Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ciudad de México, México, 6 de enero de 2025 .

Segundo Bloque de Programas

Ponce Fragoso Emmanuel

Teoria de Computaciòn
Genaro Juarez
5BM2

Índice

1. Objetivo General de las Prácticas

El propósito de estas prácticas es consolidar el conocimiento y las habilidades en el diseño, implementación y análisis de modelos computacionales fundamentales para el reconocimiento y procesamiento de lenguajes formales. A través del desarrollo de diferentes programas, se busca que el estudiante aplique conceptos teóricos relacionados con autómatas finitos, autómatas de pila, gramáticas generativas y máquinas de Turing, permitiendo la resolución de problemas específicos en áreas como el reconocimiento de patrones, la derivación de lenguajes y el análisis sintáctico.

2. Buscador de Palabras

2.1. Objetivo Específico

El objetivo específico del Programa 3 es desarrollar un autómata finito determinista (DFA), derivado de la transformación de un autómata finito no determinista (NFA), para reconocer y procesar palabras clave específicas del conjunto: {acoso, acecho, agresión, víctima, violación, violencia, machista}. Este objetivo implica diseñar un NFA inicial que reconozca dichas palabras y convertirlo a su equivalente DFA mediante cálculos detallados, utilizando el método de subconjuntos y tablas de transición, todo documentado en el reporte. Además, se implementará un programa que permita analizar textos desde archivos locales o páginas web, identificando las palabras clave, contando sus apariciones y localizando su posición en el archivo (coordenadas x, y).

2.2. Introducción

El Programa 3 tiene como objetivo desarrollar un autómata finito determinista (DFA) capaz de reconocer palabras clave del conjunto {acoso, acecho, agresión, víctima, violación, violencia, machista}. Para ello, se diseña un autómata finito no determinista (NFA) que luego se transforma a su equivalente DFA mediante el método de subconjuntos, documentando cada paso del proceso.

El programa permite analizar textos desde archivos locales, identificando las palabras clave, contando sus ocurrencias y localizando su posición exacta en el archivo. Además, genera una representación gráfica del DFA y un registro detallado de las transiciones de estado por cada carácter leído.

2.3. Metodología

El desarrollo del Programa 3 se divide en varias etapas, desde el diseño teórico del autómata hasta su implementación y evaluación práctica. A continuación, se describen los pasos principales con un ejemplo ilustrativo para el reconocimiento de una palabra clave:

Diseño del NFA (Autómata Finito No Determinista)

Se construye un NFA que permita reconocer las palabras clave del conjunto dado. Cada palabra se representa como un camino en el autómata, donde los nodos son los estados y las transiciones corresponden a los caracteres de la palabra.

Ejemplo: Para la palabra .ªcoso", se define un NFA con las siguientes transiciones:

$$q_0 \xrightarrow{a} q_1 \xrightarrow{c} q_2 \xrightarrow{o} q_3 \xrightarrow{s} q_4 \xrightarrow{o} q_5$$

donde q_0 es el estado inicial y q_5 el estado final.

Conversión del NFA a DFA

Utilizando el método de subconjuntos, se transforma el NFA diseñado a su equivalente DFA. Este proceso incluye la construcción de tablas de transición que combinan estados no deterministas en estados únicos deterministas.

Ejemplo: El subconjunto $\{q_2, q_3\}$ del NFA podría transformarse en un único estado Q_A del DFA.

Implementación del DFA en Código

Se programa el DFA para que lea un archivo de texto (local o de internet) y procese cada carácter, verificando si las palabras clave son reconocidas.

Identificación y Localización de Palabras Clave

A medida que el DFA procesa el texto, registra cada aparición de una palabra clave, incluyendo su posición (x, y) en términos de líneas y columnas.

Ejemplo: En un archivo donde la palabra .ªcoso.ªparece en la línea 3, columna 15, el programa registra:

Palabra encontrada: .ªcoso", Posición: (3, 15).

Registro del Proceso del DFA

El programa genera un archivo que documenta cada transición de estado durante la evaluación del texto, permitiendo un análisis detallado del comportamiento del autómata.

Visualización Gráfica del DFA

Se utiliza una herramienta de graficación (como Graphviz o Matplotlib) para representar visualmente el DFA, mostrando los estados y transiciones de manera clara.

Ejemplo de Aplicación

Si el archivo de entrada contiene el texto:

```
La violencia de género es un problema global. El acoso y el acecho son manifestaciones graves.
```

El programa identifica las palabras clave "violencia", .ªcosoz .ªcecho", registrando:

- Palabra: "violencia", Posición: (1, 4).
- Palabra: .ªcoso", Posición: (2, 2).
- Palabra: .ªcecho", Posición: (2, 5).

Este registro se complementa con el historial completo de transiciones del DFA en un archivo generado automáticamente.

2.4. Desarrollo

Libreria

Usadas: graphviz: Sirve para crear y visualizar diagramas dirigidos, como árboles o flujos. pandas: Es una librería para manipular y analizar datos en estructuras como DataFrames. matplotlib: Se usa para crear gráficos visuales, como líneas y barras, a partir de datos.

```
from graphviz import Digraph
import pandas as pd
import matplotlib.pyplot as plt
```

Transiciones

El código define un conjunto de transiciones para un autómata determinista finito (DFA) en un diccionario de Python. Cada estado del autómata (como 'q0', 'q1', etc.) tiene transiciones especificadas para diferentes caracteres, lo que indica a qué estado debe moverse el autómata al leer una letra de entrada. Este DFA está diseñado para reconocer patrones específicos en una secuencia de texto, moviéndose entre estados según las reglas definidas para cada carácter.

```
estados_DFA = {
    'q0':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q1':{'a':'q1','b':'q0','c':'q4','d':'q0','e':'q0','f':'q0','g':'q5','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q2':{'a':'q6','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q3':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q7','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q8','ó':'q0','ú':'q0'},
    'q4':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q9','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q10','p':'q0','q'
        :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q5':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q11','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q6':{'a':'q1','b':'q0','c':'q12','d':'q0','e':'q0','f':'q0','g':'q5','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q7':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q13','p':'q0','q'
        :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q8':{'a':'q1','b':'q0','c':'q14','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q9':{'a':'q1','b':'q0','c':'q15','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z'
        :'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q10':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
        q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
        'q0','r':'q0','s':'q16','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q11':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q17','f':'q0','g':'q0','h':'q0','i':
        'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
        :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q12':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q9','f':'q0','g':'q0','h':'q18','i':
        'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q10','p':'q0','q
        ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
        z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
```

```
'q13':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
    q0','j':'q0','k':'q0','l':'q19','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q14':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
    q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
    'q0','r':'q0','s':'q0','t':'q20','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q15':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q21','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q16':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
    q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q22','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q17':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
    q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q':
    'q0','r':'q0','s':'q23','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q18':{'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':'
    q24','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q19': {'a':'q25','b':'q0','c':'q0','d':'q0','e':'q26','f':'q0','g':'q0','h':'q0','i
    ':'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','
    q':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0',
    'z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q20': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q27','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q21': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q28','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q22': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q23': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q29','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q24': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q30','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q25': {'a':'q1','b':'q0','c':'q31','d':'q0','e':'q0','f':'q0','g':'q5','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q26': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q32','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q27': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q33','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q28': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
```

```
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q29': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q34','ú':'q0'},
'q30': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q35','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q31': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q9','f':'q0','g':'q0','h':'q0','i':
    'q36','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q10','p':'q0','
    q':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0',
    'z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q32': {'a':'q1','b':'q0','c':'q37','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q33': {'a':'q38','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q34': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q39','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q35': {'a':'q40','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q36': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q41','ú':'q0'},
'q37': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q42','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q38': {'a':'q1','b':'q0','c':'q43','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q39': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q40': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
    :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
    ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q41': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
    'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q44','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q42': {'a':'q45','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
    ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
    z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
'q43': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q18','i'
    :'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q
```

```
        ':'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','
        z':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q44': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
        'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
        :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'},
    'q45': {'a':'q1','b':'q0','c':'q0','d':'q0','e':'q0','f':'q0','g':'q0','h':'q0','i':
        'q0','j':'q0','k':'q0','l':'q0','m':'q2','n':'q0','ñ':'q0','o':'q0','p':'q0','q'
        :'q0','r':'q0','s':'q0','t':'q0','u':'q0','v':'q3','w':'q0','x':'q0','y':'q0','z
        ':'q0','á':'q0','é':'q0','í':'q0','ó':'q0','ú':'q0'}
}
```

Estados Finales

La función analizar_archivo parece diseñada para analizar un archivo de texto y buscar palabras clave específicas como .ªcoso", .ªcecho", "violencia", entre otras, registrando cuántas veces aparecen y dónde se ubican. Esto se logra utilizando un diccionario llamado estados_finales, donde cada clave (como 'q22' o 'q28') representa un estado asociado a una palabra clave, junto con sus propiedades: ocurrencias (un contador inicializado en 0) y ubicaciones (una lista vacía para almacenar las posiciones donde aparece la palabra).

```
def analizar_archivo(archivo_texto):
    estados_finales = {
        'q22': {
            'palabra': 'acoso',
            'longitud': 5,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q28': {
            'palabra': 'acecho',
            'longitud': 6,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q38': {
            'palabra': 'víctima',
            'longitud': 7,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q39': {
            'palabra': 'agresión',
            'longitud': 8,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q40': {
            'palabra': 'machista',
            'longitud': 8,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q44': {
            'palabra': 'violación',
            'longitud': 9,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        },
        'q45': {
```

```
            'palabra': 'violencia',
            'longitud': 9,  # Longitud predefinida de la palabra
            'ocurrencias': 0,
            'ubicaciones': []
        }
    }
```

Implementa un análisis de texto usando un Autómata Finito Determinista (DFA) para identificar palabras clave en un archivo. Inicializa el estado actual en 'q0' y utiliza una lista, registro_estados_DFA, para registrar las transiciones entre estados. Abre el archivo especificado en modo lectura con codificación y recorre línea por línea, aumentando el contador de líneas y columnas según avanza. Para cada carácter, actualiza el estado_actual utilizando la tabla de transiciones estados_DFA, retrocediendo al estado inicial 'q0' si no hay una transición válida. Si el estado actual pertenece a los estados finales, incrementa el contador de ocurrencias asociado a ese estado y calcula la posición en el texto donde se encuentra la palabra clave, añadiéndola a una lista de ubicaciones. Además, registra cada transición (estado previo, carácter leído, nuevo estado) en registro_estados_DFA para rastrear el flujo del DFA durante la lectura del archivo.

```
    estado_actual = 'q0'
    registro_estados_DFA = []
    with open(archivo_texto, "r", encoding="utf-8") as archivo:
        linea_actual = 0
        for linea in archivo:
            columna_actual = 1
            linea_actual += 1
            for caracter in linea:
                columna_actual += 1
                estado_anterior = estado_actual
                estado_actual = estados_DFA.get(estado_actual, {}).get(caracter, 'q0')
                if estado_actual in estados_finales:
                    estados_finales[estado_actual]['ocurrencias'] += 1
                    longitud_palabra = estados_finales[estado_actual]['longitud']
                    estados_finales[estado_actual]['ubicaciones'].append((columna_actual
                        - longitud_palabra, linea_actual))
                registro_estados_DFA.append((estado_anterior, caracter, estado_actual))

    return estados_finales, registro_estados_DFA
```

Exportar los Resultados

La función exportar_resultados genera un archivo de texto con los resultados del análisis de palabras, escribiendo el contenido de un diccionario estados_finales en formato legible. Abre el archivo especificado en archivo_salida en modo escritura con codificación, escribe un encabezado, y luego recorre el diccionario para cada estado, exportando la palabra analizada, el número de ocurrencias y sus ubicaciones en el texto procesado. Esto organiza la información en secciones claras dentro del archivo de salida.

```
def exportar_resultados(archivo_salida, estados_finales):
    with open(archivo_salida, 'w', encoding='utf-8') as archivo:
        archivo.write("Resultados del análisis:\n")
        for estado, detalles in estados_finales.items():
            archivo.write(f"Palabra: {detalles['palabra']}\n")
            archivo.write(f"  Ocurrencias: {detalles['ocurrencias']}\n")
            archivo.write(f"  Ubicaciones: {detalles['ubicaciones']}\n\n")
```

Exportar los Estados

La función exportar_estados_DFA toma dos parámetros: archivo_estados_DFA (el nombre del archivo donde se guardará la información) y registro (una lista de transiciones en el autómata determinista

finito, DFA). Abre el archivo especificado en modo de escritura con codificación. Luego, escribe un encabezado "Historial de estados_DFA:.[en] el archivo. Posteriormente, recorre cada transición en el registro, donde cada transición está formada por tres elementos: el estado inicial, el símbolo y el estado final.

```
def exportar_estados_DFA(archivo_estados_DFA, registro):
    with open(archivo_estados_DFA, 'w', encoding='utf-8') as archivo:
        archivo.write("Historial de estados_DFA:\n")
        for transicion in registro:
            estado_inicial, simbolo, estado_final = transicion
            archivo.write(f"8({estado_inicial}, '{simbolo}') -> {estado_final}\n")
```

Creaciòn de NFA

Genera un gráfico visual de un Autómata Finito No Determinista (NFA) utilizando el paquete graphviz. Recibe como parámetros el NFA, el estado inicial, los estados de aceptación, y opcionalmente el nombre del archivo de salida. Crea un gráfico dirigido configurado para mostrar los nodos de izquierda a derecha. Luego, agrega un nodo vacío como punto de inicio y lo conecta al estado inicial. Recorre las transiciones del NFA, añadiendo al gráfico los nodos correspondientes y dibujando las aristas etiquetadas con los símbolos que representan las transiciones. Los estados de aceptación se distinguen mediante un diseño de nodo de doble círculo y la función guarda y renderiza el gráfico como un archivo de imagen PNG y lo muestra al usuario.

```
def create_nfa_graph(nfa, initial_state, accepting_states, output_file='nfa_graph'):
    # Crear el gráfico
    nfa_graph = Digraph(format='png', engine='dot')

    # Configuración de dirección izquierda a derecha
    nfa_graph.attr(rankdir='LR')
    nfa_graph.attr('node', shape='circle')

    # Agregar el estado inicial
    nfa_graph.node("", shape="none")
    nfa_graph.edge("", str(initial_state))

    # Agregar nodos y transiciones
    for state, transitions in nfa.items():
        for symbol, next_states in transitions.items():
            for next_state in next_states:
                nfa_graph.edge(str(state), str(next_state), label=symbol)

    # Agregar estados finales
    for accepting_state in accepting_states:
        nfa_graph.node(str(accepting_state), shape='doublecircle')

    # Guardar y renderizar el gráfico
    nfa_graph.render(output_file, view=True)

# Definir los estados y transiciones
nfa = {
    1: {'a': [2, 7, 13], 'v': [21, 28, 37], 'm': [46], 'E': [1]},
    2: {'c': [3]},
    3: {'o': [4]},
    4: {'s': [5]},
    5: {'o': [6]},
    7: {'c': [8]},
    8: {'e': [9]},
    9: {'c': [10]},
    10: {'h': [11]},
    11: {'o': [12]},
```

```
    13: {'g': [14]},
    14: {'r': [15]},
    15: {'e': [16]},
    16: {'s': [17]},
    17: {'i': [18]},
    18: {'o': [19]},
    19: {'n': [20]},
    21: {'i': [22]},
    22: {'c': [23]},
    23: {'t': [24]},
    24: {'i': [25]},
    25: {'m': [26]},
    26: {'a': [27]},
    28: {'i': [29]},
    29: {'o': [30]},
    30: {'l': [31]},
    31: {'a': [32]},
    32: {'c': [33]},
    33: {'i': [34]},
    34: {'o': [35]},
    35: {'n': [36]},
    37: {'i': [38]},
    38: {'o': [39]},
    39: {'l': [40]},
    40: {'e': [41]},
    41: {'n': [42]},
    42: {'c': [43]},
    43: {'i': [44]},
    44: {'a': [45]},
    46: {'a': [47]},
    47: {'c': [48]},
    48: {'h': [49]},
    49: {'i': [50]},
    50: {'s': [51]},
    51: {'t': [52]},
    52: {'a': [53]},
}

initial_state = 1
accepting_states = {6, 12, 20, 27, 36, 45, 53}
```

Creaciòn de DFA

Crea un gráfico que representa un Autómata Finito Determinista (DFA) utilizando el paquete graphviz. Define un conjunto de estados finales (estados_finales) y utiliza un objeto Digraph para construir el gráfico. Para cada estado y sus transiciones en el diccionario estados_DFA, agrega nodos al gráfico: los estados finales se representan con doble círculo, mientras que los demás tienen forma de círculo simple. Luego, para cada símbolo en las transiciones, agrega una arista dirigida entre el estado actual y el estado destino, etiquetada con el símbolo de la transición. Finalmente, renderiza y guarda el gráfico en un archivo PNG llamado grafo_DFA y lo muestra automáticamente al usuario.

```
def generar_grafo():
    estados_finales = {'q22', 'q28', 'q38', 'q39', 'q40', 'q44', 'q45'}
    grafo_DFA = Digraph(format='png')
    for estado, trans in estados_DFA.items():
        if estado in estados_finales:
            grafo_DFA.node(estado, estado, shape='doublecircle')
        else:
            grafo_DFA.node(estado, estado, shape='circle')
```

```
        for simbolo, estado_destino in trans.items():
            grafo_DFA.edge(estado, estado_destino, label=simbolo)
    grafo_DFA.render('grafo_DFA', view=True)
```

Generaciòn de Tabla

La función toma como argumento un diccionario estados_DFA y un nombre de archivo nombre_archivo para guardar la tabla como una imageny dentro de la función, se construye el DataFrame y se genera la tabla.

```
def guardar_tabla_estados_DFA(estados_DFA, nombre_archivo='tabla_estados_DFA.png'):
    # Crear un DataFrame a partir del diccionario
    df = pd.DataFrame(estados_DFA).T

    # Crear la figura y el eje
    fig, ax = plt.subplots(figsize=(15, 8))  # Ajusta el tamaño de la figura

    # Dibujar la tabla
    ax.axis('off')  # Eliminar los ejes
    tabla = ax.table(
        cellText=df.values,
        colLabels=df.columns,
        rowLabels=df.index,
        cellLoc='center',
        loc='center',
    )

    # Ajustar el tamaño de la fuente
    tabla.auto_set_font_size(False)
    tabla.set_fontsize(8)
    tabla.auto_set_column_width(col=list(range(len(df.columns))))

    # Guardar la tabla como PNG
    plt.savefig(nombre_archivo, bbox_inches='tight', dpi=300)
    print(f"Tabla guardada como '{nombre_archivo}'.")
```

Generar los Archivos y Grafos

La función procesar_datos coordina varias tareas relacionadas con el análisis de texto utilizando un DFA (Autómata Finito Determinista) y la generación de gráficos de autómatas. Define los nombres de los archivos de entrada y salida: uno para el texto de entrada, otro para los resultados, y un tercero para el historial de estados del DFA. Llama a la función analizar_archivo, que analiza el texto y devuelve los estados finales alcanzados y un registro de las transiciones del DFA. Exporta los resultados a un archivo de texto y genera un gráfico visual del NFA (Autómata Finito No Determinista) utilizando create_nfa_graph. Luego, guarda el historial de estados del DFA en un archivo y genera un gráfico del DFA mediante generar_grafo. Finalmente, imprime mensajes informativos en cada paso para confirmar la realización de las tareas. La función principal ejecuta procesar_datos al ser invocada directamente.

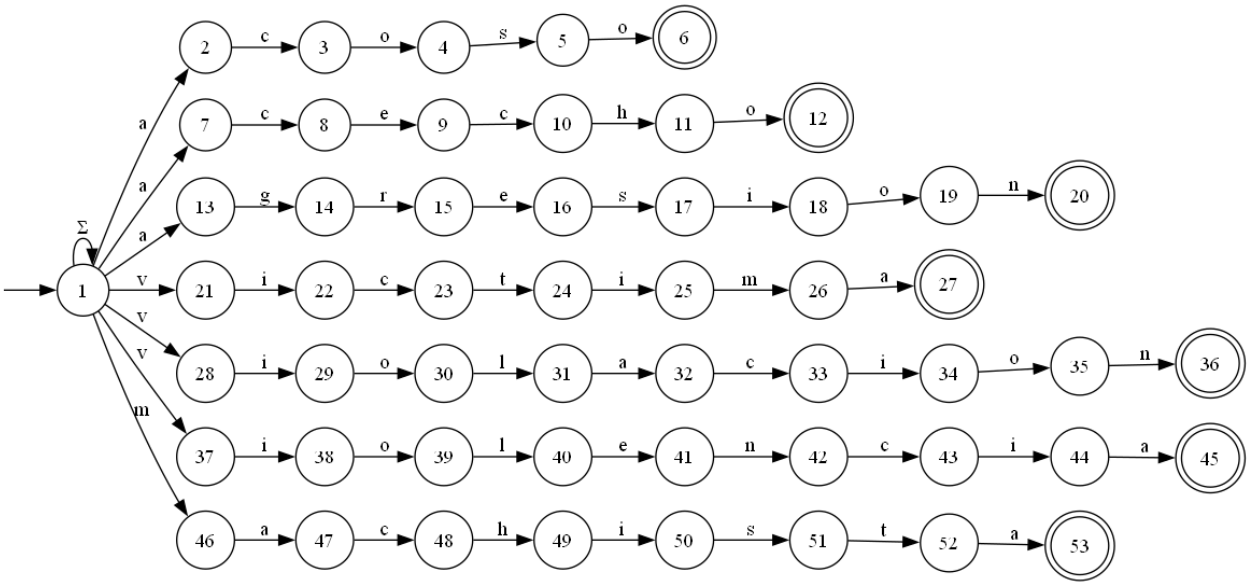
```
def procesar_datos():
    archivo_entrada = 'TextoPrueba.txt'
    archivo_salida_resultados = 'resultados.txt'
    archivo_salida_estados_DFA = 'estados_DFA.txt'

    print("Analizando archivo de texto usando el DFA...")
    estados_finales, registro_estados_DFA = analizar_archivo(archivo_entrada)

    exportar_resultados(archivo_salida_resultados, estados_finales)
```

```
    print(f"Resultados exportados a {archivo_salida_resultados}")

    guardar_tabla_estados_DFA(estados_DFA)
    print("Tabla de Estados de DFA generado y guardado.")

    create_nfa_graph(nfa, initial_state, accepting_states)
    print("Grafo del NFA generado y guardado.")

    exportar_estados_DFA(archivo_salida_estados_DFA, registro_estados_DFA)
    print(f"Historial de estados_DFA exportado a {archivo_salida_estados_DFA}")

    generar_grafo()
    print("Grafo del DFA generado y guardado.")

if __name__ == "__main__":
    procesar_datos()
```

2.5. Tabla

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	ñ	o	p	q	r	s	t	u	v	w	x	y	z	á	é	í	ó	ú
q0	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q1	q1	q0	q4	q0	q0	q0	q5	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q2	q6	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q3	q1	q0	q0	q0	q0	q0	q0	q0	q7	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q8	q0	q0
q4	q1	q0	q0	q0	q9	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q10	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q5	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q11	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q6	q1	q0	q12	q0	q0	q0	q5	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q7	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q13	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q8	q1	q0	q14	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q9	q1	q0	q15	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q10	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q16	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q11	q1	q0	q0	q0	q17	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q12	q1	q0	q0	q0	q9	q0	q0	q18	q0	q0	q0	q0	q2	q0	q0	q10	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q13	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q19	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q14	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q20	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q15	q1	q0	q0	q0	q0	q0	q0	q21	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q16	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q22	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q17	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q23	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q18	q1	q0	q0	q0	q0	q0	q0	q0	q24	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q19	q25	q0	q0	q0	q26	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q20	q1	q0	q0	q0	q0	q0	q0	q0	q27	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q21	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q28	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q22	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q23	q1	q0	q0	q0	q0	q0	q0	q0	q29	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q24	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q30	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q25	q1	q0	q31	q0	q0	q0	q5	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q26	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q32	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q27	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q33	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q28	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q29	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q34	q0
q30	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q35	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q31	q1	q0	q0	q0	q9	q0	q0	q0	q36	q0	q0	q0	q2	q0	q0	q10	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q32	q1	q0	q37	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q33	q38	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q34	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q39	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q35	q40	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q36	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q41	q0
q37	q1	q0	q0	q0	q0	q0	q0	q0	q42	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q38	q1	q0	q43	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q39	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q40	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q41	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q44	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q42	q45	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q43	q1	q0	q0	q0	q0	q0	q0	q18	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q44	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0
q45	q1	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	q2	q0	q0	q0	q0	q0	q0	q0	q0	q0	q3	q0	q0	q0	q0	q0	q0	q0	q0	q0

2.6. Grafo NFA



2.7. Texto a Analizar

La violencia machista es una de las formas más extendidas de opresión, afectando principalmente a mujeres y niñas. Las víctimas suelen enfrentar múltiples tipos de acoso, desde el acecho constante hasta agresiones verbales y físicas. Este fenómeno, que muchas veces comienza con pequeños actos de intimidación, puede escalar a formas más graves de violencia, como la agresión física o sexual, e incluso culminar en situaciones de violación.

El acoso callejero, laboral y digital constituye una de las formas más normalizadas de esta problemática. Las mujeres que son objeto de acecho, ya sea por exparejas, desconocidos o incluso compañeros de trabajo, viven con un miedo constante que afecta su calidad de vida. Este acecho puede manifestarse en seguimientos, mensajes no deseados y control obsesivo, representando una clara amenaza a su libertad y seguridad.

Por otro lado, la agresión, en cualquiera de sus formas, busca someter a las víctimas y perpetuar una relación de poder desigual. La violación, como expresión máxima de esta violencia, no solo daña el cuerpo, sino que también deja profundas heridas emocionales y psicológicas que pueden acompañar a las víctimas durante toda su vida.

Combatir la violencia machista requiere un esfuerzo colectivo. Denunciar cada caso de acoso, agresión o acecho, apoyar a las víctimas y rechazar las actitudes que perpetúan esta cultura son pasos fundamentales. Solo a través de la visibilización y el compromiso podremos erradicar la violencia de género y garantizar una sociedad más justa y libre de machismo.

2.8. Resultados del Texto Analizado

```
Resultados del análisis:
Palabra: acoso
  Ocurrencias: 3
  Ubicaciones: [(165, 1), (4, 3), (87, 7)]

Palabra: acecho
  Ocurrencias: 4
  Ubicaciones: [(181, 1), (137, 3), (278, 3), (105, 7)]

Palabra: víctima
  Ocurrencias: 4
  Ubicaciones: [(120, 1), (78, 5), (304, 5), (126, 7)]

Palabra: agresión
  Ocurrencias: 3
  Ubicaciones: [(368, 1), (19, 5), (94, 7)]

Palabra: machista
  Ocurrencias: 2
  Ubicaciones: [(14, 1), (23, 7)]

Palabra: violación
  Ocurrencias: 2
  Ubicaciones: [(431, 1), (134, 5)]

Palabra: violencia
  Ocurrencias: 5
  Ubicaciones: [(4, 1), (349, 1), (175, 5), (13, 7), (285, 7)]
```

2.9. Grafo DFA

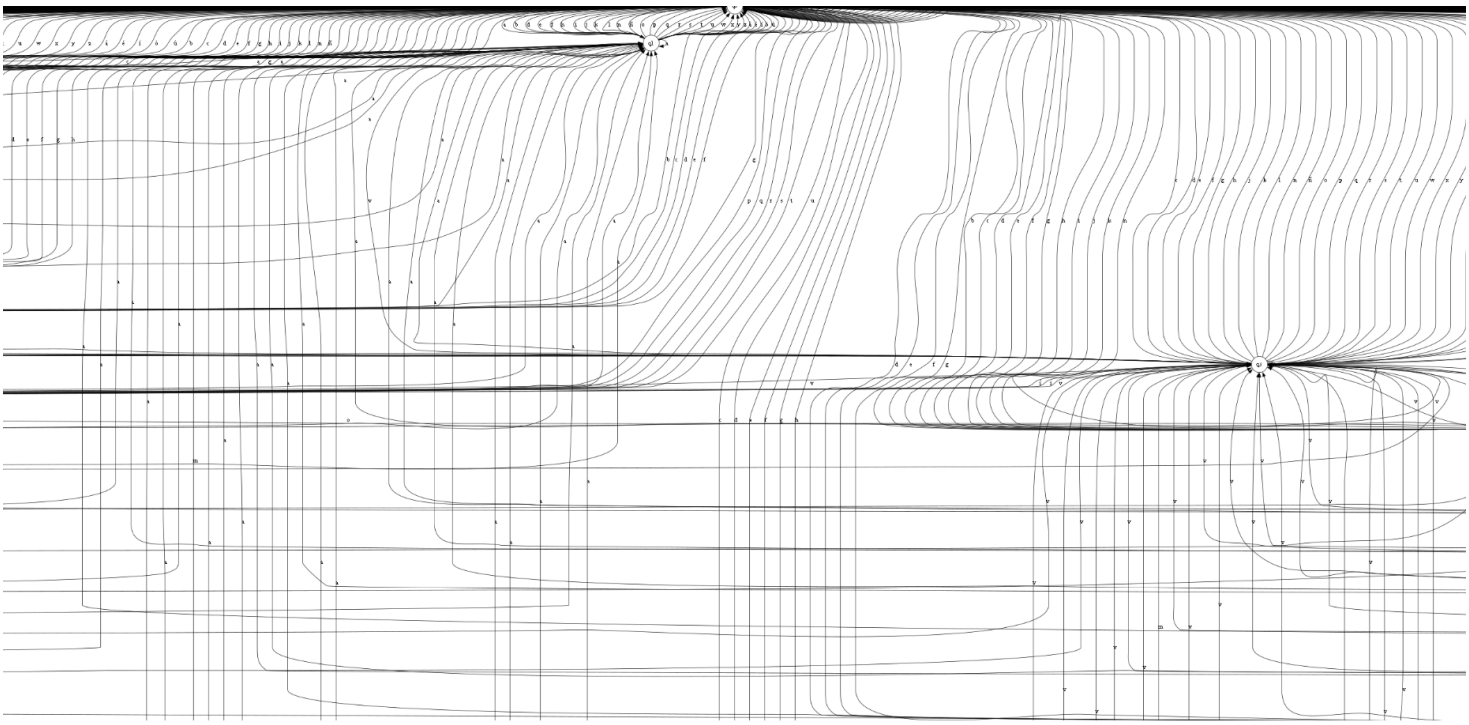


Figura 1: Un Fragmento del DFA

3. Autómata de Pila

3.1. Objetivo Especìfico

El objetivo del programa es implementar un autómata de pila que evalúe y reconozca el lenguaje libre de contexto $L = \{0^n 1^n \mid n \geq 1\}$, permitiendo procesar cadenas ingresadas manualmente o generadas automáticamente (hasta 100,000 caracteres). El programa debe registrar las descripciones instantáneas (IDs) del procesamiento del autómata, mostrando en pantalla y guardando en un archivo los estados, contenido de la pila y la entrada restante en cada pasoy se incluirá una animación gráfica del funcionamiento del autómata para cadenas de hasta 10 caracteres, brindando una visualización interactiva del proceso.

3.2. Introducciòn

En el campo de la teoría de autómatas, los autómatas de pila son herramientas fundamentales para el reconocimiento de lenguajes libres de contexto, los cuales son de gran importancia en el procesamiento de lenguajes formales y programación de compiladores, se busca entender y demostrar cómo una máquina con pila puede manejar este tipo de lenguajes, procesando secuencias de longitud variable y registrando el estado de cada transición.

3.3. Metodología

El objetivo del programa es desarrollar un autómata de pila que sea capaz de reconocer el lenguaje. Este lenguaje está compuesto por cadenas formadas por una cantidad igual de ceros seguidos por unos, y es un ejemplo típico de un lenguaje libre de contexto.

1. Estructura del autómata de pila

El autómata de pila se implementa mediante una estructura de pila en Python, representada por las clases `NodoPila` y `EstructuraPila`. La clase `NodoPila` define un nodo de la pila que contiene un valor y una referencia al siguiente nodo. La clase `EstructuraPila` gestiona la pila, proporcionando métodos para insertar y quitar elementos de la misma. El autómata de pila inicia con una pila vacía y procesará la cadena de entrada desde el primer carácter hasta el último.

2. Generación de la cadena

La cadena de entrada puede ser proporcionada por el usuario o generada aleatoriamente. Si el usuario opta por la opción automática, la longitud de la cadena será aleatoria, con un máximo de 100,000 caracteres. La cadena generada puede contener tanto ceros como unos en posiciones aleatorias.

3. Evaluación del autómata de pila

El autómata sigue una serie de transiciones basadas en el símbolo de entrada y el símbolo en la cima de la pila. El autómata comienza en el estado q y procesa la cadena de la siguiente manera:

- Si el símbolo de entrada es 0, se inserta un x en la pila.
- Si el símbolo de entrada es 1, se extrae un x de la pila. Si la pila está vacía antes de que se termine de procesar la cadena, la cadena es rechazada.
- Si la cadena está vacía y la pila también está vacía (solo contiene el símbolo z), la cadena es aceptada.

A medida que el autómata procesa cada símbolo, se registran las transiciones y se generan descripciones instantáneas de los estados. Estas descripciones se guardan en un archivo de texto y se muestran en la pantalla.

4. Animación del autómata

Si la cadena tiene 10 caracteres o menos, el programa anima el autómata utilizando la librería `turtle`. El estado del autómata y la pila se muestran en tiempo real, con cada transición representada gráficamente. El cuadro y las flechas indican el flujo de las transiciones, mientras que la pila se visualiza en la parte inferior de la ventana.

5. Ejemplo de ejecución

Supongamos que el usuario ingresa la cadena "000111". El autómata comenzará en el estado q, con la cadena "000111z la pila vacía (z). Procesando la cadena:

- Primero, el autómata lee el primer 0, inserta un x en la pila y avanza al siguiente símbolo.
- Luego, lee el siguiente 0, inserta otro x en la pila y continúa.
- Después, lee el 1, extrae un x de la pila y avanza.
- Finalmente, procesa los dos 1 restantes, extrayendo los x de la pila hasta que la pila esté vacía.
- Al finalizar, si la cadena está vacía y la pila solo contiene z, la cadena es aceptada.

El proceso de cada transición se mostrará en pantalla y se almacenará en un archivo `Resultado.txt`.

3.4. Desarrollo

Librerìas

Usadas:
random: En Python permite generar números aleatorios. Puedes utilizarla para obtener valores aleatorios dentro de un rango determinado, para elegir elementos de una secuencia de manera aleatoria, entre otras funcionalidades.
turtle: Es una librería en Python que proporciona una forma divertida de aprender a programar gráficos. Básicamente, simula un "tortuga"que se mueve en la pantalla y dibuja líneas.
La línea **from time import sleep** en Python importa la función sleep desde el módulo time. El propósito de la función sleep es pausar o retrasar la ejecución del programa durante un período específico de tiempo, en segundos.

```
import random
import turtle
from time import sleep
```

Pila

La clase `NodoPila` es una representación de un nodo en una estructura de datos tipo *pila* (stack). Un nodo en una pila generalmente tiene dos atributos:

- **dato**: almacena el valor del nodo, que puede ser cualquier tipo de dato. En este caso, el valor se pasa como argumento al crear el nodo.
- **siguiente**: es una referencia al siguiente nodo en la pila. Inicialmente se establece como `None`, lo que significa que no hay ningún nodo siguiente cuando se crea un nuevo nodo.

```
class NodoPila:
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None
```

La clase `EstructuraPila` implementa una estructura de datos tipo **pila** (*stack*) basada en nodos. Esta estructura sigue el principio de **LIFO** (*Last In, First Out*), donde el último elemento que se inserta es el primero en salir. A continuación, se explican los métodos principales de la clase:

Estructura de la Pila

- `__init__(self)`: Este es el constructor de la clase. Inicializa el atributo `tope`, que apunta al nodo en la parte superior de la pila. Al inicio, la pila está vacía, por lo que `tope` se establece como `None`.
- `insertar(self, dato)`: Este método permite agregar un nuevo elemento a la pila.
- `quitar(self)`: Este método elimina y devuelve el elemento en el tope de la pila.

```
class EstructuraPila:
    def __init__(self):
        self.tope = None

    def insertar(self, dato):
        if self.tope is None:
            self.tope = NodoPila(dato)
            return
        nuevo_nodo = NodoPila(dato)
        nuevo_nodo.siguiente = self.tope
        self.tope = nuevo_nodo

    def quitar(self):
        if self.tope is None:
            return "z"
        temp = self.tope
        self.tope = self.tope.siguiente
        return temp.dato

    # Concatenar ceros y unos
    cadena = ceros + unos

    return cadena
```

Generaciòn de Cadena Aleatoria

Parametros:
Longitud par: Si el número aleatorio es impar, se ajusta para garantizar que sea par.
Mitades iguales: Calcula la mitad de la longitud y genera 0 para la primera mitad y 1 para la segunda mitad.
Ordenado: Los 0 siempre estarán primero, seguidos por los 1, tal como pediste.

```
def generarCadena():
    # Generar un número aleatorio dentro del rango de 2 a 100,000, asegurándonos de que
        sea par
    numeroAleatorio = random.randint(2, 100000)
    if numeroAleatorio % 2 != 0:  # Asegurarse de que el número sea par
        numeroAleatorio += 1
```

```
    mitad = numeroAleatorio // 2  # Dividir la longitud entre dos
    ceros = '0' * mitad            # Generar la mitad de ceros
    unos = '1' * mitad             # Generar la mitad de unos
```

Dibujar con Turtle

- **Parámetro:**
  - `turtle_instance`: Es una instancia del objeto `Turtle`, que se utiliza para dibujar en la pantalla.
- **Dibujo del cuadro:**
  - La función comienza configurando la velocidad del dibujo (`speed(8)`) para que sea relativamente rápido.
  - Utiliza `penup()` para mover la posición del lápiz sin dibujar y establece el punto inicial del cuadro en (-100, 100).
  - Activa el lápiz con `pendown()` y comienza a rellenar el cuadro con `begin_fill()`.
  - Dentro de un bucle de 4 iteraciones, dibuja un rectángulo alternando los colores de relleno entre azul claro (`lightblue`) y verde claro (`lightgreen`), con bordes de color azul (`pencolor("blue")`).
  - Después de dibujar el cuadro, cierra el relleno con `end_fill()`.
- **Dibujo de las flechas:**
  - La primera flecha apunta hacia arriba, comenzando desde el centro superior del cuadro (-50, 100).
    - Mueve el lápiz a la posición inicial con `penup()`.
    - Dibuja una línea recta hacia arriba y luego agrega dos líneas diagonales para formar la punta de la flecha.
  - La segunda flecha apunta hacia abajo, comenzando desde el centro inferior del cuadro (-50, 0).
    - Siguiendo el mismo proceso, dibuja la línea recta hacia abajo y las líneas diagonales para formar la punta de la flecha.

```
def dibujarCuadroYFlechas(turtle_instance):
    # Dibuja el cuadro
    turtle_instance.speed(8)
    turtle_instance.penup()
    turtle_instance.setpos(-100, 100)
    turtle_instance.pendown()
    turtle_instance.begin_fill()
    for lado in range(4):
        if lado % 2 == 0:
            turtle_instance.fillcolor("lightblue")  # Cambiar color del cuadro
            turtle_instance.pencolor("blue")
        else:
            turtle_instance.fillcolor("lightgreen")  # Cambiar color del cuadro
            turtle_instance.pencolor("blue")
        turtle_instance.forward(100)
        turtle_instance.right(90)
    turtle_instance.end_fill()

    # Dibuja las flechas
    turtle_instance.penup()
    turtle_instance.setpos(-50, 100)
```

```
    turtle_instance.pendown()
    turtle_instance.setpos(-50, 150)
    turtle_instance.setpos(-45, 140)
    turtle_instance.setpos(-50, 150)
    turtle_instance.setpos(-55, 140)

    turtle_instance.penup()
    turtle_instance.setpos(-50, 0)
    turtle_instance.pendown()
    turtle_instance.setpos(-50, -50)
    turtle_instance.setpos(-45, -40)
    turtle_instance.setpos(-50, -50)
    turtle_instance.setpos(-55, -40)

    turtle_instance.hideturtle()
```

Animaciòn

La función `animarAutomata(cadena)` tiene como objetivo simular el funcionamiento de un autómata de pila utilizando la librería `turtle` para representar gráficamente las transiciones de estado, el contenido de la pila y la cadena restante. A continuación, se explican las principales partes de la función:

Inicialización

La función comienza creando una instancia de `EstructuraPila`, que representa la pila del autómata, y una instancia de `turtle.Turtle()`, utilizada para dibujar en la ventana gráfica. Se configura la velocidad de la tortuga con `speed(8)` y se dibujan los elementos iniciales (el cuadro y las flechas) mediante la función auxiliar `dibujarCuadroYFlechas()`.

Preparación de Variables

Se inicializan las variables `cadenaMod`, que contiene la cadena a procesar, y `cadenaPila`, que representa el contenido actual de la pila, comenzando con el símbolo base `z`. También se define la variable `resultado`, que almacenará el seguimiento de las transiciones del autómata en formato texto.

Validación de la Cadena

Se realiza una validación preliminar de la cadena. Si el primer carácter es `'1'`, se considera automáticamente no aceptada y se termina el proceso.

Procesamiento de la Cadena

El procesamiento de la cadena se realiza utilizando un bucle `for` que recorre cada carácter de la cadena. Se emplea una estructura de control `match-case` para definir las transiciones entre los estados del autómata:

- **Estado 0:** Si el carácter actual es `'0'`, se inserta un símbolo `'x'` en la pila, se elimina el primer carácter de `cadenaMod` y se actualiza `cadenaPila`. Si el carácter es `'1'`, se cambia al estado 1, se elimina un elemento de la pila y se ajustan las variables.
- **Estado 1:** Si el carácter actual es `'1'`, se elimina un elemento de la pila. Si la pila llega a su símbolo base `z`, se considera que la cadena ha sido procesada correctamente. Si se encuentra un carácter no válido, el autómata rechaza la cadena.

Finalización

Al finalizar el recorrido de la cadena: La función utiliza `turtle` para mostrar el resultado final gráficamente, junto con el contenido de la pila. Además, se escribe el contenido de `resultado` en un archivo de texto llamado `Resultado.txt`.

```
def animarAutomata(cadena):
    pila = EstructuraPila()
    turtle_instance = turtle.Turtle()  # Instancia de turtle
    turtle_instance.speed(8)

    # Dibujar el cuadro y las flechas al inicio
    dibujarCuadroYFlechas(turtle_instance)

    # Cadena y pila inicial
    cadenaMod = cadena
    cadenaPila = "z"
    resultado = "(q," + cadenaMod + "," + cadenaPila + ")    \n"
    turtle_instance.penup()
    turtle_instance.setpos(-53, 160)
    turtle_instance.pendown()
    turtle_instance.write(cadenaMod, False, align="left", font=("Lucida Console", 20))

    cont = -80
    for letra in cadenaPila:
        turtle_instance.penup()
        turtle_instance.setpos(-50, cont)
        turtle_instance.pendown()
        turtle_instance.write(letra, False, align="center", font=("Lucida Console", 20))
        cont -= 20
    sleep(1)

    if cadenaMod[0:1] == '1':
        resultado = "La cadena no es aceptada"
    else:
        turtle_instance.clear()
        valido = 1
        estado = 0
        for i in cadena:
            turtle_instance.clear()
            dibujarCuadroYFlechas(turtle_instance)  # Re-dibujar el cuadro y las flechas
                antes de cada acción
            match estado:
                case 0:
                    if int(i) == 0:
                        pila.insertar("x")
                        cadenaMod = cadenaMod[1:]
                        cadenaPila = "X" + cadenaPila
                        resultado += "(q," + cadenaMod + "," + cadenaPila + ")    \n"
                    elif int(i) == 1:
                        estado = 1
                        cadenaMod = cadenaMod[1:]
                        cadenaPila = cadenaPila[1:]
                        if cadenaMod == "":
                            resultado += "(p,e," + cadenaPila + ")    \n"
                            cadenaMod = "e"
                        else:
                            resultado += "(p," + cadenaMod + "," + cadenaPila + ")    \n"
                        pila.quitar()
                case 1:
                    if int(i) == 1:
                        dato = pila.quitar()
```

```
                    if dato == "z":
                        turtle_instance.penup()
                        turtle_instance.setpos(-53, 160)
                        turtle_instance.pendown()
                        turtle_instance.write(cadenaMod, False, align="left", font=(
                            "Lucida Console", 20))
                        turtle_instance.penup()
                        turtle_instance.setpos(-50, 40)
                        turtle_instance.pendown()
                        turtle_instance.write('f', False, align="center", font=("
                            Lucida Console", 20))
                        cont = -80
                        for letra in cadenaPila:
                            turtle_instance.penup()
                            turtle_instance.setpos(-50, cont)
                            turtle_instance.pendown()
                            turtle_instance.write(letra, False, align="center", font
                                =("Lucida Console", 20))
                            cont -= 20
                        sleep(1)
                        break
                    cadenaMod = cadenaMod[1:]
                    cadenaPila = cadenaPila[1:]
                    if cadenaMod == "":
                        resultado += "(p,e," + cadenaPila + ")    \n"
                        cadenaMod = "e"
                    else:
                        resultado += "(p," + cadenaMod + "," + cadenaPila + ")    \n"
                else:
                    turtle_instance.penup()
                    turtle_instance.setpos(-53, 160)
                    turtle_instance.pendown()
                    turtle_instance.write(cadenaMod, False, align="left", font=("
                        Lucida Console", 20))
                    turtle_instance.penup()
                    turtle_instance.setpos(-50, 40)
                    turtle_instance.pendown()
                    turtle_instance.write('f', False, align="center", font=("Lucida
                        Console", 20))
                    cont = -80
                    for letra in cadenaPila:
                        turtle_instance.penup()
                        turtle_instance.setpos(-50, cont)
                        turtle_instance.pendown()
                        turtle_instance.write(letra, False, align="center", font=("
                            Lucida Console", 20))
                        cont -= 20
                    sleep(1)
                    break

            turtle_instance.penup()
            turtle_instance.setpos(-53, 160)
            turtle_instance.pendown()
            turtle_instance.write(cadenaMod, False, align="left", font=("Lucida Console"
                , 20))
            turtle_instance.penup()
            turtle_instance.setpos(-50, 40)
            turtle_instance.pendown()

            if cadenaMod[0:1] == '0':
                turtle_instance.write('q', False, align="center", font=("Lucida Console"
```

```
                , 20))
        else:
            turtle_instance.write('p', False, align="center", font=("Lucida Console"
                , 20))
        cont = -80
        for letra in cadenaPila:
            turtle_instance.penup()
            turtle_instance.setpos(-50, cont)
            turtle_instance.pendown()
            turtle_instance.write(letra, False, align="center", font=("Lucida
                Console", 20))
            cont -= 20
        sleep(1)

    turtle_instance.penup()
    turtle_instance.setpos(-53, 160)
    turtle_instance.pendown()

    dato = pila.quitar()
    if dato == "z" or valido == 0:
        resultado += "(f,e,z)"
        turtle_instance.clear()
        dibujarCuadroYFlechas(turtle_instance)  # Dibujar las flechas y el cuadro al
            final
        turtle_instance.penup()
        turtle_instance.setpos(-53, 160)
        turtle_instance.pendown()
        turtle_instance.write('e', False, align="left", font=("Lucida Console", 20))
        turtle_instance.penup()
        turtle_instance.setpos(-50, 40)
        turtle_instance.pendown()
        turtle_instance.write('f', False, align="center", font=("Lucida Console",
            20))
        cont = -80
        for letra in cadenaPila:
            turtle_instance.penup()
            turtle_instance.setpos(-50, cont)
            turtle_instance.pendown()
            turtle_instance.write(letra, False, align="center", font=("Lucida
                Console", 20))
            cont -= 20
        sleep(1)

        turtle_instance.penup()
        turtle_instance.setpos(-153, 200)
        turtle_instance.pendown()
        turtle_instance.write('La cadena es aceptada', False, align="left", font=("
            Lucida Console", 20))

        resultado += "\nLa cadena es aceptada"
    else:
        if cadenaMod == "":
            resultado += "(f,e," + cadenaPila + ")\n"
        else:
            resultado += "(f," + cadenaMod + "," + cadenaPila + ")\n"
        turtle_instance.clear()
        dibujarCuadroYFlechas(turtle_instance)  # Dibujar las flechas y el cuadro al
            final
        turtle_instance.penup()
        turtle_instance.setpos(-153, 200)
        turtle_instance.pendown()
```

```
            turtle_instance.write('La cadena no es aceptada', False, align="left", font
                =("Lucida Console", 20))
            resultado += "\nLa cadena no es aceptada"

    # Imprimir en consola el resultado
    print(resultado)

    with open("Resultado.txt", "w", encoding="utf-8") as f:
        f.write(resultado)
```

Menú de Opciones

La función presenta al usuario un menú para elegir entre dos opciones:

1. Ingresar manualmente una cadena para evaluar.
2. Generar automáticamente una cadena utilizando la función `generarCadena()`.

El usuario debe ingresar el número correspondiente a su opción. La entrada se guarda en la variable `opcion`, que se evalúa posteriormente.

Ingreso de la Cadena

- Si el usuario elige la opción 1, se solicita la entrada manual de una cadena mediante la función `input()`, con un mensaje que indica que la longitud máxima permitida es de 100,000 caracteres.
- Si el usuario elige la opción 2, se genera automáticamente una cadena utilizando la función `generarCadena()`. La cadena generada se muestra en pantalla utilizando `print()`.

La cadena seleccionada o generada se almacena en la variable `cadena`.

Evaluación de la Longitud de la Cadena

La función evalúa el valor de `contador` para decidir el flujo del programa:

- Si la longitud de la cadena es menor o igual a 100,000:
  - Si la longitud es menor o igual a 10, se inicia la animación del autómata con la función `animarAutomata(cadena)`. Antes de esto, se configura una ventana gráfica utilizando `turtle.Screen()`:
    - El color de fondo de la ventana se establece en negro con `screen.bgcolor("black")`.
    - Se ajusta el tamaño de la ventana a un ancho de 600 píxeles y una altura de 400 píxeles mediante `screen.setup(width=600, height=400)`.

```
def main():
    print("Seleccione una opción:")
    print("1. Ingresar la cadena manualmente")
    print("2. Generar una cadena automáticamente")

    opcion = int(input("Ingrese el número de su opción: "))

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

```
    # Contador manual
    contador = 0
    for _ in cadena:
        contador += 1

    if contador <= 100000:
        if contador <= 10:
            screen = turtle.Screen()
            screen.bgcolor("black")  # Cambiar el color de fondo a negro
            screen.setup(width=600, height=400)  # Hacer la ventana más pequeña
            animarAutomata(cadena=cadena)
            screen.mainloop()
        else:
            print("La cadena es demasiado larga para animar el autómata.")
    else:
        print("La cadena excede el límite máximo de caracteres (100,000).")

if __name__ == "__main__":
    main()
```

3.5. Ejecuciones del Programa

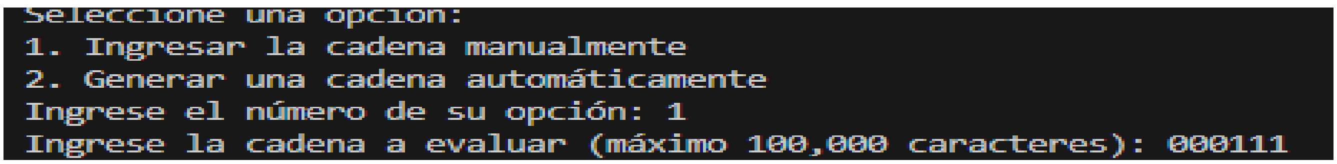


Figura 2: Generaciòn de Cadena Manual.

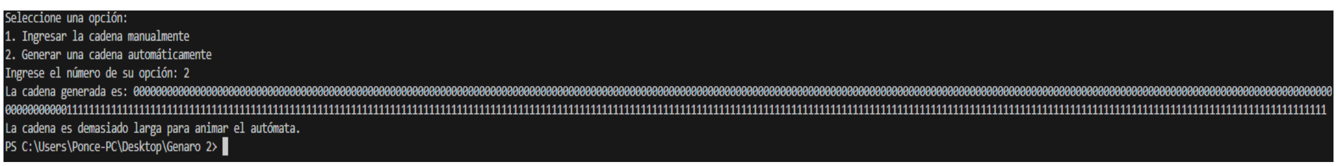


Figura 3: Generaciòn de Cadena Aleatoria.

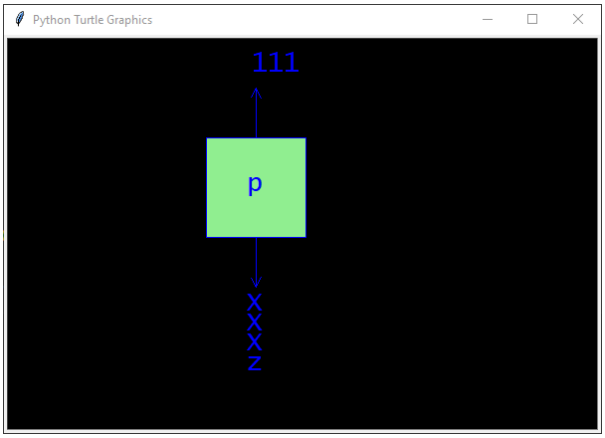


Figura 4: Salida del autòmata de pila y su animaciòn del autòmata de pila.

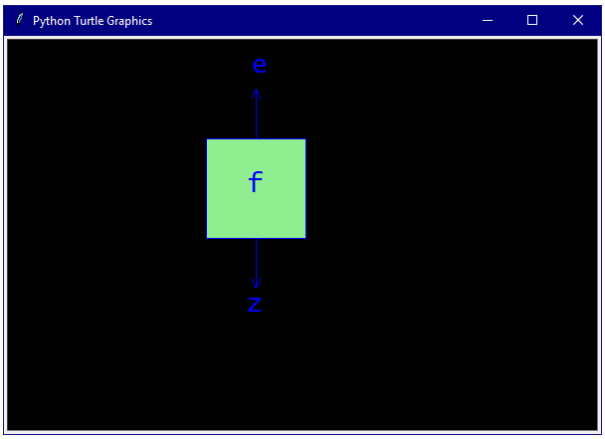


Figura 5: El Estado Final de la Pila.

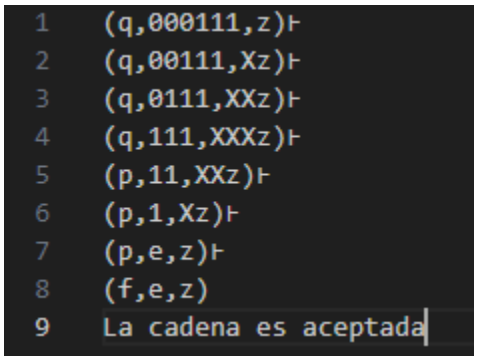


Figura 6: Salida del TXT.

3.6. Resultados

El programa analiza cadenas para verificar si pertenecen a un lenguaje definido por el autómata de pila.

Cadenas Aceptadas: Cuando la cadena tiene la forma 0^n1^n, el autómata la reconoce como válida. El programa registra esta aceptación en un archivo y lo representa en la animación mostrando que se alcanzó el estado final f con la pila completamente vacía.

Cadenas Rechazadas: Si la cadena no cumple con el formato 0^n1^n, el autómata la rechaza. El programa escribe este resultado en un archivo y lo ilustra en la animación indicando que se llegó a un estado de error o que la pila no quedó correctamente vacía.

4. Backus-Naur condicional IF

4.1. Objetivo Específico

El objetivo específico del programa es desarrollar un generador de derivaciones de una gramática Backus-Naur (BNF) que defina el condicional `IF` en el lenguaje de programación. El programa debe permitir derivar de manera automática la gramática hasta generar un número determinado de expresiones `IF` según lo especifique el usuario o la máquina, mostrando cada paso de las derivaciones en un archivo. Además, el programa debe generar un segundo archivo que contenga el pseudo-código correspondiente a la cadena de derivación generada. El límite de derivaciones está establecido en 1000.

4.2. Introducción

La gramática BNF es una forma formal de describir la sintaxis de un lenguaje de programación. En particular, el condicional `IF` es una de las estructuras de control más utilizadas, y su correcta derivación es fundamental para comprender cómo se pueden generar expresiones condicionales de manera automática.

El programa desarrollado tiene como objetivo permitir la generación de expresiones `IF` de acuerdo a la gramática especificada, permitiendo al usuario definir el número de derivaciones. El proceso de derivación se realiza de manera automática, y los resultados se almacenan en archivos.

4.3. Metodología

El programa implementa un generador de derivaciones basado en la gramática Backus-Naur (BNF) para el condicional `IF`. Comienza con el estado inicial `S` y realiza derivaciones aleatorias hasta alcanzar el número de pasos especificado, reemplazando los símbolos `S` y `A` por sus producciones correspondientes: $S \rightarrow iCtSA$ y $A \rightarrow ; eS \mid \epsilon$.

El usuario puede elegir entre dos modos:

- **Manual:** Especificando el número de pasos (máximo 1000).
- **Automático:** Generando un número aleatorio de pasos.

4.4. Desarrollo

Librerìas

Usadas:
random: En Python permite generar números aleatorios. Puedes utilizarla para obtener valores aleatorios dentro de un rango determinado, para elegir elementos de una secuencia de manera aleatoria, entre otras funcionalidades.

```
import random
```

Derivar la Secuencia

- $S \rightarrow (iCtSA)$
- $A \rightarrow (; eS)$
- $A \rightarrow \varepsilon$

En cada iteración:

1. Se identifica un símbolo no terminal (`S` o `A`).

2. Se selecciona al azar cuál reemplazar.

3. Se aplica la regla correspondiente, y la cadena derivada se almacena.

4. El proceso continúa hasta agotar los pasos o símbolos no terminales.

```
# Función principal para procesar la gramática
def procesar_gramatica(cadena, max_pasos):
    resultados = [f"Paso 1: ({cadena})"]  # Lista para almacenar las derivaciones
    contador_pasos = 1  # Contador del paso actual

    while max_pasos > 0:  # Iteramos mientras haya pasos restantes
        reemplazos_disponibles = []

        # Verificar qué símbolos pueden reemplazarse
        if 'S' in cadena:
            reemplazos_disponibles.append('S')
        if 'A' in cadena:
            reemplazos_disponibles.append('A')

        # Si no hay más símbolos, salimos del ciclo
        if not reemplazos_disponibles:
            break

        # Elegimos al azar el símbolo que vamos a reemplazar
        simbolo = random.choice(reemplazos_disponibles)

        if simbolo == 'A':
            # Decidir entre las dos reglas de producción de A
            if random.choice([0, 1]) == 0:
                cadena = cadena.replace('A', '(;eS)', 1)
                resultados.append(f"Paso {contador_pasos + 1}: A -> ;eS: ({cadena})")
            else:
                cadena = cadena.replace('A', '', 1)
                resultados.append(f"Paso {contador_pasos + 1}: A -> E: ({cadena})")
        elif simbolo == 'S':
            # Regla de producción de S
            cadena = cadena.replace('S', '(iCtSA)', 1)
            resultados.append(f"Paso {contador_pasos + 1}: S -> iCtSA: ({cadena})")

        max_pasos -= 1
        contador_pasos += 1

    return resultados
```

Pseudocódigo

1. Sustituir los símbolos de la gramática por constructos de pseudocódigo:

   - $i \rightarrow$ `if (`
   - $C \rightarrow$ `cond`
   - $t \rightarrow$ `) then {`
   - $;e \rightarrow$ `} else {`

2. Equilibrar las llaves abiertas y cerradas.

3. Mejorar el formato con saltos de línea y tabulación.

```
# Función para convertir la gramática en pseudocódigo
def generar_pseudocodigo(cadena_final):
    codigo = cadena_final.replace('i', 'if (') \
                         .replace('C', 'cond') \
                         .replace('t', ') then {') \
                         .replace(';e', '} else {')

    # Equilibrar llaves abiertas y cerradas
    llaves_abiertas = codigo.count('{')
    llaves_cerradas = codigo.count('}')
    codigo += '}' * (llaves_abiertas - llaves_cerradas)

    # Mejorar formato del pseudocódigo
    codigo = codigo.replace(')', ')\n') \
                   .replace('{', '\n{\n\t') \
                   .replace('}', '\n}\n')

    return codigo
```

Descripción del código principal

- **Definición del límite de derivaciones:** Se establece un valor máximo de derivaciones (`MAX_DERIVACIONES`) igual a 1000.
- **Selección del modo de operación:** Se permite al usuario elegir entre:
  - **Modo manual:** El usuario ingresa el número de pasos de derivación, limitado a 1000.
  - **Modo automático:** Se genera un número aleatorio de pasos entre 1 y 1000.
- **Derivación de secuencias:** Llama a la función `derivar_secuencia` con el estado inicial `'S'` y el número de pasos seleccionados. Los resultados de las derivaciones se almacenan en una lista llamada `historial`.
- **Almacenamiento del registro:** El historial de derivaciones se guarda en el archivo `derivaciones.txt`.
- **Conversión a pseudocódigo:** La última derivación generada se convierte a pseudocódigo utilizando la función `convertir_a_pseudocodigo`. El pseudocódigo resultante se guarda en el archivo `pseudocodigo.txt`.
- **Mensajes informativos:** Se notifica al usuario sobre la creación de los archivos `derivaciones.txt` y `pseudocodigo.txt`.

```
# Parámetro para limitar la cantidad máxima de derivaciones
MAX_PASOS = 1000

# Entrada del usuario para seleccionar el modo
try:
    modo = int(input("Selecciona el modo: (1: Manual, 2: Automático): "))
    if modo == 1:
        pasos = int(input(f"Ingrese el número de pasos (máximo {MAX_PASOS}): "))
        if pasos > MAX_PASOS:
            pasos = MAX_PASOS
    elif modo == 2:
        pasos = random.randint(1, MAX_PASOS)
    else:
        print("Modo inválido. Por favor selecciona 1 o 2.")
        exit()
```

```
except ValueError:
    print("Entrada inválida. Debes ingresar un número.")
    exit()

# Procesamiento de la gramática inicial
cadena_inicial = 'S'
derivaciones_generadas = procesar_gramatica(cadena_inicial, pasos)

# Guardar derivaciones en un archivo
with open('Derivaciones.txt', 'w', encoding='utf-8') as archivo:
    for linea in derivaciones_generadas:
        archivo.write(linea + '\n')

# Generar pseudocódigo a partir de la última derivación
if derivaciones_generadas:
    ultima_cadena = derivaciones_generadas[-1].split(": ")[-1]
    pseudocodigo_final = generar_pseudocodigo(ultima_cadena)

    # Guardar el pseudocódigo en un archivo
    with open('Pseudocodigo.txt', 'w', encoding='utf-8') as archivo_pseudocodigo:
        archivo_pseudocodigo.write(pseudocodigo_final)

# Mensajes al usuario
print("Derivaciones guardadas en 'Derivaciones.txt'")
print("Pseudocódigo guardado en 'Pseudocodigo.txt'")
```

4.5. Ejecuciones del Programa

```
Selecciona el modo: (1: Manual, 2: Automático): 1
Ingrese el número de pasos (máximo 1000): 10
Derivaciones guardadas en 'Derivaciones.txt'
Pseudocódigo guardado en 'Pseudocodigo.txt'
```

Figura 7: Ejecución del Programa para Manual.

```
Selecciona el modo: (1: Manual, 2: Automático): 2
Derivaciones guardadas en 'Derivaciones.txt'
Pseudocódigo guardado en 'Pseudocodigo.txt'
```

Figura 8: Ejecución del Programa para Automàtico.

```
Paso 1: (S)
Paso 2: S -> iCtSA: ((iCtSA))
Paso 3: A -> ε: ((iCtS))
Paso 4: S -> iCtSA: ((iCt(iCtSA)))
Paso 5: S -> iCtSA: ((iCt(iCt(iCtSA)A)))
Paso 6: S -> iCtSA: ((iCt(iCt(iCt(iCtSA)A)A)))
Paso 7: A -> ;eS: ((iCt(iCt(iCt(iCtS(;eS))A)A)))
Paso 8: S -> iCtSA: ((iCt(iCt(iCt(iCt(iCtSA)(;eS))A)A)))
Paso 9: S -> iCtSA: ((iCt(iCt(iCt(iCt(iCtSA)A)(;eS))A)A)))
Paso 10: A -> ;eS: ((iCt(iCt(iCt(iCt(iCt(iCtS(;eS))A)(;eS))A)A)))
Paso 11: A -> ε: ((iCt(iCt(iCt(iCt(iCt(iCtS(;eS)))(;eS))A)A)))
```

Figura 9: Derivaciones.

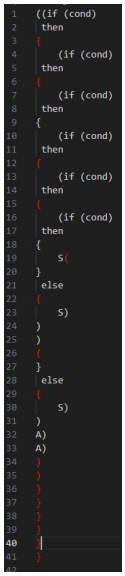


Figura 10: Pseudocódigo.

5. Máquina de Turing

5.1. Objetivo Específico

El objetivo del programa es implementar una Máquina de Turing que reconozca el lenguaje $\{0^n1^n \mid n \geq 1\}$, permitiendo ingresar o generar cadenas de hasta 1000 caracteres, simulando cada paso de la computación y registrando las descripciones instantáneas en un archivo de texto. El programa debe animar la ejecución de la máquina para cadenas de hasta 10 caracteres, mostrando visualmente los cambios en la cinta, el estado y la posición del cabezal, y determinar si la cadena es aceptada o rechazada según las reglas del lenguaje.

5.2. Introducciòn

La Máquina de Turing es un modelo matemático fundamental en la teoría de la computación, utilizado para describir y estudiar la computabilidad de los lenguajes formales, que consiste en cadenas de ceros seguidas por el mismo número de unos. Este lenguaje es conocido por su propiedad de exigir una cantidad igual de ceros y unos en una secuencia específica y la simulación de la máquina, se observarán las transiciones entre los diferentes estados y los cambios en la cinta de la máquina, con el fin de determinar si una cadena dada pertenece o no al lenguaje descrito.

5.3. Metodología

El programa simula una Máquina de Turing utilizando una lista enlazada que representa la cinta. Se ofrece dos opciones para la entrada:

- El usuario ingresa una cadena manualmente con caracteres '0' y '1'.
- O se genera una cadena aleatoria de longitud entre 1 y 1000.

La máquina de Turing sigue las siguientes reglas de transición:

- Estado 0: Reemplaza el primer '0' por 'X' y se mueve a la derecha.
- Estado 1: Reemplaza el primer '1' por 'Y' y se mueve a la izquierda.
- Estado 2: Retrocede buscando 'X' o 'Y', luego se mueve a la derecha.
- Estado 3: Si encuentra un espacio en blanco ('B'), acepta la cadena.

El proceso es visualizado y mostrando la cinta, el cabezal y el estado actual en cada paso y el programa indica si la cadena es aceptada o no, y guarda el resultado en un archivo de texto. La interfaz permite al usuario elegir entre ingresar una cadena o generar una aleatoria.

5.4. Desarrollo

Librerìas

Usadas:
random: En Python permite generar números aleatorios. Puedes utilizarla para obtener valores aleatorios dentro de un rango determinado, para elegir elementos de una secuencia de manera aleatoria, entre otras funcionalidades.
turtle: Es una librería en Python que proporciona una forma divertida de aprender a programar gráficos. Básicamente, simula un "tortuga"que se mueve en la pantalla y dibuja líneas.
time en Python Python importa todas las funciones, constantes y clases del módulo estándar time.

```
from turtle import *
from time import *
import random
```

Nodos

Es una estructura básica para representar un nodo en una lista doblemente enlazada. Su constructor recibe dos parámetros: `valor`, que almacena el valor del nodo, y `siguiente`, que apunta al nodo anterior en la lista. Los atributos `derecha` e `izquierda` se inicializan como `None`, representando la ausencia de nodos adyacentes hasta que se asignen explícitamente.

```
class Nodo:
    def __init__(self, valor, siguiente):
        self.valor = valor
        self.derecha = None
        self.izquierda = siguiente
```

Lista

Representa una lista doblemente enlazada. Su constructor inicializa el atributo `inicio` como `None`, lo que significa que la lista está vacía al principio.

El método `agregar` permite añadir un nuevo nodo al final de la lista. Si la lista está vacía, se crea el primer nodo con el valor proporcionado. En caso contrario, el método recorre la lista hasta encontrar el último nodo, y luego agrega un nuevo nodo con el valor dado. El nuevo nodo apunta hacia el nodo previamente final, y el último nodo de la lista se actualiza para que su `derecha` apunte al nuevo nodo.

```
class Lista:
    def __init__(self):
        self.inicio = None

    def agregar(self, valor):
        if self.inicio is None:
            self.inicio = Nodo(valor, None)
            return

        nodo_actual = self.inicio

        while nodo_actual.derecha is not None:
            nodo_actual = nodo_actual.derecha

        nuevo_nodo = Nodo(valor, None)
        nuevo_nodo.izquierda = nodo_actual
        nodo_actual.derecha = nuevo_nodo
```

Ejecutar Maquina

Parámetros de entrada

La función recibe dos parámetros:

- `cadena`: La secuencia de caracteres que se va a procesar. Esta cadena debe contener caracteres que la máquina pueda interpretar (por ejemplo, '0', '1', 'B', etc.).
- `longitud_max` (opcional): El valor máximo de la longitud de la cadena. El valor predeterminado es 10.

Validación de la longitud de la cadena

El primer paso en la ejecución de la máquina es verificar si la longitud de la cadena supera el valor máximo permitido (`longitud_max`). Si la longitud es mayor, la función termina y se guarda un mensaje en un archivo de texto, indicando que la cadena no es aceptada debido a su longitud.

Creación de la lista de nodos

Una vez validada la longitud de la cadena, esta se convierte en una lista de nodos. Cada nodo representa un carácter de la cadena y tiene enlaces tanto al nodo siguiente (`derecha`) como al nodo anterior (`izquierda`). Se utiliza la clase `Lista` y la clase `Nodo` para manejar la cadena de manera eficiente.

Representación gráfica con `turtle`

Para visualizar el procesamiento de la máquina de Turing, se emplea la biblioteca `turtle`. A continuación, se configura la ventana gráfica con un fondo negro y se preparan los elementos gráficos. En cada iteración, la máquina de Turing es representada visualmente en la pantalla mediante cuadros (celdas) y el estado actual es mostrado en el gráfico.

Ejecución de la máquina de Turing

- `estado 0`: Si lee un '0', cambia el valor a 'X' y se mueve a la derecha. Si lee un 'Y', pasa al `estado 3`.
- `estado 1`: Si lee un '0', continúa moviéndose a la derecha. Si lee un '1', cambia el valor a 'Y' y se mueve a la izquierda. Si lee un 'Y', continúa moviéndose a la derecha.
- `estado 2`: Si lee un '0', continúa moviéndose a la izquierda. Si lee un 'X', cambia el valor a 'X' y se mueve a la derecha. Si lee un 'Y', continúa moviéndose a la izquierda.
- `estado 3`: Si lee un 'Y', se mueve a la derecha. Si lee un 'B', alcanza el `estado 4`, indicando que la cadena es aceptada.

Cada vez que se cambia de estado, la pantalla gráfica se actualiza, mostrando el nuevo estado y el contenido de la cinta. Además, se visualizan las reglas que se están aplicando en ese momento.

Estado de aceptación

Si la máquina llega al `estado 4`, la cadena es aceptada. El mensaje correspondiente se muestra tanto en la pantalla como en un archivo de texto llamado `Maquina_Turing.txt`. Si la máquina no alcanza el estado de aceptación, se muestra un mensaje de rechazo.

```
def ejecutar_maquina(cadena, longitud_max=10):
    contador = 0
    for caracter in cadena:
        contador += 1
    if contador > longitud_max:
        print(f"Error: La cadena excede el máximo permitido de {longitud_max} caracteres
            .")
        mensaje = f"La cadena '{cadena}' no es aceptada debido a su longitud."
        with open("Maquina_Turing.txt.txt", "w", encoding="utf-8") as archivo:
            archivo.write(mensaje)
        return

    lista = Lista()
    for caracter in cadena:
```

```
        lista.agregar(caracter)

    nodo_actual = lista.inicio
    mensaje = ""

    # Configuración de la ventana gráfica
    setup(width=1000, height=600)
    speed(0)
    bgcolor("black")

    # Dibujo de las celdas sin la tabla de transiciones
    penup()
    setpos(-300, 100)
    pendown()
    color("white")

    # Usamos un contador en lugar de len(cadena)
    for _ in range(contador + 1):
        fillcolor("darkblue")
        begin_fill()
        for _ in range(4):
            forward(50)
            right(90)
        forward(50)
        end_fill()

    # Crear el objeto turtle solo si la ventana está activa
    turtle = Turtle()
    turtle.hideturtle()

    estado = 0
    nodo_final = None
    colores = ["red", "green", "blue", "purple", "orange"]

    while estado != 4:
        if nodo_actual is None:
            if nodo_final is not None:
                nodo_final.derecha = Nodo("B", siguiente=nodo_final)

                nodo_actual = nodo_final.derecha
            else:
                nodo_actual = Nodo("B", siguiente=None)
                nodo_final = nodo_actual

        nodo_temp = nodo_actual

        idt1 = "q" + str(estado) + "_"
        idt2 = ""
        idt_sin = ""
        id_sin = ""

        while nodo_temp is not None:
            if nodo_temp is not None:
                idt1 += nodo_temp.valor
                idt_sin += nodo_temp.valor
            nodo_temp = nodo_temp.derecha
        nodo_temp = nodo_actual.izquierda
        while nodo_temp is not None:
            if nodo_temp is not None:
                idt2 += nodo_temp.valor
            nodo_temp = nodo_temp.izquierda
```

```
        mensaje += idt2[::-1] + idt1 + "\n"
        id_sin = idt2[::-1] + idt_sin

        turtle.clear()
        pos_x = -280
        for i, char in enumerate(id_sin):
            turtle.penup()
            turtle.setpos(pos_x, 67)
            turtle.pendown()
            turtle.color(colores[i % len(colores)])
            turtle.write(char, False, align="left", font=("Arial", 12, "normal"))  #
                Cambié la fuente aquí
            pos_x += 50

        pos_x2 = -280 + (50 * len(idt2))
        turtle.penup()
        turtle.setpos(pos_x2, 167)
        turtle.pendown()
        turtle.color("cyan")
        turtle.write("q" + str(estado), False, align="left", font=("Courier", 12, "bold"
            ))  # Cambié la fuente aquí
        turtle.penup()
        turtle.setpos(pos_x2, 160)
        turtle.pendown()
        turtle.setpos(pos_x2, 110)
        turtle.setpos(pos_x2 - 5, 115)
        turtle.setpos(pos_x2, 110)
        turtle.setpos(pos_x2 + 5, 115)
        turtle.penup()
        turtle.setpos(pos_x2, -10)
        sleep(0.5)

        # Mostrar reglas lentamente
        turtle.clear()
        turtle.penup()
        turtle.setpos(-280, -20)
        turtle.pendown()
        turtle.color("yellow")
        turtle.write(f"Reglas en uso: Estado: q{estado}", align="left", font=("Times New
            Roman", 12, "italic"))  # Cambié la fuente aquí
        sleep(0.5)

        match estado:
            case 0:
                if nodo_actual.valor == '0':
                    estado = 1
                    nodo_actual.valor = "X"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                elif nodo_actual.valor == 'Y':
                    estado = 3
                    nodo_actual.valor = "Y"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                else:
                    break
            case 1:
                if nodo_actual.valor == '0':
```

```
                    estado = 1
                    nodo_actual.valor = "0"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                elif nodo_actual.valor == '1':
                    estado = 2
                    nodo_actual.valor = "Y"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.izquierda
                elif nodo_actual.valor == 'Y':
                    estado = 1
                    nodo_actual.valor = "Y"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                else:
                    break
            case 2:
                if nodo_actual.valor == '0':
                    estado = 2
                    nodo_actual.valor = "0"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.izquierda
                elif nodo_actual.valor == 'X':
                    estado = 0
                    nodo_actual.valor = "X"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                elif nodo_actual.valor == 'Y':
                    estado = 2
                    nodo_actual.valor = "Y"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.izquierda
                else:
                    break
            case 3:
                if nodo_actual.valor == 'Y':
                    estado = 3
                    nodo_actual.valor = "Y"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                elif nodo_actual.valor == 'B':
                    estado = 4
                    nodo_actual.valor = "B"
                    if nodo_actual.derecha is None:
                        nodo_final = nodo_actual
                    nodo_actual = nodo_actual.derecha
                else:
                    break

    if estado == 4:
        mensaje += "\n La cadena es aceptada"
        print("La cadena es aceptada")
        turtle.clear()
        pos_x = -280
```

```
        for i in id_sin:
            turtle.penup()
            turtle.setpos(pos_x, 67)
            turtle.pendown()
            turtle.write(i, False, align="left", font=("Arial", 10, "normal"))  # Cambié
                la fuente aquí
            pos_x += 50
        pos_x2 = -280 + (50 * len(idt2))
        turtle.penup()
        turtle.setpos(pos_x2, 167)
        turtle.write("q" + str(estado), False, align="left", font=("Courier", 10, "bold"
            ))  # Cambié la fuente aquí
        turtle.penup()
        turtle.setpos(-250, 200)
        turtle.pendown()
        turtle.color("green")
        turtle.write("La cadena es aceptada", align="left", font=("Times New Roman", 12,
             "italic"))  # Cambié la fuente aquí
    else:
        mensaje += "\n La cadena no es aceptada"
        print("La cadena no es aceptada")
        turtle.penup()
        turtle.setpos(-250, 200)
        turtle.pendown()
        turtle.color("red")
        turtle.write("La cadena no es aceptada", align="left", font=("Times New Roman",
            12, "italic"))  # Cambié la fuente aquí

    print(mensaje)
    with open("Maquina_Turing.txt", "w", encoding="utf-8") as archivo:
        archivo.write(mensaje)
        archivo.close()
```

Despliegue del menú

Se imprimen en pantalla las opciones disponibles para el usuario:

- **Opción 1:** Permite al usuario ingresar una cadena manualmente.
- **Opción 2:** Genera una cadena aleatoria de forma automática.

Selección de opción

El programa solicita al usuario que elija una opción:

- Si el usuario selecciona la opción `"1"`, se le solicita ingresar una cadena manualmente mediante el teclado. La cadena ingresada se pasa como argumento a la función `ejecutar_maquina`.
- Si el usuario selecciona la opción `"2"`, el programa genera una cadena de longitud aleatoria (entre 1 y 1000) compuesta únicamente por los caracteres '0' y '1'. Esta cadena también se pasa como argumento a la función `ejecutar_maquina`. Antes de ejecutarla, se imprime la cadena generada en la consola.
- Si se selecciona cualquier otra opción, el programa imprime un mensaje indicando que la opción no es válida.

```python
def mostrar_menu():
    print("Seleccione una opción:")
    print("1. Ingresar cadena manualmente.")
    print("2. Generar cadena automáticamente.")

    opcion = input("Opción: ")
    if opcion == "1":
        cadena = input("Ingrese la cadena: ")
        ejecutar_maquina(cadena=cadena)
    elif opcion == "2":
        longitud = random.randint(1, 1000)
        cadena = ''.join(random.choice("01") for _ in range(longitud))
        print(f"Cadena generada automáticamente: {cadena}")
        ejecutar_maquina(cadena=cadena)
    else:
        print("Opción no válida.")

mostrar_menu()
```

5.5. Ejecuciones del Programa

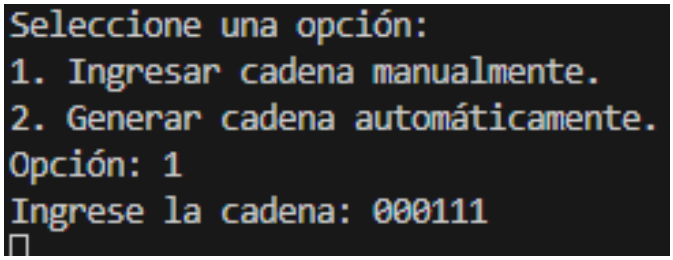


Figura 11: Ingresa la Cadena Manualmente

Figura 12: Ingresa la Cadena Automaticamente

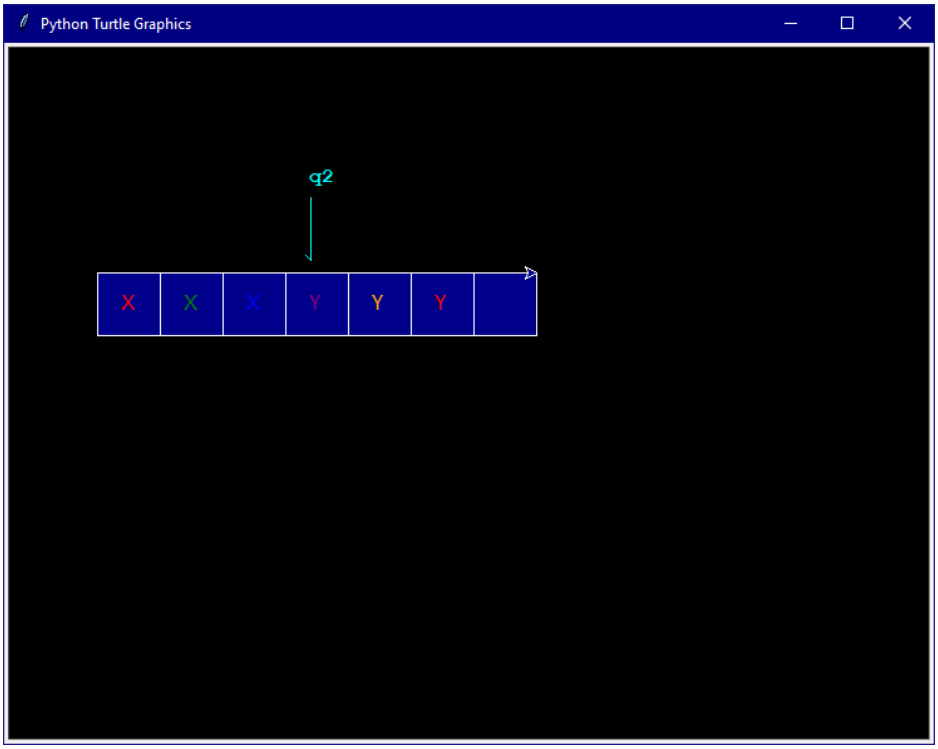


Figura 13: Transcisiòn de la Maquina Estados q2

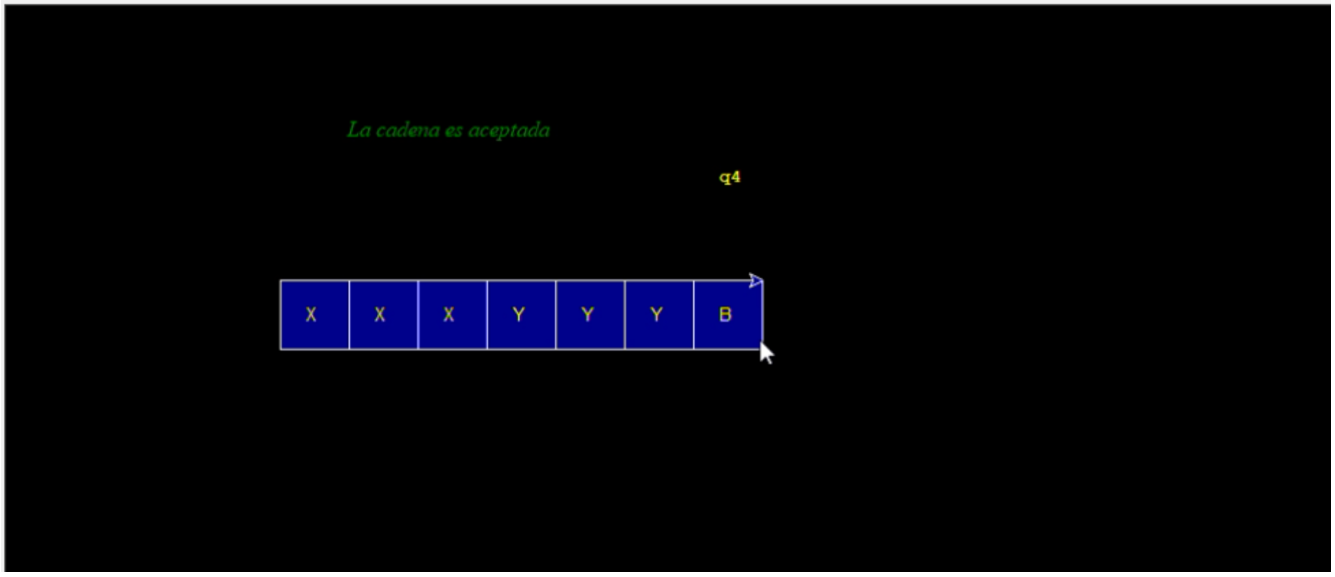


Figura 14: Animaciòn Final

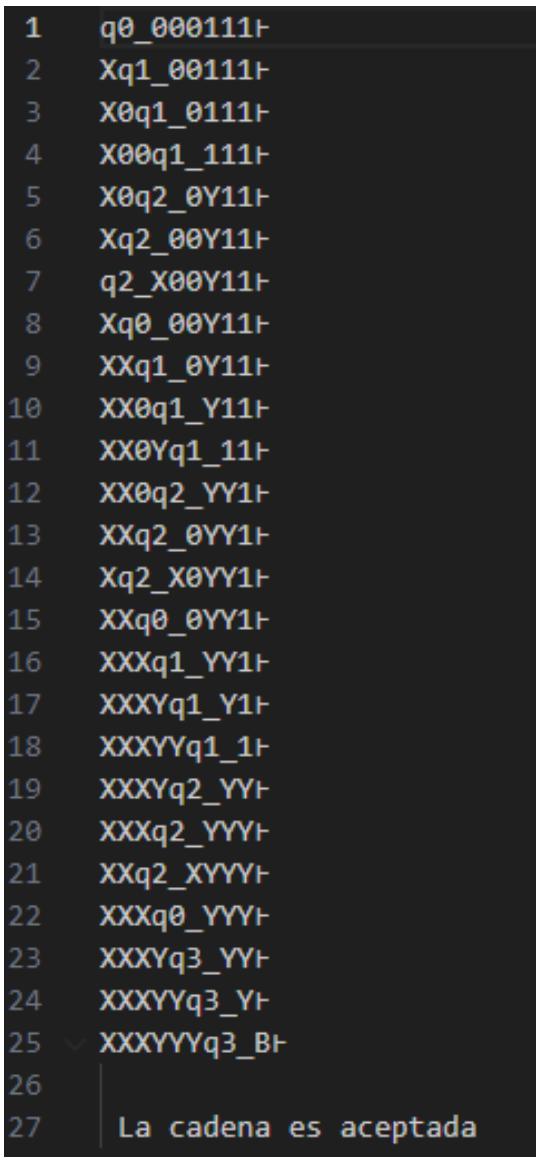


Figura 15: Archivo .TXT con las Transiciones

6. Conclusiones

Estos programas representan una aplicación integral de conceptos fundamentales de la teoría de autómatas, lenguajes formales y gramáticas, combinando tanto aspectos teóricos como prácticos. Cada implementación aborda un modelo computacional diferente, desde autómatas finitos y autómatas de pila hasta gramáticas y máquinas de Turing, demostrando cómo estos conceptos pueden ser utilizados para resolver problemas de reconocimiento de patrones, procesamiento de lenguajes y simulación de cómputo.

6.1. Aprendizaje Obtenido

Durante la realización de estos programas se adquirieron conocimientos fundamentales sobre los modelos computacionales y su aplicación práctica. Se profundizó en la teoría de autómatas, desde los autómatas finitos y autómatas de pila hasta las máquinas de Turing, comprendiendo sus diferencias y cómo son utilizados en el trabajo con lenguajes regulares, libres de contexto y recursivamente enumerables. También se reforzaron las habilidades para la conversión de modelos como la conversión de un NFA a un DFA, mediante la estructuración y organización de las habilidades como el uso de estados conjuntos y tablas de transición.

6.2. Limitaciones y mejoras

Los programas desarrollados presentan ciertas limitaciones que podrían ser abordadas en futuras implementaciones. Una de las principales limitaciones es la restricción de tamaño al procesar cadenas largas o manejar grandes volúmenes de datos, lo cual puede resultar en un uso ineficiente de recursos y las animaciones están limitadas a cadenas cortas, lo que restringe su utilidad para visualizar procesos más complejos, la complejidad técnica de ciertas transformaciones, como la conversión de NFA a DFA o las derivaciones gramaticales, puede dificultar su comprensión para usuarios con poca experiencia. También se identificó una dependencia de elementos externos, como bibliotecas y conexión a internet, que podrían generar fallos en determinadas condiciones.

7. Referencias

Referencias

[1] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation* (3rd ed.). Pearson. Fundamental text for understanding automata, Turing machines, and formal languages.

[2] Menezes, M. (2008). *Formal Languages and Automata Theory*. Elsevier. Covers the application of formal languages and automata in computational problem-solving.

[3] Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1), 230–265. Seminal work introducing the concept of Turing machines and their significance in computation.