



INSTITUTO POLITECNICO NACIONAL

Escuela Superior de Cómputo

Autómata de pila

Teoría de la computación

Alumno:

Reyes Garnelo Uziel Bruno

Profesor: Genaro Juárez Martinez

Grupo: 5BM1

14 de junio de 2023

1. Introducción:

Un autómata de pila es un tipo de máquina abstracta que se utiliza en el campo de la teoría de la computación. A diferencia de los autómatas finitos, los autómatas de pila tienen una memoria adicional llamada "pila", que es una estructura de datos LIFO (Last In, First Out).

Los autómatas de pila son utilizados en el estudio de gramáticas libres de contexto, reconocimiento de lenguajes y en la construcción de compiladores y analizadores sintácticos. Permiten modelar una amplia gama de lenguajes y estructuras de datos más complejas que los autómatas finitos.

En resumen, los autómatas de pila son máquinas abstractas que incluyen una pila como una memoria adicional. Utilizan una función de transición para cambiar de estado en función de la entrada y el contenido de la pila. Son herramientas fundamentales en la teoría de la computación para estudiar gramáticas y lenguajes formales.

2. Presentación del problema:

Programar un autómata de pila que sirva para reconocer el lenguaje libre de contexto $0^n 1^n | n \geq 1$.

Adicionalmente, el programa debe de contar con las siguientes características:

- La cadena puede ser ingresada por el usuario o automáticamente. Si es aleatoriamente, la cadena no podrá ser mayor a 100,000 caracteres.
- Mandar a un archivo y en pantalla la evaluación del autómata a través de descripciones instantáneas (IDs).
- Animar el autómata de pila, solo si la cadena es menor igual a 10 caracteres.

3. Desarrollo:

3.1. Programación del autómata

El lenguaje seleccionado para resolver este problema es Python.

A continuación, mostraremos el código desarrollado.

```
1 import random % Libreria 'random'
2 import os % Libreria para manejo de archivos
```

Usamos la librería 'random' para la generación de números. A demás, usamos la librería 'os' para el manejo de archivos.

```

1 print('Automata de pila!')
2 # Modo de creacion de la cadena
3 print('Deseas que el lenguaje se genere de forma manual o automatica?')
4 modo = int(input((" 0. Manual\n 1. Automatica\n")))
5 cadena = ''
6 # Manual: Pedimos al usuario que introduzca la cadena
7 if modo == 0:
8     cadena = str(input('Cadena binaria --> '))
9 # Automatico: Creamos la cadena con 0's y 1's (50% probabilidad por elemento)
10 else:
11     n = random.randint(0,100000)
12     cadena += '0'
13     for i in range(n-1):
14         cadena += str(random.randint(0,1))

```

En el bloque anterior, se pregunta al usuario si quiere ingresar la cadena ó si quiere que se genere automáticamente. Si el usuario decide ingresarla el sistema pide la cadena.

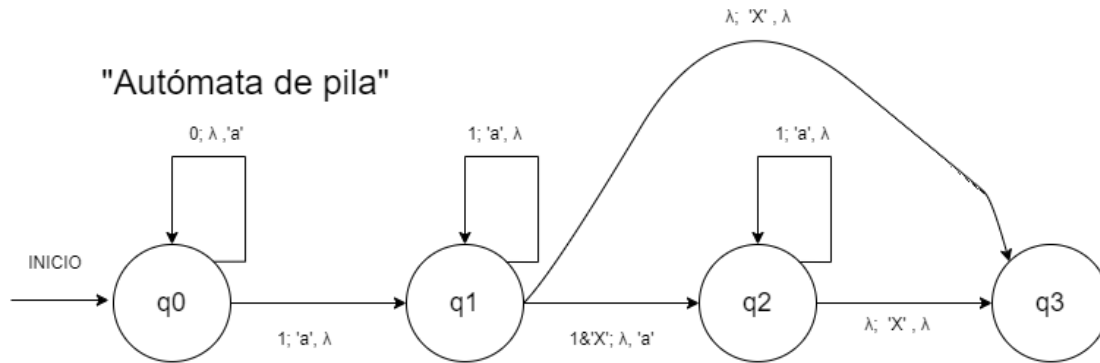
Si se debe generar automáticamente, se genera un número aleatorio del 1 al 100,000, que sera la longitud de la cadena. Se concatena a una cadena vacía un 0 o 1, de forma aleatoria "n"veces.

```

1 # Definimos estructura "Pila"
2 class Pila:
3     def __init__(self):
4         self.items = ['X'] # Instanciamos la pila con una 'X' en el fondo
5
6     def agregar(self, item): # Agregamos un elemento a la pila
7         self.items.append(item)
8
9     def extraer(self): # Extraemos el ultimo elemento de la pila
10        self.items.pop()
11
12    def inspeccionar(self): # Consultamos el tope de la pila
13        return self.items[len(self.items)-1]
14
15    def imprimir(self): # Imprimimos la pila completa
16        contenido = ''
17        for item in self.items[::-1]:
18            contenido += item
19        return contenido

```

Para definir la 'pila', usamos una clase que cuando se inicializa, comienza con un valor de "X", y definimos algunos métodos que definen la estructura de pila.



El anterior diagrama de es la representación del autómata de pila que propongo funciona para resolver el problema presentado. Es común encontrar un diagrama de autómata diferente para validar palindromos, sin embargo, creo que tiene un error, cuando recibe cadenas del tipo 0001111, (cuando tiene más 1's que 0's) la cadena es incorrecta y el autómata la acepta, por lo cual, se propone un autómata con cuatro estados para evitar este problema.

```

1  # Definimos el automata de pila
2  class PDA:
3      def __init__(self, cadena):
4          self.cadena = cadena # Cadena que se va a evaluar
5          self.q0 = True # Estado q0
6          self.q1 = False # Estado q1
7          self.q2 = False # Estado q2
8          self.q3 = False # Estado q3
9          self.stack = Pila() # Instanciamos una pila
10
11     def escribirArchivo(self, historial):
12         with open('./Automata de pila/Historial.txt', mode='w', encoding='utf-8')
13             as archivo:
14                 archivo.write(historial)
15                 archivo.close()
16
17     def evaluar(self): # Control de estados finitos
18         historial = '(q0 , '+cadena+' , '+str(self.stack.imprimir())+')\n'
19         for i in range(len(self.cadena)): # Recorremos la cadena
20             if self.q0: # Cuando estado q0 esta activo
21                 historial += '(q0 , '
22                 if self.cadena[i] == '0':
23                     self.stack.agregar('a')
24                 elif self.cadena[i] == '1' and self.stack.inspeccionar() == 'a':
25                     self.stack.extraer()
26                     self.q0 = False
27                     self.q1 = True
28             elif self.q1: # Cuando estado q1 esta activo
29                 historial += '(q1 , '
30                 if self.cadena[i] == '1' and self.stack.inspeccionar() == 'a':

```

```

30         self.stack.extraer()
31     elif self.cadena[i] == '1' and self.stack.inspeccionar() == 'X':
32         self.stack.agregar('a')
33         self.q1 = False
34         self.q2 = True
35     elif self.q2: # Cuando estado q2 esta activo
36         historial += '(q2 , '
37         if self.cadena[i] == '1':
38             self.stack.agregar('a')
39         historial += str(self.cadena[i+1:])+', '+str(self.stack.imprimir())
40         historial += ')\n'
41     if self.stack.inspeccionar() == 'X': # Cuando termina la cadena
42         verificamos el tope de la pila
43         self.q1 = False
44         self.q2 = False
45         self.q3 = True # Si el tope de la pila es 'X', la vaciamos y
46         aceptamos la cadena
47         historial += '(q3 , e , )\n'
48     if self.q3:
49         historial += 'Cadena Aceptada!'
50     else:
51         historial += 'Cadena Rechazada!'
52     self.escribirArchivo(historial)
53     return self.q3

```

En el fragmento de código anterior, mostramos el autómata de pila. Explicaremos sus partes. Vemos que el autómata también es una clase, dentro de su método constructor recibe una cadena que almacena en su atributo `cadena`. Tiene los cuatro estados descritos en el diagrama anterior, tomando valores booleanos (True, False), además, instancia una pila.

En su método `.escribirArchivo`, recibe una cadena llamada `historial`, lo que hace es abrir y/o crear un archivo llamado "Historial.txt", dentro del archivo escribe el historial y cierra el archivo.

El método `.evaluar`, es el control de estados finitos. Primero crea la variable `historial`, guarda la cadena con la que inicia y escribe el triplete que describe el ID. Posteriormente itera en cada elemento de la cadena. Con una estructura de control `if`, verifica en qué estado se encuentra el autómata.

- Si está en el estado `q0`, agrega al historial el estado actual, si lee '0' agrega una 'a' a la pila, si lee '1' y el tope de la pila es 'a', extrae el valor de la pila y pasa al estado `q1`
- Si está en el estado `q1`, agrega al historial el estado actual, si lee '1' y el tope de la pila es 'a', extrae el valor de la pila, si lee '1' y el tope de la pila es 'X' agrega una 'a' a la pila y pasa al estado `q2`
- Si está en el estado `q2`, agrega al historial el estado actual, si lee '1' agrega una 'a' a la pila.

Cuando termina de leer la cadena, que es lo mismo que leer un carácter vacío, revisa si hasta el tope de la pila hay una 'X', si lo hay, cambia al estado `q3`, se vacía la pila y se agrega el ID.

Finalmente, verifica que este en el estado q3, si lo esta significa que acepta la cadena, sino la rechaza. Escribe el historial en el archivo y retorna el estado q3.

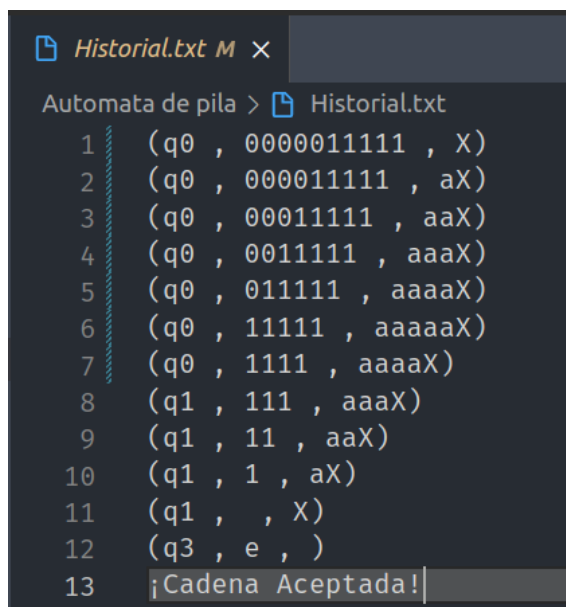
```
1 print(cadena) # Mostramos la cadena a evaluar
2 automata = PDA(cadena) # Instanciamos el automata con la cadena a evaluar
3 print(automata.evaluar()) # Usamos el metodo evaluar para saber si la cadena es
  valida o no
```

Finalmente, tomamos la cadena almacenada para evaluar, la pasamos al autómata "PDA", y ejecutamos su método evaluar que llama a la pila y a los demas métodos de su clase.

Veamos un ejemplo de cómo funciona el programa.

```
(base) bruno-rg@bruno-rg ~/Documents/6to Semestre/TC main /bin/python3 "/home/bruno-rg/
Documents/6to Semestre/TC/Automata de pila/AutomataDePila.py"
Automata de pila!
Deseas que el lenguaje se genere de forma manual o automática?
 0. Manual
 1. Automatica
0
Cadena binaria --> 0000011111
0000011111
True
```

Se seleccionó el modo manual, se ingreso la cadena binaria '0000011111' que es un palíndromo y la cadena fue aceptada.



```
Automata de pila > Historial.txt
1 (q0 , 0000011111 , X)
2 (q0 , 000011111 , aX)
3 (q0 , 00011111 , aaX)
4 (q0 , 0011111 , aaaX)
5 (q0 , 011111 , aaaaX)
6 (q0 , 11111 , aaaaaX)
7 (q0 , 1111 , aaaaX)
8 (q1 , 111 , aaaX)
9 (q1 , 11 , aaX)
10 (q1 , 1 , aX)
11 (q1 , , X)
12 (q3 , e , )
13 ¡Cadena Aceptada!
```

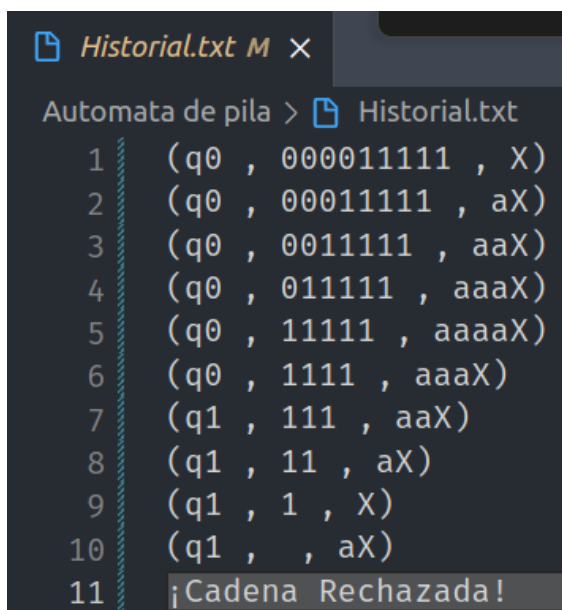
La salida del programa fue la anterior donde vemos la evaluación paso a paso del autómata y verifi-

camos que efectivamente, se vacio la pila y se aceptó.

Ahora, veamos un ejemplo donde la cadena no debería ser aceptada.

```
(base) bruno-rg@bruno-rg ~/Documents/6to Semestre/TC ➤ main ± /bin/python3 "/home/bruno-r
g/Documents/6to Semestre/TC/Automata de pila/AutomataDePila.py"
Automata de pila!
Deseas que el lenguaje se genere de forma manual o automática?
0. Manual
1. Automatica
0
Cadena binaria --> 000011111
000011111
False
```

Se seleccionó el modo manual, se ingreso la cadena binaria '000011111' que no es un palíndromo y la cadena fue rechazada.



```
Automata de pila > Historial.txt
1 (q0 , 000011111 , X)
2 (q0 , 00011111 , aX)
3 (q0 , 0011111 , aaX)
4 (q0 , 011111 , aaaX)
5 (q0 , 11111 , aaaaX)
6 (q0 , 1111 , aaaX)
7 (q1 , 111 , aaX)
8 (q1 , 11 , aX)
9 (q1 , 1 , X)
10 (q1 , , aX)
11 ¡Cadena Rechazada!
```

La salida del programa fue la anterior donde vemos la evaluación paso a paso del autómata y verificamos que efectivamente, la cadena se rechaza por que aún hay elementos en la pila.

3.2. Animación del autómatas:

```
1 import pygame
2 import threading
3
4 def animacion():
5     # Dimensiones de la ventana
6     tamaño = (800, 1000)
7
8     # Dimensiones del rectángulo
9     rectángulo = (int(tamaño[0]/5), int((tamaño[1]/10)*8))
10
11     # Abrimos archivo y preparamos los datos
12     datos = []
13     archivo = open('./Automata de pila/Historial.txt', mode='r', encoding='utf-8'
14         )
15     líneas = archivo.readlines()
16     archivo.close()
17     tamaño_pila = 0
18     for línea in líneas[:-1]:
19         dato = línea.replace('\n', '').replace('(', '').replace(')', '').replace(' ',
20             ), '').split(',')
21         if len(dato[2]) > tamaño_pila:
22             tamaño_pila = len(dato[2])
23         datos.append(dato)
24     resultado = líneas[-1]
25     # Inicializar Pygame
26     pygame.init()
27
28     # Crear la ventana
29     pantalla = pygame.display.set_mode(tamaño)
30     pygame.display.set_caption("Automata de pila")
31
32     # Colores
33     BLANCO = (255, 255, 255)
34     NEGRO = (0, 0, 0)
35     VERDE = (78, 205, 74)
36
37     # Crear un objeto de fuente
38     fuente = pygame.font.Font(None, 36)
39     fuente_cadena = pygame.font.Font(None, 36)
40     # Ciclo principal del juego
41     ejecucion = True
42     while ejecucion:
43         # Manejo de eventos
44         for evento in pygame.event.get():
45             if evento.type == pygame.QUIT:
46                 ejecucion = False
47
48         # Limpiar la ventana
```



```

47     pantalla.fill(BLANCO)
48
49     for iteracion in datos:
50         # Limpiar el texto
51         pantalla.fill(BLANCO)
52         estado_iteracion = iteracion[0]
53         cadena_iteracion = iteracion[1]
54         pila_iteracion = iteracion[2]
55
56         # Crear una superficie de texto
57         texto_superficie_resultado, texto_rectangulo_resultado =
            fuente_cadena.render(resultado, NEGRO)
58         # Centrar el texto en la coordenada
59         texto_rectangulo_resultado.center = (tamano[0]//2, tamano[1]//20)
60         # Dibujar el texto en la ventana
61         pantalla.blit(texto_superficie_resultado, texto_rectangulo_resultado)
62         # Dibujar el rectangulo
63         pygame.draw.rect(pantalla, NEGRO, ((tamano[0]/5)*3, (tamano[0]/10),
            rectangulo[0], rectangulo[1]), 3)
64
65         # Dibujar las cajas para los estados
66         ancho_division = rectangulo[1] // 4
67         for i in range(1, 4):
68             y = (ancho_division * i) + tamano[1]//10
69             pygame.draw.line(pantalla, NEGRO, ((tamano[0]/5)*3, y), ((tamano
                [0]/5)*4, y), width=3)
70
71         # Escribir el estado dentro de cada caja
72         x = tamano[0] // 5
73         estado = 0
74         for j in range(1,8,2):
75             estado_texto = 'q'+str(estado)
76             if estado_texto == estado_iteracion:
77                 pygame.draw.circle(pantalla, VERDE, (x*3+x/2, tamano[1]//10 +
                    (rectangulo[1]//4//2)*j), radius= 40 )
78
79                 texto = fuente.render(estado_texto, True, NEGRO)
80                 pantalla.blit(texto, (x*3+x/2 - 12, tamano[1]//10 + (rectangulo
                    [1]//4//2)*j - 15))
81                 estado += 1
82
83
84         x_abs_texto = tamano[0] // 5
85         x_abs_texto += x_abs_texto//2
86
87         y_abs_texto = tamano[1] // 10 * 9
88         alto_division = rectangulo[1]// tamano_pila // 2
89         k = 1
90         for c in pila_iteracion[::-1]:

```

```

91         # Crear una superficie de texto
92         texto_superficie, texto_rectangulo = fuente_cadena.render(c,
93             NEGRO)
94         # Centrar el texto en la coordenada
95         texto_rectangulo.center = (x_abs_texto, y_abs_texto -
96             alto_division*(k+1) + alto_division//2)
97         # Dibujar el texto en la ventana
98         pantalla.blit(texto_superficie, texto_rectangulo)
99         k += 1
100
101     # Crear una superficie de texto
102     texto_superficie, texto_rectangulo = fuente_cadena.render(
103         cadena_iteracion, NEGRO)
104     # Centrar el texto en la coordenada
105     texto_rectangulo.center = (tamano[0]//2, (tamano[1]//20) * 19)
106     # Dibujar el texto en la ventana
107     pantalla.blit(texto_superficie, texto_rectangulo)
108     # Actualizar la ventana
109     pygame.display.flip() https://www.overleaf.com/project/648
110     a212c22026a5b7e4a52fa
111
112     # Esperar un momento para que se muestre el texto
113     if estado_iteracion == 'q3':
114         pygame.time.wait(3000)
115     else:
116         pygame.time.wait(400)
117
118     # Salir del programa
119     pygame.quit()
120
121 if __name__ == '__main__':
122     # Crear un hilo para ejecutar la aplicacion de Pygame
123     pygame_thread = threading.Thread(target=animacion)
124     pygame_thread.start()
125
126     # Esperar hasta que el hilo de Pygame termine
127     pygame_thread.join()

```

En el anterior fragmento se encuentra el código programado para la animación del autómeta. Debido a que todo el código se encuentra comentado casi línea por línea, (donde no, es por que son calculos de ventana o posiciones de elementos), no veo la necesidad de volver a explicar código.

Veámos en imágenes la ejecución del programa.

