



Escuela Superior de Cómputo

TEORÍA DE LA COMPUTACIÓN

PRÁCTICA 6: *AUTÓMATA DE PILA*

Autor:

Rodrigo Gerardo Trejo Arriaga

Diciembre 2023

Práctica 6:

AUTÓMATA DE PILA

I. Introducción

Los autómatas de pila son un tipo de autómata abstracto que puede ser visto como una extensión de los autómatas finitos. Son particularmente útiles para el análisis de ciertos tipos de lenguajes formales como los lenguajes libres de contexto. Un autómata de pila hace uso de una pila para mantener un historial de información que le permite realizar transiciones que dependen no solo del estado actual y la entrada, sino también del contenido de la pila [1].

II. Definición Formal [2]

Un autómata de pila se define como una 6-tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$ donde:

- Q es un conjunto finito de estados.
- Σ es un conjunto finito de símbolos de entrada, conocido como el alfabeto de entrada.
- Γ es un conjunto finito de símbolos de pila.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados de aceptación.

III. Funcionamiento

El funcionamiento de un autómata de pila se basa en su capacidad para leer símbolos de entrada y manipular la pila según la función de transición δ . En cada paso, el autómata [2]:

1. Observa el símbolo actual de la entrada y el símbolo en la cima de la pila.
2. Realiza una transición a un nuevo estado y modifica la pila según δ .
3. Puede leer el siguiente símbolo de entrada o repetir el paso con el mismo símbolo.

IV. Ejemplo

Consideremos un autómata de pila simple que reconoce el lenguaje $\{a^n b^n | n \geq 0\}$. La idea es usar la pila para contar el número de a s y luego verificar que hay un número igual de b s.

V. Instrucciones

El programa diseñado debe cumplir con las siguientes especificaciones:

V.1. Entrada de la Cadena

- La cadena a evaluar puede ser ingresada por el usuario o generada automáticamente.
- En caso de generación automática, la longitud de la cadena no debe exceder los 100,000 caracteres.

V.2. Evaluación del Autómata

- El programa debe mostrar la evaluación del autómata a través de descripciones instantáneas (IDs).
- Esta evaluación debe ser mostrada tanto en pantalla como guardada en un archivo.

V.3. Animación del Autómata

- El autómata de pila debe ser animado si la longitud de la cadena es menor o igual a 10 caracteres.

VI. Resultados

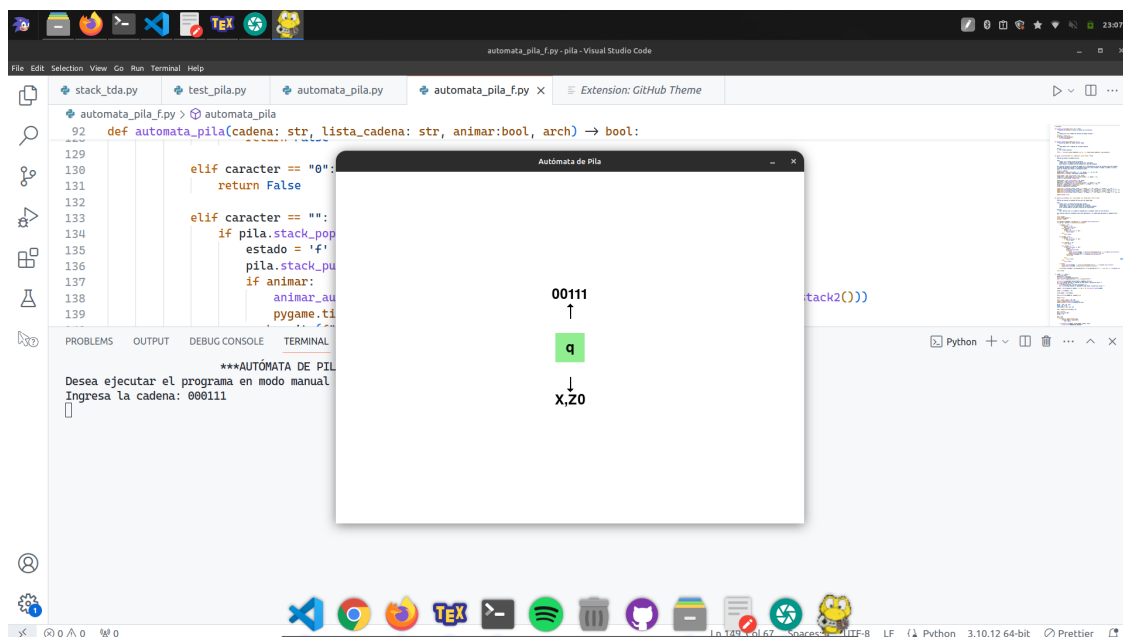


Figura 1: Modo manual – Animación de la pila

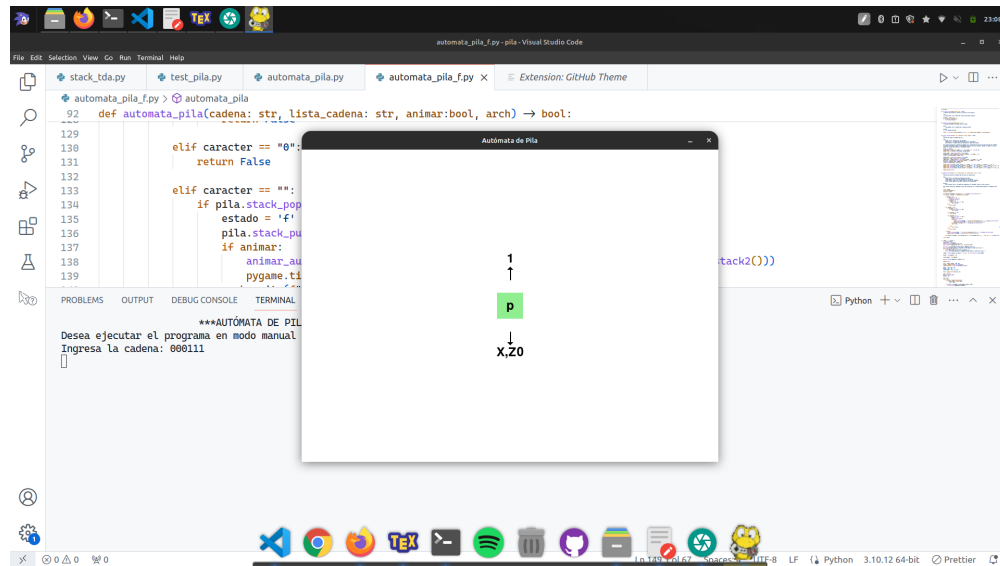


Figura 2: Modo manual - Animación de la pila

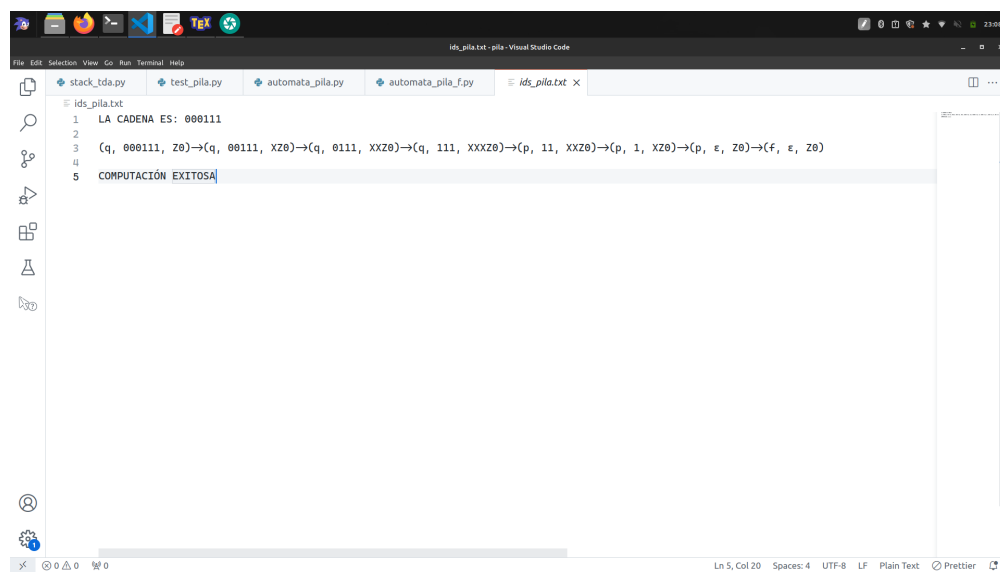


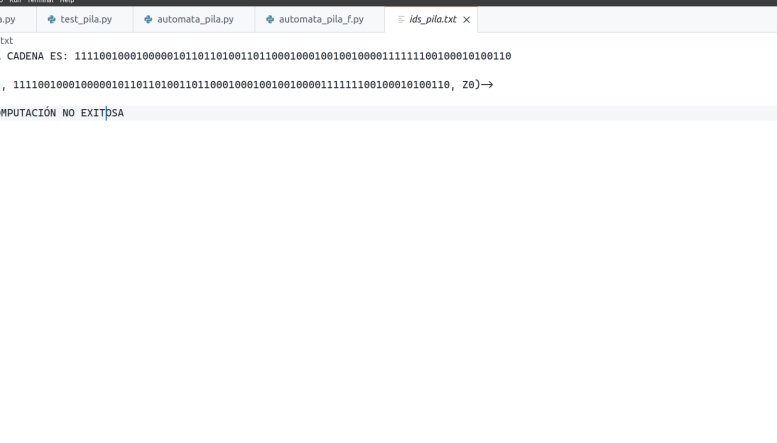
Figura 3: Archivo del modo manual

The screenshot displays the Visual Studio Code interface with a Python file named `automata_pila.py` open. The code implements a stack using a list. The `Stack` class has methods `push`, `pop`, and `isEmpty`. The `main` function tests the stack with a sequence of numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100. The terminal output shows the execution results, including the stack state after each operation.

```

1  # Implementación de una Pila (Stack) usando una lista
2
3  class Stack:
4      def __init__(self):
5          self.items = []
6
7      def push(self, item):
8          self.items.append(item)
9
10     def pop(self):
11         if not self.isEmpty():
12             return self.items.pop()
13         return None
14
15     def isEmpty(self):
16         return len(self.items) == 0
17
18     def __str__(self):
19         return str(self.items)
20
21     def __repr__(self):
22         return str(self.items)
23
24     def __len__(self):
25         return len(self.items)
26
27     def __getitem__(self, index):
28         return self.items[index]
29
30     def __setitem__(self, index, value):
31         self.items[index] = value
32
33     def __delitem__(self, index):
34         del self.items[index]
35
36     def __iter__(self):
37         return iter(self.items)
38
39     def __reversed__(self):
40         return reversed(self.items)
41
42     def __contains__(self, item):
43         return item in self.items
44
45     def __eq__(self, other):
46         return self.items == other.items
47
48     def __neq__(self, other):
49         return self.items != other.items
50
51     def __lt__(self, other):
52         return self.items < other.items
53
54     def __gt__(self, other):
55         return self.items > other.items
56
57     def __le__(self, other):
58         return self.items <= other.items
59
60     def __ge__(self, other):
61         return self.items >= other.items
62
63     def __add__(self, other):
64         return self.items + other.items
65
66     def __sub__(self, other):
67         return self.items - other.items
68
69     def __mul__(self, other):
70         return self.items * other.items
71
72     def __div__(self, other):
73         return self.items / other.items
74
75     def __mod__(self, other):
76         return self.items % other.items
77
78     def __pow__(self, other):
79         return self.items ** other.items
80
81     def __divmod__(self, other):
82         return self.items // other.items, self.items % other.items
83
84     def __radd__(self, other):
85         return other + self.items
86
87     def __rsub__(self, other):
88         return other - self.items
89
90     def __rmul__(self, other):
91         return other * self.items
92
93     def __rdiv__(self, other):
94         return other / self.items
95
96     def __rmod__(self, other):
97         return other % self.items
98
99     def __rpow__(self, other):
100        return other ** self.items
101
102     def __divmod__(self, other):
103        return other // self.items, other % self.items
104
105     def __rdivmod__(self, other):
106        return other // self.items, other % self.items
107
108     def __rdivmod__(self, other):
109        return other // self.items, other % self.items
110
111     def __rdivmod__(self, other):
112        return other // self.items, other % self.items
113
114     def __rdivmod__(self, other):
115        return other // self.items, other % self.items
116
117     def __rdivmod__(self, other):
118        return other // self.items, other % self.items
119
120     def __rdivmod__(self, other):
121        return other // self.items, other % self.items
122
123     def __rdivmod__(self, other):
124        return other // self.items, other % self.items
125
126     def __rdivmod__(self, other):
127        return other // self.items, other % self.items
128
129     def __rdivmod__(self, other):
130        return other // self.items, other % self.items
131
132     def __rdivmod__(self, other):
133        return other // self.items, other % self.items
134
135     def __rdivmod__(self, other):
136        return other // self.items, other % self.items
137
138     def __rdivmod__(self, other):
139        return other // self.items, other % self.items
140
141     def __rdivmod__(self, other):
142        return other // self.items, other % self.items
143
144     def __rdivmod__(self, other):
145        return other // self.items, other % self.items
146
147     def __rdivmod__(self, other):
148        return other // self.items, other % self.items
149
150     def __rdivmod__(self, other):
151        return other // self.items, other % self.items
152
153     def __rdivmod__(self, other):
154        return other // self.items, other % self.items
155
156     def __rdivmod__(self, other):
157        return other // self.items, other % self.items
158
159     def __rdivmod__(self, other):
160        return other // self.items, other % self.items
161
162     def __rdivmod__(self, other):
163        return other // self.items, other % self.items
164
165     def __rdivmod__(self, other):
166        return other // self.items, other % self.items
167
168     def __rdivmod__(self, other):
169        return other // self.items, other % self.items
170
171     def __rdivmod__(self, other):
172        return other // self.items, other % self.items
173
174     def __rdivmod__(self, other):
175        return other // self.items, other % self.items
176
177     def __rdivmod__(self, other):
178        return other // self.items, other % self.items
179
180     def __rdivmod__(self, other):
181        return other // self.items, other % self.items
182
183     def __rdivmod__(self, other):
184        return other // self.items, other % self.items
185
186     def __rdivmod__(self, other):
187        return other // self.items, other % self.items
188
189     def __rdivmod__(self, other):
190        return other // self.items, other % self.items
191
192     def __rdivmod__(self, other):
193        return other // self.items, other % self.items
194
195     def __rdivmod__(self, other):
196        return other // self.items, other % self.items
197
198     def __rdivmod__(self, other):
199        return other // self.items, other % self.items
200
201     def __rdivmod__(self, other):
202        return other // self.items, other % self.items
203
204     def __rdivmod__(self, other):
205        return other // self.items, other % self.items
206
207     def __rdivmod__(self, other):
208        return other // self.items, other % self.items
209
210     def __rdivmod__(self, other):
211        return other // self.items, other % self.items
212
213     def __rdivmod__(self, other):
214        return other // self.items, other % self.items
215
216     def __rdivmod__(self, other):
217        return other // self.items, other % self.items
218
219     def __rdivmod__(self, other):
220        return other // self.items, other % self.items
221
222     def __rdivmod__(self, other):
223        return other // self.items, other % self.items
224
225     def __rdivmod__(self, other):
226        return other // self.items, other % self.items
227
228     def __rdivmod__(self, other):
229        return other // self.items, other % self.items
230
231     def __rdivmod__(self, other):
232        return other // self.items, other % self.items
233
234     def __rdivmod__(self, other):
235        return other // self.items, other % self.items
236
237     def __rdivmod__(self, other):
238        return other // self.items, other % self.items
239
240     def __rdivmod__(self, other):
241        return other // self.items, other % self.items
242
243     def __rdivmod__(self, other):
244        return other // self.items, other % self.items
245
246     def __rdivmod__(self, other):
247        return other // self.items, other % self.items
248
249     def __rdivmod__(self, other):
250        return other // self.items, other % self.items
251
252     def __rdivmod__(self, other):
253        return other // self.items, other % self.items
254
255     def __rdivmod__(self, other):
256        return other // self.items, other % self.items
257
258     def __rdivmod__(self, other):
259        return other // self.items, other % self.items
260
261     def __rdivmod__(self, other):
262        return other // self.items, other % self.items
263
264     def __rdivmod__(self, other):
265        return other // self.items, other % self.items
266
267     def __rdivmod__(self, other):
268        return other // self.items, other % self.items
269
270     def __rdivmod__(self, other):
271        return other // self.items, other % self.items
272
273     def __rdivmod__(self, other):
274        return other // self.items, other % self.items
275
276     def __rdivmod__(self, other):
277        return other // self.items, other % self.items
278
279     def __rdivmod__(self, other):
280        return other // self.items, other % self.items
281
282     def __rdivmod__(self, other):
283        return other // self.items, other % self.items
284
285     def __rdivmod__(self, other):
286        return other // self.items, other % self.items
287
288     def __rdivmod__(self, other):
289        return other // self.items, other % self.items
290
291     def __rdivmod__(self, other):
292        return other // self.items, other % self.items
293
294     def __rdivmod__(self, other):
295        return other // self.items, other % self.items
296
297     def __rdivmod__(self, other):
298        return other // self.items, other % self.items
299
300     def __rdivmod__(self, other):
301        return other // self.items, other % self.items
302
303     def __rdivmod__(self, other):
304        return other // self.items, other % self.items
305
306     def __rdivmod__(self, other):
307        return other // self.items, other % self.items
308
309     def __rdivmod__(self, other):
310        return other // self.items, other % self.items
311
312     def __rdivmod__(self, other):
313        return other // self.items, other % self.items
314
315     def __rdivmod__(self, other):
316        return other // self.items, other % self.items
317
318     def __rdivmod__(self, other):
319        return other // self.items, other % self.items
320
321     def __rdivmod__(self, other):
322        return other // self.items, other % self.items
323
324     def __rdivmod__(self, other):
325        return other // self.items, other % self.items
326
327     def __rdivmod__(self, other):
328        return other // self.items, other % self.items
329
330     def __rdivmod__(self, other):
331        return other // self.items, other % self.items
332
333     def __rdivmod__(self, other):
334        return other // self.items, other % self.items
335
336     def __rdivmod__(self, other):
337        return other // self.items, other % self.items
338
339     def __rdivmod__(self, other):
340
```

Figura 4: Modo automático



```
1 LA CADENA ES: 1111001000100000101101101001101100010001001000011111100100010100110
2
3 (q, 1111001000100000101101101001101100010001001000011111100100010100110, Z0)→
4
5 COMPUTACIÓN NO EXITOSA
```

Figura 5: Archivo del modo automático

Conclusión

La práctica con autómatas de pila ha proporcionado una comprensión profunda de su importancia en la teoría de la computación y su aplicación en el análisis de lenguajes libres de contexto. A través de la implementación de un autómata de pila que valida cadenas en el lenguaje específico 0^n1^n , hemos podido observar de primera mano cómo estos modelos teóricos pueden ser transformados en herramientas de software efectivas.

Esta práctica ha demostrado que, aunque los autómatas de pila son conceptualmente más complejos que los autómatas finitos, su capacidad para manejar una memoria auxiliar en forma de pila les permite reconocer patrones y estructuras que van más allá de las capacidades de los autómatas finitos. En particular, la habilidad para equilibrar y comparar segmentos de una cadena basándose en su longitud y contenido es fundamental en el análisis sintáctico en compiladores y en el procesamiento del lenguaje natural.

De esta manera, esta práctica ha reforzado la relevancia de los autómatas de pila en la computación teórica y su aplicabilidad en problemas del mundo real, proveyendo una base sólida para futuras exploraciones en el campo de la teoría de la computación y el diseño de lenguajes de programación.

Bibliografía

- [1] Wikipedia. "Autómata con pila - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. Accedido el 23 de diciembre de 2023. [En línea]. Disponible: https://es.wikipedia.org/wiki/Autómata_con_pila
- [2] "Automata de pila". Lenguajes Formales y Autómatas. Accedido el 23 de diciembre de 2023. [En línea]. Disponible: https://ivanvladimir.gitlab.io/lfya_book/docs/06dependedelcontexto/03autómatadepila-de-terministico/

VII. Anexo – Código de Implementación

```

1      '''
2
3      INSTITUTO POLITECNICO NACIONAL
4      ESCUELA SUPERIOR DE COMPUTO
5
6      INGENIERIA EN INTELIGENCIA ARTIFICIAL
7
8      TEORIA DE LA COMPUTACION
9      AUTOMATA DE PILA
10
11     GRUPO: 5BM1
12     ALUMNO: TREJO ARRIAGA RODRIGO GERARDO
13
14     ESTE PROGRAMA GENERA UN AUTOMATA DE PILA QUE VALIDA SI UNA
15         CADENA PERTENECE AL LENGUAJE 0^n1^n Y:
16     i) SOLICITA AL USUARIO UNA CADENA A VALIDAR O LA GENERA DE
17         MANERA RANDOM
18     ii) ANIMA EL AUTOMATA DE PILA SI LA LONGITUD DE LA PALABRA ES
19         MENOR A 10
20     iii) GENERA LA HISTORIA DE ID's EN UN ARCHIVO DE TEXTO
21
22     ULTIMA MODIFICACION: 23/12/2023
23     '''
24
25     #
26     -----
27
28     # MODULOS Y LIBRERIAS IMPORTADAS
29
30     import pygame
31     import sys
32     from stack_tda import Stack
33     import os
34     import random
35
36     #
37     -----
38
39     # FUNCIONES
40
41     def eliminar_archs(nombre_arch: str) -> None:
42         """Funcion que elimina un archivo si existe en el
43             directorio
44
45             Args:
46             nombre_arch (str): Nombre del archivo que deseas eliminar
47         """
48         archivo1 = nombre_arch
49         if os.path.exists(archivo1):
50             os.remove(archivo1)

```

```
44
45     def generar_cadena(long_cadena:int) -> str:
46         """Funcion que genera la cadena binaria random
47
48         Args:
49             long_cadena (int): Longitud de la palabra binaria
50
51         Returns:
52             str: Palabra binarias
53         """
54         return ''.join([str(random.randint(0, 1)) for _ in range(
55             random.randint(2, long_cadena))])
56
57     def animar_automata(estado: str, cadena:str, pila: Stack)
58         -> None:
59         """
60         Funcion que anima el automata de pila.
61
62         Args:
63             estado (str): Estado actual del automata.
64             cadena (str): La cadena de entrada que se esta procesando.
65             pila (Stack): El objeto pila que representa la pila del
66                 automata.
67
68         Esta funcion actualiza la ventana de pygame con la
69             representacion visual del estado actual del automata,
70             la cadena de entrada restante y el contenido actual de la
71             pila. Dibuja el estado, la cadena y la pila,
72             junto con flechas para indicar el movimiento actual.
73         """
74         screen.fill(WHITE)
75         estado_autom = pygame.Rect(width // 2 - 25, height // 2 -
76             25, 50, 50)
77         pygame.draw.rect(screen, LIGHT_GREEN, estado_autom)
78
79         texto_estado = font.render(estado, True, BLACK)
80         estado_rect = texto_estado.get_rect(center=(width // 2,
81             height // 2))
82         screen.blit(texto_estado, estado_rect)
83
84         cadena_letra = font.render(cadena, True, BLACK)
85         pila_letra = font.render(pila, True, BLACK)
86         cadena_rect = cadena_letra.get_rect(center=(width // 2,
87             height // 2 - 90))
88         pila_rect = pila_letra.get_rect(center=(width // 2, height
89             // 2 + 90))
90         screen.blit(cadena_letra, cadena_rect)
91         screen.blit(pila_letra, pila_rect)
92
93         pygame.draw.line(screen, BLACK, (width // 2, height // 2 -
94             50), (width // 2, height // 2 - 70), 2)
95         pygame.draw.polygon(screen, BLACK, [(width // 2, height //
96             2 - 75), (width // 2 - 5, height // 2 - 70), (width // 2
```



```

    + 5, height // 2 - 70]])
87     pygame.draw.line(screen, BLACK, (width // 2, height // 2 +
        50), (width // 2, height // 2 + 70), 2)
88     pygame.draw.polygon(screen, BLACK, [(width // 2, height //
        2 + 75), (width // 2 - 5, height // 2 + 70), (width // 2
        + 5, height // 2 + 70)])
89
90     pygame.display.flip()
91
92
93     def automata_pila(cadena: str, lista_cadena: str, animar:bool,
        arch) -> bool:
94         """
95         Funcion que ejecuta el automata de pila para una cadena
            dada.
96
97         Args:
98         cadena (str): La cadena de entrada para validar.
99         lista_cadena (str): Lista de caracteres de la cadena.
100        animar (bool): Indica si se debe animar el proceso del
            automata.
101        arch: Archivo donde se escribe el historial de transiciones
            .
102
103        Returns:
104        bool: Retorna True si la cadena es aceptada por el automata
            , False en caso contrario.
105
106        Esta funcion ejecuta el automata de pila para determinar si
            la cadena dada pertenece al lenguaje 0^n1^n.
107        """
108
109        global estado
110        lista_cadena.append("")
111        str_aux = cadena
112
113        arch.write(f"({estado}, {str_aux[:]}), {''.join(pila.
            read_stack2())})->")
114        for num_car, caracter in enumerate(lista_cadena):
115
116            if estado == "q":
117                if caracter == "0":
118                    pila.stack_push("X")
119                elif caracter == "1":
120                    estado = "p"
121                if pila.stack_pop() == "Z0":
122                    return False
123                else:
124                    return False
125
126            elif estado == "p":
127                if caracter == "1":
128                    if pila.stack_pop() == "Z0":
129                        return False

```

```
130
131     elif caracter == "0":
132         return False
133
134     elif caracter == "":
135         if pila.stack_pop() == "Z0":
136             estado = 'f'
137             pila.stack_push('Z0')
138             if animar:
139                 animar_automata(estado, ''.join(lista_cadena[num_car:]), ',
140                     '.join(pila.read_stack2()))
141                 pygame.time.delay(1000) # Retardo para visualizar cada
142                     paso
143                 arch.write(f"({estado}, e, {''.join(pila.read_stack2())})")
144             return True
145
146     else:
147         return False
148
149     if animar:
150         animar_automata(estado, ''.join(lista_cadena[num_car+1:]),
151             ', '.join(pila.read_stack2()))
152         pygame.time.delay(1000) # Retardo para visualizar cada
153             paso
154
155         arch.write(f"({estado}, {str_aux[num_car+1:] if str_aux[
156             num_car+1:] != '' else 'e'}, {''.join(pila.read_stack2(
157             )))>->")
158
159     return False
160
161 if __name__ == "__main__":
162     os.system('clear')
163     nombre_arch = "ids_pila"
164     eliminar_archs(f"{nombre_arch}.txt")
165     arch = open(f"{nombre_arch}.txt", "a+", encoding="utf-8")
166
167     print("\t\t\t\t\t***AUTOMATA DE PILA PARA EL LENGUAJE [0^n1^n]"
168         )
169     op = input("Desea ejecutar el programa en modo manual o
170         automatico? [a/m]: ")
171     while op!="a" and op != "m":
172         print("Modo invalido, intentelo nuevamente :")
173         op = input("Desea ejecutar el programa en modo manual o
174             automatico? [a/m]: ")
175
176     cadena = input("Ingresa la cadena: ") if op == "m" else
177         generar_cadena(100000)
178
179     print(f"La cadena es {cadena}") if op == "a" else None
```

```
174     animar = len(cadena) <= 10
175
176     lista_cadena = list(cadena)
177
178     arch.write(f"LA CADENA ES: {cadena}\n\n")
179
180     pygame.init()
181
182     size = width, height = 800, 600
183     screen = pygame.display.set_mode(size)
184     pygame.display.set_caption("Automata de Pila")
185
186     WHITE = (255, 255, 255)
187     BLACK = (0, 0, 0)
188     LIGHT_GREEN = (144, 238, 144)
189
190     font = pygame.font.Font(None, 36)
191
192     pila = Stack()
193     pila.stack_push("Z0")
194     estado = "q"
195
196     run = True
197     while run:
198         for event in pygame.event.get():
199             if event.type == pygame.QUIT:
200                 run = False
201
202         if automata_pila(cadena, lista_cadena, animar, arch):
203             print("\n\t>>>COMPUTACION EXITOSA")
204             arch.write("\n\nCOMPUTACION EXITOSA")
205         else:
206             print("\n\t>>>COMPUTACION NO EXITOSA")
207             arch.write("\n\nCOMPUTACION NO EXITOSA")
208
209     run = False
210
211     pygame.quit()
212     sys.exit()
```