 0):
        return True
    else:
        return False

# Funciones para el automata

def PruebaImpresion(edo, cadena, Pila):
    print(edo)
    print(cadena)
    if (Pila == ""):
        print("Z")
    else:
        print(Pila)

def escribirDI(archivo, q, w, a):
    if (a == ""):
        archivo.write("(" + q + ", " + w + ", Z) \n")
    else:
        archivo.write("(" + q + ", " + w + ", " + a + ") \n")

def procesar(cadena):
    salida = open("salida.txt", 'w')
    Pila = []
    edo = "q"
    count = 0

    for c in cadena: # Recorre toda la cadena
        #PruebaImpresion(edo, cadena[count:len(cadena)], ''.join(Pila))
        escribirDI(salida, edo, cadena[count:len(cadena)], ''.join(Pila))
        if (edo == "q"):
            if (c == "1"):
                if (not esVacia(Pila)):
                    edo = "p"
                    Pila.pop()
                else:
                    edo = "error"
                    break
            elif (c == "0"):
                edo = "q"
                Pila = push(Pila, 'X')
            else:
                edo = "error"
                break
        elif (edo == "p"):
            if (c == "1"):
                if (not esVacia(Pila)):
                    edo = "p"

```

```

        Pila.pop()
    else:
        edo = "error"
        break
    else:
        edo = "error"
        break
    else:
        edo = "error"
        break
    count = count + 1

if((edo == "p") and (count == len(cadena)) and esVacia(Pila)):
    print("\nLa cadena " + cadena + " pertenece al lenguaje , revisa el lenguaje")
    edo = "f"
    #PruebaImpresion(edo, "E", ''.join(Pila))
    escribirDI(salida, edo, "e", ''.join(Pila))
else:
    print("\nLa cadena " + cadena + " no pertenece al lenguaje , revisa el lenguaje")

salida.close()

inicio()

```

4. Maquina de Turing.

Reconoce el lenguaje $\{0^n 1^n \mid n \geq 1\}$. El programa recibe una cadena definida por el usuario, o bien, una cadena generada por el programa, esta cadena tiene una longitud máxima de 1000 caracteres. La salida consiste en las descripciones instantáneas de cada paso que se hace en la computación.

Las transiciones se describen en la siguiente tabla:

| Estado | 0 | 1 | X | Y | B |
|----------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| q ₀ | (q ₁ , X, R) | - | - | (q ₃ , Y, R) | - |
| q ₁ | (q ₁ , 0, R) | (q ₂ , Y, L) | - | (q ₁ , Y, R) | - |
| q ₂ | (q ₂ , 0, L) | - | (q ₀ , X, R) | (q ₂ , Y, L) | - |
| q ₃ | - | - | - | (q ₃ , Y, R) | (q ₄ , B, R) |
| q ₄ | - | - | - | - | - |

4.1. Pruebas de ejecución.

Ejecución con una cadena valida.

Maquina de Turing.

- 1) Ingresar una cadena.
- 2) Generar una cadena al azar.
- 3) Generar una cadena valida.
- 4) Salir

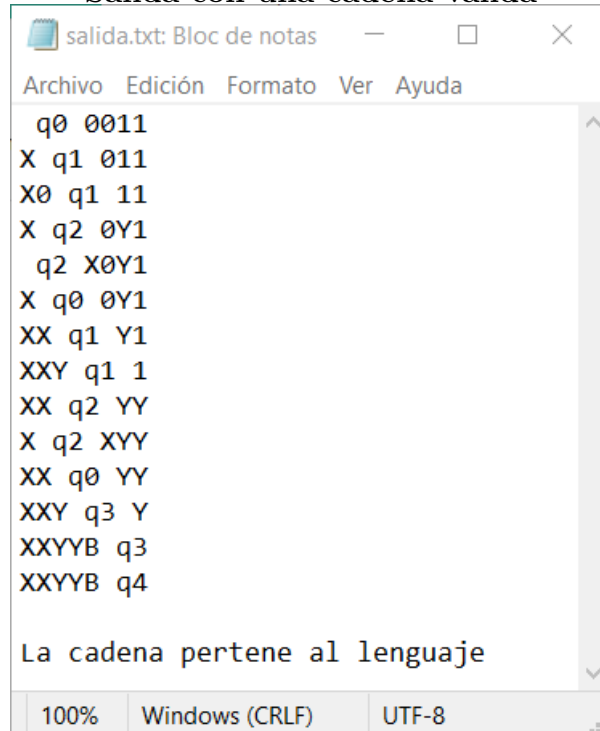
Ingrese una opcion: 1

Ingresa una cadena para evaluar, debe tener menos de 1000 caracteres: 0011

Procesando la cadena: 0011

La cadena pertenece al lenguaje, revisa el archivo.

Salida con una cadena valida



```
salida.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
q0 0011
X q1 011
X0 q1 11
X q2 0Y1
  q2 X0Y1
X q0 0Y1
XX q1 Y1
XXY q1 1
XX q2 YY
X q2 XYY
XX q0 YY
XXY q3 Y
XXYYB q3
XXYYB q4

La cadena pertenece al lenguaje
100%  Windows (CRLF)  UTF-8
```

Ejecución con una cadena invalida.

Maquina de Turing.

- 1) Ingresar una cadena.
- 2) Generar una cadena al azar.
- 3) Generar una cadena valida.
- 4) Salir

Ingrese una opcion: 1

Ingresa una cadena para evaluar, debe tener menos de 1000 caracteres: 00110

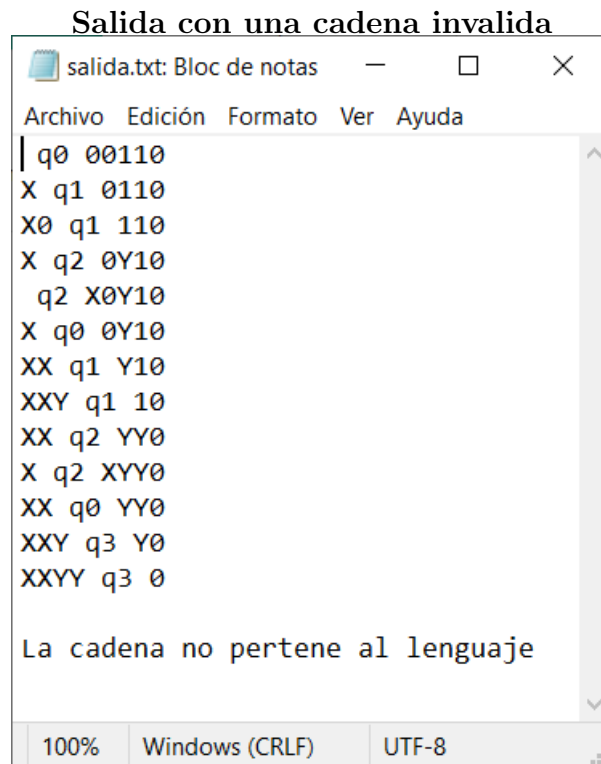
Procesando la cadena: 00110

La cadena no pertenece al lenguaje, revisa el archivo.

Maquina de Turing.

- 1) Ingresar una cadena.
- 2) Generar una cadena al azar.
- 3) Generar una cadena valida.
- 4) Salir

Ingrese una opcion: █



4.2. Código.

MaquinaTuring.py

Listing 4: Programa para reconocer el lenguaje.

```

import random

def menu():
    while(True):
        print("\nMaquina de Turing.")
        print("\t1) Ingresar una cadena.")
        print("\t2) Generar una cadena al azar.")
        print("\t3) Generar una cadena valida.")
        print("\t4) Salir")
        try:
            opcion = int(input("Ingresa una opcion: "))
            break
        except:
            print("Ingresa una opcion valida.")
    return opcion

def PedirCadena():
    cadena = input("Ingresa una cadena para evaluar, debe tener menos de 100")
    return cadena

def PedirTam():
    tam = int(input("Ingresa el tamaño de la cadena a generar, debe ser menor"))
    return tam

```

```

def GenerarCadena(tam):
    cadena = ""
    tam = int(tam)
    for i in range(0,tam):
        car = random.randint(0,1)
        if(car == 1):
            cadena = cadena + "1"
        else:
            cadena = cadena + "0"
    return cadena

def GenValida(tam):
    cadena = ""
    for i in range(0,tam):
        if(i < tam/2):
            cadena = cadena + "0"
        else:
            cadena = cadena + "1"
    return cadena

def esCadValida(cad):
    if(len(cad) <= 1000):
        return True
    else:
        return False

def esTamValido(tam):
    if(tam <= 1000 and tam > 0):
        return True
    else:
        return False

def hacerTamValido(tam):
    if(esPar(tam)):
        return tam
    else:
        print("_tam_" + str(tam) + ")_necesita_ser_par,_se_le_sumara_1.")
        return tam+1

def inicio():
    while(True):
        opc = int(menu())
        if(opc == 1):
            cadena = PedirCadena()
            if(esCadValida(cadena)):
                procesar(cadena)
            else:
                print("La_cadena_" + cadena + "_no_es_valida.")
        elif(opc == 2):
            tam = PedirTam()

```

```

        if(esTamValido(tam)):
            cadena = GenerarCadena(tam)
            procesar(cadena)
        else:
            print("_" + str(tam) + "_no_es_valido.")
    elif(opc == 3):
        tam = PedirTam()
        if(esTamValido(tam)):
            tam = hacerTamValido(tam)
            cadena = GenValida(tam)
            procesar(cadena)
        else:
            print("_" + str(tam) + "_no_es_valido.")
    elif(opc == 4):
        print("_Fin_del_programa")
        break
    else:
        print("Ingresa una opcion valida.")

def esPar(n):
    if((n%2) == 0):
        return True
    else:
        return False

def reemplazar(cadena, posicion, reemplazo):
    cl = list(cadena)
    cl[posicion] = reemplazo
    return ''.join(cl)

def esB(pos, tamC):
    if(pos >= tamC):
        return True
    elif(int(pos) < 0):
        return True
    else:
        return False

def ImpDI(archivo, cinta, pos, edo):
    i = 0
    pasoEdo = False
    for s in cinta:
        if(i == pos):
            archivo.write("_" + edo + "_")
            pasoEdo = True
            archivo.write(s)
            i+=1
    if(not pasoEdo):
        archivo.write("B_" + edo)
    archivo.write("\n")

```

```

def procesar(cadena):
    q = "q0"
    p = 0
    salida = open("salida.txt", "w")
    print("_Procesando_la_cadena:_ " + cadena)
    while(True):
        ImpDI(salida, cadena, p, q)
        if(q == "q0"):
            if(not esB(p, len(cadena))):
                simbolo = cadena[p]
                if(simbolo == "0"):
                    q = "q1"
                    cadena = reemplazar(cadena, p, "X")
                    p += 1
                elif(simbolo == "Y"):
                    q = "q3"
                    p += 1
                else:
                    break
            else:
                break
        elif(q == "q1"):
            if(not esB(p, len(cadena))):
                simbolo = cadena[p]
                if(simbolo == "0"):
                    p += 1
                elif(simbolo == "1"):
                    q = "q2"
                    cadena = reemplazar(cadena, p, "Y")
                    p -= 1
                elif(simbolo == "Y"):
                    p += 1
                else:
                    break
            else:
                break
        elif(q == "q2"):
            if(not esB(p, len(cadena))):
                simbolo = cadena[p]
                if(simbolo == "0"):
                    p -= 1
                elif(simbolo == "X"):
                    q = "q0"
                    p += 1
                elif(simbolo == "Y"):
                    p -= 1
                else:
                    break
            else:

```

```

        break
    elif(q == "q3"):
        if(esB(p, len(cadena))):
            q = "q4"
            p += 1
        else:
            simbolo = cadena[p]
            if(simbolo == "Y"):
                q = "q3"
                p += 1
            else:
                break
    elif(q == "q4"):
        break
    else:
        break

if(q == "q4"):
    print("La cadena pertenece al lenguaje , revisa el archivo.")
    salida.write("\nLa cadena pertenece al lenguaje")
else:
    print("La cadena no pertenece al lenguaje , revisa el archivo.")
    salida.write("\nLa cadena no pertenece al lenguaje")
salida.close()

inicio()

```