Instituto Politécnico Nacional

Escuela Superior de Cómputo

Teoría de la Computación

Programas Bloque 2

Profesor: Dr. Juarez Martinez Genaro

Alumno: Yangfeng Chen

chen1436915478@gmail.com

GRUPO
5BM2

4 de enero de 2025

Índice

1. Introducción

En el presente reporte se documenta el desarrollo e implementación de una serie de programas diseñados para resolver problemas fundamentales de la teoría de autómatas y lenguajes formales. Estos problemas incluyen el diseño y simulación de autómatas finitos deterministas y no deterministas, autómatas de pila, gramáticas en notación Backus-Naur y máquinas de Turing. Cada programa aborda un tema clave en esta área, implementando las soluciones con rigor matemático y técnico, y generando resultados que validan las propiedades de los modelos computacionales correspondientes.

El reporte incluye la descripción teórica, los cálculos y procesos involucrados, así como la implementación en código fuente y los resultados obtenidos. Los programas desarrollados son los siguientes:

1. **Buscador de Palabras mediante un Autómata Finito Determinista (DFA):** Este programa implementa un autómata que reconoce un conjunto predefinido de palabras relacionadas con violencia de género. Incluye el diseño del NFA inicial, su conversión a DFA mediante el método de subconjuntos, y una funcionalidad que permite identificar, contar y ubicar estas palabras en un archivo de texto, documentando cada paso del análisis.

2. **Reconocimiento de Lenguaje con un Autómata de Pila (PDA):** Este programa simula un autómata de pila que reconoce cadenas del lenguaje $\{0^n1^n | n \geq 1\}$. Se incluye una animación para cadenas cortas y un registro detallado de las transiciones y descripciones instantáneas, tanto en pantalla como en archivos.

3. **Derivación de Gramáticas en Notación Backus-Naur:** Este programa genera derivaciones automáticas de la gramática definida para un condicional IF. El usuario puede especificar el número de derivaciones deseadas, y el programa registra cada paso del proceso, además de generar un pseudo-código correspondiente a la gramática derivada.

4. **Simulación de una Máquina de Turing:** Este programa implementa una máquina de Turing que reconoce el lenguaje $\{0^n1^n | n \geq 1\}$, basado en el diseño del ejercicio 8.2 del libro de John Hopcroft. El programa genera las descripciones instantáneas de cada paso de la computación y permite la animación del proceso para cadenas pequeñas.

2. Marco Teórico

2.1. Introducción de NFA a DFA

En la teoría de autómatas y lenguajes formales, los autómatas finitos son modelos matemáticos fundamentales utilizados para representar lenguajes regulares. Los dos tipos principales de autómatas finitos son los **Autómatas Finitos No Deterministas (NFA)** y los **Autómatas Finitos Deterministas (DFA)**. Aunque los NFA y DFA tienen diferencias estructurales y operativas, ambos son equivalentes en poder expresivo, lo que significa que cualquier lenguaje que puede ser reconocido por un NFA también puede ser reconocido por un DFA. La conversión de un NFA a un DFA es un procedimiento importante que demuestra esta equivalencia.

2.2. Definición de NFA

Un **Autómata Finito No Determinista (NFA)** se define como una quíntupla $M = (Q, \Sigma, \delta, q_0, F)$, donde:

- Q: Conjunto finito de estados.
- Σ: Alfabeto finito de entrada.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$: Función de transición, que puede asignar un conjunto de estados a un par estado-símbolo.
- $q_0 \in Q$: Estado inicial.
- $F \subseteq Q$: Conjunto de estados de aceptación.

En un NFA, las transiciones pueden ser no deterministas, permitiendo múltiples estados de destino o incluso transiciones vacías (ϵ-transiciones).

2.3. Definición de DFA

Un **Autómata Finito Determinista (DFA)** es una variante restringida de los autómatas finitos y se define como una quíntupla $M = (Q, \Sigma, \delta, q_0, F)$, donde:

- Q: Conjunto finito de estados.
- Σ: Alfabeto finito de entrada.
- $\delta : Q \times \Sigma \to Q$: Función de transición determinista, que asigna exactamente un estado a cada par estado-símbolo.
- $q_0 \in Q$: Estado inicial.
- $F \subseteq Q$: Conjunto de estados de aceptación.

A diferencia del NFA, un DFA no permite transiciones ambiguas ni transiciones vacías.

Aunque los NFA pueden parecer más potentes debido a su flexibilidad, ambos tipos de autómatas son equivalentes en términos de los lenguajes que pueden reconocer. Esto significa que, dado un NFA, siempre es posible construir un DFA que acepte el mismo lenguaje.

2.4. Proceso de Conversión: NFA a DFA

El algoritmo de conversión de NFA a DFA se basa en la construcción del **Autómata del Conjunto de Estados (Subset Construction)**. Este proceso implica crear un DFA cuyas transiciones reflejan los subconjuntos de estados alcanzables en el NFA. Los pasos principales son los siguientes:

1. **Estado Inicial**: El estado inicial del DFA corresponde al conjunto de estados alcanzables desde el estado inicial del NFA mediante ϵ-transiciones.

2. **Construcción de Estados**: Los estados del DFA representan subconjuntos de estados del NFA. Cada subconjunto de Q es un posible estado del DFA.

3. **Transiciones**:

   - Para cada estado del DFA (un subconjunto de Q) y cada símbolo $a \in \Sigma$, se calcula el conjunto de estados alcanzables en el NFA desde los estados del subconjunto actual mediante una transición con a, seguido de ϵ-transiciones.

4. **Estados de Aceptación**: Un estado del DFA se considera de aceptación si contiene al menos un estado de aceptación del NFA.

5. **Optimización**: En muchos casos, el DFA resultante puede tener estados innecesarios o redundantes, que pueden ser eliminados mediante técnicas de minimización.

2.5. Autómata de Pila (PDA)

El autómata de pila es un modelo computacional ampliamente utilizado en la teoría de lenguajes formales y autómatas para reconocer lenguajes libres de contexto. Estos lenguajes son un nivel más complejo en la jerarquía de Chomsky que los lenguajes regulares y son esenciales para describir estructuras jerárquicas como las gramáticas de los lenguajes de programación, expresiones matemáticas y estructuras anidadas.

Un **Autómata de Pila (PDA, por sus siglas en inglés)** es un autómata finito extendido con una pila, que proporciona memoria adicional para el procesamiento de cadenas. Formalmente, un PDA se define como una 7-tupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

donde:

- Q: Conjunto finito de estados.
- Σ: Alfabeto de entrada.
- Γ: Alfabeto de la pila.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$: Función de transición, que describe cómo el autómata cambia de estado y modifica la pila.
- $q_0 \in Q$: Estado inicial.
- $Z_0 \in \Gamma$: Símbolo inicial de la pila.

- $F \subseteq Q$: Conjunto de estados finales.

El PDA se caracteriza por su capacidad de manipular una pila de manera que puede empujar (push) o sacar (pop) elementos de esta estructura, permitiendo almacenar información sobre el contexto.

2.6. Funcionamiento de un Autómata de Pila

El autómata de pila procesa una cadena de entrada y toma decisiones basadas en tres elementos:

1. El símbolo actual de la entrada.
2. El estado actual del autómata.
3. El símbolo en la parte superior de la pila.

La función de transición (δ) puede realizar las siguientes operaciones:

- Cambiar de estado.
- Empujar uno o más símbolos en la pila.
- Sacar (hacer pop) un símbolo de la pila.
- Leer un símbolo de entrada o ignorarlo (ϵ-transiciones).

El PDA acepta una cadena de entrada si:

1. Llega a un estado de aceptación después de procesar toda la entrada (aceptación por estado final), o
2. La pila está vacía (aceptación por pila vacía).

2.7. Notación Backus-Naur

La Notación Backus-Naur (BNF) es una forma estándar de expresar gramáticas libres de contexto de manera precisa y legible. Fue desarrollada inicialmente para describir la sintaxis del lenguaje de programación ALGOL, pero su uso se ha extendido ampliamente en la especificación de lenguajes de programación y lenguajes formales.

Una gramática en notación Backus-Naur consta de los siguientes elementos:

- **Símbolos no terminales**: Representan categorías abstractas o partes de la estructura que necesitan ser desarrolladas. Se denotan típicamente con letras mayúsculas (por ejemplo, S, A).
- **Símbolos terminales**: Son los elementos finales del lenguaje y no pueden derivarse más. Por ejemplo, palabras clave como `if`, `else` o símbolos como `;`.
- **Reglas de producción**: Describen cómo un símbolo no terminal puede ser sustituido por una secuencia de símbolos terminales y/o no terminales.
- **Símbolo inicial**: Es el punto de partida para generar cadenas en el lenguaje.

2.8. Máquina de Turing

Una **Máquina de Turing (MT)** es un autómata teórico que opera sobre una cinta infinita dividida en celdas. Cada celda puede contener un símbolo de un alfabeto finito, y la máquina tiene un cabezal de lectura/escritura que puede moverse hacia la izquierda o hacia la derecha.

Formalmente, una MT se define como una 7-tupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

donde:

- Q: Conjunto finito de estados.
- Σ: Alfabeto de entrada (excluye el símbolo blanco B).
- Γ: Alfabeto de la cinta ($\Sigma \subseteq \Gamma$ y contiene B).
- $\delta : \{Q, \Gamma\} \to \{Q, \Gamma, \{L, R\}\}$: Función de transición, que indica el cambio de estado, el símbolo que se escribe en la cinta y el movimiento del cabezal (L: izquierda, R: derecha).
- $q_0 \in Q$: Estado inicial.
- B: Símbolo blanco de la cinta.
- F: Estados finales.

El funcionamiento de una MT se describe mediante los siguientes pasos:

1. La máquina comienza en el estado inicial q_0, con el cabezal apuntando al primer símbolo de la cadena de entrada en la cinta.
2. En cada paso, la máquina lee el símbolo en la celda actual, consulta la función de transición (δ) y realiza tres acciones:
   - Cambia al estado indicado.
   - Escribe un símbolo en la celda actual.
   - Mueve el cabezal hacia la izquierda (L) o hacia la derecha (R).
3. El proceso continúa hasta que la máquina alcanza un estado de aceptación (F) o hasta que alcanza una transición invalida.

3. Desarrollo

3.1. Programa 3. Buscador de palabras

El objetivo del programa es leer un archivo de texto o leer el texto plano de la web, identificar las ocurrencias de las palabras claves, contar cuántas veces aparecen e indicar la posición de cada ocurrencia en el archivo. Además, se generará un archivo de salida que muestre el proceso de evaluación del autómata para cada carácter leído, incluyendo el cambio de estados (historial de cómputo).

3.1.1. Diseño del NFA

El primer paso es modelar un NFA que reconozca la unión de las palabras del conjunto dado. Cada palabra puede verse como una secuencia de caracteres, donde para su reconocimiento se necesita un camino que, partiendo de un estado inicial, avance sobre transiciones etiquetadas con los caracteres correspondientes, hasta llegar a un estado final.

En este caso usaremos el conjunto de palabras: {acoso, acecho, agresión, víctima, violación, violencia, machista}.

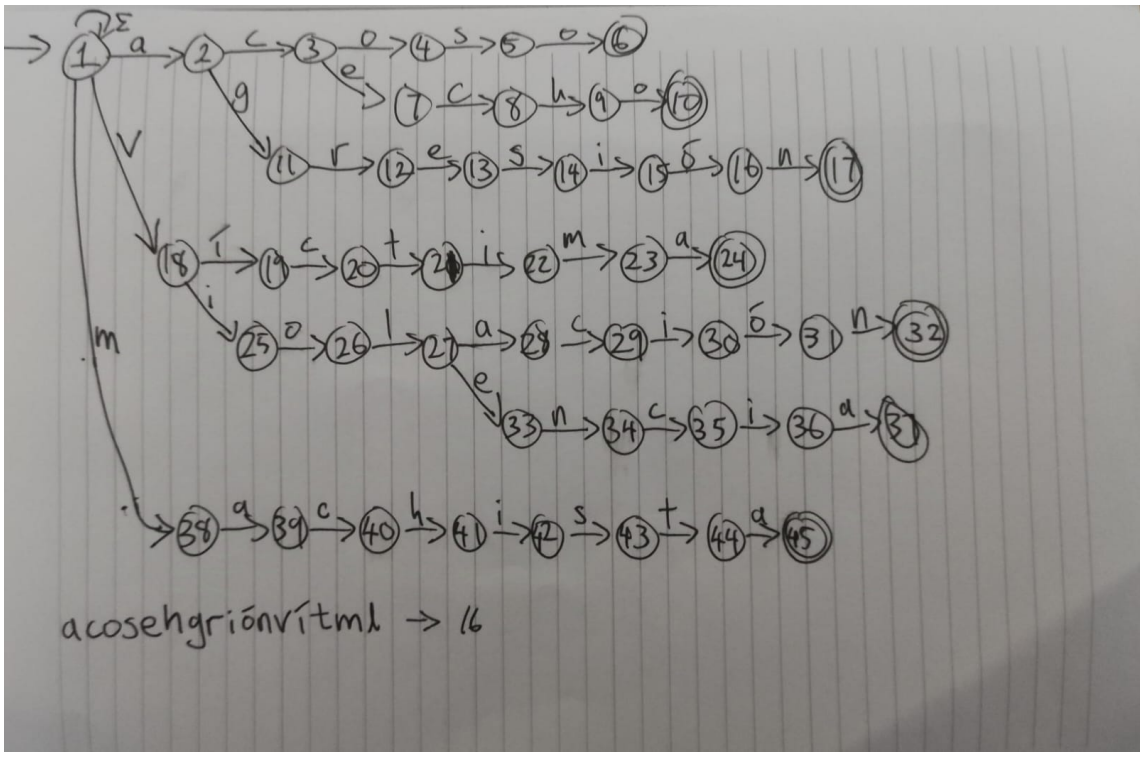

Figura 1: Diseño de la grafica del NFA segun el conjunto de palabras.

3.1.2. Transformación del NFA a DFA

Una vez construido el NFA, se procede a la conversión al DFA mediante el método de subconjuntos.

1. Tomar el conjunto de estados ϵ-cerradura (alcanzables mediante transiciones épsilon) desde el estado inicial del NFA como el estado inicial del DFA.

2. Para cada símbolo del alfabeto (en este caso, caracteres que aparecen en las palabras clave), se calculan las transiciones del NFA y se encuentra la ϵ-cerradura de los estados resultantes. Esto genera un nuevo conjunto de estados que se convierte en un estado del DFA.

3. El proceso se repite hasta que no surjan nuevos conjuntos de estados.

		Alfabeto															
Estados		a	c	o	s	e	h	g	r	i	ó	n	v	í	t	m	l
1	{1}	{1, 2}											{1, 18}			{1, 38}	
2	{1, 2}	{1, 2}	{1, 3}					{1, 11}					{1, 18}			{1, 38}	
3	{1, 18}	{1, 2}								{1, 25}			{1, 18}	{1, 19}		{1, 38}	
4	{1, 38}	{1, 2, 39}											{1, 18}			{1, 38}	
5	{1, 3}	{1, 2}		{1, 4}		{1, 7}							{1, 18}			{1, 38}	
6	{1, 11}	{1, 2}							{1, 12}				{1, 18}			{1, 38}	
7	{1, 25}	{1, 2}		{1, 26}									{1, 18}			{1, 38}	
8	{1, 19}	{1, 2}	{1, 20}										{1, 18}			{1, 38}	
9	{1, 2, 39}	{1, 2}	{1, 3, 40}					{1, 11}					{1, 18}			{1, 38}	
10	{1, 4}	{1, 2}			{1, 5}								{1, 18}			{1, 38}	
11	{1, 7}	{1, 2}	{1, 8}										{1, 18}			{1, 38}	
12	{1, 12}	{1, 2}				{1, 13}							{1, 18}			{1, 38}	
13	{1, 26}	{1, 2}											{1, 18}			{1, 38}	{1, 27}
14	{1, 20}	{1, 2}											{1, 18}		{1, 21}	{1, 38}	
15	{1, 3, 40}	{1, 2}		{1, 4}		{1, 7}	{1, 41}						{1, 18}			{1, 38}	
16	{1, 5}	{1, 2}		{1, 6}									{1, 18}			{1, 38}	
17	{1, 8}	{1, 2}					{1, 9}						{1, 18}			{1, 38}	
18	{1, 13}	{1, 2}			{1, 14}								{1, 18}			{1, 38}	
19	{1, 27}	{1, 2, 28}				{1, 33}							{1, 18}			{1, 38}	
20	{1, 21}	{1, 2}								{1, 22}			{1, 18}			{1, 38}	
21	{1, 41}	{1, 2}								{1, 42}			{1, 18}			{1, 38}	
22	{1, 6}	{1, 2}											{1, 18}			{1, 38}	
23	{1, 9}	{1, 2}		{1, 10}									{1, 18}			{1, 38}	
24	{1, 14}	{1, 2}								{1, 15}			{1, 18}			{1, 38}	
25	{1, 2, 28}	{1, 2}	{1, 3, 29}					{1, 11}					{1, 18}			{1, 38}	
26	{1, 33}	{1, 2}										{1, 34}	{1, 18}			{1, 38}	
27	{1, 22}	{1, 2}											{1, 18}			{1, 23, 38}	
28	{1, 42}	{1, 2}			{1, 43}								{1, 18}			{1, 38}	
29	{1, 10}	{1, 2}											{1, 18}			{1, 38}	
30	{1, 15}	{1, 2}									{1, 16}		{1, 18}			{1, 38}	
31	{1, 3, 29}	{1, 2}		{1, 4}		{1, 7}				{1, 30}			{1, 18}			{1, 38}	
32	{1, 34}	{1, 2}	{1, 35}										{1, 18}			{1, 38}	
33	{1, 23, 38}	{1, 2, 24, 39}											{1, 18}			{1, 38}	
34	{1, 43}	{1, 2}											{1, 18}		{1, 44}	{1, 38}	
35	{1, 16}	{1, 2}										{1, 17}	{1, 18}			{1, 38}	
36	{1, 30}	{1, 2}									{1, 31}		{1, 18}			{1, 38}	
37	{1, 35}	{1, 2}								{1, 36}			{1, 18}			{1, 38}	
38	{1, 2, 24, 39}	{1, 2}	{1, 3, 40}					{1, 11}					{1, 18}			{1, 38}	
39	{1, 44}	{1, 2, 45}											{1, 18}			{1, 38}	
40	{1, 17}	{1, 2}											{1, 18}			{1, 38}	
41	{1, 31}	{1, 2}										{1, 32}	{1, 18}			{1, 38}	
42	{1, 36}	{1, 2, 37}											{1, 18}			{1, 38}	
43	{1, 2, 45}	{1, 2}	{1, 3}					{1, 11}					{1, 18}			{1, 38}	
44	{1, 32}	{1, 2}											{1, 18}			{1, 38}	
45	{1, 2, 37}	{1, 2}	{1, 3}					{1, 11}					{1, 18}			{1, 38}	

Figura 2: Tabla de subconjuntos

En los espacios en blanco, son subconjuntos {1} o en la figura 3 son celdas para el nodo 1.

Las celdas marcadas con el color amarillo son estados finales para la DFA.

		Alfabeto															
Estados		a	c	o	s	e	h	g	r	i	ó	n	v	í	t	m	l
1	1	2											3			4	
2	2	2	5					6					3			4	
3	3	2								7			3	8		4	
4	4	9											3			4	
5	5	2		10		11							3			4	
6	6	2							12				3			4	
7	7	2		13									3			4	
8	8	2	14										3			4	
9	9	2	15					6					3			4	
10	10	2			16								3			4	
11	11	2	17										3			4	
12	12	2				18							3			4	
13	13	2											3			4	19
14	14	2											3		20	4	
15	15	2		10		11	21						3			4	
16	16	2		22									3			4	
17	17	2					23						3			4	
18	18	2			24								3			4	
19	19	25				26							3			4	
20	20	2								27			3			4	
21	21	2								28			3			4	
22	22	2											3			4	
23	23	2		29									3			4	
24	24	2								30			3			4	
25	25	2	31					6					3			4	
26	26	2										32	3			4	
27	27	2											3			33	
28	28	2			34								3			4	
29	29	2											3			4	
30	30	2									35		3			4	
31	31	2		10		11				36			3			4	
32	32	2	37										3			4	
33	33	38											3			4	
34	34	2											3		39	4	
35	35	2										40	3			4	
36	36	2									41		3			4	
37	37	2								42			3			4	
38	38	2	15					6					3			4	
39	39	43											3			4	
40	40	2											3			4	
41	41	2										44	3			4	
42	42	45											3			4	
43	43	2	5					6					3			4	
44	44	2											3			4	
45	45	2	5					6					3			4	

Figura 3: Tabla de subconjuntos con estados asignados

3.1.3. Función `automata_buscador_palabra`

Esta función identifica si una palabra ingresada corresponde a una de las palabras reservadas, basándose en la tabla de transiciones.

```
    def automata_buscador_palabra(word, historial, transiciones):

        estados_finales = {'22', '29', '38', '40', '43', '44', '45'}
        current_state = '1'

        for char in word:
            if char in transiciones[current_state]:
                next_state = transiciones[current_state][char]
            else:
                next_state = '1'
            historial.append((char, current_state, next_state))
            current_state = next_state

        return current_state in estados_finales
```

3.1.4. Función `procesar_contenido`

Procesa el contenido de un texto para buscar palabras reservadas mediante el DFA y genera un historial de transiciones y un reporte en un archivo programa3_resultado_palabras.txt con las palabras encontradas.

```
        def procesar_contenido(contenido, salida_historial, transiciones):
            palabras_reservadas = {'acoso', 'acecho', 'agresi n', 'v ctima',
                'violaci n', 'violencia', 'machista'}
            conteo_palabras = {palabra: [] for palabra in palabras_reservadas}

            with open(salida_historial, "w", encoding="utf-8") as historial_file:
                for x, linea in enumerate(contenido.splitlines(), start=1):
                    linea_sin_puntuacion = linea.translate(str.maketrans('', '',
                        string.punctuation))
                    palabras = linea_sin_puntuacion.split()
```

```
                for y, palabra in enumerate(palabras, start=1):
                    historial = []
                    es_palabra_reservada =
                        automata_buscador_palabra(palabra.lower(), historial,
                        transiciones)

                    historial_file.write(f"Palabra: {palabra}\n")
                    for char, estado_actual, estado_siguiente in historial:
                        historial_file.write(f"  {char}: {estado_actual} ->
                            {estado_siguiente}\n")
                    historial_file.write("\n")

                    if es_palabra_reservada and palabra.lower() in
                        palabras_reservadas:
                        conteo_palabras[palabra.lower()].append((x, y))

        with open("Bloque_2\\programa3_resultado_palabras.txt", "w",
            encoding="utf-8") as resultado_file:
            for palabra, posiciones in conteo_palabras.items():
                resultado_file.write(f"{palabra}: {len(posiciones)} ocurrencias\n")
                for posicion in posiciones:
                    resultado_file.write(f"  Linea {posicion[0]}, Palabra
                        {posicion[1]}\n")
```

3.1.5. Función `obtener_texto_de_url`

Obtiene el texto sin formato de una página web.

```
    def obtener_texto_de_url(url):
        try:
            respuesta = requests.get(url)
            respuesta.raise_for_status()
            soup = BeautifulSoup(respuesta.text, 'html.parser')
            return soup.get_text()
        except requests.RequestException as e:
            print(f"Error al obtener la URL: {e}")
            return ""
```

3.1.6. Función `grafica_dfa`

Genera y visualiza un grafo que representa el autómata finito determinista (DFA).

```
    def grafica_dfa(transiciones, estados_finales):
        estados_finales_list = list(estados_finales)
        G = nx.DiGraph()

        for nodo in transiciones:
            G.add_node(nodo)


        # Agregar las transiciones como aristas y combinar etiquetas
        edge_labels_dict = {}  # Diccionario para agrupar etiquetas por aristas
        for nodo_actual, transiciones_letras in transiciones.items():
            for letra, nodo_destino in transiciones_letras.items():
                edge = (nodo_actual, nodo_destino)
                if edge not in edge_labels_dict:
                    edge_labels_dict[edge] = []
                edge_labels_dict[edge].append(letra)


        for (nodo_origen, nodo_destino), letras in edge_labels_dict.items():
            G.add_edge(nodo_origen, nodo_destino, label=",".join(letras))


        node_colors = ['lightblue' if nodo not in estados_finales_list else
            'lightgreen' for nodo in G.nodes()]

        pos = nx.spring_layout(G, seed=42)  # Posiciones para los nodos
        plt.figure(figsize=(8, 6))
```

```
            nx.draw(G, pos, with_labels=True, node_color=node_colors, node_size=2000,
                font_size=12, font_weight="bold", arrows=True)

            edge_labels = nx.get_edge_attributes(G, 'label')
            nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

            # Mostrar el grafo
            plt.title("Grafo de Transiciones")
            plt.show()
```

3.1.7. Función `programa3`

Función principal del programa 3, coordina la ejecución del programa principal, incluyendo la selección de la fuente de entrada (archivo de texto o página web) y el procesamiento de contenido.

- Ofrece al usuario opciones para procesar un archivo de texto o extraer contenido de una página web.
- Carga y procesa el contenido seleccionado utilizando las funciones auxiliares.
- Llama a `procesar_contenido` para identificar palabras reservadas y registrar el historial de transiciones.
- Opcionalmente, solicita al usuario mostrar la gráfica del DFA utilizando la función `grafica_dfa`.

```
        def programa3():
            # alfabeto
            chars = ['a', 'c', 'o', 's', 'e', 'h', 'g', 'r', 'i', '  ', 'n', 'v', '  ',
                't', 'm', 'l']

            # todos los caracteres llevan a '1'
            def state_transitions(**overrides):
                base = {ch: '1' for ch in chars}
                base.update(overrides)
                return base

            transiciones = {
                '1': state_transitions(a='2', v='3', m='4'),
                '2': state_transitions(a='2', v='3', m='4', c='5', g='6'),
                '3': state_transitions(a='2', v='3', m='4', i='7',   ='8'),
                '4': state_transitions(a='9', v='3', m='4'),
                '5': state_transitions(a='2', v='3', m='4', o='10', e='11'),
                '6': state_transitions(a='2', v='3', m='4', r='12'),
                '7': state_transitions(a='2', v='3', m='4', o='13'),
                '8': state_transitions(a='2', v='3', m='4', c='14'),
                '9': state_transitions(a='2', v='3', m='4', g='6', c='15'),
                '10': state_transitions(a='2', v='3', m='4', s='16'),
                '11': state_transitions(a='2', v='3', m='4', c='17'),
                '12': state_transitions(a='2', v='3', m='4', e='18'),
                '13': state_transitions(a='2', v='3', m='4', l='19'),
                '14': state_transitions(a='2', v='3', m='4', t='20'),
                '15': state_transitions(a='2', v='3', m='4', o='10', e='11', h='21'),
                '16': state_transitions(a='2', v='3', m='4', o='22'),
                '17': state_transitions(a='2', v='3', m='4', h='23'),
                '18': state_transitions(a='2', v='3', m='4', s='24'),
                '19': state_transitions(a='25', v='3', m='4', e='26'),
                '20': state_transitions(a='2', v='3', m='4', i='27'),
                '21': state_transitions(a='2', v='3', m='4', i='28'),
                '22': state_transitions(a='2', v='3', m='4'),
                '23': state_transitions(a='2', v='3', m='4', o='29'),
                '24': state_transitions(a='2', v='3', m='4', i='30'),
                '25': state_transitions(a='2', v='3', m='4', g='6', c='31'),
                '26': state_transitions(a='2', v='3', m='4', n='32'),
                '27': state_transitions(a='2', v='3', m='33'),
                '28': state_transitions(a='2', v='3', m='4', s='34'),
                '29': state_transitions(a='2', v='3', m='4'),
```

```
        '30': state_transitions(a='2', v='3', m='4',   ='35'),
        '31': state_transitions(a='2', v='3', m='4', o='10', e='11', i='36'),
        '32': state_transitions(a='2', v='3', m='4', c='37'),
        '33': state_transitions(a='38', v='3', m='4'),
        '34': state_transitions(a='2', v='3', m='4', t='39'),
        '35': state_transitions(a='2', v='3', m='4', n='40'),
        '36': state_transitions(a='2', v='3', m='4',   ='41'),
        '37': state_transitions(a='2', v='3', m='4', i='42'),
        '38': state_transitions(a='2', v='3', m='4', c='15', g='6'),
        '39': state_transitions(a='43', v='3', m='4'),
        '40': state_transitions(a='2', v='3', m='4'),
        '41': state_transitions(a='2', v='3', m='4', n='44'),
        '42': state_transitions(a='45', v='3', m='4'),
        '43': state_transitions(a='2', v='3', m='4', c='5', g='6'),
        '44': state_transitions(a='2', v='3', m='4'),
        '45': state_transitions(a='2', v='3', m='4', c='5', g='6'),
    }



    print("Seleccione la opci n de entrada:")
    print("1. Leer desde un archivo de texto")
    print("2. Leer desde una p gina web")
    opcion = input("Ingrese el n mero de su elecci n: ")

    if opcion == "1":
        #ruta_archivo = input("Ingrese la ruta del archivo de texto: ")
        try:
            with open('Bloque_2\\Programa 3 Buscador de palabras\\texto.txt',
                "r", encoding="utf-8") as archivo:
                contenido = archivo.read()
        except FileNotFoundError:
            print("Archivo no encontrado.")
            return
    elif opcion == "2":
        url = input("Ingrese la URL de la p gina web: ")
        contenido = obtener_texto_de_url(url)
        if not contenido:
            print("No se pudo obtener el contenido de la p gina web.")
            return
    else:
        print("Opci n no v lida.")
        return

    procesar_contenido(contenido,
        "Bloque_2\\programa3_historial_transiciones.txt", transiciones)

    estados_finales = {'22', '29', '38', '40', '43', '44', '45'}
    gra = input(" Desea mostrar la grafica DFA? (s/n): ").strip().lower()
    if gra == 's':
        grafica_dfa(transiciones, estados_finales)
```

3.1.8. Ejecución del programa 3

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
```

Figura 4: Seleccionamos el programa Buscador de palabras

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
Seleccione la opción de entrada:
1. Leer desde un archivo de texto
2. Leer desde una página web
Ingrese el número de su elección: 1
```

Figura 5: Leer desde un archivo .txt

Bloque_2 > programa3_resultado_palabras.txt

```
1  víctima: 1 ocurrencias
2      Linea 4, Palabra 10
3  violación: 1 ocurrencias
4      Linea 3, Palabra 11
5  violencia: 2 ocurrencias
6      Linea 1, Palabra 2
7      Linea 6, Palabra 13
8  acoso: 1 ocurrencias
9      Linea 2, Palabra 7
10 machista: 1 ocurrencias
11     Linea 1, Palabra 3
12 acecho: 1 ocurrencias
13     Linea 2, Palabra 11
14 agresión: 1 ocurrencias
15     Linea 3, Palabra 4
16
```

Figura 6: resultados_palabras.txt

Bloque_2 > programa3_historial_transiciones.txt

```
1  Palabra: La
2      l: 1 -> 1
3      a: 1 -> 2
4
5  Palabra: violencia
6      v: 1 -> 3
7      i: 3 -> 7
8      o: 7 -> 13
9      l: 13 -> 19
10     e: 19 -> 26
11     n: 26 -> 32
12     c: 32 -> 37
13     i: 37 -> 42
14     a: 42 -> 45
15
16 Palabra: machista
17     m: 1 -> 4
18     a: 4 -> 9
19     c: 9 -> 15
20     h: 15 -> 21
21     i: 21 -> 28
22     s: 28 -> 34
23     t: 34 -> 39
24     a: 39 -> 43
25
26 Palabra: se
27     s: 1 -> 1
28     e: 1 -> 1
29
30 Palabra: manifiesta
31     m: 1 -> 4
32     a: 4 -> 9
33     n: 9 -> 1
34     i: 1 -> 1
35     f: 1 -> 1
36     i: 1 -> 1
```

Figura 7: historial_transiciones.txt

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
Seleccione la opción de entrada:
1. Leer desde un archivo de texto
2. Leer desde una página web
Ingrese el número de su elección: 1
¿Desea mostrar la grafica DFA? (s/n): s
```

Figura 8: Mostrar la gráfica DFA

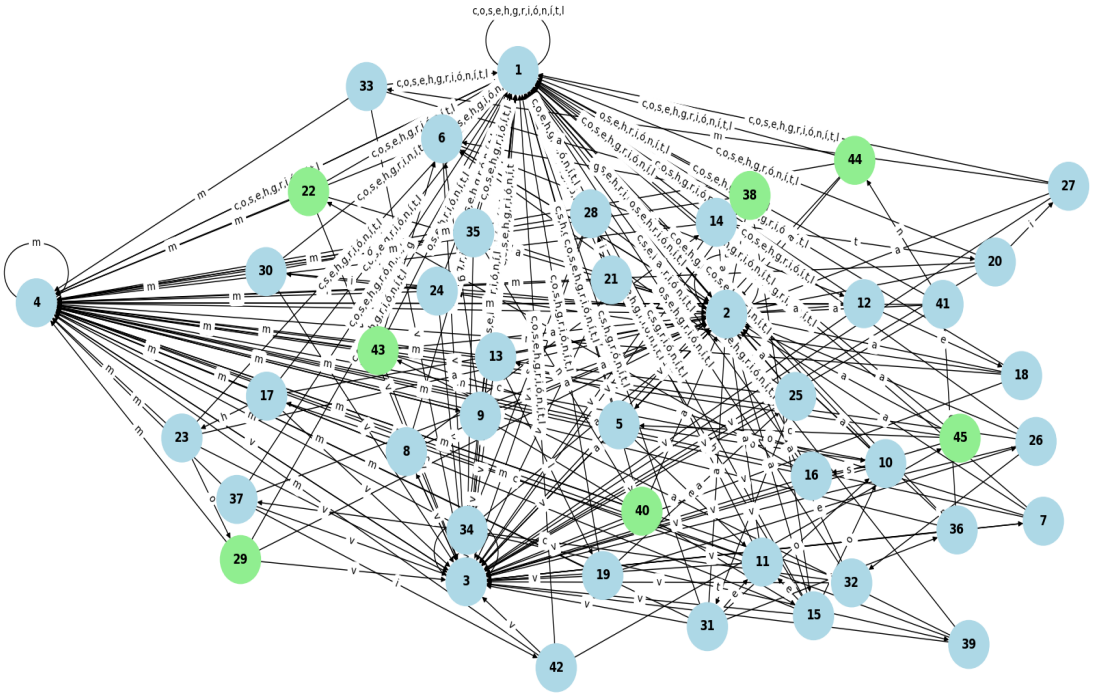

Figura 9: Gráfica DFA, 1 es estado inicial, color verde son estados finales

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
Seleccione la opción de entrada:
1. Leer desde un archivo de texto
2. Leer desde una página web
Ingrese el número de su elección: 2
```

Figura 10: Opción 2, leer en una pagina web

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
Seleccione la opción de entrada:
1. Leer desde un archivo de texto
2. Leer desde una página web
Ingrese el número de su elección: 2
Ingrese la URL de la página web: https://riodoce.mx/2024/12/17/un-blindaje-perforado-por-la-violencia-en-mazatlan/
```

Figura 11: Ingresando URL

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 1
Seleccione la opción de entrada:
1. Leer desde un archivo de texto
2. Leer desde una página web
Ingrese el número de su elección: 2
Ingrese la URL de la página web: https://riodoce.mx/2024/12/17/un-blindaje-perforado-por-la-violencia-en-mazatlan/
¿Desea mostrar la grafica DFA? (s/n): n
```

Figura 12: No graficar DFA

Bloque_2 > programa3_resultado_palabras.txt

```
violencia: 7 ocurrencias
  Linea 9, Palabra 6
  Linea 137, Palabra 6
  Linea 246, Palabra 6
  Linea 261, Palabra 1
  Linea 276, Palabra 67
  Linea 277, Palabra 23
  Linea 300, Palabra 4
víctima: 0 ocurrencias
acoso: 0 ocurrencias
machista: 0 ocurrencias
acecho: 0 ocurrencias
violación: 0 ocurrencias
agresión: 0 ocurrencias

```

Figura 13: pagina web resultados_palabras.txt

Bloque_2 > programa3_historial_transiciones.txt

```

Palabra: por
  p: 1 -> 1
  o: 1 -> 1
  r: 1 -> 1

Palabra: la
  l: 1 -> 1
  a: 1 -> 2

Palabra: violencia
  v: 1 -> 3
  i: 3 -> 7
  o: 7 -> 13
  l: 13 -> 19
  e: 19 -> 26
  n: 26 -> 32
  c: 32 -> 37
  i: 37 -> 42
  a: 42 -> 45

Palabra: en
  e: 1 -> 1
  n: 1 -> 1

Palabra: Mazatlán
  m: 1 -> 4
  a: 4 -> 9
  z: 9 -> 1
  a: 1 -> 2
  t: 2 -> 1
  l: 1 -> 1
  á: 1 -> 1
  n: 1 -> 1

```

Figura 14: pagina web historial_transiciones.txt

3.2. Programa 4. Autómata de pila

El objetivo principal del programa es implementar un autómata de pila para reconocer el lenguaje libre de contexto

$$\{0^n 1^n \mid n \geq 1\}.$$

El programa evaluará cadenas de entrada siguiendo las reglas de este lenguaje, generará descripciones instantáneas (IDs) de su ejecución, y presentará una animación del autómata cuando las cadenas sean de longitud menor o igual a 10 caracteres.

3.2.1. Clase `PushdownAutomaton`

Primero creamos una clase de PDA, con sus parámetros correspondientes, que son:

- Q: Conjunto finito de estados.
- Σ: Alfabeto de la cadena.
- Γ: Alfabeto de la pila.
- δ: Transiciones.
- $q_0 \in Q$: Estado inicial.
- Z_0: Símbolo inicial de la pila.
- F: Estados finales.

3.2.2. Función `transitions`

Obtiene las transiciones válidas desde el estado actual dado el símbolo de entrada y el tope de la pila.

```
    def _transitions(self, q, a, X):
        if (q, a, X) in self.delta:
            return self.delta[(q, a, X)]
        return []
```

3.2.3. Función `simulate`

Simula el comportamiento del autómata de pila para una cadena de entrada w. Realiza las transiciones paso a paso, registrando las descripciones instantáneas (IDs).

```
    def simulate(self, w):
    stack = [(self.q0, 0, [self.Z0], [])]
    visited = set()
    last_dead_end_path = None

    w_len = sum(1 for _ in w)

    while stack:
        q, i, pila, path = stack.pop()
        # Convertimos pila a string
        stack_str = "".join(pila)
        # w_restante es w[i:] si i < w_len, sino "  "
        w_restante = w[i:] if i < w_len else "  "

        current_config = (q, w_restante, stack_str)
        current_path = path + [current_config]
```

```
17
18              # Checar aceptaci n (i == w_len)
19              if i == w_len and q in self.F:
20                  return True, current_path
21
22              conf_signature = (q, i, tuple(pila))
23              if conf_signature in visited:
24                  continue
25              visited.add(conf_signature)
26
27              did_move = False
28              a = w[i] if i < w_len else None
29
30              # Transiciones consumiendo entrada
31              if a is not None and pila:
32                  X = pila[0]
33                  for (q_next, to_push) in self._transitions(q, a, X):
34                      did_move = True
35                      new_stack = pila[1:]
36                      if to_push != "  ":
37                          # Insertar s mbolos en orden inverso
38                          for sym in reversed(to_push):
39                              new_stack.insert(0, sym)
40                      stack.append((q_next, i+1, new_stack, current_path))
41
42              # Transiciones epsilon
43              if pila:
44                  X = pila[0]
45                  for (q_next, to_push) in self._transitions(q, "  ", X):
46                      did_move = True
47                      new_stack = pila[1:]
48                      if to_push != "  ":
49                          for sym in reversed(to_push):
50                              new_stack.insert(0, sym)
51                      stack.append((q_next, i, new_stack, current_path))
52
53              if not did_move:
54                  last_dead_end_path = current_path
55
56          return False, last_dead_end_path
```

3.2.4. Función `animarAutomata`

Visualiza el proceso del autómata de pila mediante animaciones gráficas, mostrando las configuraciones del autómata (estado, pila y cadena restante) paso a paso.

```
1           def animarAutomata(path, accepted):
2           screen = turtle.Screen()
3           screen.title("Animaci n␣PDA")
4           t = turtle.Turtle()
5           t.hideturtle()
6           t.speed(0)
7
8           # Dibujar marco
9           t.penup()
10          t.goto(-100, 100)
11          t.pendown()
12          t.color("black")
13          t.begin_fill()
14          fill_colors = ["yellow", "green"]
15          for lado in range(4):
16              t.fillcolor(fill_colors[lado % 2])
17              t.forward(100)
18              t.right(90)
19          t.end_fill()
20
21          # Flechas
22          t.penup()
23          t.goto(-50, 100)
24          t.pendown()
25          t.goto(-50, 150)
26          t.goto(-45, 140)
```

```
        t.goto(-50, 150)
        t.goto(-55, 140)

        t.penup()
        t.goto(-50, 0)
        t.pendown()
        t.goto(-50, -50)
        t.goto(-45, -40)
        t.goto(-50, -50)
        t.goto(-55, -40)

        writer = turtle.Turtle()
        writer.hideturtle()
        writer.speed(0)

        path_length = sum(1 for _ in path)

        # Iterar sobre configuraciones
        index_gen = (i for i in range(path_length))  # Generador para indices
        for i in index_gen:
            q, w_rest, stack_str = path[i]
            writer.clear()
            writer.penup()
            writer.goto(-53, 160)
            writer.pendown()

            # Si w_rest == "" -> "  "
            w_rest_len = sum(1 for _ in w_rest)
            if w_rest_len == 0:
                w_rest = "  "

            writer.write(w_rest, False, align="left", font=("Arial", 20))

            # Estado
            writer.penup()
            writer.goto(-50, 40)
            writer.pendown()
            writer.write(q, False, align="center", font=("Arial", 20))

            # Pila
            cont = -80
            for letra in stack_str:
                writer.penup()
                writer.goto(-50, cont)
                writer.pendown()
                writer.write(letra, False, align="center", font=("Arial", 20))
                cont -= 20

            sleep(1)

        # Mensaje final
        writer.penup()
        writer.goto(-153, 200)
        writer.pendown()
        if accepted:
            writer.write('La␣cadena␣es␣aceptada', False, align="left", font=("Arial",
                20))
        else:
            writer.write('La␣cadena␣no␣es␣aceptada', False, align="left", font=("Arial",
                20))

        screen.mainloop()
```

3.2.5. Función generarCadena

Genera una cadena válida del lenguaje con una longitud máxima que no podrá ser mayor a 100,000 caracteres.

```
    def generarCadena(max_length=100000):
    asignacion = random.randint(1, max_length)
    cadena = ""
    if asignacion > 0:
        cadena += str(0) * asignacion  # Agregar ceros
        cadena += str(1) * asignacion  # Agregar unos
```

```
7         return cadena
```

3.2.6. Función `Programa4`

Funcion principal del programa 4 que permite al usuario seleccionar entre ingresar manualmente una cadena o generar una automáticamente, simula el autómata de pila y muestra o guarda los resultados. Simula el autómata para evaluar la cadena seleccionada. Genera descripciones instantáneas (IDs) del proceso. Una animación gráfica si el usuario lo solicita y la cadena es corta.

- **Conjunto de estados** (Q): $\{q, p, f\}$.
  - q: Estado inicial.
  - p: Estado intermedio utilizado durante la validación de la cadena.
  - f: Estado final de aceptación.
- **Alfabeto de entrada** (Σ): $\{0, 1\}$.
- **Alfabeto de la pila** (Γ): $\{Z, X\}$.
  - Z: Símbolo inicial de la pila.
  - X: Símbolo que representa cada cero leído.
- **Estado inicial** (q_0): q.
- **Símbolo inicial de la pila** (Z_0): Z.
- **Conjunto de estados finales** (F): $\{f\}$.

La **función de transición** (δ) del autómata de pila está definida de la siguiente forma:

- Si el autómata está en el estado q y lee un 0:
  - Si el tope de la pila es Z, apila XZ y permanece en q.
  - Si el tope de la pila es X, apila otro X y permanece en q.
- Si el autómata está en el estado q y lee un 1:
  - Si el tope de la pila es X, desapila X y transita al estado p.
- Si el autómata está en el estado p y lee un 1:
  - Si el tope de la pila es X, desapila X y permanece en p.
- Si el autómata está en el estado p y encuentra una transición ϵ (sin consumir entrada):
  - Si el tope de la pila es Z, transita al estado f sin modificar la pila.

```
        Q = {"q", "p", "f"}
           = {"0", "1"}
           = {"Z", "X"}
        q0 = "q"
        Z0 = "Z"
        F = {"f"}

           = {
            ("q", "0", "Z"): [("q", "XZ")],
            ("q", "0", "X"): [("q", "XX")],
            ("q", "1", "X"): [("p", "  ")],
            ("p", "1", "X"): [("p", "  ")],
            ("p", "  ", "Z"): [("f", "Z")]
        }

        pda = PushdownAutomaton(Q,   ,   ,   , q0, Z0, F)

        print("Seleccione una opci n:")
        print("1. Ingresar la cadena manualmente")
        print("2. Generar una cadena autom ticamente")

        opcion = int(input("Ingrese el n mero de su opci n: "))

        if opcion == 1:
            w = input("Ingrese la cadena a evaluar (m ximo 100,000 caracteres): ")
        elif opcion == 2:
            w = generarCadena()
            print(f"La cadena generada es: {w}")
        else:
            print("Opci n no v lida.")
            return

        # Simular el PDA
        accepted, path = pda.simulate(w)

        # Guardar el resultado en resultado.txt
        if path is not None:
            path_length = sum(1 for _ in path)
            with open("Bloque_2\\programa4_resultado.txt", "w", encoding="utf-8") as f:
                path_list = [c for c in path]
                for i in range(path_length):
                    q, w_rest, st = path_list[i]
                    if sum(1 for _ in w_rest) == 0:
                        w_rest = "  "
                    # si es la  ltima  y aceptada, sin
                    if accepted and i == (path_length - 1):
                        f.write(f"({q}, {w_rest}, {st})\n")
                    else:
                        if i < (path_length - 1):
                            f.write(f"({q}, {w_rest}, {st})   \n")
                        else:
                            f.write(f"({q}, {w_rest}, {st})\n")
            print("Procedimiento guardado en resultado.txt")
        else:
            print("No se generaron configuraciones (path es None).")

        if accepted:
            print(f"La cadena {w} es aceptada por el PDA.")
        else:
            print(f"La cadena {w} NO es aceptada por el PDA.")

        # Si la longitud de la cadena es <= 10, preguntar si se desea mostrar la
            animaci n
        w_length = sum(1 for _ in w)
        if w_length <= 10 and path is not None:
            opcion_animacion = input(" Desea  mostrar la animaci n? (s/n): 
                ").strip().lower()
            if opcion_animacion == 's':
                animarAutomata(path, accepted)
            else:
                print("Animaci n omitida.")
        elif w_length > 10:
            print("La cadena supera los 10 caracteres. No se mostrar  la animaci n.")
```

3.2.7. Ejecución del programa 4

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 2
```

Figura 15: Opción 2 Programa Autómata de pila

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 2
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
```

Figura 16: Ingresar cadena manualmente

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 2
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
```

Figura 17: Ingresar la cadena "0011"

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 2
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
Procedimiento guardado en resultado.txt
La cadena 0011 es aceptada por el PDA.
```

Figura 18: Guardado los resultados en el txt

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 2
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
Procedimiento guardado en resultado.txt
La cadena 0011 es aceptada por el PDA.
¿Desea mostrar la animación? (s/n): s
```

Figura 19: Mostrar animación

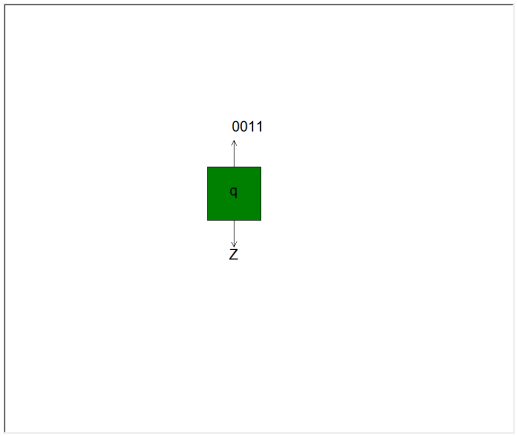

Figura 20: Animación 1

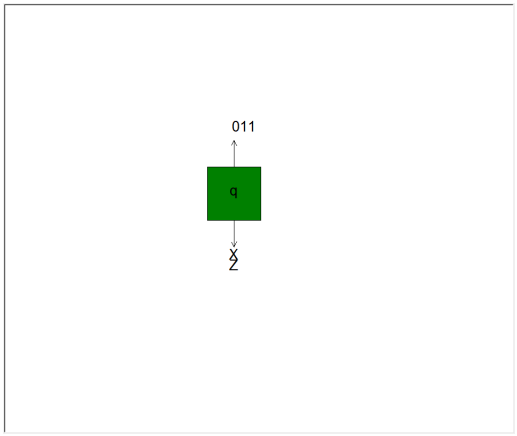

Figura 21: Animación 2

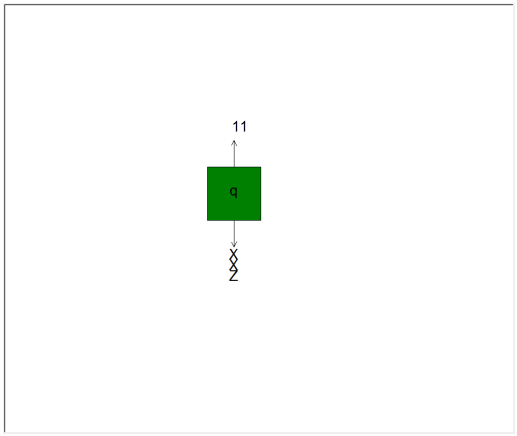

Figura 22: Animación 3

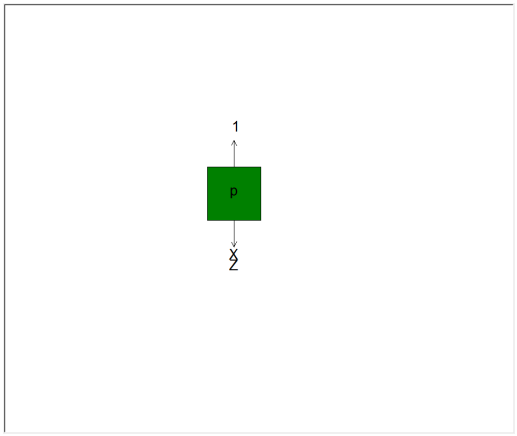

Figura 23: Animación 4

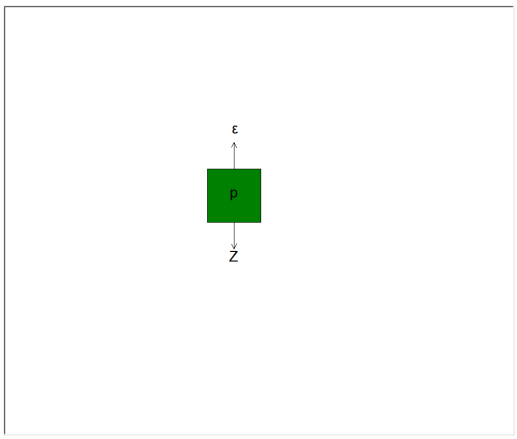

Figura 24: Animación 5

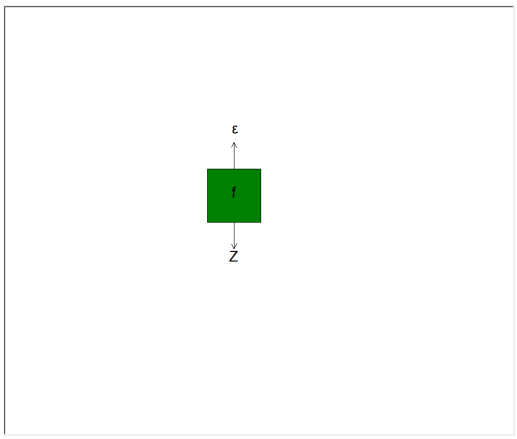

Figura 25: Animación 6

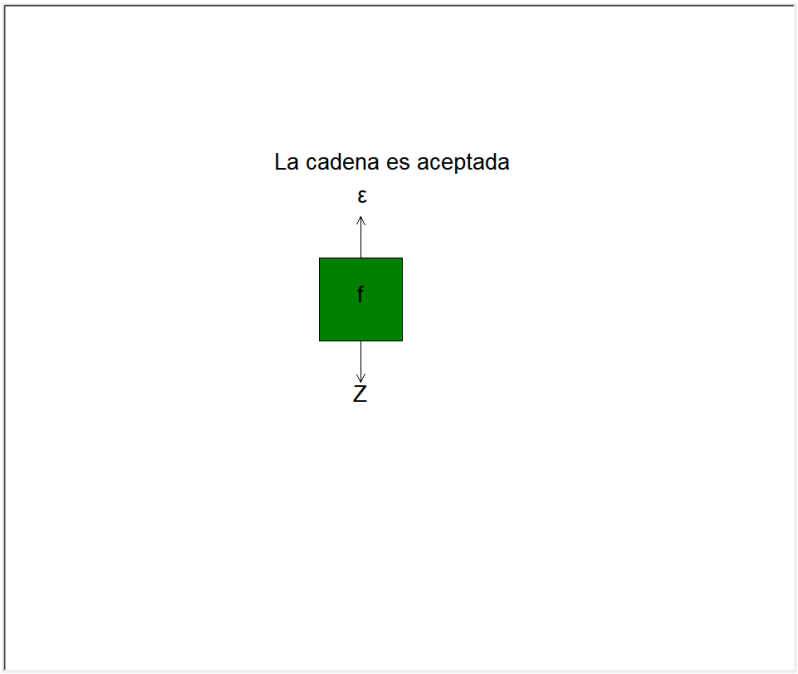

Figura 26: Animación 7

3.3. Programa 5. Backus-Naur Condicional IF

Programa que implemente la gramática Backus-Naur (BNF) para el condicional IF, permitiendo generar derivaciones automáticas hasta un número definido por el usuario o por la máquina. Este programa muestra los pasos de las derivaciones y genera un archivo con el pseudocódigo correspondiente a la cadena derivada.

Entrada: El usuario define un número máximo de derivaciones deseadas o permite que el programa lo determine automáticamente. El límite superior de derivaciones está establecido en 1000 pasos.

Proceso de Derivación:

- El proceso inicia con el símbolo inicial S.
- Las derivaciones se realizan de forma aleatoria, respetando las reglas de producción de la gramática:
  - $S \rightarrow iCtSA$
  - $A \rightarrow ; eS$ (continuación con ELSE)
  - $A \rightarrow \epsilon$ (terminación del condicional)
- Cada paso de la derivación es almacenado para ser registrado posteriormente.

3.3.1. Función `derivar_gramatica`

Esta función realiza derivaciones de una gramática basada en reglas predefinidas, comenzando desde un símbolo inicial (S) y aplicando un número especificado de pasos.

```
def derivar_gramatica(S, pasos):
derivaciones = [f"Paso 1: {S}"]
paso_actual = 2

while pasos > 0:
    # Identificar las posibles opciones de reemplazo en S
    opciones = [simbolo for simbolo in ['S', 'A'] if simbolo in S]

    if not opciones:
        break  # Salir si no hay m s s mbolos para derivar

    # Elegir un s mbolo al azar de las opciones disponibles
    eleccion = random.choice(opciones)

    if eleccion == 'A':
        if random.choice([True, False]):
            S = S.replace('A', '(;eS)', 1)
            derivaciones.append(f"Paso {paso_actual}: Aplicamos A -> ;eS: {S}")
        else:
            S = S.replace('A', '', 1)
            derivaciones.append(f"Paso {paso_actual}: Aplicamos A ->  : {S}")
    elif eleccion == 'S':
        S = S.replace('S', '(iCtSA)', 1)
        derivaciones.append(f"Paso {paso_actual}: Aplicamos S -> iCtSA: {S}")

    pasos -= 1
    paso_actual += 1

return derivaciones
```

3.3.2. Función `convertir_a_pseudocodigo`

Convierte una expresión derivada de la gramática en pseudocódigo legible basado en la estructura derivada. Analiza la cadena de forma recursiva para interpretar los símbolos y generar pseudocódigo.

- i se interpreta como un bloque if
- ; indica el inicio de un bloque *else*
- S representa una declaración general (*statement*)
-) indica el final de un bloque.

```
def convertir_a_pseudocodigo(expression):

    expression = expression[1:-1]  # Eliminar los par ntesis externos
    def parse_expression(expr, indent=0):
        result = ""

        while expr:
            char = expr[0]
            expr = expr[1:]

            if char == 'i':  # if
                result += "␣" * indent + "if␣(cond)␣then\n"
                result += "␣" * indent + "{\n"
                nested, expr = parse_expression(expr, indent + 4)
                result += nested
                result += "␣" * indent + "}\n"
            elif char == 'A':  # then
                pass
            elif char == 'S':  # statement
                result += "␣" * indent + "statement\n"
                break
            elif char == ';':  # else starts
                result += "␣" * indent + "else\n"
                result += "␣" * indent + "{\n"
                nested, expr = parse_expression(expr, indent + 4)
                result += nested
                result += "␣" * indent + "}\n"
            elif char == ')':  # end of a block
                indent -= 4
                break

        return result, expr

    pseudocode, _ = parse_expression(expression)
    return pseudocode
```

3.3.3. Función `programa5`

Esta función guía las diferentes etapas, desde la interacción con el usuario hasta la generación de derivaciones de una gramática, su conversión en pseudocódigo, y el almacenamiento de los resultados en archivos.

El primer paso es determinar el *modo de ejecución*. El programa solicita al usuario si desea trabajar en modo manual (ingresando directamente el número de derivaciones) o en modo automático (generando un número aleatorio de derivaciones). En el modo manual, se pide al usuario un número de derivaciones dentro del rango de 1 a 1000, mientras que en el modo automático, el programa selecciona un valor aleatorio dentro del mismo límite.

La siguiente etapa es la *generación de derivaciones*. Aquí, el programa llama a la función `derivar_gramatica` con el número de pasos determinado en la etapa anterior. Cada derivación se registra y se guarda en un archivo llamado `programa5_Derivaciones.txt`, lo que permite conservar el proceso de derivación paso a paso.

Una vez completadas las derivaciones, se pasa a la *conversión a pseudocódigo*. El programa toma la última derivación generada, la transforma en pseudocódigo estructurado utilizando la función `convertir_a_pseudocodigo`, y guarda el resultado en un archivo llamado `programa5_Pseudocodigo.txt`.

```
        def programa5():
            max_derivaciones = 1000

            # Solicitar al usuario el modo de ejecuci n
            try:
                modo = int(input("Elige el modo de ejecuci n (1 para manual, 2 para autom tico): "))
            except ValueError:
                print("Entrada inv lida. Se seleccionar el modo autom tico.")
                modo = 2

            # Determinar el n mero de derivaciones
            if modo == 1:
                try:
                    num_derivaciones = int(input(f"Ingrese el n mero de derivaciones (hasta {max_derivaciones}): "))
                    num_derivaciones = min(max_derivaciones, max(1, num_derivaciones))
                except ValueError:
                    print("Entrada inv lida. Se usar el n mero m ximo de derivaciones.")
                    num_derivaciones = max_derivaciones
            else:
                num_derivaciones = random.randint(1, max_derivaciones)

            # Derivaciones y generaci n de pseudo-c digo
            S = 'S'
            derivaciones = derivar_gramatica(S, num_derivaciones)

            with open('Bloque_2\\programa5_Derivaciones.txt', 'w', encoding='utf-8') as f:
                f.write("\n".join(derivaciones))

            if derivaciones:
                ultima_derivacion = derivaciones[-1].split(": ")[-1]
                pseudocodigo = convertir_a_pseudocodigo(ultima_derivacion)
                with open('Bloque_2\\programa5_Pseudocodigo.txt', 'w', encoding='utf-8') as f:
                    f.write(pseudocodigo)

            print("Las derivaciones se han guardado en 'Derivaciones.txt'")
            print("El pseudo-c digo se ha guardado en 'Pseudocodigo.txt'")
```

3.3.4. Ejecución del programa 5

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 3
```

Figura 27: Elegir el programa BNC IF

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 3
Elige el modo de ejecución (1 para manual, 2 para automático): 1
```

Figura 28: Modo manual

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 3
Elige el modo de ejecución (1 para manual, 2 para automático): 1
Ingrese el número de derivaciones (hasta 1000): 6
```

Figura 29: Número de derivaciones

```
================= MENU =================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
========================================

Selecciona una opción: 3
Elige el modo de ejecución (1 para manual, 2 para automático): 1
Ingrese el número de derivaciones (hasta 1000): 6
Las derivaciones se han guardado en 'Derivaciones.txt'
El pseudo-código se ha guardado en 'Pseudocodigo.txt'
```

Figura 30: Historial guardado en los archivos txt

```
Paso 1: S
Paso 2: Aplicamos S -> iCtSA: (iCtSA)
Paso 3: Aplicamos S -> iCtSA: (iCt(iCtSA)A)
Paso 4: Aplicamos A -> ε: (iCt(iCtS)A)
Paso 5: Aplicamos A -> ;eS: (iCt(iCtS)(;eS))
Paso 6: Aplicamos S -> iCtSA: (iCt(iCt(iCtSA))(;eS))
Paso 7: Aplicamos S -> iCtSA: (iCt(iCt(iCt(iCtSA)A))(;eS))
```

Figura 31: Derivaciones

```
if (cond) then
{
    if (cond) then
    {
        if (cond) then
        {
            if (cond) then
            {
                statement
            }
        }
    }
}
else
{
    statement
}
```

Figura 32: Pseudocodigo

3.4. Programa 6. Máquina de Turing

El objetivo del programa es implementar esta Máquina de Turing que reconoce el lenguaje $\{0^n1^n \mid n \geq 1\}$, como se describe en el libro de John Hopcroft (Ejercicio 8.2, Segunda Edición).

1. El programa debe recibir una cadena definida por el usuario o que sea generada automáticamente. La cadena tendrá una longitud máxima de 1000 caracteres.

2. La salida del programa debe escribirse en un archivo de texto y debe incluir descripciones instantáneas de cada paso de la computación.

La Máquina de Turing se define como:

$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Q: Conjunto finito de estados. $\{q_0, q_1, q_2, q_3, q_4\}$
- Σ: Alfabeto de entrada. $\{0, 1\}$
- Γ: Alfabeto de la cinta. $\{0, 1, X, Y, B\}$
- $\delta : \{q, X\} \rightarrow \{p, Y, D\}$: Función de transición. p y q son estados, X y Y son para sustituir en la cinta, D es la dirección.
- $q_0 \in Q$: Estado inicial.
- B: Símbolo blanco de la cinta.
- F: Estados finales.

3.4.1. Clase `TuringMachine`

Simula una máquina de Turing y contiene métodos para procesar cadenas de entrada y animar el funcionamiento de la máquina. En su núcleo, la clase incluye atributos como el conjunto de estados, el alfabeto de entrada, el alfabeto de la cinta, el estado inicial, los estados finales, y la cinta que almacena la representación de la cadena a procesar, así como el cabezal que apunta a la posición actual en la cinta.

```
    def __init__(self, Q,   ,   ,   , q0, B, F):
        self.Q = Q          # Conjunto de estados
        self.   =             # Alfabeto de entrada
        self.   =             # Alfabeto de la cinta
        self.   =             # Funci n de transici n (diccionario)
        self.q0 = q0        # Estado inicial
        self.B = B          # S mbolo en blanco
        self.F = F          # Conjunto de estados finales
        self.cinta = []     # Representaci n de la cinta (se inicializa con una
            cadena)
        self.cabezal = 0    # Posici n del cabezal
        self.estado = q0    # Estado actual
```

3.4.2. Función `cargar_cinta`

El método cargar_cinta permite cargar una cadena en la cinta de la máquina, rodeándola con símbolos blancos para delimitarla.

```
    def cargar_cinta(self, cadena):
        # Inicializa la cinta
        self.cinta = [self.B] + list(cadena) + [self.B]
        self.cabezal = 1
```

3.4.3. Función avanzar

La función principal que simula el funcionamiento de la máquina de Turing es avanzar. Este método ejecuta una transición de la máquina en función de la configuración actual del estado y el símbolo leído en la posición del cabezal. Si existe una transición válida, el método actualiza el estado de la máquina, escribe en la cinta, y mueve el cabezal según la dirección especificada (izquierda o derecha). Si no hay transición válida, la máquina termina su ejecución. También se asegura de extender la cinta automáticamente cuando el cabezal se mueve fuera de los límites iniciales.

```
    def avanzar(self):
        if self.estado in self.F:
            return True  # La cadena es aceptada si estamos en un estado final

        len_cinta = sum(1 for _ in self.cinta)

        simbolo_actual = self.cinta[self.cabezal] if self.cabezal < len_cinta else
            self.B
        transicion = self.  .get((self.estado, simbolo_actual))

        if not transicion:
            return False

        nuevo_estado, nuevo_simbolo, direccion = transicion
        self.cinta[self.cabezal] = nuevo_simbolo  # Escribir en la cinta
        self.estado = nuevo_estado  # Cambiar de estado

        # dirrecion
        if direccion == 'R':
            self.cabezal += 1
            if self.cabezal == len_cinta:  # A adir blanco si se sale de la cinta
                self.cinta.append(self.B)
        elif direccion == 'L':
            self.cabezal -= 1
            if self.cabezal < 0:  # A adir blanco a la izquierda si es necesario
                self.cinta.insert(0, self.B)
                self.cabezal = 0

        return None
```

3.4.4. Función ejecutar_con_animacion

Para mejorar la experiencia visual, el método ejecutar_con_animacion añade la capacidad de animar gráficamente el funcionamiento de la máquina usando el módulo turtle. Este método también escribe en un archivo de texto el estado de la máquina en cada paso, lo que permite documentar las descripciones instantáneas de la computación. Se representa visualmente la cinta y el cabezal en cada momento del proceso, permitiendo una comprensión más clara de cómo avanza la máquina paso a paso. Este método se puede ejecutar con o sin animación, dependiendo de la longitud de la cadena y las preferencias del usuario.

```
    def ejecutar_con_animacion(self, archivo_salida="salida.txt",
        mostrar_animacion=True):
        if mostrar_animacion:
            screen = Screen()
            screen.setup(width=800, height=400)
            screen.tracer(0)

            tr = Turtle()
            tr.hideturtle()
            tr.penup()

            def dibujar_cinta():
                tr.clear()
                x_inicio = -300
                for i, simbolo in enumerate(self.cinta):
```

```
                    x = x_inicio + i * 50
                    tr.setpos(x, 0)
                    tr.pendown()
                    for _ in range(4):
                        tr.forward(50)
                        tr.right(90)
                    tr.penup()
                    tr.setpos(x-25, -30)
                    tr.write(simbolo, align="center", font=("Arial", 16, "normal"))

                # Dibujar el cabezal el palo
                tr.setpos(x_inicio + self.cabezal * 50 - 25, 50)
                tr.write(f"{self.estado}", align="center", font=("Arial", 16,
                    "bold"))
                tr.setpos(x_inicio + self.cabezal * 50 - 25, 40)
                tr.setheading(270)
                tr.pendown()
                tr.forward(30)
                tr.penup()

            dibujar_cinta()
            screen.update()

        with open(archivo_salida, "w", encoding="utf-8") as archivo:
            while True:
                # Escribir el estado actual en el archivo
                cinta_con_estado = "".join(
                    self.cinta[:self.cabezal] +
                    [f"{self.estado}_"] +
                    self.cinta[self.cabezal:]
                )
                archivo.write(cinta_con_estado + "    \n")

                resultado = self.avanzar()
                if mostrar_animacion:
                    dibujar_cinta()
                    screen.update()
                    sleep(1)

                if resultado is not None:  # Terminar la ejecuci n
                    if mostrar_animacion:
                        screen.bye()
                    return resultado
```

3.4.5. Función programa6

configura un ejemplo práctico de la máquina de Turing. Define la máquina y su función de transición para procesar cadenas que pertenecen al lenguaje $\{0^n1^n \mid n \geq 1\}$ El usuario puede ingresar manualmente una cadena o generar una automáticamente. Si la longitud de la cadena es adecuada, se ofrece la posibilidad de ver una animación del proceso. La salida de cada paso, junto con el resultado final de aceptación o rechazo, se escribe en un archivo de texto. Esta implementación demuestra cómo las máquinas de Turing pueden usarse para procesar lenguajes formales y ofrecer una visión computacional detallada de sus operaciones.

Utilizando la tabla de transición, generamos lo siguiente:

322 CHAPTER 8. INTRODUCTION TO TURING MACHINES

State	Symbol 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	–	–	–	–	–

Figure 8.9: A Turing machine to accept $\{0^n1^n \mid n \geq 1\}$

Figura 33: Tabla de transición para TM

```
def programa6():
    Q = {'q0', 'q1', 'q2', 'q3', 'q4'}  # Conjunto de estados
       = {'0', '1'}                      # Alfabeto de entrada
       = {'0', '1', 'X', 'Y', 'B'}       # Alfabeto de la cinta
    B = 'B'                              # Blanco
    F = {'q4'}                           # Conjunto de estados finales
       = {                               # Funci n de transici n
        ('q0', '0'): ('q1', 'X', 'R'),
        ('q0', 'Y'): ('q3', 'Y', 'R'),
        ('q1', '0'): ('q1', '0', 'R'),
        ('q1', '1'): ('q2', 'Y', 'L'),
        ('q1', 'Y'): ('q1', 'Y', 'R'),
        ('q2', '0'): ('q2', '0', 'L'),
        ('q2', 'X'): ('q0', 'X', 'R'),
        ('q2', 'Y'): ('q2', 'Y', 'L'),
        ('q3', 'Y'): ('q3', 'Y', 'R'),
        ('q3', 'B'): ('q4', 'B', 'R'),
    }
    q0 = 'q0'  # Estado inicial


    tm = TuringMachine(Q,   ,   ,   , q0, B, F)


    print("Seleccione una opci n:")
    print("1. Ingresar la cadena manualmente")
    print("2. Generar una cadena autom ticamente")

    opcion = int(input("Ingrese el n mero de su opci n: "))

    if opcion == 1:
        cadena = input("Ingrese la cadena a evaluar (m ximo 100,000
            caracteres): ")
    elif opcion == 2:
        cadena = generarCadena()
        print(f"La cadena generada es: {cadena}")

    cadena_length = sum(1 for _ in cadena)
    if cadena_length <= 10:
        # Preguntar si se desea animaci n
        opcion_animacion = input(" Desea  mostrar la animaci n? (s/n): 
            ").strip().lower()
        mostrar_animacion = opcion_animacion == 's'
    else:
        print("La cadena supera los 10 caracteres. No se puede animar.")
        return

    # Cargar una cadena de entrada
    #cadena = input("Ingrese la cadena a evaluar: ")
    tm.cargar_cinta(cadena)



    resultado =
        tm.ejecutar_con_animacion(archivo_salida="Bloque_2\\programa6_salidaTM.txt",
```

```
                mostrar_animacion=mostrar_animacion)
53              if resultado:
54                  print(f"\nLa␣cadena␣{cadena}␣ES␣aceptada.␣Los␣pasos␣se␣ha␣guardado␣en␣
                        'salidaTM.txt'.\n")
55              else:
56                  print(f"\nLa␣cadena␣{cadena}␣NO␣es␣aceptada.␣Los␣pasos␣se␣ha␣guardado␣en␣
                        'salidaTM.txt'.\n")
```

3.4.6. Ejecución del programa 6

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 4
```

Figura 34: Elegir el programa TM

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 4
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
```

Figura 35: Ingresar la cadena manualmente

```
================ MENU ================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
======================================

Selecciona una opción: 4
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
```

Figura 36: ingresar la cadena '0011'

```
================ MENU ================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
======================================

Selecciona una opción: 4
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
¿Desea mostrar la animación? (s/n): s
```

Figura 37: Mostrar la animación

```
================== MENU ==================
Selecciona un programa para ejecutar:
1. Programa Buscador de palabras
2. Programa Automata de pila
3. Programa Backus-Naur Condicional IF
4. Programa Máquina de Turing
5. Salir
==========================================

Selecciona una opción: 4
Seleccione una opción:
1. Ingresar la cadena manualmente
2. Generar una cadena automáticamente
Ingrese el número de su opción: 1
Ingrese la cadena a evaluar (máximo 100,000 caracteres): 0011
¿Desea mostrar la animación? (s/n): s

La cadena 0011 ES aceptada. Los pasos se ha guardado en 'salidaTM.txt'.
```

Figura 38: Mensaje de la cadena aprobada

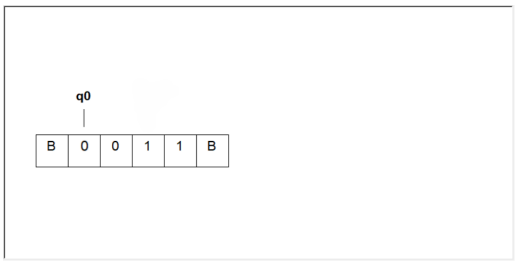

Figura 39: Animación 1

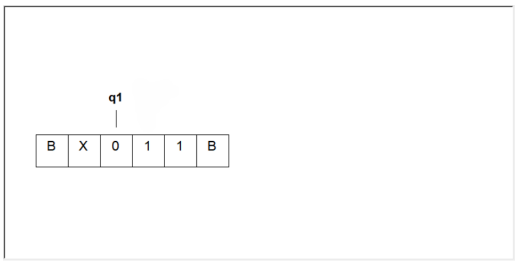

Figura 40: Animación 2

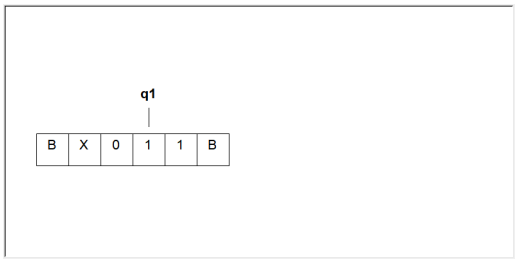

Figura 41: Animación 3

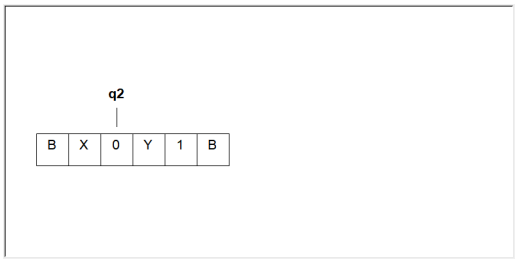

Figura 42: Animación 4

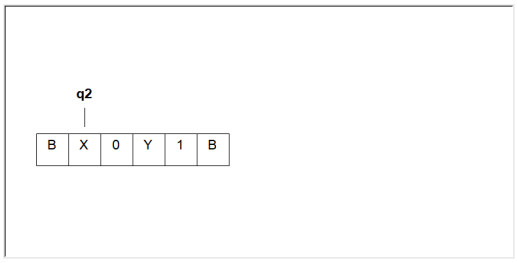

Figura 43: Animación 5

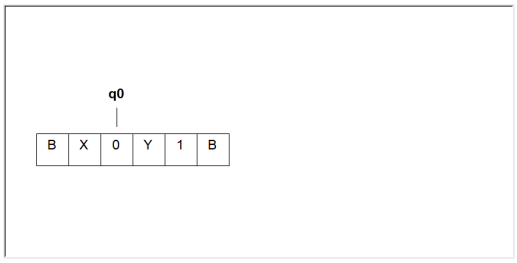

Figura 44: Animación 6

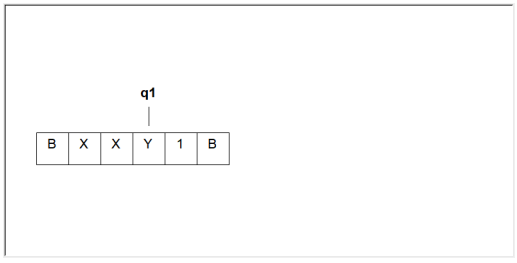

Figura 45: Animación 7

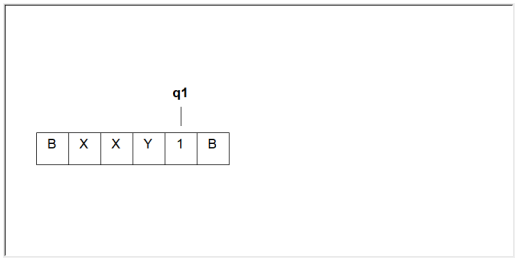

Figura 46: Animación 8

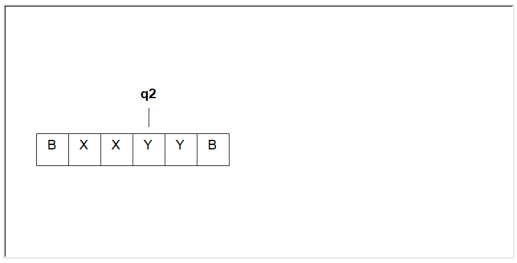

Figura 47: Animación 9

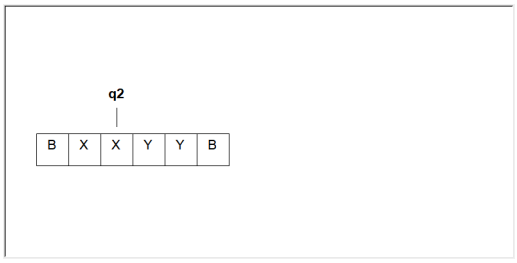

Figura 48: Animación 10

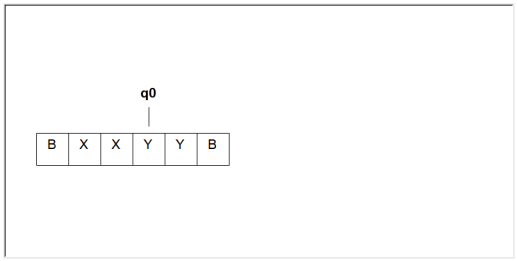

Figura 49: Animación 11

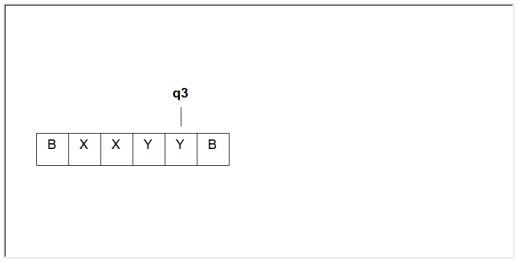

Figura 50: Animación 12

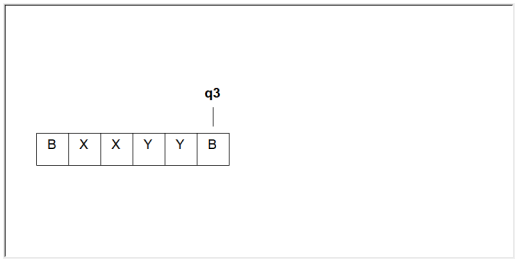

Figura 51: Animación 13

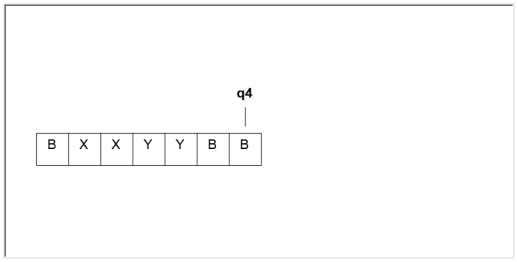

Figura 52: Animación 14

4. Conclusión

En este reporte, hemos examinado varias estructuras fundamentales en la teoría de la computación, incluyendo los autómatas finitos deterministas, los autómatas de pila, la gramática Backus-Naur y las máquinas de Turing. Cada una de estas estructuras proporciona una perspectiva única para modelar y comprender cómo se procesan y reconocen las cadenas en diferentes lenguajes formales.

Los autómatas finitos deterministas son herramientas clave para reconocer lenguajes regulares, ofreciendo un enfoque simple y claro basado en una secuencia de transiciones deterministas. En contraste, los autómatas de pila amplían esta capacidad al poder manejar lenguajes más complejos gracias a su memoria adicional, lo que les permite aceptar lenguajes no regulares de forma eficiente.

la gramática Backus-Naur refuerza la conexión entre la teoría de lenguajes formales y su aplicación práctica, destacando la utilidad de las gramáticas en la representación de lógica y algoritmos en un formato estructurado. La generación de archivos de texto para las derivaciones y el pseudocódigo asegura además la trazabilidad y documentación del proceso.

Por último, las máquinas de Turing representan el modelo más poderoso en términos de capacidad computacional teórica. Estas máquinas nos permiten explorar los límites de lo que es computable, ya que pueden simular cualquier algoritmo. Además, son una base esencial para el estudio de la decidibilidad y la complejidad computacional, proporcionando una visión profunda de los fundamentos de la computación.

5. Referencias Bibliográficas

Hopcroft, J., Motwani, R., Ullman, J. (2001). *Introduction to Automata Theory, Languages, and Computation* https://www-2.dc.uba.ar/staff/becher/Hopcroft-Motwani-Ullman-2
pdf

Jeffrey D. Ullman. (2009). *CS154: Introduction to Automata and Complexity Theory* http://infolab.stanford.edu/~ullman/ialc/spr10/spr10.html#LECTURE%20NOTES

6. Anexo

6.1. Código completo de los programas implementado en Python

```
import random
import string
import turtle
import requests
import networkx as nx
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from time import sleep
from turtle import *


######################################## Programa 3 Buscador de palabras
    ############################################################

def automata_buscador_palabra(word, historial, transiciones):

    estados_finales = {'22', '29', '38', '40', '43', '44', '45'}
    current_state = '1'

    for char in word:
        if char in transiciones[current_state]:
            next_state = transiciones[current_state][char]
        else:
            next_state = '1'
        historial.append((char, current_state, next_state))
        current_state = next_state

    return current_state in estados_finales


def procesar_contenido(contenido, salida_historial, transiciones):
    palabras_reservadas = {'acoso', 'acecho', 'agresi n', 'v ctima', 'violaci n',
        'violencia', 'machista'}
    conteo_palabras = {palabra: [] for palabra in palabras_reservadas}

    with open(salida_historial, "w", encoding="utf-8") as historial_file:
        for x, linea in enumerate(contenido.splitlines(), start=1):
            linea_sin_puntuacion = linea.translate(str.maketrans('', '',
                string.punctuation))
            palabras = linea_sin_puntuacion.split()

            for y, palabra in enumerate(palabras, start=1):
                historial = []
                es_palabra_reservada = automata_buscador_palabra(palabra.lower(),
                    historial, transiciones)

                historial_file.write(f"Palabra: {palabra}\n")
                for char, estado_actual, estado_siguiente in historial:
                    historial_file.write(f"  {char}: {estado_actual} -> {estado_siguiente}\n")
                historial_file.write("\n")

                if es_palabra_reservada and palabra.lower() in palabras_reservadas:
                    conteo_palabras[palabra.lower()].append((x, y))

    with open("Bloque_2\\programa3_resultado_palabras.txt", "w", encoding="utf-8") as
        resultado_file:
        for palabra, posiciones in conteo_palabras.items():
            resultado_file.write(f"{palabra}: {len(posiciones)} ocurrencias\n")
            for posicion in posiciones:
                resultado_file.write(f"  Linea {posicion[0]}, Palabra {posicion[1]}\n")


def obtener_texto_de_url(url):
    try:
        respuesta = requests.get(url)
        respuesta.raise_for_status()
        soup = BeautifulSoup(respuesta.text, 'html.parser')
        return soup.get_text()
    except requests.RequestException as e:
        print(f"Error al obtener la URL: {e}")
```

```
        return ""


def grafica_dfa(transiciones, estados_finales):
    estados_finales_list = list(estados_finales)
    G = nx.DiGraph()

    for nodo in transiciones:
        G.add_node(nodo)


    # Agregar las transiciones como aristas y combinar etiquetas
    edge_labels_dict = {}  # Diccionario para agrupar etiquetas por aristas
    for nodo_actual, transiciones_letras in transiciones.items():
        for letra, nodo_destino in transiciones_letras.items():
            edge = (nodo_actual, nodo_destino)
            if edge not in edge_labels_dict:
                edge_labels_dict[edge] = []
            edge_labels_dict[edge].append(letra)


    for (nodo_origen, nodo_destino), letras in edge_labels_dict.items():
        G.add_edge(nodo_origen, nodo_destino, label=",".join(letras))


    node_colors = ['lightblue' if nodo not in estados_finales_list else 'lightgreen' for
        nodo in G.nodes()]

    pos = nx.spring_layout(G, seed=42)  # Posiciones para los nodos
    plt.figure(figsize=(8, 6))
    nx.draw(G, pos, with_labels=True, node_color=node_colors, node_size=2000,
        font_size=12, font_weight="bold", arrows=True)

    edge_labels = nx.get_edge_attributes(G, 'label')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)

    # Mostrar el grafo
    plt.title("Grafo de Transiciones")
    plt.show()



def programa3():
    # alfabeto
    chars = ['a', 'c', 'o', 's', 'e', 'h', 'g', 'r', 'i', '  ', 'n', 'v', '  ', 't',
        'm', 'l']

    # todos los caracteres llevan a '1'
    def state_transitions(**overrides):
        base = {ch: '1' for ch in chars}
        base.update(overrides)
        return base

    transiciones = {
        '1': state_transitions(a='2', v='3', m='4'),
        '2': state_transitions(a='2', v='3', m='4', c='5', g='6'),
        '3': state_transitions(a='2', v='3', m='4', i='7',   ='8'),
        '4': state_transitions(a='9', v='3', m='4'),
        '5': state_transitions(a='2', v='3', m='4', o='10', e='11'),
        '6': state_transitions(a='2', v='3', m='4', r='12'),
        '7': state_transitions(a='2', v='3', m='4', o='13'),
        '8': state_transitions(a='2', v='3', m='4', c='14'),
        '9': state_transitions(a='2', v='3', m='4', g='6', c='15'),
        '10': state_transitions(a='2', v='3', m='4', s='16'),
        '11': state_transitions(a='2', v='3', m='4', c='17'),
        '12': state_transitions(a='2', v='3', m='4', e='18'),
        '13': state_transitions(a='2', v='3', m='4', l='19'),
        '14': state_transitions(a='2', v='3', m='4', t='20'),
        '15': state_transitions(a='2', v='3', m='4', o='10', e='11', h='21'),
        '16': state_transitions(a='2', v='3', m='4', o='22'),
        '17': state_transitions(a='2', v='3', m='4', h='23'),
        '18': state_transitions(a='2', v='3', m='4', s='24'),
        '19': state_transitions(a='25', v='3', m='4', e='26'),
        '20': state_transitions(a='2', v='3', m='4', i='27'),
        '21': state_transitions(a='2', v='3', m='4', i='28'),
        '22': state_transitions(a='2', v='3', m='4'),
```

```
        '23': state_transitions(a='2', v='3', m='4', o='29'),
        '24': state_transitions(a='2', v='3', m='4', i='30'),
        '25': state_transitions(a='2', v='3', m='4', g='6', c='31'),
        '26': state_transitions(a='2', v='3', m='4', n='32'),
        '27': state_transitions(a='2', v='3', m='33'),
        '28': state_transitions(a='2', v='3', m='4', s='34'),
        '29': state_transitions(a='2', v='3', m='4'),
        '30': state_transitions(a='2', v='3', m='4',   ='35'),
        '31': state_transitions(a='2', v='3', m='4', o='10', e='11', i='36'),
        '32': state_transitions(a='2', v='3', m='4', c='37'),
        '33': state_transitions(a='38', v='3', m='4'),
        '34': state_transitions(a='2', v='3', m='4', t='39'),
        '35': state_transitions(a='2', v='3', m='4', n='40'),
        '36': state_transitions(a='2', v='3', m='4',   ='41'),
        '37': state_transitions(a='2', v='3', m='4', i='42'),
        '38': state_transitions(a='2', v='3', m='4', c='15', g='6'),
        '39': state_transitions(a='43', v='3', m='4'),
        '40': state_transitions(a='2', v='3', m='4'),
        '41': state_transitions(a='2', v='3', m='4', n='44'),
        '42': state_transitions(a='45', v='3', m='4'),
        '43': state_transitions(a='2', v='3', m='4', c='5', g='6'),
        '44': state_transitions(a='2', v='3', m='4'),
        '45': state_transitions(a='2', v='3', m='4', c='5', g='6'),
    }



    print("Seleccione la opci n de entrada:")
    print("1. Leer desde un archivo de texto")
    print("2. Leer desde una p gina web")
    opcion = input("Ingrese el n mero de su elecci n: ")

    if opcion == "1":
        #ruta_archivo = input("Ingrese la ruta del archivo de texto: ")
        try:
            with open('Bloque_2\\Programa 3 Buscador de palabras\\texto.txt', "r",
                encoding="utf-8") as archivo:
                contenido = archivo.read()
        except FileNotFoundError:
            print("Archivo no encontrado.")
            return
    elif opcion == "2":
        url = input("Ingrese la URL de la p gina web: ")
        contenido = obtener_texto_de_url(url)
        if not contenido:
            print("No se pudo obtener el contenido de la p gina web.")
            return
    else:
        print("Opci n no v lida.")
        return

    procesar_contenido(contenido, "Bloque_2\\programa3_historial_transiciones.txt",
        transiciones)

    estados_finales = {'22', '29', '38', '40', '43', '44', '45'}
    gra = input(" Desea  mostrar la grafica DFA? (s/n): ").strip().lower()
    if gra == 's':
        grafica_dfa(transiciones, estados_finales)




######################################### Programa 4 Automata de pila
    ################################################################
class PushdownAutomaton:
    def __init__(self, Q,   ,   ,   , q0, Z0, F):
        self.Q = Q
        self.Sigma =
        self.Gamma =
        self.delta =
        self.q0 = q0
        self.Z0 = Z0
        self.F = F

    def _transitions(self, q, a, X):
        if (q, a, X) in self.delta:
```

```
            return self.delta[(q, a, X)]
        return []

    def simulate(self, w):
        stack = [(self.q0, 0, [self.Z0], [])]
        visited = set()
        last_dead_end_path = None

        w_len = sum(1 for _ in w)

        while stack:
            q, i, pila, path = stack.pop()
            # Convertimos pila a string
            stack_str = "".join(pila)
            # w_restante es w[i:] si i < w_len, sino "  "
            w_restante = w[i:] if i < w_len else "  "

            current_config = (q, w_restante, stack_str)
            current_path = path + [current_config]

            # Checar aceptaci n (i == w_len)
            if i == w_len and q in self.F:
                return True, current_path

            conf_signature = (q, i, tuple(pila))
            if conf_signature in visited:
                continue
            visited.add(conf_signature)

            did_move = False
            a = w[i] if i < w_len else None

            # Transiciones consumiendo entrada
            if a is not None and pila:
                X = pila[0]
                for (q_next, to_push) in self._transitions(q, a, X):
                    did_move = True
                    new_stack = pila[1:]
                    if to_push != "  ":
                        # Insertar s mbolos en orden inverso
                        for sym in reversed(to_push):
                            new_stack.insert(0, sym)
                    stack.append((q_next, i+1, new_stack, current_path))

            # Transiciones epsilon
            if pila:
                X = pila[0]
                for (q_next, to_push) in self._transitions(q, "  ", X):
                    did_move = True
                    new_stack = pila[1:]
                    if to_push != "  ":
                        for sym in reversed(to_push):
                            new_stack.insert(0, sym)
                    stack.append((q_next, i, new_stack, current_path))

            if not did_move:
                last_dead_end_path = current_path

        return False, last_dead_end_path


def animarAutomata(path, accepted):
    screen = turtle.Screen()
    screen.title("Animaci n␣PDA")
    t = turtle.Turtle()
    t.hideturtle()
    t.speed(0)

    # Dibujar marco
    t.penup()
    t.goto(-100, 100)
    t.pendown()
    t.color("black")
    t.begin_fill()
    fill_colors = ["yellow", "green"]
    for lado in range(4):
```

```
        t.fillcolor(fill_colors[lado % 2])
        t.forward(100)
        t.right(90)
    t.end_fill()

    # Flechas
    t.penup()
    t.goto(-50, 100)
    t.pendown()
    t.goto(-50, 150)
    t.goto(-45, 140)
    t.goto(-50, 150)
    t.goto(-55, 140)

    t.penup()
    t.goto(-50, 0)
    t.pendown()
    t.goto(-50, -50)
    t.goto(-45, -40)
    t.goto(-50, -50)
    t.goto(-55, -40)

    writer = turtle.Turtle()
    writer.hideturtle()
    writer.speed(0)

    path_length = sum(1 for _ in path)

    # Iterar sobre configuraciones
    index_gen = (i for i in range(path_length))  # Generador para indices
    for i in index_gen:
        q, w_rest, stack_str = path[i]
        writer.clear()
        writer.penup()
        writer.goto(-53, 160)
        writer.pendown()

        # Si w_rest == "" -> "  "
        w_rest_len = sum(1 for _ in w_rest)
        if w_rest_len == 0:
            w_rest = "  "

        writer.write(w_rest, False, align="left", font=("Arial", 20))

        # Estado
        writer.penup()
        writer.goto(-50, 40)
        writer.pendown()
        writer.write(q, False, align="center", font=("Arial", 20))

        # Pila
        cont = -80
        for letra in stack_str:
            writer.penup()
            writer.goto(-50, cont)
            writer.pendown()
            writer.write(letra, False, align="center", font=("Arial", 20))
            cont -= 20

        sleep(5)

    # Mensaje final
    writer.penup()
    writer.goto(-153, 200)
    writer.pendown()
    if accepted:
        writer.write('La cadena es aceptada', False, align="left", font=("Arial", 20))
    else:
        writer.write('La cadena no es aceptada', False, align="left", font=("Arial", 20))

    screen.mainloop()


def generarCadena(max_length=100000):
    asignacion = random.randint(1, max_length)
    cadena = ""
```

```
    if asignacion > 0:
        cadena += str(0) * asignacion  # Agregar ceros
        cadena += str(1) * asignacion  # Agregar unos
    return cadena



def programa4():
    Q = {"q", "p", "f"}
       = {"0", "1"}
       = {"Z", "X"}
    q0 = "q"
    Z0 = "Z"
    F = {"f"}

       = {
        ("q", "0", "Z"): [("q", "XZ")],
        ("q", "0", "X"): [("q", "XX")],
        ("q", "1", "X"): [("p", "  ")],
        ("p", "1", "X"): [("p", "  ")],
        ("p", "  ", "Z"): [("f", "Z")]
    }

    pda = PushdownAutomaton(Q,   ,   ,   , q0, Z0, F)

    print("Seleccione una opci n:")
    print("1. Ingresar la cadena manualmente")
    print("2. Generar una cadena autom ticamente")

    opcion = int(input("Ingrese el n mero de su opci n: "))

    if opcion == 1:
        w = input("Ingrese la cadena a evaluar (m ximo 100,000 caracteres): ")
    elif opcion == 2:
        w = generarCadena()
        print(f"La cadena generada es: {w}")
    else:
        print("Opci n no v lida.")
        return

    # Simular el PDA
    accepted, path = pda.simulate(w)

    # Guardar el resultado en resultado.txt
    if path is not None:
        path_length = sum(1 for _ in path)
        with open("Bloque_2\\programa4_resultado.txt", "w", encoding="utf-8") as f:
            path_list = [c for c in path]
            for i in range(path_length):
                q, w_rest, st = path_list[i]
                if sum(1 for _ in w_rest) == 0:
                    w_rest = "  "
                # si es la  ltima  y aceptada, sin
                if accepted and i == (path_length - 1):
                    f.write(f"({q}, {w_rest}, {st})\n")
                else:
                    if i < (path_length - 1):
                        f.write(f"({q}, {w_rest}, {st})   \n")
                    else:
                        f.write(f"({q}, {w_rest}, {st})\n")
        print("Procedimiento guardado en resultado.txt")
    else:
        print("No se generaron configuraciones (path es None).")

    if accepted:
        print(f"La cadena {w} es aceptada por el PDA.")
    else:
        print(f"La cadena {w} NO es aceptada por el PDA.")

    # Si la longitud de la cadena es <= 10, preguntar si se desea mostrar la animaci n
    w_length = sum(1 for _ in w)
    if w_length <= 10 and path is not None:
        opcion_animacion = input(" Desea  mostrar la animaci n? (s/n): 
            ").strip().lower()
        if opcion_animacion == 's':
            animarAutomata(path, accepted)
```

```
        else:
            print("Animaci n omitida.")
    elif w_length > 10:
        print("La cadena supera los 10 caracteres. No se mostrar  la animaci n.")




######################################### Programa 5 Backus-Naur Condicional IF
    ################################################
def derivar_gramatica(S, pasos):
    derivaciones = [f"Paso 1: {S}"]
    paso_actual = 2

    while pasos > 0:
        # Identificar las posibles opciones de reemplazo en S
        opciones = [simbolo for simbolo in ['S', 'A'] if simbolo in S]

        if not opciones:
            break  # Salir si no hay m s s mbolos para derivar

        # Elegir un s mbolo al azar de las opciones disponibles
        eleccion = random.choice(opciones)

        if eleccion == 'A':
            if random.choice([True, False]):
                S = S.replace('A', '(;eS)', 1)
                derivaciones.append(f"Paso {paso_actual}: Aplicamos A -> ;eS: {S}")
            else:
                S = S.replace('A', '', 1)
                derivaciones.append(f"Paso {paso_actual}: Aplicamos A ->   : {S}")
        elif eleccion == 'S':
            S = S.replace('S', '(iCtSA)', 1)
            derivaciones.append(f"Paso {paso_actual}: Aplicamos S -> iCtSA: {S}")

        pasos -= 1
        paso_actual += 1

    return derivaciones


def convertir_a_pseudocodigo(expression):

    expression = expression[1:-1]  # Eliminar los par ntesis externos
    def parse_expression(expr, indent=0):
        result = ""

        while expr:
            char = expr[0]
            expr = expr[1:]

            if char == 'i':  # if
                result += " " * indent + "if (cond) then\n"
                result += " " * indent + "{\n"
                nested, expr = parse_expression(expr, indent + 4)
                result += nested
                result += " " * indent + "}\n"
            elif char == 'A':  # then
                pass
            elif char == 'S':  # statement
                result += " " * indent + "statement\n"
                break
            elif char == ';':  # else starts
                result += " " * indent + "else\n"
                result += " " * indent + "{\n"
                nested, expr = parse_expression(expr, indent + 4)
                result += nested
                result += " " * indent + "}\n"
            elif char == ')':  # end of a block
                indent -= 4
                break

        return result, expr

    pseudocode, _ = parse_expression(expression)
```

```
    return pseudocode


def programa5():
    max_derivaciones = 1000

    # Solicitar al usuario el modo de ejecuci n
    try:
        modo = int(input("Elige el modo de ejecuci n (1 para manual, 2 para
            autom tico): "))
    except ValueError:
        print("Entrada inv lida. Se seleccionar el modo autom tico.")
        modo = 2

    # Determinar el n mero de derivaciones
    if modo == 1:
        try:
            num_derivaciones = int(input(f"Ingrese el n mero de derivaciones (hasta
                {max_derivaciones}): "))
            num_derivaciones = min(max_derivaciones, max(1, num_derivaciones))
        except ValueError:
            print("Entrada inv lida. Se usar el n mero m ximo de derivaciones.")
            num_derivaciones = max_derivaciones
    else:
        num_derivaciones = random.randint(1, max_derivaciones)

    # Derivaciones y generaci n de pseudo-c digo
    S = 'S'
    derivaciones = derivar_gramatica(S, num_derivaciones)

    with open('Bloque_2\\programa5_Derivaciones.txt', 'w', encoding='utf-8') as f:
        f.write("\n".join(derivaciones))

    if derivaciones:
        ultima_derivacion = derivaciones[-1].split(": ")[-1]
        pseudocodigo = convertir_a_pseudocodigo(ultima_derivacion)
        with open('Bloque_2\\programa5_Pseudocodigo.txt', 'w', encoding='utf-8') as f:
            f.write(pseudocodigo)

    print("Las derivaciones se han guardado en 'Derivaciones.txt'")
    print("El pseudo-c digo se ha guardado en 'Pseudocodigo.txt'")




######################################### Programa 6 M quina de Turing
    ############################################################
class TuringMachine:
    def __init__(self, Q,   ,   ,   , q0, B, F):
        self.Q = Q          # Conjunto de estados
        self.   =             # Alfabeto de entrada
        self.   =             # Alfabeto de la cinta
        self.   =             # Funci n de transici n (diccionario)
        self.q0 = q0        # Estado inicial
        self.B = B          # S mbolo en blanco
        self.F = F          # Conjunto de estados finales
        self.cinta = []     # Representaci n de la cinta (se inicializa con una cadena)
        self.cabezal = 0    # Posici n del cabezal
        self.estado = q0    # Estado actual

    def cargar_cinta(self, cadena):
        # Inicializa la cinta
        self.cinta = [self.B] + list(cadena) + [self.B]
        self.cabezal = 1

    def avanzar(self):
        if self.estado in self.F:
            return True  # La cadena es aceptada si estamos en un estado final

        len_cinta = sum(1 for _ in self.cinta)

        simbolo_actual = self.cinta[self.cabezal] if self.cabezal < len_cinta else self.B
        transicion = self.  .get((self.estado, simbolo_actual))

        if not transicion:
```

```
            return False

        nuevo_estado, nuevo_simbolo, direccion = transicion
        self.cinta[self.cabezal] = nuevo_simbolo  # Escribir en la cinta
        self.estado = nuevo_estado  # Cambiar de estado

        # dirrecion
        if direccion == 'R':
            self.cabezal += 1
            if self.cabezal == len_cinta:  # A adir blanco si se sale de la cinta
                self.cinta.append(self.B)
        elif direccion == 'L':
            self.cabezal -= 1
            if self.cabezal < 0:  # A adir blanco a la izquierda si es necesario
                self.cinta.insert(0, self.B)
                self.cabezal = 0

        return None  # Continuar la ejecuci n

    def ejecutar_con_animacion(self, archivo_salida="salida.txt",
        mostrar_animacion=True):
        if mostrar_animacion:
            screen = Screen()
            screen.setup(width=800, height=400)
            screen.tracer(0)

            tr = Turtle()
            tr.hideturtle()
            tr.penup()

            def dibujar_cinta():
                tr.clear()
                x_inicio = -300
                for i, simbolo in enumerate(self.cinta):
                    x = x_inicio + i * 50
                    tr.setpos(x, 0)
                    tr.pendown()
                    for _ in range(4):
                        tr.forward(50)
                        tr.right(90)
                    tr.penup()
                    tr.setpos(x-25, -30)
                    tr.write(simbolo, align="center", font=("Arial", 16, "normal"))

                # Dibujar el cabezal el palo
                tr.setpos(x_inicio + self.cabezal * 50 - 25, 50)
                tr.write(f"{self.estado}", align="center", font=("Arial", 16, "bold"))
                tr.setpos(x_inicio + self.cabezal * 50 - 25, 40)
                tr.setheading(270)
                tr.pendown()
                tr.forward(30)
                tr.penup()

            dibujar_cinta()
            screen.update()

        with open(archivo_salida, "w", encoding="utf-8") as archivo:
            while True:
                # Escribir el estado actual en el archivo
                cinta_con_estado = "".join(
                    self.cinta[:self.cabezal] +
                    [f"{self.estado}_"] +
                    self.cinta[self.cabezal:]
                )
                archivo.write(cinta_con_estado + "   \n")

                resultado = self.avanzar()
                if mostrar_animacion:
                    dibujar_cinta()
                    screen.update()
                    sleep(1)

                if resultado is not None:  # Terminar la ejecuci n
                    if mostrar_animacion:
                        screen.bye()
                    return resultado
```

```
662
663
664 def generarCadena(max_length=1000):
665     asignacion = random.randint(1, max_length)
666     cadena = ""
667     if asignacion > 0:
668         cadena += str(0) * asignacion  # Agregar ceros
669         cadena += str(1) * asignacion  # Agregar unos
670     return cadena
671
672
673
674 def programa6():
675     Q = {'q0', 'q1', 'q2', 'q3', 'q4'}  # Conjunto de estados
676        = {'0', '1'}                      # Alfabeto de entrada
677        = {'0', '1', 'X', 'Y', 'B'}       # Alfabeto de la cinta
678     B = 'B'                             # Blanco
679     F = {'q4'}                          # Conjunto de estados finales
680        = {                               # Funci n de transici n
681         ('q0', '0'): ('q1', 'X', 'R'),
682         ('q0', 'Y'): ('q3', 'Y', 'R'),
683         ('q1', '0'): ('q1', '0', 'R'),
684         ('q1', '1'): ('q2', 'Y', 'L'),
685         ('q1', 'Y'): ('q1', 'Y', 'R'),
686         ('q2', '0'): ('q2', '0', 'L'),
687         ('q2', 'X'): ('q0', 'X', 'R'),
688         ('q2', 'Y'): ('q2', 'Y', 'L'),
689         ('q3', 'Y'): ('q3', 'Y', 'R'),
690         ('q3', 'B'): ('q4', 'B', 'R'),
691     }
692     q0 = 'q0'  # Estado inicial
693
694
695     tm = TuringMachine(Q,   ,   ,   , q0, B, F)
696
697
698     print("Seleccione una opci n:")
699     print("1. Ingresar la cadena manualmente")
700     print("2. Generar una cadena autom ticamente")
701
702     opcion = int(input("Ingrese el n mero de su opci n: "))
703
704     if opcion == 1:
705         cadena = input("Ingrese la cadena a evaluar (m ximo 100,000 caracteres): ")
706     elif opcion == 2:
707         cadena = generarCadena()
708         print(f"La cadena generada es: {cadena}")
709
710     cadena_length = sum(1 for _ in cadena)
711     if cadena_length <= 10:
712         # Preguntar si se desea animaci n
713         opcion_animacion = input(" Desea  mostrar la animaci n? (s/n): 
            ").strip().lower()
714         mostrar_animacion = opcion_animacion == 's'
715     else:
716         print("La cadena supera los 10 caracteres. No se puede animar.")
717         return
718
719     # Cargar una cadena de entrada
720     #cadena = input("Ingrese la cadena a evaluar: ")
721     tm.cargar_cinta(cadena)
722
723
724
725     resultado =
            tm.ejecutar_con_animacion(archivo_salida="Bloque_2\\programa6_salidaTM.txt",
            mostrar_animacion=mostrar_animacion)
726     if resultado:
727         print(f"\nLa cadena {cadena} ES aceptada. Los pasos se ha guardado en 
            'salidaTM.txt'.\n")
728     else:
729         print(f"\nLa cadena {cadena} NO es aceptada. Los pasos se ha guardado en 
            'salidaTM.txt'.\n")
730
731
732
```

```
733
734 ##################### MENU PRINCIPAL ####################
735
736 def mostrar_menu():
737     print("\n==================␣MENU␣==================")
738     print("Selecciona␣un␣programa␣para␣ejecutar:")
739     print("1.␣Programa␣Buscador␣de␣palabras")
740     print("2.␣Programa␣Automata␣de␣pila")
741     print("3.␣Programa␣Backus-Naur␣Condicional␣IF")
742     print("4.␣Programa␣M quina␣de␣Turing")
743     print("5.␣Salir")
744     print("==========================================\n")
745
746
747 def main():
748     while True:
749         mostrar_menu()
750         opcion = input("Selecciona␣una␣opci n:␣")
751
752         if opcion == "1":
753             programa3()
754         elif opcion == "2":
755             programa4()
756         elif opcion == "3":
757             programa5()
758         elif opcion == "4":
759             programa6()
760         elif opcion == "5":
761             print("Saliendo␣del␣programa...")
762             break
763         else:
764             print("Opci n␣no␣v lida.␣Intenta␣de␣nuevo.")
765
766
767 if __name__ == "__main__":
768     main()
```