



## **Instituto Politecnico Nacional Escuela Superior de Computo**

Materia: Teoría de la Computación  
Grupo: 5BM1

Profesor: Genaro Juarez Martínez  
Periodo: 2024/02

### **Bloque de programas 01:**

**Generador de potencias: de un alfabeto binario, de rutas ganadoras en un tablero de ajedrez; y buscador de palabras.**

Realizado por:  
**Carrillo Barreiro José Emiliano**



**Escuela Superior de Computo  
Fecha: 7 de abril de 2024**

# Índice

<b>1 Resumen.</b>	<b>4</b>
<b>2 Introducción.</b>	<b>4</b>
<b>3 Marco Teorico.</b>	<b>5</b>
3.1 Teoría de Automátas . . . . .	5
3.2 Alfabetos . . . . .	5
3.3 Cadena de caracteres . . . . .	5
3.4 Cadena vacio. . . . .	6
3.5 Longitud de cadenas. . . . .	6
3.6 Potencias de un alfabeto. . . . .	6
3.7 Concatenacion de cadenas. . . . .	6
3.8 Lenguajes. . . . .	6
3.9 Autómatas Finitos No Deterministas (AFN) . . . . .	6
3.9.1 Aplicaciones de los AFN . . . . .	6
3.10 Equivalencia de autómatas finitos deterministas y no deterministas . . . . .	6
3.11 AFD para reconocer un conjunto de palabras clave . . . . .	7
<b>4 Desarrollo</b>	<b>8</b>
4.1 Especificaciones . . . . .	8
4.2 Lenguajes . . . . .	8
4.2.1 C++ . . . . .	9
4.2.2 Matlab . . . . .	9
4.3 Librerias . . . . .	10
<b>5 Implementación</b>	<b>11</b>
5.1 Practica 01: Universo de Cadenas Binarias. . . . .	11
5.1.1 Libreria . . . . .	11
5.1.2 Main . . . . .	11
5.1.3 Menu . . . . .	12
5.1.4 Solicitudes . . . . .	14
5.1.5 Logica del problema . . . . .	15
5.1.6 Llamada a graficar . . . . .	16
5.1.7 Graficar . . . . .	16
5.2 Practica03: Tablero de Ajedrez . . . . .	18
5.2.1 <i>PRACTICA03.h</i> . . . . .	18
5.2.2 <i>mainP03.cpp</i> . . . . .	19
5.2.3 <i>menuP03.cpp</i> . . . . .	19
5.2.4 <i>solicitudesP03.cpp</i> . . . . .	21
5.2.5 <i>automataP03.cpp</i> . . . . .	24
5.2.6 <i>jugadasP03.cpp</i> . . . . .	25
5.2.7 <i>animarP03.cpp</i> . . . . .	26
5.2.8 <i>leercsv.m</i> . . . . .	27
5.2.9 <i>GraficoAutomata.m</i> . . . . .	28
5.2.10 <i>animacionP03.m</i> . . . . .	29
5.3 Practica04: Buscador de palabras. . . . .	31
5.3.1 <i>Conversion de NFA a DFA</i> . . . . .	31
5.3.2 <i>PRACTICA04.h</i> . . . . .	33
5.3.3 <i>mainP04.cpp</i> . . . . .	34
5.3.4 <i>menuP04.cpp</i> . . . . .	34
5.3.5 <i>solicitudesP04.cpp</i> . . . . .	36
5.3.6 <i>automataP04.cpp</i> . . . . .	39
5.3.7 <i>palabrasP04.cpp</i> . . . . .	40
5.3.8 <i>matlabP04.cpp</i> . . . . .	41
5.3.9 <i>leerPagWebs.m</i> . . . . .	41
5.3.10 <i>GraficoAutomata.m</i> . . . . .	42
5.4 Compendio . . . . .	45

<b>6 Resultados</b>	<b>46</b>
6.1 Resultados: Universo de cadenas binarias . . . . .	46
6.2 Resultados: Tablero de ajedrez . . . . .	47
6.3 Resultados: Buscador de palabras . . . . .	50
<b>7 Conclusiones</b>	<b>52</b>

# Índice de Listados de C++

1	PRACTICA01.h	11
2	mainP01.cpp — main	12
3	menuP01.cpp — menu	12
4	menuP01.cpp — ingresarOpcion	13
5	menuP01.cpp — ejecucion	13
6	menuP01.cpp — medirTiempo	14
7	solicitudesP01.cpp — ingresarPotencia	14
8	solicitudesP01.cpp — randomPotencia	14
9	solicitudesP01.cpp — ingresarCaracteres	14
10	solicitudesP01.cpp — opcionGraficar	15
11	P01.h — generarCombinaciones	15
12	P01.h — graficar	16
13	graficoMatlab.m — Script	16
14	PRACTICA03.h	18
15	mainP03.cpp	19
16	menuP03.cpp — menuP03	19
17	menuP03.cpp — ingresarOpcionP03	20
18	menuP03.cpp — ejecucionP03	21
19	menuP03.cpp — medirTiempoP03	21
20	solicitudesP03.cpp — solicitarCadenaP03	21
21	solicitudesP03.cpp — GenerarCadenaP03	22
22	solicitudesP03.cpp — guardarJugadasEnArchivo	23
23	solicitudesP03.cpp — opcionAnimarP03	24
24	automataP03.cpp — crearCasilla	24
25	automataP03.cpp — crearTablero	25
26	jugadasP03.cpp — encontrarJugadas	26
27	animarP03.cpp — animarP03	27
28	PRACTICA04.h	33
29	mainP04.cpp	34
30	menuP04.cpp — menuP04	34
31	menuP04.cpp — ingresarOpcionP04	35
32	menuP04.cpp — ejecucionP04	35
33	menuP04.cpp — medirTiempoP04	36
34	solicitudesP04.cpp — solicitarCadenaP04	36
35	solicitudesP04.cpp — leerCSV	37
36	solicitudesP04.cpp — guardarURL	38
37	solicitudesP04.cpp — leerArchivo	38
38	solicitudesP04.cpp — guardarPREnArchivo	38
39	solicitudesP04.cpp — opcionAnimarP04	39
40	automataP04.cpp — crearEstadoP04	39
41	automataP04.cpp — crearAutomataP04	40
42	palabrasP04 — encontrarPalabrasReservadas	40
43	matlabP04.cpp — EjecutarMatlabP04	41
44	Compendio01.cpp — main	45

# Índice de Listados de MATLAB

1	Código en MATLAB	16
2	leercsv	27
3	GraficoAutomata	28
4	animacionP03	30
5	leerPagWebs	42
6	GraficoAutomataP04	42

## **Índice de figuras**

1	Ejecucion de la Practica 01 en terminal. . . . .	46
2	Grafico resultante con potencia = 28 . . . . .	47
3	Ejecucion de la Practica 03 en terminal. . . . .	47
4	Automata Resultante P04 . . . . .	48
5	Tablero de ajedrez . . . . .	48
6	Ejecucion de la Practica 04 en terminal. . . . .	51
7	Automata Resultante P04 . . . . .	51

## **Índice de cuadros**

1	Tabla de estados del automata NFA. . . . .	32
2	Tabla de Conversion NFA a DFA . . . . .	32
3	Tabla de Estados del DFA . . . . .	50

# 1 Resumen.

El presente informe detalla el desarrollo de tres programas en el contexto de la teoría de autómatas y la programación interactiva. Cada programa aborda distintos aspectos y funcionalidades, con el objetivo común de ejecutarse de manera automática y permitir la interacción manual según las necesidades del usuario.

El primer programa, *Universo de Cadenas Binarias*, se enfoca en la generación y manipulación de cadenas binarias de longitud variable. Permite al usuario o al programa determinar la longitud de las cadenas, generando la salida en formato de conjunto y guardándola en un archivo de texto. Además, se realiza un análisis detallado para una longitud específica, incluyendo cálculos y gráficos.

El segundo programa, *Tablero de Ajedrez*, simula el movimiento de piezas en un tablero de  $4 \times 4$ , con la posibilidad de movimientos ortogonales y diagonales. Implementa un juego para dos jugadores, generando rutas de movimiento y detectando posibles movimientos ganadores. Se ofrece la opción de juego automático o manual, con la posibilidad de introducir o generar aleatoriamente la cadena de movimientos.

El tercer programa, *Buscador de Palabras*, desarrolla un autómata que reconoce un conjunto específico de palabras. Se diseña y transforma un NFA a DFA, permitiendo la lectura de un archivo de texto o página web para identificar las palabras reservadas y contar su ocurrencia. Se genera un archivo que registra la evaluación del autómata por cada carácter leído, y se grafica el DFA resultante.

Cada programa se acompaña de su respectivo código fuente en LaTeX, proporcionando una visión detallada de la implementación. El informe final se presenta en un único PDF que recopila todos los detalles de los programas, sus funcionalidades y los resultados obtenidos durante su ejecución.

# 2 Introducción.

En el presente informe se detallan las actividades realizadas en el marco de un proyecto que engloba tres programas distintos, cada uno abordando diferentes aspectos de la teoría de autómatas y la programación de sistemas interactivos. Estos programas han sido desarrollados con el objetivo de ejecutarse de manera automática, permitiendo al usuario interactuar con ellos de forma manual según sea necesario. A continuación, se presenta un resumen de cada programa junto con sus características principales:

## Programa 1: Universo de Cadenas Binarias

En este programa se aborda la generación y manipulación del universo de cadenas binarias de longitud variable, representadas por el conjunto  $\Sigma^n$ . Se permite al usuario o al programa determinar el valor de  $n$ , con un rango válido entre 0 y 1000. El programa puede ejecutarse tanto en modo automático como manual, generando la salida en formato de conjunto y guardándola en un archivo de texto. Además, se realiza un análisis específico para  $n = 28$ , incluyendo cálculos y gráficos del número de unos en cada cadena, tanto lineal como con logaritmo base 10.

## Programa 2: Tablero de Ajedrez

Este programa simula el movimiento de piezas en un tablero de ajedrez de  $4 \times 4$ , permitiendo movimientos ortogonales y diagonales. Se implementa un juego para dos jugadores con reglas específicas, generando las rutas de movimiento y detectando posibles movimientos ganadores. El programa puede funcionar en modo automático o manual, con la opción de introducir la cadena de movimientos o generarla aleatoriamente. Se grafica el tablero y se muestra la red (NFA) generada por los movimientos de ambos jugadores.

## Programa 3: Buscador de Palabras

En este programa se desarrolla un autómata que reconoce un conjunto de palabras específicas. Se diseña y transforma un NFA a DFA, mostrando todo el proceso en detalle. El programa lee un archivo de texto o una página web, identifica las palabras reservadas utilizando el DFA y cuenta su ocurrencia, indicando su posición en el archivo. Se genera un archivo que registra la evaluación del autómata por cada carácter leído, mostrando el cambio de estado. Además, se grafica el DFA generado.

Cada programa está acompañado por su respectivo código fuente en LaTeX, permitiendo una revisión detallada de la implementación. El objetivo final es generar un único programa y PDF que recopile todos los detalles de los programas y sus resultados, el cual será entregado a través de la plataforma designada.

## 3 Marco Teorico.

### 3.1. Teoría de Automátas.

La teoría de autómatas es un campo fundamental en la ciencia de la computación que se ocupa del estudio de dispositivos abstractos de cálculo. Estos dispositivos, conocidos como *máquinas*, son modelos matemáticos que nos permiten entender y analizar el comportamiento de sistemas computacionales. Desde sus primeros pasos en la década de 1930 con los trabajos de Alan Turing, la teoría de autómatas ha evolucionado y se ha ramificado en diversos aspectos que abarcan desde autómatas finitos simples hasta conceptos más complejos como gramáticas formales y problemas computacionales.

El punto de partida de la teoría de autómatas se encuentra en los estudios de Turing sobre las *máquinas de Turing*, dispositivos abstractos capaces de realizar cualquier cálculo computacional. Turing propuso estas máquinas como un modelo universal de computación, estableciendo así los fundamentos teóricos de lo que hoy entendemos como computación. A partir de este trabajo pionero, otros investigadores comenzaron a explorar variantes más simples de las máquinas de Turing, dando lugar a los autómatas finitos.

Los autómatas finitos son modelos computacionales simples que representan sistemas con un número limitado de estados y una capacidad limitada de procesamiento. Estos dispositivos se utilizan para modelar sistemas de control, reconocer patrones en cadenas de símbolos y resolver problemas de decisión. Además, los autómatas finitos están estrechamente relacionados con las gramáticas formales, ya que ambos se utilizan para describir y generar lenguajes formales.

La teoría de autómatas también abarca el estudio de problemas computacionales y la clasificación de su complejidad. Investigadores como Stephen Cook han contribuido significativamente al campo al desarrollar técnicas para clasificar los problemas en función de su dificultad computacional. Esta clasificación ha llevado a la identificación de problemas que pueden resolverse eficientemente y problemas que son inherentemente difíciles de resolver.[?]

### 3.2. Alfabetos.

Un alfabeto, representado convencionalmente por el símbolo  $\Sigma$ , es un conjunto finito y no vacío de símbolos. Estos símbolos pueden ser números, letras, caracteres especiales, o cualquier otro tipo de elemento que se utilice para formar cadenas de caracteres. Algunos ejemplos comunes de alfabetos incluyen:

1.  $\Sigma = \{0, 1\}$ : el alfabeto binario, utilizado en sistemas informáticos para representar datos de manera binaria.
2.  $\Sigma = \{a, b, \dots, z\}$ : el conjunto de todas las letras minúsculas del alfabeto latino.
3. El conjunto de todos los caracteres ASCII o el conjunto de todos los caracteres AS-CII imprimibles, utilizado en programación y comunicaciones para representar texto y caracteres especiales.[?]

### 3.3. Cadena de caracteres.

Una cadena de caracteres, también conocida como palabra en algunos contextos, es una secuencia finita de símbolos seleccionados de un alfabeto específico. Por ejemplo, la cadena «01101» es una cadena del alfabeto binario  $\Sigma = \{0, 1\}$ , mientras que «holá» es una cadena del alfabeto de letras minúsculas del alfabeto latino. Incluso la cadena vacía, representada por  $\epsilon$ , es una cadena que puede construirse en cualquier alfabeto.[?]

### 3.4. Cadena vacío.

La cadena vacía es aquella que no contiene ningún símbolo y se denota por  $\epsilon$ . Aunque no contiene símbolos, sigue siendo una cadena válida y puede ser considerada como el elemento neutro de la concatenación de cadenas.[?]

### 3.5. Longitud de cadenas.

La longitud de una cadena se refiere al número de símbolos que contiene. Por ejemplo, la cadena «01101» tiene una longitud de 5, mientras que la cadena vacía tiene una longitud de 0. La notación estándar para indicar la longitud de una cadena  $w$  es  $|w|$ . Por lo tanto,  $|01101| = 5$  y  $|\epsilon| = 0$ .[?]

### 3.6. Potencias de un alfabeto.

Si  $\Sigma$  es un alfabeto, podemos expresar el conjunto de todas las cadenas de una determinada longitud de dicho alfabeto utilizando una notación exponencial. Definimos  $\Sigma^k$  como el conjunto de todas las cadenas de longitud  $k$ , donde cada símbolo de la cadena pertenece a  $\Sigma$ .[?]

### 3.7. Concatenacion de cadenas.

La concatenación de dos cadenas  $x$  e  $y$ , denotada como  $xy$ , consiste en unir la cadena  $x$  seguida de la cadena  $y$ . Por ejemplo, si  $x = "abc"$  y  $y = "def"$ , entonces  $xy = "abcdef"$ . Esta operación es fundamental en la manipulación de cadenas y es comúnmente utilizada en la construcción de algoritmos y programas.[?]

### 3.8. Lenguajes.

Un conjunto de cadenas, todas ellas seleccionadas de un  $\Sigma^*$ , donde  $\Sigma$  es un determinado alfabeto, se denomina lenguaje. Si  $\Sigma$  es un alfabeto y  $L \subseteq \Sigma^*$ , entonces  $L$  es un lenguaje de  $\Sigma$ . Los lenguajes pueden interpretarse como conjuntos de cadenas válidas según ciertos criterios. Por ejemplo, en el lenguaje inglés, el conjunto de palabras válidas representa un lenguaje. Del mismo modo, en programación, el conjunto de programas válidos en un lenguaje de programación determinado constituye un lenguaje. Los lenguajes pueden ser finitos o infinitos, dependiendo de la cantidad de cadenas que los componen.[?]

### 3.9. Autómatas Finitos No Deterministas (AFN)

Los autómatas finitos no deterministas (AFN) son una clase de autómatas que poseen la capacidad de estar en varios estados simultáneamente. Esta característica se manifiesta en la capacidad del autómata para “conjeturar” sobre su entrada en ciertos momentos durante su procesamiento. Por ejemplo, al buscar determinadas secuencias de caracteres, como palabras clave, dentro de una cadena de texto extensa, el AFN puede “conjeturar” el inicio de una de

estas cadenas y emplear una serie de estados para verificar la presencia de la cadena, carácter por carácter.[?]

### 3.9.1. Aplicaciones de los AFN

Un ejemplo claro de la aplicación de los AFN se encuentra en la búsqueda de patrones en texto, como la búsqueda de palabras clave o la validación de formatos específicos. Además, los AFN se utilizan en el análisis léxico de compiladores, donde se emplean para reconocer tokens en un programa fuente.

## 3.10. Equivalencia de autómatas finitos deterministas y no deterministas

La equivalencia entre Autómatas Finitos Deterministas (AFD) y Autómatas Finitos No Deterministas (AFN) es un tema fundamental en la teoría de autómatas. Aunque puede ser más fácil construir un AFN para algunos lenguajes que para un AFD, como en el caso del lenguaje de cadenas que terminan en '01', resulta sorprendente que cualquier lenguaje que puede ser descrito por un AFN también puede ser descrito por algún AFD. Además, en la práctica, un AFD tiene aproximadamente tantos estados como un AFN, aunque puede tener más transiciones. Sin embargo, en el peor de los casos, un AFD puede tener el doble de estados que un AFN más pequeño para el mismo lenguaje.

La demostración de que los AFD pueden hacer lo que hacen los AFN implica una importante *construcción* conocida como la construcción de subconjuntos. Esta construcción implica construir todos los subconjuntos del conjunto de estados del AFN. En general, muchas demostraciones en la teoría de autómatas implican construir un autómata a partir de otro, lo que subraya la importancia de comprender la construcción de subconjuntos como un ejemplo de cómo se puede describir formalmente un autómata en función de los estados y transiciones de otro, sin necesidad de conocer los detalles específicos de este último.

La construcción de subconjuntos comienza con un AFN  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  y tiene como objetivo describir un AFD  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  tal que  $L(D) = L(N)$ . Los componentes de  $D$  se construyen de la siguiente manera:

- $Q_D$  es el conjunto de subconjuntos de  $Q_N$ , es decir, el conjunto potencia de  $Q_N$ . Por lo tanto, si  $Q_N$  tiene  $n$  estados, entonces  $Q_D$  tendrá  $2^n$  estados. Sin embargo, no todos estos estados son necesariamente accesibles desde el estado inicial de  $D$ , por lo que algunos estados pueden ser eliminados.
- $F_D$  es el conjunto de subconjuntos de  $Q_N$  que incluyen al menos un estado de aceptación de  $N$ .
- Para cada conjunto  $S \subseteq Q_N$  y para cada símbolo de entrada  $a \in \Sigma$ ,  $\delta_D(S, a)$  se define como la unión de los estados a los que se puede llegar desde algún estado en  $S$  con la entrada  $a$  según la función de transición de  $N$ .

Esta construcción nos permite obtener un AFD equivalente al AFN original, lo que demuestra la equivalencia de ambos tipos de autómatas.[?]

### 3.11. AFD para reconocer un conjunto de palabras clave

En el contexto de la teoría de autómatas, una aplicación práctica es el reconocimiento de un conjunto de palabras clave mediante un Autómata Finito Determinista (AFD). Esta tarea se puede abordar mediante la construcción de subconjuntos a partir de un Autómata Finito No Determinista (AFN) diseñado específicamente para las palabras clave.

Cuando aplicamos la construcción de subconjuntos a un AFN diseñado a partir de un conjunto de palabras clave, observamos que el número de estados del AFD resultante nunca es mayor que el número de estados del AFN. Esta observación es alentadora ya que, en el caso peor, el número de estados en el AFD crece de manera exponencial. Por lo tanto, es comúnmente

utilizado el método de diseñar un AFN para las palabras clave y luego construir un AFD a partir de él.

Las reglas para construir el conjunto de estados del AFD son las siguientes:

- a) Si  $q_0$  es el estado inicial del AFN, entonces  $\{q_0\}$  es uno de los estados del AFD.
- b) Suponemos que para alguno de los estados del AFN,  $p$ , se llega a él desde el estado inicial siguiendo un camino cuyos símbolos son  $a_1a_2\dots a_m$ . Luego, uno de los estados del AFD es el conjunto de estados del AFN constituido por:
  1.  $q_0$ .
  2.  $p$ .
  3. Cualquier otro estado del AFN al que se pueda llegar desde  $q_0$  siguiendo un camino cuyas etiquetas sean un sufijo de  $a_1a_2\dots a_m$ , es decir, cualquier secuencia de símbolos de la forma  $a_ja_{j+1}\dots a_m$ .

En general, existirá un estado del AFD para cada estado  $p$  del AFN. Sin embargo, en el paso (b), dos estados pueden llevar al mismo conjunto de estados del AFN y, por tanto, ser representados por un único estado en el AFD. Por ejemplo, si dos de las palabras clave comienzan por la misma letra, por ejemplo  $a$ , entonces los dos estados del AFN a los que se puede llegar desde  $q_0$  a través del arco etiquetado con  $a$  llevarán al mismo conjunto de estados del AFN y, por tanto, se reducirán a uno en el AFD.[?]

## 4 Desarrollo de la Practica.

### 4.1. información del sistema.

La siguiente información fue extraída gracias al comando systeminfo en cmd:

1	Nombre de host:	CARBAJE
2	Nombre del sistema operativo:	Microsoft Windows 11 Home
3	Version del sistema operativo:	10.0.22631 N/D Compilacion 22631
4	Fabricante del sistema operativo:	Microsoft Corporation
5	Configuracion del sistema operativo:	Estacion de trabajo independiente
6	Tipo de compilacion del sistema operativo:	Multiprocessor Free
7	Propiedad de:	emi.cruzazul@hotmail.com
8	Organizacion registrada:	
9	Fecha de instalacion original:	04/03/2024, 8:09:03
10	Tiempo de arranque del sistema:	12/03/2024, 9:41:56
11	Fabricante del sistema:	GIGABYTE
12	Modelo el sistema:	G5 KF5
13	Tipo de sistema:	x64-based PC
14	Procesador(es):	1 Procesadores instalados. [01]: Intel64 Family 6 Model 186 Stepping 2 GenuineIntel ~2400 Mhz
15	Version del BIOS:	INSYDE Corp. FD06, 03/11/2023
16	Directorio de Windows:	C:\Windows
17	Directorio de sistema:	C:\Windows\system32
18	Dispositivo de arranque:	\Device\HarddiskVolume1
19	Configuracion regional del sistema:	es;Espanol (internacional)
20	Idioma de entrada:	en-us;Ingles (Estados Unidos)
21	Zona horaria:	(UTC-06:00) Ciudad de Mexico, Monterrey
22	Cantidad total de memoria fisica:	16.088 MB
23	Memoria fisica disponible:	7.991 MB
24	Memoria virtual: tamano maximo:	65.240 MB
25	Memoria virtual: disponible:	53.369 MB
26	Memoria virtual: en uso:	11.871 MB
27	Ubicacion(es) de archivo de paginacion:	C:\pagefile.sys
28	Dominio:	WORKGROUP
29	Servidor de inicio de sesion:	\CARBAJE
30	Revision(es):	5 revision(es) instaladas. [01]: KB5034467 [02]: KB5027397

```
34 [03]: KB5036212
35 [04]: KB5034848
36 [05]: KB5035226
37 Tarjeta(s) de red:
38     instaladas.
39             3 Tarjetas de interfaz de red
40
41             [01]: Realtek PCIe GbE Family
42                 Controller
43                     Nombre de conexion: Ethernet
44                     Estado:           Medios
45                     desconectados
46             [02]: Bluetooth Device (Personal Area
47                 Network)
48                     Nombre de conexion: Conexion
49                     de red Bluetooth
50                     Estado:           Medios
51                     desconectados
52             [03]: Intel(R) Wi-Fi 6E AX211 160MHz
53                     Nombre de conexion: Wi-Fi
54                     DHCP habilitado: Si
```

## 4.2. Lenguajes de programación usados.

La combinación de C++ y MATLAB ofrece una solución integral para diseñar, implementar y analizar los programas *universo, tablero y buscador de palabras* de manera eficaz y precisa. C++ proporciona un control preciso sobre los recursos del sistema y una alta eficiencia computacional, mientras que MATLAB ofrece herramientas avanzadas de visualización y cálculo numérico, facilitando el análisis y la comprensión de los resultados. Juntos, estos lenguajes permiten abordar los desafíos específicos de cada programa de manera efectiva.

### 4.2.1. C++

La elección de C++ para la elaboración de la lógica del programa se basa en su eficiencia, control sobre los recursos, flexibilidad y compatibilidad. Estas características hacen que sea una opción sólida y adecuada para implementar los programas descritos anteriormente de manera óptima y eficiente. A continuación se enumera a detenimiento las ventajas mencionadas:

- 1. Eficiencia y rendimiento:** C++ es conocido por ser un lenguaje de programación de alto rendimiento. Esto significa que los programas escritos en C++ tienden a ejecutarse más rápido y a consumir menos recursos que aquellos escritos en lenguajes de más alto nivel, como MATLAB. Dado que estamos trabajando en la implementación de los programas, donde la eficiencia es crucial, el uso de C++ puede garantizar un rendimiento óptimo.
- 2. Control sobre los recursos:** C++ proporciona un alto grado de control sobre la gestión de memoria y otros recursos del sistema. Esto es especialmente importante en aplicaciones donde se manejan grandes volúmenes de datos o se realizan operaciones intensivas en términos de recursos. En el contexto de los programas, donde estamos trabajando con grandes conjuntos de datos, este control adicional puede ser beneficioso.
- 3. Flexibilidad y versatilidad:** C++ es un lenguaje multiparadigma que permite programar en diferentes estilos, como programación orientada a objetos, programación genérica y programación procedural. Esta versatilidad ofrece la posibilidad de diseñar y estructurar el código de manera óptima según las necesidades específicas del problema. En el caso de la implementación de este conjunto de programas, esta flexibilidad nos es útil para organizar y modularizar el código de manera eficiente.
- 4. Compatibilidad y portabilidad:** C++ es un lenguaje ampliamente utilizado y está disponible en una amplia variedad de plataformas y sistemas operativos. Esto garantiza que el código desarrollado en C++ pueda ejecutarse en diferentes entornos sin mayores modificaciones, lo que aumenta la portabilidad y la interoperabilidad de la solución.

#### 4.2.2. Matlab

La elección de MATLAB para la elaboración de la graficación y animación del programa se basa en su facilidad de prototipado, sus capacidades avanzadas de cálculo numérico y matemático, sus herramientas de visualización y su integración con otras herramientas. Estas características hacen que sea una opción sólida y eficaz para la representación visual y el análisis de los resultados del conjunto de programas. A continuación se enlistan las ventajas mencionadas:

1. **Facilidad de prototipado y desarrollo rápido:** MATLAB es conocido por su capacidad para el prototipado rápido y el desarrollo eficiente de algoritmos. Proporciona una amplia gama de funciones y herramientas integradas que facilitan la implementación de algoritmos complejos con un código más compacto y legible. Esto es especialmente útil en el contexto de la práctica, donde la experimentación y la iteración rápida son fundamentales para el diseño y la optimización del conjunto de programas.
2. **Amplio conjunto de herramientas para cálculos numéricos y matemáticos:** MATLAB ofrece una amplia gama de funciones y herramientas especializadas para realizar cálculos numéricos y matemáticos avanzados. Esto incluye funciones para operaciones con matrices, álgebra lineal, transformadas, optimización y simulación, entre otros. Estas herramientas son fundamentales para la manipulación y el procesamiento de datos en el contexto que estamos trabajando.
3. **Gráficos y visualización avanzados:** MATLAB cuenta con potentes capacidades de gráficos y visualización que facilitan la representación visual de datos y resultados. Esto es especialmente importante en el contexto de la práctica, donde se desea visualizar y analizar los resultados de los programas, como patrones de secuencias generadas y animaciones. Las capacidades de graficación de MATLAB permiten crear gráficos personalizados y visualizaciones interactivas para explorar y comprender mejor los datos generados por el programa.
4. **Integración con herramientas adicionales:** MATLAB se integra bien con otras herramientas y entornos de desarrollo, lo que facilita la incorporación de funcionalidades adicionales o la conexión con sistemas externos si es necesario. Esto nos resulta útil para ampliar la funcionalidad del generador de potencias o integrarlo en un flujo de trabajo más amplio.

#### 4.3. Librerías usadas

1. **iostream:** Se utiliza para la interacción básica con el usuario, permitiendo la entrada desde el teclado con `cin` y la salida en la consola con `cout`. Es esencial para la comunicación en tiempo real entre el usuario y el programa.
2. **fstream:** Esta biblioteca es crucial para la manipulación de archivos en C++. Se utiliza para abrir, leer, escribir y cerrar archivos en el sistema de archivos del ordenador. Es fundamental para leer y escribir archivos de texto o binarios, lo que permite almacenar y recuperar información de manera persistente.
3. **vector:** Se emplea para implementar contenedores de datos dinámicos en C++. Permite almacenar colecciones de elementos de tamaño variable de manera eficiente, con la capacidad de cambiar su tamaño dinámicamente según sea necesario. Es útil para manejar listas de elementos que pueden crecer o reducirse durante la ejecución del programa.
4. **string:** Esta biblioteca es esencial para manipular cadenas de caracteres en C++. Proporciona clases y funciones para trabajar con texto, como la búsqueda de subcadenas, la concatenación y la conversión entre diferentes tipos de datos. Se utiliza ampliamente en aplicaciones que involucran el procesamiento de texto.
5. **sstream:** Se utiliza para realizar operaciones de entrada y salida en cadenas de caracteres en memoria. Es útil para construir o manipular cadenas de caracteres de manera eficiente sin necesidad de interactuar con archivos. Se emplea cuando es necesario manipular datos en formato de cadena de caracteres.
6. **time.h:** Se utiliza para trabajar con el tiempo y la fecha en C. Proporciona funciones y estructuras para obtener la fecha y la hora actuales, así como para realizar operaciones de manipulación de

tiempo, como el cálculo de diferencias entre fechas. Es útil en aplicaciones que requieren el manejo de fechas y horarios.

7. **chrono**: Ofrece utilidades para medir el tiempo y realizar operaciones con intervalos de tiempo en C++. Proporciona una interfaz moderna y robusta para trabajar con el tiempo, lo que facilita la medición precisa del tiempo de ejecución de diferentes partes del programa.
8. **random**: Se utiliza para generar números aleatorios y distribuciones aleatorias en C++. Es útil para generar secuencias de números aleatorios de manera controlada y realizar experimentos que requieran aleatoriedad, como simulaciones y pruebas de rendimiento.
9. **windows.h**: Contiene funciones y estructuras para la programación de Windows en C++. Proporciona acceso a diversas funcionalidades del sistema operativo Windows, como la manipulación de procesos, hilos, memoria y entrada/salida. Se utiliza en aplicaciones que interactúan directamente con el sistema operativo Windows.
10. **unordered\_set**: Se utiliza para implementar conjuntos no ordenados en C++. Permite almacenar elementos únicos sin un orden específico y proporciona acceso rápido a los elementos mediante tablas hash. Es útil cuando se necesita una estructura de datos que garantice la unicidad de los elementos y un acceso rápido a ellos.
11. **regex**: Ofrece soporte para expresiones regulares en C++, que son patrones utilizados para buscar y manipular texto de manera eficiente. Se utiliza para realizar operaciones avanzadas de búsqueda y manipulación de cadenas de caracteres, como la validación de formatos de datos y la extracción de información específica de un texto.
12. **cstdlib**: Se utiliza para manipular cadenas, convertir tipos de datos y generar números aleatorios en C++. Forma parte de la librería estándar de C++ y proporciona funciones útiles para operaciones comunes en programación.
13. **ctime**: Proporciona funciones relacionadas con el tiempo y la fecha en C++, similar a **time.h**, pero con una interfaz más moderna y orientada a objetos. Se utiliza para obtener información sobre el tiempo de ejecución del programa y realizar operaciones relacionadas con el tiempo.
14. **iomanip**: Se utiliza para formatear la salida en streams en C++. Proporciona manipuladores de flujo que permiten establecer el ancho de campo, el número de decimales y otros aspectos del formato de salida. Es útil para mejorar la presentación de la salida en la consola o en archivos de texto.
15. **math.h**: Contiene funciones y constantes matemáticas en C++, como funciones trigonométricas, exponenciales y de redondeo. Se utiliza para realizar operaciones matemáticas avanzadas en aplicaciones que requieren cálculos numéricos.
16. **algorithm**: Se utiliza para realizar operaciones en rangos de elementos en C++, como ordenamiento, búsqueda y manipulación de contenedores. Es una biblioteca estándar de C++ que facilita la manipulación de datos y la implementación de algoritmos comunes.

## 5 Implementación

### 5.1. Practica 01: Universo de Cadenas Binarias.

#### 5.1.1. PRACTICA01.h

Este archivo, tambien llamado biblioteca a partir de ahora, tiene solamente una bolque de codigo donde fragmento de código establece las bases para el programa, incluyendo las bibliotecas necesarias y la declaración de las funciones que se utilizarán. La implementación específica de cada función y la lógica del programa se encuentra en otros lugares del código. A continuacion se deja el bloque de codigo de la biblioteca:

```
1 #include <iostream>
2 #include <string>
```

```
3 #include <fstream>
4 #include <cstdlib>
5 #include <ctime>
6 #include <vector>
7 #include <chrono>
8 #include <windows.h>
9 #include <iomanip>
10 #include <math.h>
11 #include <algorithm>
12 //##include "matplotlibcpp.h"
13
14 //funciones
15 void graficar(int&);
16 void opcionGraficar(int&);
17 void medirTiempo(std::chrono::high_resolution_clock::time_point);
18 void mostrarBarraProgreso(int&, int&);
19 void generarCombinaciones(std::ofstream&, std::ofstream&, int, int&, int&,
   char, char, std::string, int);
20 void ejecucion(int&, int, char&, char&);
21 void ingresarPotencia(int&);
22 void randomPotencia(int&);
23 void ingresarCaracteres(char&, char&);
24 int ingresarOpcion();
25 void menu(int&);
26 int main();
```

Listing 1: PRACTICA01.h

### 5.1.2. *mainP01.cpp*

En este archivo existe únicamente una función, el siguiente fragmento de código inicia la ejecución del programa, llama a una función `menu` para presentar al usuario un menú de opciones, y devuelve un valor de error al sistema operativo al finalizar la ejecución. La lógica específica del programa, incluyendo la implementación de la función `menu`, se encuentra en otros archivos que están incluidos a través del archivo de encabezado «`PRACTICA01.h`». A continuación se muestra la función `main` en cuestión:

```
1 //MAIN
2 #include "PRACTICA01.h"
3
4 int main(){
5     int error = 0;
6     menu(error);
7
8     return error;
9 }
```

Listing 2: *mainP01.cpp* — `main()`

### 5.1.3. *menuP01.cpp*

Este código es un programa que presenta un menú con varias opciones para el usuario. La función `menu` muestra el menú y solicita al usuario que elija una opción. Dependiendo de la opción seleccionada, el programa realiza diferentes acciones:

1. Si el usuario elige la opción 0, el programa se detiene y muestra un mensaje de despedida.
2. Si elige la opción 1, le permite ingresar una potencia.
3. Si elige la opción 2, genera una potencia aleatoria.
4. Si elige la opción 3, le permite cambiar los caracteres utilizados en el programa.
5. Si elige la opción 4, ejecuta una serie de acciones, incluida la generación de combinaciones y la medición del tiempo transcurrido.

A continuación se ve de manera separada cada función que compone al archivo `menuP01.cpp`:

## MenuP01

Este fragmento de código define la función `menu`, presenta un menú de opciones al usuario y ejecuta las acciones correspondientes según la opción seleccionada. A continuación el bloque de código de la función en cuestión:

```
1 void menu(int&error){  
2     int flag = -1;  
3     int potencia = 0;  
4     int iteraciones = 1;  
5     char simbolo0 = '#';  
6     char simbolo1 = '.';  
7  
8     while(flag != 0){  
9         std::cout << "BIENVENIDO A LA PRACTICA 01." << std::endl;  
10        std::cout << "OPCIONES A REALIZAR" << std::endl;  
11        std::cout << "1- Ingresar potencia." << std::endl;  
12        std::cout << "2- Dar potencia random." << std::endl;  
13        std::cout << "3- Cambiar caracteres." << std::endl;  
14        std::cout << "4- Ejecutar Programa." << std::endl;  
15        std::cout << "0- Salir del Programa." << std::endl;  
16  
17        flag = ingresarOpcion();  
18  
19        switch(flag){  
20            case 0:  
21                std::cout << "GRACIAS POR USAR EL PROGRAMA." << std::endl;  
22                std::cout << "ADIOS." << std::endl;  
23                return;  
24                break;  
25            case 1:  
26                ingresarPotencia(potencia);  
27                break;  
28            case 2:  
29                randomPotencia(potencia);  
30                break;  
31            case 3:  
32                ingresarCaracteres(simbolo0, simbolo1);  
33                break;  
34            case 4:  
35                ejecucion(potencia, iteraciones, simbolo0, simbolo1);  
36                opcionGraficar(error);  
37                break;  
38  
39        }  
40    }  
41}
```

Listing 3: Archivo: *menuP01.cpp* — Funcion: `menu`

## IngresarOpcionP01

Esta función `ingresarOpcionP01` proporciona una forma de interactuar con el usuario, permitiéndole seleccionar una opción deseada del menú mediante la entrada de un número entero en la consola. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación se anexa el bloque de código que alude a la función mencionada:

```
1 int ingresarOpcion(){  
2     int opcion= 5;  
3     std::cout << "Ingrese la opcion deseada: ";  
4     std::cin >> opcion;  
5  
6     return opcion;  
7 }
```

Listing 4: Archivo: *menuP01.cpp* — Funcion: `ingresarOpcion`

## EjecucionP01

Esta función `ejecucionP01` realiza el trabajo principal del programa, que incluye la generación de combinaciones y la medición del tiempo de ejecución. Además, gestiona la creación y escritura de archivos para almacenar los resultados de las combinaciones generadas. A continuación se anexa el bloque de código que alude a la función mencionada:

```
1 void ejecucion(int&potencia, int iteraciones, char&simbolo0, char&simbolo1){  
2     int total = 0;  
3     for (int i = 0; i <= potencia; i++)  
4         total += pow(2,i);  
5  
6     //Crear y abrir un archivo  
7     std::ofstream archivo("UniversoP01.txt");  
8     std::ofstream archivo2("salidaPractica1.csv");  
9  
10    // Iniciar el cronometro  
11    auto start = std::chrono::high_resolution_clock::now();  
12  
13    // Generar las combinaciones de longitud creciente del alfabeto  
14    archivo << '\u03B5' << ",";  
15    archivo2 << "0,0" << std::endl;  
16    generarCombinaciones(archivo, archivo2, potencia, iteraciones, total,  
17        simbolo0, simbolo1, "", 0);  
18  
19    // Detener el cronometro y mostrar el tiempo transcurrido  
20    medirTiempo(start);  
21  
22    //Mostrar iteraciones y tiempo cronometro  
23    std::cout<<"Numero de iteraciones realizadas: "<<iteraciones<<std::endl;  
24  
25    // Cerrar el archivo  
26    archivo.close();  
}
```

Listing 5: Archivo: *menuP01.cpp* — Funcion: *ejecucion()*

## MedirTiempoP01

Esta función `medirTiempo` calcula el tiempo transcurrido entre un punto de inicio y un punto de finalización utilizando la librería `<chrono>` de C++. Aquí está el bloque de código de la función:

```
1 void medirTiempo(std::chrono::high_resolution_clock::time_point start) {  
2     auto stop = std::chrono::high_resolution_clock::now();  
3     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop  
4         - start);  
5     auto milliseconds = duration.count() % 1000;  
6     auto seconds = (duration.count() / 1000) % 60;  
7     auto minutes = (duration.count() / (1000 * 60)) % 60;  
8  
8     std::cout << "Tiempo transcurrido: " << minutes << " minutos, " << seconds  
9         << " segundos, " << milliseconds << " milisegundos" << std::endl;  
9 }
```

Listing 6: Archivo: *menuP01.cpp* — Funcion *medirTiempo*

### 5.1.4. *solicitudesP01.cpp*

El archivo *solicitudesP01.cpp* contiene implementaciones de funciones relacionadas con la manipulación de datos o la interacción con el usuario en el contexto específico de la Práctica 01. Estas funciones incluyen la generación de combinaciones, la manipulación de archivos y la realización de cálculos específicos para resolver el problema abordado en la presente práctica. En esta parte del reporte se exploraran las funciones que la componen.

## IngresarPotencia

Esta función `ingresarPotencia` proporciona una forma de interactuar con el usuario, permitiéndole ingresar la potencia deseada para realizar ciertas operaciones en el programa. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación el bloque de código de la función en cuestión:

```
1 //Solicitudes
2 #include "PRACTICA01.h"
3
4 void ingresarPotencia(int&potencia){
5     std::cout << "Ingrese la potencia deseada: ";
6     std::cin >> potencia;
7 }
```

Listing 7: Archivo: *solicitudesP01.cpp* — Función `ingresarPotencia()`

## RandomPotencia

Esta función `randomPotencia` proporciona una forma de generar de manera aleatoria un valor de potencia y asignarlo a una variable, lo que puede ser útil en situaciones donde no se requiere una potencia específica y se desea una elección aleatoria.

A continuación se deja el bloque de código en cuestión:

```
1 void randomPotencia(int&potencia){
2     srand(time(0));
3     potencia = rand() % (28 - 0 + 1) + 0;
4     std::cout << "la potencia escogida de manera aleatoria fue:
5         " << potencia << ". " << std::endl;
```

Listing 8: Archivo: *solicitudesP01.cpp* — Función `randomPotencia`

## IngresarCaracteres

La función `ingresarCaracteres` solicita al usuario que ingrese dos caracteres que representarán los símbolos cero y uno del alfabeto deseado. Luego, almacena estos caracteres en las variables `simbolo0` y `simbolo1`, que se pasan por referencia a la función. A continuación se deja el código:

```
1 void ingresarCaracteres(char&simbolo0, char&simbolo1){
2     std::cout << "Ingrese el simbolo cero para el alfabeto deseado: ";
3     std::cin >> simbolo0;
4     std::cout << "Ingrese el simbolo uno para el alfabeto deseado: ";
5     std::cin >> simbolo1;
6     //Crear Archivo para detectar el 0 y uno del universo
7     std::ofstream archivo("Universo.txt");
8     archivo << simbolo0 << ',' << simbolo1;
9     archivo.close();
10 }
```

Listing 9: Archivo: *solicitudesP01.cpp* — Función `ingresarCaracteres`

## OpcionGraficar

La función `opcionGraficar` proporciona una manera de interactuar con el usuario, permitiéndole decidir si desea o no graficar el resultado del programa. La función `graficar` se llama si el usuario elige sí, de lo contrario, no se realiza ninguna acción adicional. Se deja el bloque de código que engloba a la función en cuestión:

```
1 void opcionGraficar(int&error){  
2     char opcion = 'Y';  
3     std::cout<<"Desea Graficar el resultado(Y[y]/N[n]): ";  
4     std::cin>>opcion;  
5     if (opcion=='Y' || opcion=='y')  
6         graficar(error);  
7 }
```

Listing 10: Archivo: *solicitudesP01.cpp* — Funcion opcionGraficar

### 5.1.5. *P01.cpp*

El archivo *P01.cpp* incluye únicamente a la función `generarCombinaciones` función encargada de la generación de combinaciones y el cálculo de los unos en cada combinación. Utiliza recursión para explorar todas las posibles combinaciones de longitud `len` y registra los datos en archivos de salida.

A continuación se deja el bloque de código de la función

```
1 #include "PRACTICA01.h"  
2  
3 // Funcion para generar combinaciones y calcular unos  
4 void generarCombinaciones(std::ofstream&archivo1, std::ofstream&archivo2, int len,  
5     int&i, char simbolo0, char simbolo1, std::string prefix = "", int unos = 0) {  
6     if (len == 0) {  
7         return;  
8     }  
9     for (char c : {simbolo0, simbolo1}) {  
10         std::string newPrefix = prefix + c;  
11         archivo1 << newPrefix << ",";  
12         archivo2 << i << ',' << unos + (c == simbolo1) << std::endl;  
13         i++;  
14         generarCombinaciones(archivo1, archivo2, len - 1, i, simbolo0, simbolo1,  
15             newPrefix, unos + (c == simbolo1));  
16     }  
17 }
```

Listing 11: Archivo: *P01.cpp* — Funcion generarCombinaciones

Se detalla a lujo de detalle, esta función, la función principal para la realización de esta práctica:

#### 1. Parámetros de entrada:

- `std::ofstream&archivo1`: Referencia a un archivo de salida donde se escribirán las combinaciones generadas.
- `std::ofstream&archivo2`: Referencia a otro archivo de salida donde se escribirán los datos relacionados con las combinaciones.
- `int len`: Longitud de las combinaciones a generar.
- `int&i`: Variable que lleva la cuenta del número de combinaciones generadas.
- `char simbolo0`: Símbolo cero del alfabeto.
- `char simbolo1`: Símbolo uno del alfabeto.
- `std::string prefix = ' ' '`: Prefijo de la combinación actual (se inicializa como una cadena vacía).
- `int unos = 0`: Número de unos en la combinación actual (se inicializa como cero).

#### 2. Generación de combinaciones:

- La función utiliza recursión para generar todas las combinaciones posibles de longitud `len`.
- En cada iteración, se agrega uno de los dos símbolos (`simbolo0` o `simbolo1`) al prefijo de la combinación actual.
- Se escriben las combinaciones y los datos relacionados en los archivos de salida proporcionados.

### 3. Conteo de unos:

- Se lleva un conteo de los unos en la combinación actual para escribir los datos correspondientes en el segundo archivo de salida.

#### 5.1.6. *graficarP01.cpp*

El archivo *graficarP01.cpp* incluye únicamente a la función *graficar*, la cual ejecuta un script de MATLAB para generar un gráfico. Primero, define la ruta del script y luego la ejecuta utilizando la función *system*. Se verifica si la ejecución fue exitosa y se imprime un mensaje correspondiente. Después, se espera 15 segundos antes de iniciar la graficación en MATLAB utilizando la función *Sleep* de la biblioteca *windows.h*. Esta función facilita la integración de gráficos generados en MATLAB en el flujo de trabajo del programa de C++. A continuación el bloque de código de la función en cuestión:

```
1 #include "PRACTICA01.h"
2 #include <windows.h> // Incluir la biblioteca windows.h
3
4 void graficar(int&error){
5     std::string ruta = "MATLAB -r run('graficoMatlab.m')";
6
7     error = system(ruta.c_str());
8
9     // Verificar si la ejecución fue exitosa
10    if (error == 0) {
11        std::cout << "La ruta se ejecutó correctamente." << std::endl;
12    } else {
13        std::cerr << "Error al ejecutar la ruta." << std::endl;
14    }
15
16    // Esperar un tiempo adicional
17    std::cout << "Esperando 10 segundos antes de cerrar MATLAB..." << std::endl;
18    Sleep(10000); // Espera 10000 milisegundos (equivalente a 10 segundos)
19 }
```

Listing 12: Archivo: *P01.cpp* — Funcion *graficar()*

#### 5.1.7. *graficoMatlab.m*

Este script de MATLAB configura y traza dos subgráficos: uno que muestra la cantidad de puntos por cadena y otro que muestra el logaritmo de la cantidad de puntos por cadena. Se deja el código del script en cuestión:

```
1 % Configurar el trazado
2 scatter_options = ' .';
3
4 % Inicializar variables para el trazado
5 figure;
6
7 % Nombre del archivo y longitud del bloque
8 nombre_archivo = 'salidaPractical.csv';
9 block_size = 1e6; % Por ejemplo, lee el archivo en bloques de 1 millón de tuplas
10
11 % Abrir el archivo para lectura
12 fileID = fopen(nombre_archivo, 'r');
13
14 if fileID == -1
15     error('No se pudo abrir el archivo.');
16 end
17
18 % Crear subplots
19 subplot(2, 1, 1);
20 hold on;
21
22 % Leer y graficar los datos en bloques
23 while ~feof(fileID)
24     % Leer un bloque de datos del archivo
```

```
25 data = textscan(fileID, '%f%f', 'block_size', 'Delimiter', ',', 'HeaderLines',
26 1);
27 if isempty(data{1})
28     break; % No hay mas datos para leer, salir del bucle
29 end
30
31 % Trazar los datos del bloque actual
32 scatter(data{1}, data{2}, scatter_options);
33 end
34
35 % Configurar etiquetas y titulo
36 xlabel('ID de cadena');
37 ylabel('Cantidad de puntos en la cadena');
38 title('Grafico de la cantidad de puntos por cadena');
39
40 % Crear subplots para el logaritmo
41 subplot(2, 1, 2);
42 hold on;
43
44 % Reiniciar la lectura del archivo
45 frewind(fileID);
46
47 % Leer y graficar los datos en bloques con logaritmo
48 while ~feof(fileID)
49     % Leer un bloque de datos del archivo
50     data = textscan(fileID, '%f%f', 'block_size', 'Delimiter', ',', 'HeaderLines',
51 1);
52 if isempty(data{1})
53     break; % No hay mas datos para leer, salir del bucle
54 end
55
56 % Aplicar el logaritmo a la cantidad de puntos
57 data_log = log(data{2});
58
59 % Trazar los datos del bloque actual con logaritmo
60 scatter(data{1}, data_log, scatter_options);
61 end
62
63 % Configurar etiquetas y titulo para el subplot de logaritmo
64 xlabel('ID de cadena');
65 ylabel('Log(Cantidad de puntos en la cadena)');
66 title('Grafico del logaritmo de la cantidad de puntos por cadena');
67
68 % Cerrar el archivo
69 fclose(fileID);
70
71 % Mostrar el grafico
72 hold off;
73 pause(15);
74 close all;
75 pause(5);
76 exit();
```

Listing 13: Archivo: *graficoMatlab* — Script

## 5.2. Practica03: Tablero de Ajedrez

### 5.2.1. *PRACTICA03.h*

El archivo *PRACTICA03.h* define varias estructuras y funciones relacionadas con un programa de ajedrez. Define las estructuras *Casilla* y *Tablero* para representar el tablero de ajedrez y las casillas, respectivamente. También incluye funciones para animar el tablero, encontrar jugadas que terminan en *01*, guardar las jugadas en un archivo de texto, medir el tiempo de ejecución, ejecutar la secuencia de funciones del programa, crear una cadena para el jugador dos, generar una cadena binaria aleatoria, solicitar al usuario una cadena, ingresar una opción, crear el tablero y definir sus transiciones, crear las casillas del tablero, mostrar un menú, y la función *main* que inicia la ejecución del programa.

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <time.h>
6 #include <chrono>
7 #include <random>
8 #include <windows.h>
9 //#include <mutex>
10 //#include <thread>
11
12 // Definicion de la estructura de los Casillas
13 struct Casilla {
14     int id;
15     std::vector<std::pair<char, int>> transiciones; // Transiciones: (simbolo,
16                                         Casilla_destino)
17 };
18
19 // Definicion de la estructura del automata
20 struct Tablero {
21     std::vector<Casilla> grafo;
22     int casillaInicial;
23     int casillaFinal;
24 };
25
26 //Funcion para realizar la animacion
27 void animarP03(int&);
28
29 //Funcion para solicitar al usuario la animacion
30 void opcionAnimarP03(int&);
31
32 // Funcion para encontrar todas las Jugadas al Casilla final que terminan en "01"
33 // usando DFS
34 std::vector<std::vector<int>> encontrarJugadas(const Tablero&, const std::string&,
35                                                 int, int, std::vector<int>);
36
37 // Funcion para guardar las Jugadas en un archivo de texto
38 void guardarJugadasEnArchivo(const std::vector<std::vector<int>>&, const
39                               std::string&, const std::string&, std::string);
40
41 //Funcion para medir el tiempo
42 void medirTiempoP03(std::chrono::high_resolution_clock::time_point);
43
44 //Funcion para ejecutar la secuencia de funciones del programa
45 void ejecucionP03(std::string, Tablero, Tablero, int&);
46
47 //Funcion para crear una segunda cadena para el jugador dos a partir de la
48 // original o a partir de una nueva de manera random
49 std::string crearCadena2P03(std::string);
50
51 //Funcion para generar una cadena binaria aleatoria de size aleatorio (entre uno y
52 // cien)
53 std::string generarCadenaP03(std::string);
54
55 //Funcion para solicitar al usuario la cadena
56 std::string solicitarCadenaP03();
57
58 //Funcion para ingresar la opcion:
59 int ingresarOpcionP03();
60
61 //Funcion para generar el automata y definir sus transiciones
62 Tablero crearTablero(int );
63
64 //Funcion para crear los Casillas del Tablero
65 std::vector<Casilla> crearCasilla();
```

Listing 14: Archivo: *PRACTICA03.h*

### 5.2.2. mainP03.cpp

En este archivo existe únicamente una función, el siguiente fragmento de código inicia la ejecución del programa, llama a una función `menu` para presentar al usuario un menú de opciones, y devuelve un valor de error al sistema operativo al finalizar la ejecución. La lógica específica del programa, incluyendo la implementación de la función `menu`, se encuentra en otros archivos que están incluidos a través del archivo de encabezado «`PRACTICA03.h`». A continuación se muestra la función `main` en cuestión:

```
1 #include "PRACTICA03.h"
2
3 int main() {
4     int error = 0;
5     menuP03(error);
6     return error;
7 }
```

Listing 15: Archivo: *PRACTICA03.h*

### 5.2.3. menuP03.cpp

#### menuP03

La función `menuP03` muestra un menú de opciones para interactuar con el programa de la Práctica 03. Las opciones incluyen ingresar una cadena, generar una cadena aleatoria, ejecutar el programa y salir del programa. Dependiendo de la opción seleccionada por el usuario, se realizan diferentes acciones, como solicitar una cadena al usuario, generar una cadena aleatoria, ejecutar el programa con las tablas de ajedrez proporcionadas y mostrar una animación si es posible.

La función toma como argumento una referencia a un entero `error` que se utiliza para manejar errores dentro del programa.

La función no devuelve ningún valor.

```
1 #include "PRACTICA03.h"
2
3 void menuP03(int&error) {
4     bool repetir = true;
5     int opcion = 1;
6     std::string cadena = "";
7     Tablero tablero = crearTablero(0);
8     Tablero tablero2 = crearTablero(1);
9
10
11    while(repetir){
12        std::cout << "BIENVENIDO A LA PRACTICA 03." << std::endl;
13        std::cout << "OPCIONES A REALIZAR" << std::endl;
14        std::cout << "1- Ingresar cadena." << std::endl;
15        std::cout << "2- Generar cadena random." << std::endl;
16        std::cout << "3- Ejecutar Programa." << std::endl;
17        std::cout << "0- Salir del Programa." << std::endl;
18
19        opcion = ingresarOpcionP03();
20
21        switch (opcion) {
22            case 0:
23                std::cout << "Gracias Por usar el programa..." << std::endl;
24                repetir = false;
25                break;
26            case 1:
27                cadena = solicitarCadenaP03();
28                break;
29            case 2:
30                cadena = generarCadenaP03(" ");
31                break;
32            case 3:
33                if (!cadena.empty()) {
34                    ejecucionP03(cadena, tablero, tablero2, error);
35                    opcionAnimarP03(error);
36                }
37        }
38    }
39}
```

```
36         }else {
37             std::cout<<"Favor de ingresar (o generar) una cadena binaria al
38             programa..."<<std::endl;
39         }
40         break;
41     default:
42         std::cout<<"Favor de indicar una opcion valida...."<<std::endl;
43     }
44 }
```

Listing 16: Archivo: *menuP03.cpp* — Funcion menuP03

### ingresarOpcionP03

Esta función `ingresarOpcionP01` proporciona una forma de interactuar con el usuario, permitiéndole seleccionar una opción deseada del menú mediante la entrada de un número entero en la consola. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación se anexa el bloqued de codigo que alude a la función mencionada:

```
1 int ingresarOpcionP03(){
2     int opcion= 5;
3     std::cout << "Ingrese la opcion deseada: ";
4     std::cin >> opcion;
5
6     return opcion;
7 }
```

Listing 17: Archivo: *menuP03.cpp* — Funcion menuP03

### ejecucionP03

La función `ejecucionP03` lleva a cabo la ejecución principal del programa de la Práctica 03. Recibe como parámetros una cadena binaria `cadena`, dos tableros de ajedrez `tablero` y `tablero2`, y una referencia a un entero `error` para manejar errores durante la ejecución.

En la función, se verifica si la última letra de la cadena es `w`. Si es así, se genera una cadena para el jugador 2 utilizando la función `generarCadenaP03`. Luego, se inicia un cronómetro para medir el tiempo de ejecución.

Se generan vectores de jugadas posibles al estado final para ambas cadenas utilizando la función `encontrarJugadas`.

Después de detener el cronómetro, se especifican las rutas de los archivos de salida donde se guardarán las jugadas en formato de texto y CSV.

Finalmente, se llama a la función `guardarJugadasEnArchivo` dos veces, una para cada cadena, para guardar las jugadas en los archivos especificados.

Si la cadena no termina en `w`, se muestra un mensaje indicando que la cadena no termina en un valor válido para generar rutas y se solicita al usuario que ingrese otra cadena.

La función no devuelve ningún valor.

```
1 void ejecucionP03(std::string cadena, Tablero tablero, Tablero tablero2,
2     int&error){
3     if(cadena[cadena.size()-1] == 'w') {
4         //Generar la cadena para el jugador 2
5         std::string cadena2 = generarCadenaP03(cadena);
6
7         // Iniciar el cronometro
8         auto start = std::chrono::high_resolution_clock::now();
9
10        //Generar el vector de jugadas posibles al estado final para la cadena
11        auto rutas = encontrarJugadas(tablero, cadena, 0, tablero.casillaInicial,
12            {});
```

```
11     auto rutas2 = encontrarJugadas(tablero2, cadena2, 0,
12         tablero2.casillaInicial, {});
13
14     // Detener el cronometro y mostrar el tiempo transcurrido
15     medirTiempoP03(start);
16
17     std::string rutaA01 = "jugadas.txt";
18     std::string rutaC01 = "J01.csv";
19     std::string rutaA02 = "jugadas2.txt";
20     std::string rutaC02 = "J02.csv";
21
22     guardarJugadasEnArchivo(rutas, rutaA01, rutaC01, cadena);
23     guardarJugadasEnArchivo(rutas2, rutaA02, rutaC02, cadena2);
24 } else
25     std::cout << "La cadena no termina en un valor valido para generar rutas,
26         ingrese otra cadena." << std::endl;
27 }
```

Listing 18: Archivo: *menuP03.cpp* — Funcion ejecucionP03

#### medirTiempoP03

Esta función **medirTiempo** calcula el tiempo transcurrido entre un punto de inicio y un punto de finalización utilizando la librería **<chrono>** de C++. Aquí está el bloque de código de la función:

```
1 void medirTiempoP03(std::chrono::high_resolution_clock::time_point start) {
2     auto stop = std::chrono::high_resolution_clock::now();
3     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop -
4         start);
5     auto milliseconds = duration.count() % 1000;
6     auto seconds = (duration.count() / 1000) % 60;
7     auto minutes = (duration.count() / (1000 * 60)) % 60;
8
8     std::cout << "Tiempo transcurrido: " << minutes << " minutos, " << seconds <<
9         " segundos, " << milliseconds << " milisegundos" << std::endl;
9 }
```

Listing 19: Archivo: *menuP03.cpp* — Funcion medirTiempoP03

#### 5.2.4. *solicitudesP03.cpp*

##### solicitarCadenaP03

La función **solicitarCadenaP03** solicita al usuario que ingrese una cadena binaria compuesta únicamente por los caracteres *r* y *w*. Luego, verifica que la cadena ingresada cumpla con esta restricción. Si la cadena contiene algún otro carácter, se muestra un mensaje de error y se solicita al usuario que ingrese la cadena nuevamente. El proceso se repite hasta que se ingrese una cadena válida.

Una vez que se ha ingresado una cadena válida, esta se devuelve como un objeto de tipo **std::string**.

La función no tiene argumentos de entrada y devuelve una cadena binaria válida ingresada por el usuario.

```
1 std::string solicitarCadenaP03() {
2     std::string cadena;
3     bool verificacion;
4     do {
5         std::cout << "Ingrese una cadena (solo 'r' y 'w'): ";
6         std::cin >> cadena;
7
8         verificacion = true; // Inicializamos la verificacion como verdadera
9
10        for (char c : cadena) {
11            if(c != 'r' && c != 'w') {
12                verificacion = false; // Si encontramos un caracter distinto a 'r'
13                    o 'w', marcamos la verificacion como falsa
14            }
15        }
16    } while (!verificacion);
17
18    return cadena;
19 }
```

```
13         std::cout << "La cadena ingresada contiene caracteres que no son
14             'r' ni 'w'." << std::endl;
15         std::cout << "Intentelo nuevamente." << std::endl;
16         break; // Salimos del bucle
17     }
18
19 } while(!verificacion); // Si la verificacion es falsa, continuamos
20     solicitando la cadena
21
22 }
```

Listing 20: Archivo: *solicitudesP03.cpp* — Funcion **solicitarCadenaP03**

### GenerarCadenaP03

La función **generarCadenaP03** genera una cadena binaria aleatoria compuesta únicamente por los caracteres *r* y *w*. Si se proporciona una cadena original como argumento, la función utiliza su longitud para determinar la longitud de la cadena aleatoria generada. En caso contrario, genera una longitud aleatoria entre 5 y 15 caracteres.

La función utiliza un generador de números aleatorios con una semilla basada en el tiempo actual y la librería `<random>` de C++.

Se crea una distribución uniforme para generar aleatoriamente *r* o *w*.

Luego, se genera cada carácter de la cadena de manera aleatoria, utilizando la distribución uniforme. Finalmente, se asegura de que el último carácter de la cadena sea *w* si se generó una cadena nueva o *r* si se proporcionó una cadena original.

La función imprime la longitud y la cadena generada antes de devolverla como un objeto de tipo `std::string`.

La función recibe una cadena binaria original opcional como argumento y devuelve una cadena binaria aleatoria válida.

```
1 std::string generarCadenaP03(std:: string cadenaOriginal) {
2     // Semilla para el generador de numeros aleatorios
3     std::random_device rd;
4     std::mt19937 gen(rd());
5     srand(time(NULL));
6
7     std::string cadena;
8
9     // Distribucion para generar 'r' o 'w' aleatoriamente
10    std::uniform_int_distribution<> dis(0, 1);
11
12    int longitud = (cadenaOriginal.empty()) ? rand() % (15 - 5 + 1) + 5 :
13        cadenaOriginal.size();
14
15    for (int i = 0; i < longitud; ++i) {
16        cadena += (dis(gen) == 0) ? 'r' : 'w';
17    }
18    cadena[cadena.size()-1] = (cadenaOriginal.empty()) ? 'w' : 'r';
19    std::cout << "La cadena generada fue de longitud: " << longitud << " denotada como:
20        " << cadena << std::endl;
21
22 }
```

Listing 21: Archivo: *solicitudesP03.cpp* — Funcion **GenerarCadenaP03**

### guardarJugadasEnArchivo

La función **guardarJugadasEnArchivo** recibe como argumentos un vector de vectores de enteros **rutas** que contiene las jugadas encontradas, el nombre del archivo donde se guardarán las rutas en forma-

to de texto `nombreArchivo`, el nombre del archivo donde se guardarán las rutas en formato CSV `nombreArchivoCSV`, y la cadena binaria para la cual se encontraron las jugadas `cadena`.

La función abre dos archivos, uno para guardar las jugadas en formato de texto y otro para guardarlas en formato CSV. Luego, verifica si los archivos se abrieron correctamente. Si es así, comprueba si el vector de rutas no está vacío. Si hay jugadas encontradas, escribe el encabezado en el archivo de texto y luego cada ruta en el formato adecuado, tanto en el archivo de texto como en el archivo CSV. Finalmente, imprime un mensaje indicando que se han guardado las rutas en ambos archivos. Si no se encontraron rutas para la cadena proporcionada, se imprime un mensaje correspondiente.

La función cierra ambos archivos después de escribir las rutas.

Si no se pueden abrir los archivos, se muestra un mensaje de error.

La función no devuelve ningún valor.

```
1 void guardarJugadasEnArchivo(const std::vector<std::vector<int>>&rutas, const
2   std::string&nombreArchivo, const std::string&nombreArchivoCSV, std::string
3   cadena) {
4   std::ofstream archivo(nombreArchivo);
5   std::ofstream archivoCSV(nombreArchivoCSV);
6
7   if (archivo.is_open()&&archivoCSV.is_open()) {
8     if (!rutas.empty()) {
9       archivo << "Se encontraron las siguientes jugadas para la cadea
10      <<"<<cadena<<">>:\n";
11     for(auto&ruta : rutas) {
12       archivo << "Ruta: ";
13       for(int i = ruta.size() - 1; i >= 0; --i) {
14         archivo << 'Q' << ruta[i];
15         archivoCSV << ruta[i]+1;
16         if(i != 0) {
17           archivo << " -> ";
18           archivoCSV << ',';
19         }
20       }
21       archivo << '\n';
22       archivoCSV << '\n';
23     }
24     std::cout << "Se han guardado las rutas en el archivo
25      "<<nombreArchivo<<".txt'"<<std::endl;
26     std::cout << "Se han guardado las rutas en el archivo
27      "<<nombreArchivoCSV<<".csv'"<<std::endl;
28   } else {
29     std::cout << "No se encontraron rutas para la cadena:
30      "<<cadena<<std::endl;
31     archivo << "No se encontraron rutas para la cadena:
32      "<<cadena<<std::endl;
33   }
34 }
```

Listing 22: Archivo: *solicitudesP03.cpp* — Función `guardarJugadasEnArchivo`

### opcionAnimarP03

La función `opcionAnimarP03` solicita al usuario si desea graficar el resultado de las jugadas. Recibe como argumento una referencia a un entero `error` que se utiliza para manejar errores dentro del programa.

La función muestra un mensaje preguntando al usuario si desea graficar el resultado, permitiendo que el usuario responda con *Y* o *N* (mayúscula o minúscula).

Si el usuario responde afirmativamente (*Y o y*), se llama a la función `animarP03` para mostrar la animación del resultado.

La función no devuelve ningún valor.

```
1 void opcionAnimarP03(int&error) {
2     char opcion = 'Y';
3     std::cout << "Desea Graficar el resultado(Y[y]/N[n]): ";
4     std::cin >> opcion;
5     if (opcion=='Y' || opcion=='y')
6         animarP03(error);
7 }
```

Listing 23: Archivo: *solicitudesP03.cpp* — Funcion `opcionAnimarP03`

### 5.2.5. *automataP03.cpp*

`crearCasilla`

La función `crearCasilla` genera un conjunto de Casillas que representan los estados del tablero de ajedrez. Cada Casilla tiene un identificador único y una serie de transiciones que representan los movimientos permitidos desde ese estado.

Las transiciones están definidas como pares de símbolos y destinos, donde el símbolo representa el tipo de movimiento (*r* para movimiento hacia el color rojo y *w* para movimiento hacia casillas de color blancas) y el destino es el identificador de la Casilla a la que se puede llegar con ese movimiento.

El tablero está representado por un grafo, donde cada Casilla es un nodo y las transiciones entre Casillas son las aristas del grafo.

La función retorna un vector de Casillas que representa el tablero completo.

Esta función no recibe argumentos y devuelve un vector de Casillas que representa el tablero de ajedrez con sus respectivas transiciones.

```
1 //Funcion para generar el tablero y definir sus transiciones
2 std::vector<Casilla> crearCasilla() {
3     Casilla CasillaQ0 = { 0, { { 'r', 1}, { 'r', 4}, { 'w', 5} } }; //
4     Casilla CasillaQ1 = { 1, { { 'r', 4}, { 'r', 6}, { 'w', 0}, { 'w', 2}, { 'w', 5} } };
5     Casilla CasillaQ2 = { 2, { { 'r', 1}, { 'r', 3}, { 'r', 6}, { 'w', 5}, { 'w', 7} } };
6     Casilla CasillaQ3 = { 3, { { 'r', 6}, { 'w', 2}, { 'w', 7} } };//
7     Casilla CasillaQ4 = { 4, { { 'r', 1}, { 'r', 9}, { 'w', 0}, { 'w', 5}, { 'w', 8} } };
8     Casilla CasillaQ5 = { 5, { { 'r', 1}, { 'r', 4}, { 'r', 6}, { 'r', 9}, { 'w', 0},
9                           { 'w', 2}, { 'w', 8}, { 'w', 10} } };//
10    Casilla CasillaQ6 = { 6, { { 'r', 1}, { 'r', 3}, { 'r', 9}, { 'r', 11}, { 'w', 2},
11                           { 'w', 5}, { 'w', 7}, { 'w', 10} } };//
12    Casilla CasillaQ7 = { 7, { { 'r', 3}, { 'r', 6}, { 'r', 11}, { 'w', 2}, { 'w', 10} } };
13    Casilla CasillaQ8 = { 8, { { 'r', 4}, { 'r', 9}, { 'r', 12}, { 'w', 5}, { 'w', 13} } };//
14    Casilla CasillaQ9 = { 9, { { 'r', 4}, { 'r', 6}, { 'r', 12}, { 'r', 14}, { 'w', 5},
15                           { 'w', 8}, { 'w', 10}, { 'w', 13} } };//
16    Casilla CasillaQ10 = { 10, { { 'r', 6}, { 'r', 9}, { 'r', 11}, { 'r', 14}, { 'w', 5},
17                           { 'w', 7}, { 'w', 13}, { 'w', 15} } };//
18    Casilla CasillaQ11 = { 11, { { 'r', 6}, { 'r', 14}, { 'w', 7}, { 'w', 10}, { 'w', 15} } };//
19    Casilla CasillaQ12 = { 12, { { 'r', 9}, { 'w', 8}, { 'w', 13} } };//
20    Casilla CasillaQ13 = { 13, { { 'r', 9}, { 'r', 12}, { 'r', 14}, { 'w', 8}, { 'w', 10} } };//
21    Casilla CasillaQ14 = { 14, { { 'r', 9}, { 'r', 11}, { 'w', 10}, { 'w', 13}, { 'w', 15} } };//
22    Casilla CasillaQ15 = { 15, { { 'r', 11}, { 'r', 14}, { 'w', 10} } };//
23
24    std::vector<Casilla> grafo = {CasillaQ0, CasillaQ1, CasillaQ2, CasillaQ3,
25                                   CasillaQ4, CasillaQ5, CasillaQ6,
```

```
21     CasillaQ07, CasillaQ08, CasillaQ09, CasillaQ10, CasillaQ11, CasillaQ12,
22     CasillaQ13, CasillaQ14, CasillaQ15};  
23  
24 }  
return grafo;
```

Listing 24: Archivo: *automataP03.cpp* — Funcion `crearCasilla`

#### crearTablero

La función `crearTablero` genera un autómata que representa el tablero de ajedrez con las transiciones adecuadas para cada jugador. Recibe como argumento un entero `numJugador` que indica para qué jugador se está creando el tablero.

Dentro de la función, se crea una estructura `Tablero` llamada `nuevo`. Se inicializa el grafo del tablero con las Casillas correspondientes utilizando la función `crearCasilla`. Luego, dependiendo del jugador especificado, se establece la Casilla inicial y final del tablero.

Si `numJugador` es igual a 0, significa que se está creando el tablero para el primer jugador, por lo que la Casilla inicial es la 0 y la final es la 15. Si `numJugador` es igual a 1, significa que se está creando el tablero para el segundo jugador, por lo que la Casilla inicial es la 3 y la final es la 12.

Finalmente, la función retorna el autómata `nuevo`, que contiene el grafo del tablero con las Casillas y las transiciones correspondientes.

Esta función recibe como argumento un entero `numJugador` y devuelve una estructura `Tablero` que representa el tablero de ajedrez con las transiciones adecuadas para el jugador especificado.

```
1 //Funcion para generar el automata y definir sus transiciones
2 Tablero crearTablero(int numJugador) {
3     Tablero nuevo;
4     nuevo.grafo = crearCasilla();
5     if(numJugador == 0) {
6         nuevo.casillaInicial = 0;
7         nuevo.casillaFinal = 15;
8     } else if(numJugador == 1) {
9         nuevo.casillaInicial = 3;
10        nuevo.casillaFinal = 12;
11    }
12
13    return nuevo;
14 }
```

Listing 25: Archivo: *automataP03.cpp* — Funcion `crearTablero`

#### 5.2.6. *jugadasP03.cpp*

La función `encontrarJugadas` busca todas las posibles jugadas que llevan al estado final del tablero de ajedrez, siguiendo las transiciones definidas en el tablero y basándose en una cadena dada.

Recibe como argumentos:

- Una referencia constante al tablero (`const Tablero&tablero`) que representa el tablero de ajedrez.
- Una referencia constante a una cadena (`const std::string&cadena`) que representa la secuencia de movimientos.
- Un entero `i` que indica la posición actual en la cadena.
- Un entero `posicion` que indica la casilla actual en el tablero.
- Un vector de enteros `jugada_actual` que representa la jugada actual en el tablero.

Dentro de la función, se inicializa un vector de vectores de enteros `jugadas` para almacenar todas las jugadas encontradas.

Luego, se comprueba si hemos alcanzado el final de la cadena (`i >= cadena.size()`). Si es así, se verifica si la posición actual en el tablero es la casilla final. Si lo es, se agrega la jugada actual al vector de jugadas.

Después, se recorren todas las transiciones desde la casilla actual en el tablero. Si el símbolo en la posición actual de la cadena coincide con el símbolo de la transición, se realiza una llamada recursiva para explorar esa transición. Se actualiza la jugada actual y se agregan las nuevas jugadas encontradas al vector de jugadas.

Finalmente, se retorna el vector de jugadas que contiene todas las jugadas posibles que llevan al estado final del tablero.

Esta función es responsable de encontrar todas las jugadas posibles para alcanzar el estado final del tablero de ajedrez, dado un tablero y una cadena de movimientos.

```
1 #include "PRACTICA03.h"
2
3 std::vector<std::vector<int>> encontrarJugadas(const Tablero&tablero, const
4   std::string&cadena, int i, int posicion, std::vector<int> jugada_actual){
5   std::vector<std::vector<int>> jugadas;
6
7   if(i >= cadena.size()){// Si hemos alcanzado el final de la cadena
8     if(tablero.casillaFinal == posicion) {
9       jugada_actual.push_back(posicion);
10      jugadas.push_back(jugada_actual);
11    }
12    return jugadas;
13  }
14
15  for(auto transicion : tablero.grafo[posicion].transiciones){
16    if(cadena[i] == transicion.first){
17      // Llamada recursiva para explorar la transicion
18      auto nuevas_jugadas = encontrarJugadas(tablero, cadena, i+1,
19        transicion.second, jugada_actual);
20      for(auto&jugada : nuevas_jugadas) {
21        jugada.push_back(posicion);
22        jugadas.push_back(jugada);
23      }
24    }
25  }
26}
```

Listing 26: Archivo: *jugadasP03.cpp* — Funcion encontrarJugadas

### 5.2.7. animarP03.cpp

La función `animarP03` se encarga de ejecutar un script de MATLAB que genera un gráfico del autómata y una animación de las jugadas realizadas en el tablero de ajedrez.

Recibe como argumento una referencia a un entero `error`, que se utiliza para indicar si ocurrió algún error durante la ejecución de los comandos en MATLAB.

En primer lugar, se define la ruta del script MATLAB que genera el gráfico del autómata y se ejecuta utilizando la función `system`.

Después de la ejecución, se verifica si la ejecución fue exitosa o si ocurrió algún error.

A continuación, se espera un tiempo adicional de 30 segundos antes de iniciar la animación de MATLAB. Esto se logra utilizando la función `Sleep` de la biblioteca `windows.h`.

Luego, se define la ruta del script MATLAB que genera la animación de las jugadas en el tablero de ajedrez y se ejecuta de manera similar al primer script.

Nuevamente, se verifica si la ejecución fue exitosa o si ocurrió algún error.

Finalmente, se espera un tiempo adicional de 50 segundos antes de iniciar la animación de MATLAB.

Esta función es responsable de iniciar la animación de las jugadas en el tablero de ajedrez y de mostrar cualquier error que pueda ocurrir durante el proceso.

```
1 #include "PRACTICA03.h"
2
3 void animarP03(int&error) {
4     std::string ruta = "MATLAB -r run('GraficoAutomata.m')";
5
6     error = system(ruta.c_str());
7
8     // Verificar si la ejecucion fue exitosa
9     if (error == 0) {
10         std::cout << "La ruta se ejecuto correctamente." << std::endl;
11     } else {
12         std::cerr << "Error al ejecutar la ruta." << std::endl;
13     }
14
15     // Esperar un tiempo adicional
16     std::cout << "Esperando 30 segundos antes de iniciar MATLAB..." << std::endl;
17     Sleep(30000); // Espera 10000 milisegundos (equivalente a 20 segundos)
18
19     ruta = "MATLAB -r run('animacionP03.m')";
20
21     error = system(ruta.c_str());
22
23     // Verificar si la ejecucion fue exitosa
24     if (error == 0) {
25         std::cout << "La ruta se ejecuto correctamente." << std::endl;
26     } else {
27         std::cerr << "Error al ejecutar la ruta." << std::endl;
28     }
29
30     // Esperar un tiempo adicional
31     std::cout << "Esperando 50 segundos antes de iniciar la animacion de
32         MATLAB..." << std::endl;
33     Sleep(50000); // Espera 10000 milisegundos (equivalente a 20 segundos)
34 }
```

Listing 27: Archivo: *animarP03.cpp* — Funcion animarP03

### 5.2.8. *leercsv.m*

La función *leer\_csv* en MATLAB se encarga de leer datos desde un archivo CSV y almacenarlos en una matriz. Para manejar grandes conjuntos de datos de manera eficiente, esta función lee el archivo en bloques.

Primero, abre el archivo especificado para lectura. Luego, inicializa una matriz vacía para almacenar los datos leídos. Utiliza un bucle *while* para leer los datos en bloques hasta que se alcance el final del archivo.

Dentro del bucle, utiliza la función *textscan* para leer un bloque de datos del archivo, especificando el formato de los datos y el delimitador (en este caso, una coma). Después de leer un bloque, combina los datos leídos en una matriz y continúa leyendo el siguiente bloque hasta que se hayan leído todos los datos.

Finalmente, cierra el archivo y devuelve la matriz de datos leídos.

```
1 function matriz = leer_csv(nombre_archivo)
2     % Tamanio del bloque
3     block_size = 1e6; % Por ejemplo, lee el archivo en bloques de 1 millon de
4     % tuplas
5
6     % Abrir el archivo para lectura
7     fileID = fopen(nombre_archivo, 'r');
8
9     if fileID == -1
10         error('No se pudo abrir el archivo.');
11     end
```

```
12 % Inicializar matriz para almacenar los datos
13 matriz = [];
14
15 % Leer los datos en bloques
16 while ~feof(fileID)
17     % Leer un bloque de datos del archivo
18     data = textscan(fileID, '%d%d', 'block_size', 'Delimiter', ',',
19                      'HeaderLines', 1);
20     if isempty(data{1})
21         break; % No hay mas datos para leer, salir del bucle
22     end
23
24     % Crear la matriz de salida combinando la primera columna (chars_cell) y
25     % la segunda columna (data{2})
26     matriz = [matriz; [num2cell(data{1}), num2cell(data{2})]];
27 end
28
29 % Cerrar el archivo
30 fclose(fileID);
```

Listing 28: Archivo: *leercsv* — Funcion

### 5.2.9. *GraficoAutomata.m*

Este script de MATLAB crea una representación gráfica de un autómata finito no determinista (AFND) a partir de un archivo CSV que contiene las transiciones entre sus estados. Utiliza círculos para representar los estados y líneas para las transiciones entre estos estados.

En primer lugar, el script lee el archivo CSV que contiene las transiciones del autómata y define los estados y las transiciones correspondientes. Luego, dibuja los círculos que representan los estados y las líneas que representan las transiciones entre estos estados.

El script también incluye ajustes visuales para el título del gráfico y los ejes, así como configuraciones para desactivar la cuadricula. Finalmente, cierra la figura después de un breve intervalo de tiempo.

```
1 clear
2 clc
3 close all;
4
5 %% Definiciones
6
7 %Leemos el CSV de las transiciones
8 data = readtable("TablaEstados.csv");
9 cabeceras = data.Properties.VariableNames;
10 % Definimos el radio para los estados
11 r = 2;
12
13 % Define los estados como un arreglo de matrices
14 estados = [
15     [ 0, 0]; % Estado 0
16     [10, 0]; % Estado 01
17     [20, 0]; % Estado 02
18     [30, 0]; % Estado 03
19     [ 0, 10]; % Estado 04
20     [10, 10]; % Estado 05
21     [20, 10]; % Estado 06
22     [30, 10]; % Estado 07
23     [ 0, 20]; % Estado 08
24     [10, 20]; % Estado 09
25     [20, 20]; % Estado 10
26     [30, 20]; % Estado 11
27     [ 0, 30]; % Estado 12
28     [10, 30]; % Estado 13
29     [20, 30]; % Estado 14
30     [30, 30]; % Estado 15
31 ];
32
33 % Dibujar el escenario
34 figure;
```

```
35 hold on;
36
37 %% Dibujar los circulos y las lineas de conexion al estado 0
38 for i = 1:size(estados, 1)
39     centro = estados(i,:);
40     rectangle('Position', [centro(1)-r, centro(2)-r, 2*r, 2*r], 'Curvature', [1,
41         1], 'FaceColor', 'w', 'EdgeColor', 'k', 'LineWidth', 2);
42     if ismember(i, [13, 16])
43         rectangle('Position', [centro(1)-r+0.8, centro(2)-r+0.8, 1.2*r, 1.2*r],
44             'Curvature', [1, 1], 'FaceColor', 'w', 'EdgeColor', 'k', 'LineWidth',
45             1);
46     end
47     text(centro(1) - r/3.5, centro(2), sprintf('%02d', i), 'FontName', 'Times New
48         Roman', 'FontSize', 12, 'FontAngle', 'italic', 'Color', 'k');
49
50     %obtener las transiciones
51     filaTransiciones = data{i, 2:end};
52     for j = 1 : numel(filaTransiciones)
53         if ~isnan(filaTransiciones(j))
54             punto0 = centro;
55             punto1 = estados(filaTransiciones(j) + 1, :);
56             dis = sqrt((punto1(1)-punto0(1)+r)^2 + (punto1(2)-punto0(2)-r)^2 );
57             x1 = punto0(1) + (punto1(1) - punto0(1)) * r/dis;
58             y2 = punto0(2) + (punto1(2) - punto0(2)) * r/dis;
59             % Calcular el punto medio entre los puntos inicial y final de la linea
60             medio_x = (centro(1) - r + punto1(1)) / 2;
61             medio_y = (centro(2) + punto1(2)) / 2;
62
63             % Obtener la cabecera correspondiente
64             cabecera = cabeceras{j + 1};
65
66             % Dibujar la conexion al estadoDestino
67             plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'k', 'LineWidth',
68                 2, 'Color', [0, 0, 0, 1], 'DisplayName', 'Ruta');
69             text(medio_x+1, medio_y+1, cabecera, 'Color', 'black',
70                 'HorizontalAlignment', 'center', 'VerticalAlignment', 'middle');
71         end
72     end
73 end
74
75 %% Ajustes de los ejes y fondo del ploteo
76 axis equal;% Ajusta la relacion de aspecto para que los circulos se vean como
77 % círculos
78 axis off;
79
80 grid off; % Activa la cuadricula para una mejor visualizacion
81
82 %% Ajustes del titulo del ploteo
83
84 [titulo,subtitulo] = title('Automata Practica 03','Automata Finito No
85     Determinista', 'FontName', 'Times New Roman');
86 titulo.FontName = 'Times New Roman';
87 subtitle.FontName = 'Times New Roman';
88 titulo.FontSize = 16;
89 titulo.FontWeight = "bold";
90 subtitle.FontAngle = 'italic';
91
92 hold off;
93 %% Paso 4: Terminar el programa
94 pause(5);
95 close all
96 exit();
```

Listing 29: Archivo: *GraficoAutomata* — Script

### 5.2.10. animacionP03.m

Este script de MATLAB crea una animación de un juego entre dos jugadores en un tablero cuadrado de tamaño  $4 \times 4$ . Utiliza imágenes de fichas para representar a cada jugador y lee las jugadas de ambos

desde archivos CSV. La animación muestra las fichas moviéndose por el tablero siguiendo las jugadas registradas en los archivos CSV.

En primer lugar, el script lee los archivos CSV que contienen las jugadas de cada jugador y carga las imágenes de las fichas de ambos jugadores. Luego, define las dimensiones del tablero y crea una matriz que representa las posiciones en el tablero.

En el siguiente paso, el script dibuja el tablero y coloca las fichas de los jugadores en las posiciones correspondientes a sus jugadas. La animación avanza mostrando cómo se mueven las fichas en cada jugada.

Finalmente, el script pausa la animación durante unos segundos antes de cerrar la figura y finalizar el proceso.

```
1 clc;
2 clear;
3 close all;
4
5 %% Paso 1: Obtener la informacion
6 %Leer CSV de las jugadas del jugador uno y del jugador 2
7 J01 = readmatrix("J01.csv");
8 J02 = readmatrix("J02.csv");
9
10 % Leer la imagen
11 FichaJ01 = imread("C:\Users\emicr\Pictures\stikers vscode\Mythrai_1.png");
12
13 FichaJ02 = imread("C:\Users\emicr\Pictures\stikers vscode\pyra1_1.png");
14
15 %imfinfo("C:\Users\emicr\Pictures\stikers vscode\Mythrai_1.png")
16 %imfinfo("C:\Users\emicr\Pictures\stikers vscode\pyra1_1.png")
17
18 % Define el canal alfa y normalizamos al rango [0,1]
19 %canalAlfaFichaJ01 = double(FichaJ01(:, :, 1:4)) / 255;
20
21 % Corregir la orientacion de las imagenes
22 FichaJ01 = flipud(FichaJ01);
23 FichaJ02 = flipud(FichaJ02);
24
25 % Obtener la cantidad de jugadas (columnas)
26 longitudJugada = size(J01, 2);
27
28 % Tamanio del Tablero nxn
29 n = 4;
30
31 % Numero de jugada a graficar (fila de la matriz)
32 k1 = 1 ;
33 k2 = 1;
34
35 %Matriz de las coordenadas de las casillas del tablero
36 ruta = [4, 1; 4, 2; 4, 3; 4, 4; 3, 1; 3, 2; 3, 3; 3, 4; 2, 1; 2, 2; 2, 3; 2, 4; 1,
1; 1,2; 1, 3; 1, 4];
37
38 %% Paso 2: Dibujar y animar los elementos del escenario:
39 figure;
40 for t = 1 : longitudJugada
41     clf;
42     axis equal off;
43     % Ajustes del titulo del ploteo
44     [titulo,subtitulo] = title('Animacion Practica 03','Universo de Jugadas
        Ganadoras para dos jugadores', 'FontName','Times New Roman');
45     titulo.FontName = 'Times New Roman';
46     subtitle.FontName = 'Times New Roman';
47     titulo.FontSize = 16;
48     titulo.FontWeight = "bold";
49     subtitle.FontAngle = 'italic';
50
51     % Dibujar los cuadrados del tablero
52     for i = 1:n
53         for j = 1:n
54             if mod(i+j,2) == 0 % Si la suma de fila y columna es par, color rojo
55                 color = [0, 0, 0];
56             else % Si la suma de fila y columna es impar, color negro
57                 color = [1, 1, 1];
58             end
59             rectangle('Position',[j,i], 'FaceColor',color);
60         end
61     end
62
63     % Dibujar las fichas
64     if k1 <= longitudJugada
65         % Jugador 1 (Blanco)
66         FichaJ01 = im2double(FichaJ01);
67         FichaJ01 = FichaJ01(:,:,1:4);
68         FichaJ01 = FichaJ01 / 255;
69         FichaJ01 = flipud(FichaJ01);
70         FichaJ01 = im2uint8(FichaJ01);
71         FichaJ01 = im2colormap(FichaJ01, 'white');
72         FichaJ01 = im2uint8(FichaJ01);
73         FichaJ01 = im2colormap(FichaJ01, 'white');
74         FichaJ01 = im2colormap(FichaJ01, 'white');
75         FichaJ01 = im2colormap(FichaJ01, 'white');
76         FichaJ01 = im2colormap(FichaJ01, 'white');
77         FichaJ01 = im2colormap(FichaJ01, 'white');
78         FichaJ01 = im2colormap(FichaJ01, 'white');
79         FichaJ01 = im2colormap(FichaJ01, 'white');
80         FichaJ01 = im2colormap(FichaJ01, 'white');
81         FichaJ01 = im2colormap(FichaJ01, 'white');
82         FichaJ01 = im2colormap(FichaJ01, 'white');
83         FichaJ01 = im2colormap(FichaJ01, 'white');
84         FichaJ01 = im2colormap(FichaJ01, 'white');
85         FichaJ01 = im2colormap(FichaJ01, 'white');
86         FichaJ01 = im2colormap(FichaJ01, 'white');
87         FichaJ01 = im2colormap(FichaJ01, 'white');
88         FichaJ01 = im2colormap(FichaJ01, 'white');
89         FichaJ01 = im2colormap(FichaJ01, 'white');
90         FichaJ01 = im2colormap(FichaJ01, 'white');
91         FichaJ01 = im2colormap(FichaJ01, 'white');
92         FichaJ01 = im2colormap(FichaJ01, 'white');
93         FichaJ01 = im2colormap(FichaJ01, 'white');
94         FichaJ01 = im2colormap(FichaJ01, 'white');
95         FichaJ01 = im2colormap(FichaJ01, 'white');
96         FichaJ01 = im2colormap(FichaJ01, 'white');
97         FichaJ01 = im2colormap(FichaJ01, 'white');
98         FichaJ01 = im2colormap(FichaJ01, 'white');
99         FichaJ01 = im2colormap(FichaJ01, 'white');
100        FichaJ01 = im2colormap(FichaJ01, 'white');
101        FichaJ01 = im2colormap(FichaJ01, 'white');
102        FichaJ01 = im2colormap(FichaJ01, 'white');
103        FichaJ01 = im2colormap(FichaJ01, 'white');
104        FichaJ01 = im2colormap(FichaJ01, 'white');
105        FichaJ01 = im2colormap(FichaJ01, 'white');
106        FichaJ01 = im2colormap(FichaJ01, 'white');
107        FichaJ01 = im2colormap(FichaJ01, 'white');
108        FichaJ01 = im2colormap(FichaJ01, 'white');
109        FichaJ01 = im2colormap(FichaJ01, 'white');
110        FichaJ01 = im2colormap(FichaJ01, 'white');
111        FichaJ01 = im2colormap(FichaJ01, 'white');
112        FichaJ01 = im2colormap(FichaJ01, 'white');
113        FichaJ01 = im2colormap(FichaJ01, 'white');
114        FichaJ01 = im2colormap(FichaJ01, 'white');
115        FichaJ01 = im2colormap(FichaJ01, 'white');
116        FichaJ01 = im2colormap(FichaJ01, 'white');
117        FichaJ01 = im2colormap(FichaJ01, 'white');
118        FichaJ01 = im2colormap(FichaJ01, 'white');
119        FichaJ01 = im2colormap(FichaJ01, 'white');
120        FichaJ01 = im2colormap(FichaJ01, 'white');
121        FichaJ01 = im2colormap(FichaJ01, 'white');
122        FichaJ01 = im2colormap(FichaJ01, 'white');
123        FichaJ01 = im2colormap(FichaJ01, 'white');
124        FichaJ01 = im2colormap(FichaJ01, 'white');
125        FichaJ01 = im2colormap(FichaJ01, 'white');
126        FichaJ01 = im2colormap(FichaJ01, 'white');
127        FichaJ01 = im2colormap(FichaJ01, 'white');
128        FichaJ01 = im2colormap(FichaJ01, 'white');
129        FichaJ01 = im2colormap(FichaJ01, 'white');
130        FichaJ01 = im2colormap(FichaJ01, 'white');
131        FichaJ01 = im2colormap(FichaJ01, 'white');
132        FichaJ01 = im2colormap(FichaJ01, 'white');
133        FichaJ01 = im2colormap(FichaJ01, 'white');
134        FichaJ01 = im2colormap(FichaJ01, 'white');
135        FichaJ01 = im2colormap(FichaJ01, 'white');
136        FichaJ01 = im2colormap(FichaJ01, 'white');
137        FichaJ01 = im2colormap(FichaJ01, 'white');
138        FichaJ01 = im2colormap(FichaJ01, 'white');
139        FichaJ01 = im2colormap(FichaJ01, 'white');
140        FichaJ01 = im2colormap(FichaJ01, 'white');
141        FichaJ01 = im2colormap(FichaJ01, 'white');
142        FichaJ01 = im2colormap(FichaJ01, 'white');
143        FichaJ01 = im2colormap(FichaJ01, 'white');
144        FichaJ01 = im2colormap(FichaJ01, 'white');
145        FichaJ01 = im2colormap(FichaJ01, 'white');
146        FichaJ01 = im2colormap(FichaJ01, 'white');
147        FichaJ01 = im2colormap(FichaJ01, 'white');
148        FichaJ01 = im2colormap(FichaJ01, 'white');
149        FichaJ01 = im2colormap(FichaJ01, 'white');
150        FichaJ01 = im2colormap(FichaJ01, 'white');
151        FichaJ01 = im2colormap(FichaJ01, 'white');
152        FichaJ01 = im2colormap(FichaJ01, 'white');
153        FichaJ01 = im2colormap(FichaJ01, 'white');
154        FichaJ01 = im2colormap(FichaJ01, 'white');
155        FichaJ01 = im2colormap(FichaJ01, 'white');
156        FichaJ01 = im2colormap(FichaJ01, 'white');
157        FichaJ01 = im2colormap(FichaJ01, 'white');
158        FichaJ01 = im2colormap(FichaJ01, 'white');
159        FichaJ01 = im2colormap(FichaJ01, 'white');
160        FichaJ01 = im2colormap(FichaJ01, 'white');
161        FichaJ01 = im2colormap(FichaJ01, 'white');
162        FichaJ01 = im2colormap(FichaJ01, 'white');
163        FichaJ01 = im2colormap(FichaJ01, 'white');
164        FichaJ01 = im2colormap(FichaJ01, 'white');
165        FichaJ01 = im2colormap(FichaJ01, 'white');
166        FichaJ01 = im2colormap(FichaJ01, 'white');
167        FichaJ01 = im2colormap(FichaJ01, 'white');
168        FichaJ01 = im2colormap(FichaJ01, 'white');
169        FichaJ01 = im2colormap(FichaJ01, 'white');
170        FichaJ01 = im2colormap(FichaJ01, 'white');
171        FichaJ01 = im2colormap(FichaJ01, 'white');
172        FichaJ01 = im2colormap(FichaJ01, 'white');
173        FichaJ01 = im2colormap(FichaJ01, 'white');
174        FichaJ01 = im2colormap(FichaJ01, 'white');
175        FichaJ01 = im2colormap(FichaJ01, 'white');
176        FichaJ01 = im2colormap(FichaJ01, 'white');
177        FichaJ01 = im2colormap(FichaJ01, 'white');
178        FichaJ01 = im2colormap(FichaJ01, 'white');
179        FichaJ01 = im2colormap(FichaJ01, 'white');
180        FichaJ01 = im2colormap(FichaJ01, 'white');
181        FichaJ01 = im2colormap(FichaJ01, 'white');
182        FichaJ01 = im2colormap(FichaJ01, 'white');
183        FichaJ01 = im2colormap(FichaJ01, 'white');
184        FichaJ01 = im2colormap(FichaJ01, 'white');
185        FichaJ01 = im2colormap(FichaJ01, 'white');
186        FichaJ01 = im2colormap(FichaJ01, 'white');
187        FichaJ01 = im2colormap(FichaJ01, 'white');
188        FichaJ01 = im2colormap(FichaJ01, 'white');
189        FichaJ01 = im2colormap(FichaJ01, 'white');
190        FichaJ01 = im2colormap(FichaJ01, 'white');
191        FichaJ01 = im2colormap(FichaJ01, 'white');
192        FichaJ01 = im2colormap(FichaJ01, 'white');
193        FichaJ01 = im2colormap(FichaJ01, 'white');
194        FichaJ01 = im2colormap(FichaJ01, 'white');
195        FichaJ01 = im2colormap(FichaJ01, 'white');
196        FichaJ01 = im2colormap(FichaJ01, 'white');
197        FichaJ01 = im2colormap(FichaJ01, 'white');
198        FichaJ01 = im2colormap(FichaJ01, 'white');
199        FichaJ01 = im2colormap(FichaJ01, 'white');
200        FichaJ01 = im2colormap(FichaJ01, 'white');
201        FichaJ01 = im2colormap(FichaJ01, 'white');
202        FichaJ01 = im2colormap(FichaJ01, 'white');
203        FichaJ01 = im2colormap(FichaJ01, 'white');
204        FichaJ01 = im2colormap(FichaJ01, 'white');
205        FichaJ01 = im2colormap(FichaJ01, 'white');
206        FichaJ01 = im2colormap(FichaJ01, 'white');
207        FichaJ01 = im2colormap(FichaJ01, 'white');
208        FichaJ01 = im2colormap(FichaJ01, 'white');
209        FichaJ01 = im2colormap(FichaJ01, 'white');
210        FichaJ01 = im2colormap(FichaJ01, 'white');
211        FichaJ01 = im2colormap(FichaJ01, 'white');
212        FichaJ01 = im2colormap(FichaJ01, 'white');
213        FichaJ01 = im2colormap(FichaJ01, 'white');
214        FichaJ01 = im2colormap(FichaJ01, 'white');
215        FichaJ01 = im2colormap(FichaJ01, 'white');
216        FichaJ01 = im2colormap(FichaJ01, 'white');
217        FichaJ01 = im2colormap(FichaJ01, 'white');
218        FichaJ01 = im2colormap(FichaJ01, 'white');
219        FichaJ01 = im2colormap(FichaJ01, 'white');
220        FichaJ01 = im2colormap(FichaJ01, 'white');
221        FichaJ01 = im2colormap(FichaJ01, 'white');
222        FichaJ01 = im2colormap(FichaJ01, 'white');
223        FichaJ01 = im2colormap(FichaJ01, 'white');
224        FichaJ01 = im2colormap(FichaJ01, 'white');
225        FichaJ01 = im2colormap(FichaJ01, 'white');
226        FichaJ01 = im2colormap(FichaJ01, 'white');
227        FichaJ01 = im2colormap(FichaJ01, 'white');
228        FichaJ01 = im2colormap(FichaJ01, 'white');
229        FichaJ01 = im2colormap(FichaJ01, 'white');
230        FichaJ01 = im2colormap(FichaJ01, 'white');
231        FichaJ01 = im2colormap(FichaJ01, 'white');
232        FichaJ01 = im2colormap(FichaJ01, 'white');
233        FichaJ01 = im2colormap(FichaJ01, 'white');
234        FichaJ01 = im2colormap(FichaJ01, 'white');
235        FichaJ01 = im2colormap(FichaJ01, 'white');
236        FichaJ01 = im2colormap(FichaJ01, 'white');
237        FichaJ01 = im2colormap(FichaJ01, 'white');
238        FichaJ01 = im2colormap(FichaJ01, 'white');
239        FichaJ01 = im2colormap(FichaJ01, 'white');
240        FichaJ01 = im2colormap(FichaJ01, 'white');
241        FichaJ01 = im2colormap(FichaJ01, 'white');
242        FichaJ01 = im2colormap(FichaJ01, 'white');
243        FichaJ01 = im2colormap(FichaJ01, 'white');
244        FichaJ01 = im2colormap(FichaJ01, 'white');
245        FichaJ01 = im2colormap(FichaJ01, 'white');
246        FichaJ01 = im2colormap(FichaJ01, 'white');
247        FichaJ01 = im2colormap(FichaJ01, 'white');
248        FichaJ01 = im2colormap(FichaJ01, 'white');
249        FichaJ01 = im2colormap(FichaJ01, 'white');
250        FichaJ01 = im2colormap(FichaJ01, 'white');
251        FichaJ01 = im2colormap(FichaJ01, 'white');
252        FichaJ01 = im2colormap(FichaJ01, 'white');
253        FichaJ01 = im2colormap(FichaJ01, 'white');
254        FichaJ01 = im2colormap(FichaJ01, 'white');
255        FichaJ01 = im2colormap(FichaJ01, 'white');
256        FichaJ01 = im2colormap(FichaJ01, 'white');
```

```
57         color = [1, 1, 1];
58     end
59     rectangle('Position', [j-1, i-1, 1, 1], 'FaceColor', color);
60 end
61
62 % Colocar las Etiquetas de filas y columnas
63 for i = 1:n
64     text(-0.25, i-0.5, char('a' + i - 1), 'HorizontalAlignment', 'Center',
65           'VerticalAlignment', 'Middle');
66     text(i-0.5, -0.25, num2str(i), 'HorizontalAlignment', 'Center',
67           'VerticalAlignment', 'Middle');
68 end
69
70 % Obtener la posicion actual del jugador 1 y mostrar su ficha
71 hold on
72 posicionActualJ01 = ruta(J01(k1, t), :);
73 image([posicionActualJ01(2)-1, posicionActualJ01(2)], [posicionActualJ01(1)-1,
74         posicionActualJ01(1)], FichaJ01);
75 hold off
76
77 % Obtener la posicion actual del jugador 2 y mostrar su ficha
78 posicionActualJ02 = ruta(J02(k2, t), :);
79 % Esperar un corto tiempo antes de mostrar la ficha del jugador 2
80 pause(0.5);
81 hold on
82 image([posicionActualJ02(2)-1, posicionActualJ02(2)], [posicionActualJ02(1)-1,
83         posicionActualJ02(1)], FichaJ02);
84 hold off
85
86 %Forzar a matlab a dibujar la figura justo en ese momento
87 %drawnow;
88 pause(0.5)
89
90 % Avanzar al siguiente movimiento de cada jugador
91 if t < longitudJugada
92     while true
93         if ((ruta(J01(k2, t), 1) == ruta(J02(k2, t+1), 1)&& ruta(J01(k2, t), 2)
94             == ruta(J02(k2, t+1), 2)) || ...
95             (ruta(J01(k2, t+1), 1) == ruta(J02(k2, t), 1)&& ruta(J01(k2, t+1),
96                 2) == ruta(J02(k2, t), 2)) )&&...
97             (ruta(J01(k2, t+1), 1) == ruta(J02(k2, t+1), 1)&& ruta(J01(k2, t+1),
98                 2) == ruta(J02(k2, t+1), 2))
99         if randi([0,1])
100             k1 = k1+1;
101         else
102             k2 = k2+1;
103         end
104     else
105         break; % Sal del bucle si las rutas son diferentes
106     end
107 end
108
109 pause (10);
110 close all;
111 pause (5);
112 exit();
```

Listing 30: Archivo: *animacionP03* — Script

### 5.3. Practica04: Buscador de palabras.

#### 5.3.1. Conversion de NFA a DFA

A continuacion se dejan las tablas de estados usadas para el proceso de conversion del automata que describe el NFA a un DFA:

ESTADOS/ALFABETO	E	S	C	U	L	A	T	D	I	N	R	O	F	M	Z	EstadoFinal
0	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	
1				2												
2				3												
3						4										
4					5											
5							6									
6								7								
7																*
8									9							
9										10						
10											11					
11												12				
12													13			
13														14		
14															15	
15																16
16																17
17																18
18																
19																*
20																
21																
22																23
23																
24																
25																*
26																
27																28
28																
29																30
30																
31																*
32																
33																
34																
35																
36																
37																*
38																
39																
40																
41																
42																
43																*

Cuadro 1: Tabla de estados del automata NFA.

#	Subconjuntos	E	S	C	U	L	A	T	D	I	N	R	O	F	M	Z	ESPACIO
0	0	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
1	{0,1,8}	{0,1,8}	{0,2,9}	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
2	{0,2,9}	{0,1,8}	0	{0,3}	0	0	0	{0,10}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
3	{0,3}	{0,1,8}	0	{0,31}	{0,4}	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
4	{0,4}	{0,5}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
5	{0,5}	{0,1,8}	0	{0,31}	0	{0,6}	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
6	{0,6}	{0,1,8}	0	{0,31}	0	0	0	{0,7}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
7	{0,7}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
8	{0,10}	{0,1,8}	0	{0,31}	{0,11}	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
9	{0,11}	{0,1,8}	0	{0,31}	0	0	0	0	{0,12}	0	0	{0,19,25}	0	0	{0,37}	0	0
10	{0,12}	{0,1,8}	0	{0,31}	0	0	0	0	0	{0,13}	0	{0,19,25}	0	0	{0,37}	0	0
11	{0,13}	{0,1,8}	0	{0,31}	0	0	{0,14}	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
12	{0,14}	{0,1,8}	0	{0,31}	0	0	0	0	0	{0,15}	0	{0,19,25}	0	0	{0,37}	0	0
13	{0,15}	{0,1,8}	0	{0,31}	0	0	0	{0,16}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
14	{0,16}	{0,1,8,17}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
15	{0,1,8,17}	{0,1,8}	{0,2,9,18}	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
16	{0,2,9,18}	{0,1,8}	0	{0,3}	0	0	0	{0,10}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
17	{0,19,25}	{0,20}	0	{0,31}	0	0	0	0	0	{0,26}	0	{0,19,25}	0	0	{0,37}	0	0
18	{0,20}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	{0,21}	0	{0,19,25}	0	0	{0,37}	0
19	{0,21}	{0,1,8}	0	{0,22}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
20	{0,22}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	{0,23}	0	{0,37}	0	0
21	{0,23}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,24,25}	0	0	{0,37}	0	0
22	{0,19,24,25}	{0,20}	0	{0,31}	0	0	0	0	0	{0,26}	0	{0,19,25}	0	0	{0,37}	0	0
23	{0,26}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,27}	{0,37}	0
24	{0,27}	{0,1,8}	0	{0,31}	0	{0,28}	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
25	{0,28}	{0,1,8,29}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
26	{0,1,8,29}	{0,1,8}	{0,2,9,30}	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
27	{0,2,9,30}	{0,1,8}	0	{0,3}	0	0	0	{0,10}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
28	{0,31}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,32}	0	0	{0,37}	0	0
29	{0,32}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
30	{0,33}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,34}	0	0
31	{0,34}	{0,35}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
32	{0,35}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,36}	{0,19,25}	0	{0,37}	0	0
33	{0,36}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
34	{0,37}	{0,1,8}	0	{0,31}	0	0	{0,38}	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
35	{0,38}	{0,1,8}	0	{0,31}	0	0	0	{0,39}	0	0	0	{0,19,25}	0	0	{0,37}	0	0
36	{0,39}	{0,1,8}	0	{0,31}	0	0	{0,40}	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
37	{0,40}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,41}	{0,19,25}	0	0	{0,37}	0
38	{0,41}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	{0,42}	0
39	{0,42}	{0,1,8}	0	{0,31}	0	0	{0,43}	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0
40	{0,43}	{0,1,8}	0	{0,31}	0	0	0	0	0	0	0	{0,19,25}	0	0	{0,37}	0	0

Cuadro 2: Tabla de Conversion NFA a DFA

### 5.3.2. PRACTICA04.h

El archivo PRACTICA04.h contiene definiciones de estructuras y funciones para el programa de búsqueda de palabras en un autómata.

Las estructuras definidas incluyen EstadoP04, que representa un estado en el autómata, y AutomataP04, que representa el autómata completo con su grafo de estados. También se define la estructura Palabra para almacenar palabras encontradas junto con sus coordenadas en un archivo de texto, y ListaPalabras para mantener una lista de estas palabras.

Las funciones incluyen la ejecución del programa, la animación de resultados utilizando MATLAB, la búsqueda de palabras reservadas en el autómata, la escritura de palabras en un archivo de texto, la medición del tiempo de ejecución, la lectura de un archivo de texto para extraer palabras y sus posiciones, la lectura de una tabla de estados del autómata desde un archivo CSV, la generación de cadenas binarias aleatorias, la solicitud de cadenas al usuario, la creación del autómata y definición de sus transiciones, la creación de estados del autómata, y el menú principal del programa.

```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <string>
5 #include <sstream>
6 #include <time.h>
7 #include <chrono>
8 #include <random>
9 #include <windows.h>
10 #include <unordered_set>
11 #include <regex>

12 // Definicion de la estructura de los EstadoP04s
13 struct EstadoP04 {
14     int id;
15     std::vector<std::pair<char, int>> transiciones; // Transiciones: (simbolo,
16     EstadoP04_destino)
17 };
18
19 // Definicion de la estructura del automata
20 struct AutomataP04 {
21     std::vector<EstadoP04> grafo;
22     int EstadoP04Inicial;
23     std::vector<int> EstadoP04Final;
24 };
25
26 //ListaPalabras
27 struct Palabra{
28     std::string cadena;
29     std::pair<int, int> coordenada;
30 };
31
32 struct ListaPalabras{
33     std::vector<Palabra> lista;
34 };
35
36 //Funcion para realizar la animacion
37 void EjecutarMatlabP04(std::string&, int&);
38
39 //Funcion para solicitar al usuario la animacion
40 void opcionAnimarP04(std::string&, int&);
41
42 // Funcion para encontrar todas las Jugadas al EstadoP04 final que terminan en
43 // "01" usando DFS
44 ListaPalabras encontrarPalabrasReservadas(const AutomataP04&, const
45     ListaPalabras&);

46 // Funcion para guardar las Jugadas en un archivo de texto
47 void guardarPREnArchivo(ListaPalabras, const std::string&, std::string&);

48 //Funcion para medir el tiempo
49 void medirTiempoP04(std::chrono::high_resolution_clock::time_point&);

50
51 //Funcion para ejecutar la secuencia de funciones del programa
```

```
52 void ejecucionP04(std::string, AutomataP04, int&);  
53 //Funcion para almacenar la URL en un archivo  
54 void guardarURL(std::string&URL, std::string&archivoURL);  
55 //Funcion para leer un archivo de texto y devolver cada palabra dentro del texto  
56 //junto con su posicion en x y y  
57 ListaPalabras leerArchivo(const std::string&nombreArchivo);  
58 //Funcion para leer y guardar la tabla de estados del automata  
59 std::vector<std::vector<int>> leerCSV(std::string);  
60 //Funcion para generar una cadena binaria aleatoria de tamano aleatorio (entre  
61 //uno y cien)  
62 std::string generarCadenaP04(std::string);  
63 //Funcion para solicitar al usuario la cadena  
64 std::string solicitarCadenaP04();  
65 //Funcion para ingresar la opcion:  
66 int ingresarOpcionP04();  
67 //Funcion para generar el automata y definir sus transiciones  
68 AutomataP04 crearAutomataP04(std::string);  
69 //Funcion para crear los EstadoP04s del AutomataP04  
70 std::vector<EstadoP04> crearEstadoP04(std::vector<std::vector<int>>);  
71 //Funcion menu  
72 void menuP04(int&);  
73 //Funcion main  
74 int main();
```

Listing 31: Archivo: *PRACTICA03.h*

### 5.3.3. *mainP04.cpp*

En este archivo existe únicamente una función, el siguiente fragmento de código inicia la ejecución del programa, llama a una función `menu` para presentar al usuario un menú de opciones, y devuelve un valor de error al sistema operativo al finalizar la ejecución. La lógica específica del programa, incluyendo la implementación de la función `menu`, se encuentra en otros archivos que están incluidos a través del archivo de encabezado «*PRACTICA04.h*». A continuación se muestra la función `main` en cuestión:

```
1 #include "PRACTICA04.h"  
2  
3 int main(){  
4     int error = 0;  
5     menuP04(error);  
6     return error;  
7 }
```

Listing 32: Archivo: *PRACTICA03.h*

### 5.3.4. *menuP04.cpp*

#### *menuP04*

La función `menuP04` presenta un menú interactivo que permite al usuario elegir entre varias opciones. Primero, se muestra un mensaje de bienvenida y se enumeran las opciones disponibles. El usuario puede seleccionar una opción ingresando un número. Dependiendo de la opción seleccionada, se puede ingresar una URL, ejecutar el programa o salir del mismo. La función utiliza un bucle `while` para permitir que el usuario realice múltiples acciones hasta que decida salir del programa.

```
1 void menuP04(int&error) {  
2     bool repetir = true;
```

```
3 int opcion = 1;
4 std::string rutaCSV = "TablaEstadosP04.csv";
5 std::string URL = "";
6 AutomataP04 automataP04 = crearAutomataP04(rutaCSV);
7
8 while(repetir) {
9     std::cout << "BIENVENIDO A LA PRACTICA 03." << std::endl;
10    std::cout << "OPCIONES A REALIZAR" << std::endl;
11    std::cout << "1- Ingresar URL." << std::endl;
12    std::cout << "2- Ejecutar Programa." << std::endl;
13    std::cout << "0- Salir del Programa." << std::endl;
14
15    opcion = ingresarOpcionP04();
16
17    switch (opcion) {
18        case 0:
19            std::cout << "Gracias Por usar el programa..." << std::endl;
20            repetir = false;
21            break;
22        case 1:
23            URL = solicitarCadenaP04();
24            break;
25        case 2:
26            ejecucionP04(URL, automataP04, error);
27            break;
28        default:
29            std::cout << "Favor de indicar una opcion valida...." << std::endl;
30    }
31}
32}
```

Listing 33: Archivo: *menuP04.cpp* — Funcion menuP04

#### ingresarOpcionP04

Esta función `ingresarOpcionP01` proporciona una forma de interactuar con el usuario, permitiéndole seleccionar una opción deseada del menú mediante la entrada de un número entero en la consola. El valor ingresado por el usuario se utiliza posteriormente en otras partes del programa para determinar la acción a realizar. A continuación se anexa el bloqued de codigo que alude a la función mencionada:

```
1 int ingresarOpcionP04() {
2     int opcion= 5;
3     std::cout << "Ingrese la opcion deseada: ";
4     std::cin >> opcion;
5
6     return opcion;
7 }
```

Listing 34: Archivo: *menuP04.cpp* — Funcion ingresarOpcionP04

#### ejecucionP04

La función `ejecucionP04` lleva a cabo la ejecución principal del programa. Primero, verifica si la URL no está vacía y, en caso afirmativo, guarda la URL en un archivo específico. Luego, ejecuta un script de MATLAB para leer la página web asociada a la URL y extraer su texto.

A continuación, se inicia un cronómetro para medir el tiempo de ejecución del programa. Se lee el archivo de texto generado a partir de la página web para obtener las palabras contenidas en él. Estas palabras se utilizan para encontrar las palabras reservadas definidas en el autómata, utilizando la función `encontrarPalabrasReservadas`.

Posteriormente, se detiene el cronómetro y se muestra el tiempo transcurrido. Las palabras reservadas encontradas se guardan en un archivo específico. Finalmente, se ejecuta un script de MATLAB para visualizar el autómata asociado al proceso.

Esta función encapsula la lógica principal del programa, desde la obtención de datos hasta la generación de resultados y visualización de los mismos.

```
1 void ejecucionP04(std::string URL, AutomataP04 automataP04, int&error){  
2     //Generar el archivo que guardara la URL  
3     std::string archivoURL = "URL.txt";  
4     std::string rutaLeerPW = "MATLAB -r run('leerPagWebs.m')";  
5     if(!URL.empty()){  
6         guardarURL(URL, archivoURL);  
7         EjecutarMatlabP04(rutaLeerPW, error);  
8     }  
9  
10    std::string rutaAWeb = "textoWEB.txt";  
11  
12    // Iniciar el cronometro  
13    auto start = std::chrono::high_resolution_clock::now();  
14  
15    //Obtener las palabras dentro del archivo de la pagina web  
16    ListaPalabras paginaWeb = leerArchivo(rutaAWeb);  
17  
18    //Generar el vector de jugadas posibles al estado final para la cadena  
19    auto palabras = encontrarPalabrasReservadas(automataP04, paginaWeb);  
20    // Detener el cronometro y mostrar el tiempo transcurrido  
21    medirTiempoP04(start);  
22  
23    //Guardamos las 'Palabras Reservadas' (PR) en un archivo  
24    guardarPREnArchivo(palabras, "palabrasEncontradas.txt", URL);  
25  
26    std::string rutaAutomata = "MATLAB -r run('GraficoAutomata.m')";  
27    //opcionAnimarP04(rutaAutomata, error);  
28}
```

Listing 35: Archivo: *menuP04.cpp* — Funcion ejecucionP04

#### medirTiempoP04

Esta función **medirTiempo** calcula el tiempo transcurrido entre un punto de inicio y un punto de finalización utilizando la librería `<chrono>` de C++. Aquí está el bloque de código de la función:

```
1 void medirTiempoP04(std::chrono::high_resolution_clock::time_point&start) {  
2     auto stop = std::chrono::high_resolution_clock::now();  
3     auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(stop -  
4         start);  
5     auto milliseconds = duration.count() % 1000;  
6     auto seconds = (duration.count() / 1000) % 60;  
7     auto minutes = (duration.count() / (1000 * 60)) % 60;  
8  
8     std::cout << "Tiempo transcurrido: " << minutes << " minutos, " << seconds <<  
9         " segundos, " << milliseconds << " milisegundos" << std::endl;  
9 }
```

Listing 36: Archivo: *menuP04.cpp* — Funcion medirTiempoP04

#### 5.3.5. *solicitudesP04.cpp*

##### solicitarCadenaP04

La función **solicitarCadenaP04** solicita al usuario que ingrese una cadena, en este caso una URL. Luego, verifica si la cadena ingresada cumple con el formato de una URL válida utilizando expresiones regulares.

Si la cadena no cumple con el formato de una URL válida, se muestra un mensaje de error y se solicita al usuario que ingrese nuevamente la cadena hasta que se proporcione una URL válida.

Una vez que se ingresa una URL válida, esta se devuelve como resultado de la función. Esta función garantiza que solo se acepten URLs válidas como entrada para el programa.

```
1 std::string solicitarCadenaP04() {
2     std::string cadena;
3     bool verificacion;
4     do {
5         std::cout << "Ingrese una cadena (URL): ";
6         std::cin >> cadena;
7
8         // Expresion regular para validar URL
9         std::regex url_regex("(http|https)://.*");
10
11         verificacion = std::regex_match(cadena, url_regex);
12
13         if (!verificacion) {
14             std::cout << "La cadena ingresada no es una URL valida." << std::endl;
15             std::cout << "Intentelo nuevamente." << std::endl;
16         }
17
18     } while(!verificacion);
19
20     return cadena;
21 }
```

Listing 37: Archivo: *solicitudesP04.cpp* — Funcion **solicitarCadenaP04**

### leerCSV

La función `leerCSV` lee un archivo CSV y devuelve los datos en forma de una matriz bidimensional de enteros.

Primero, la función intenta abrir el archivo CSV especificado por `nombreArchivo`. Si no puede abrir el archivo, muestra un mensaje de error y devuelve una matriz vacía.

Luego, la función lee cada línea del archivo. Para cada línea, separa los valores utilizando la coma como delimitador. Los valores separados se convierten a enteros y se almacenan en un vector. Este vector representa una fila de la matriz de datos.

Finalmente, cada vector de fila se agrega a la matriz bidimensional de datos. Una vez que se leen todas las líneas del archivo, se cierra el archivo y se devuelve la matriz de datos. Si encuentra un valor que no se puede convertir a entero, muestra un mensaje de error pero continúa leyendo el resto del archivo.

```
1 std::vector<std::vector<int>> leerCSV(std::string nombreArchivo) {
2     std::vector<std::vector<int>> datos;
3     std::ifstream archivo(nombreArchivo);
4
5     if (!archivo.is_open()) {
6         std::cerr << "Error al abrir el archivo." << std::endl;
7         return datos;
8     }
9
10    std::string linea;
11    while (std::getline(archivo, linea)) {
12        std::vector<int> fila;
13        std::stringstream ss(linea);
14        std::string valor;
15
16        while (std::getline(ss, valor, ',')) {
17            try {
18                if (!valor.empty())
19                    fila.push_back(std::stoi(valor));
20            } catch (const std::invalid_argument&e) {
21                std::cerr << "Error: Valor no valido en el archivo CSV." <<
22                std::endl;
23            }
24        }
25        datos.push_back(fila);
26    }
27    archivo.close();
28 }
```

```
29     return datos;
30 }
```

Listing 38: Archivo: *solicitudesP04.cpp* — Funcion leerCSV

### guardarURL

La función `guardarURL` guarda una URL en un archivo de texto especificado.

Primero, intenta abrir el archivo especificado por `archivoURL` para escribir en él. Si puede abrir el archivo, escribe la URL en el archivo y luego lo cierra. Muestra un mensaje indicando que la URL se guardó exitosamente en el archivo.

Si no puede abrir el archivo, muestra un mensaje indicando que no se pudo abrir el archivo. No hay retorno explícito de ningún valor ya que la función solo tiene efecto de escritura en el archivo.

```
1 void guardarURL(std::string& URL, std::string& archivoURL) {
2     std::ofstream archivo(archivoURL);
3     if(archivo.is_open()){
4         archivo << URL;
5         archivo.close();
6         std::cout << "El URL se guardo exitosamente en el archivo" << std::endl;
7     }else
8         std::cout << "No se pudo abrir el archivo" << std::endl;
9 }
```

Listing 39: Archivo: *solicitudesP04.cpp* — Funcion guardarURL

### leerArchivo

La función `leerArchivo` lee un archivo de texto y devuelve una lista de palabras junto con sus coordenadas de posición en el archivo.

Primero, intenta abrir el archivo especificado por `nombreArchivo` para leerlo. Si puede abrir el archivo, lee línea por línea y procesa cada línea. Para cada línea, divide la línea en palabras y guarda cada palabra junto con sus coordenadas de posición en la lista de palabras `palabrasEnArchivo`. Las coordenadas de posición consisten en la posición en x y la posición en y de la palabra en el archivo.

Finalmente, cierra el archivo y devuelve la lista de palabras. Si no puede abrir el archivo, muestra un mensaje de error y devuelve una lista vacía de palabras.

```
1 ListaPalabras leerArchivo(const std::string& nombreArchivo) {
2     ListaPalabras palabrasEnArchivo;
3
4     std::ifstream archivo(nombreArchivo);
5     if (!archivo.is_open()) {
6         std::cerr << "No se pudo abrir el archivo " << nombreArchivo << std::endl;
7         return palabrasEnArchivo;
8     }
9
10    std::string linea;
11    int posY = 1;
12    while (std::getline(archivo, linea)) {
13        int posX = 1;
14        std::istringstream iss(linea);
15        std::string palabra;
16        while (iss >> palabra) {
17            palabrasEnArchivo.lista.push_back({palabra, {posX, posY}});
18            posX += palabra.length() + 1; // Suma la longitud de la palabra mas un
19            espacio
20        }
21        posY++; // Avanza a la siguiente linea
22    }
23    archivo.close();
24    return palabrasEnArchivo;
25 }
```

Listing 40: Archivo: *solicitudesP04.cpp* — Funcion leerArchivo

### guardarPREnArchivo

La función `guardarPREnArchivo` guarda las palabras reservadas junto con sus coordenadas de posición en un archivo de texto.

Primero, intenta abrir el archivo especificado por `nombreArchivo` para escribir en él. Si puede abrir el archivo, verifica si la lista de palabras reservadas `palabras` no está vacía. Si hay palabras reservadas, escribe un encabezado en el archivo indicando la URL de donde se trajeron las palabras. Luego, itera sobre cada palabra reservada en la lista y escribe la palabra junto con sus coordenadas de posición en el archivo.

Además, muestra por consola cada palabra reservada junto con sus coordenadas de posición. Finalmente, muestra un mensaje indicando que las palabras reservadas se han guardado en el archivo especificado.

Si la lista de palabras reservadas está vacía, indica que no se encontraron palabras reservadas para la URL especificada y escribe un mensaje correspondiente en el archivo.

Si no puede abrir el archivo, muestra un mensaje de error por consola.

```
1 //PR = Palabra Reservada
2 void guardarPREnArchivo(ListaPalabras palabras, const std::string& nombreArchivo,
3     std::string& URL) {
4     std::ofstream archivo(nombreArchivo);
5     if(archivo.is_open()) {
6         if(!palabras.lista.empty()) {
7             archivo << "Se encontraron las siguientes palabras reservadas (junto
8                 sus coordenadas) dentro de la URL '" << URL << "'<<std::endl;
9             for(auto& palabra : palabras.lista) {
10                 archivo << "Palabra: "<<palabra.cadena;
11                 archivo << '\t'Coordenada en X: '<< palabra.coordenada.first <<
12                     '\t'Coordenada en Y: '<< palabra.coordenada.second;
13                 archivo << std::endl;
14
15                 std::cout << "Palabra: "<<palabra.cadena;
16                 std::cout << '\t'Coordenada en X: '<< palabra.coordenada.first <<
17                     '\t'Coordenada en Y: '<< palabra.coordenada.second;
18                 std::cout << std::endl;
19             }
20             std::cout << "Se han guardado las palabras en el archivo
21                 "<<nombreArchivo<<".txt'"<<std::endl;
22         } else {
23             std::cout << "No se encontraron palabras reservadas para la pagina web
24                 con URL: " << URL << std::endl;
25             archivo << "No se encontraron palabras reservadas para la pagina web
26                 con URL: " << URL << std::endl;
27         }
28     } else
29         std::cout << "No se pudo abrir el archivo" << std::endl;
30 }
```

Listing 41: Archivo: *solicitudesP04.cpp* — Funcion guardarPREnArchivo

### opcionAnimarP04

La función `opcionAnimarP04` permite al usuario elegir si desea graficar el resultado. Primero, muestra un mensaje preguntando al usuario si desea graficar el resultado. Luego, espera que el usuario ingrese una respuesta (*Y* para sí, *N* para no). Si la respuesta es *Y* o *y*, llama a la función `EjecutarMatlabP04` con la ruta proporcionada y pasa el error como referencia. De lo contrario, no hace nada y la animación no se ejecuta.

```
1 void opcionAnimarP04(std::string& ruta, int& error) {
2     char opcion = 'Y';
3     std::cout << "Desea Graficar el resultado(Y[y]/N[n]): ";
4     std::cin >> opcion;
5     if (opcion=='Y' || opcion=='y')
6         EjecutarMatlabP04(ruta, error);
7 }
```

Listing 42: Archivo: *solicitudesP04.cpp* — Funcion opcionAnimarP04

### 5.3.6. *automataP04.cpp*

#### crearEstadoP04

La función `crearEstadoP04` genera el autómata a partir de la tabla de estados proporcionada. Recorre cada fila de la tabla de estados y crea un estado correspondiente en el autómata. Cada estado tiene un identificador y una lista de transiciones que consisten en pares de símbolos y el identificador del estado destino. Finalmente, devuelve el grafo generado que representa el autómata.

```
1 //Funcion para generar el AutomataP04 y definir sus transiciones
2 std::vector<EstadoP04> crearEstadoP04(std::vector<std::vector<int>> TablaEstados) {
3     std::vector<EstadoP04> grafo;
4     for(int i = 0 ; i < TablaEstados.size() ; i++){
5         grafo.push_back({TablaEstados[i][0], {
6             {'e', TablaEstados[i][1]},
7             {'s', TablaEstados[i][2]},
8             {'c', TablaEstados[i][3]},
9             {'u', TablaEstados[i][4]},
10            {'l', TablaEstados[i][5]},
11            {'a', TablaEstados[i][6]},
12            {'t', TablaEstados[i][7]},
13            {'d', TablaEstados[i][8]},
14            {'i', TablaEstados[i][9]},
15            {'n', TablaEstados[i][10]},
16            {'r', TablaEstados[i][11]},
17            {'o', TablaEstados[i][12]},
18            {'f', TablaEstados[i][13]},
19            {'m', TablaEstados[i][14]},
20            {'z', TablaEstados[i][15]},
21            {' ', TablaEstados[i][16]}
22        })
23    });
24 }
25
26     return grafo;
27 }
```

Listing 43: Archivo: *automataP04.cpp* — Funcion crearEstadoP04

#### crearAutomataP04

La función `crearAutomataP04` construye un autómata a partir de un archivo CSV que contiene la definición de los estados y sus transiciones. Utiliza la función `leerCSV` para obtener la tabla de estados desde el archivo especificado por la ruta. Luego, utiliza la función `crearEstadoP04` para generar el grafo del autómata a partir de la tabla de estados. Además, identifica el estado inicial y los estados finales del autómata y los almacena en la estructura `AutomataP04`. Finalmente, devuelve el autómata generado.

```
1 //Funcion para generar el automata y definir sus transiciones
2 AutomataP04 crearAutomataP04(std::string rutaArchivo) {
3     AutomataP04 nuevo;
4     std::vector<std::vector<int>> TablaEstados = leerCSV(rutaArchivo);
5     nuevo.grafo = crearEstadoP04(TablaEstados);
6
7     for(std::vector<int>& fila : TablaEstados){
```

```
8     if(fila[17] == 1){
9         nuevo.EstadoP04Final.push_back(fila[0]);
10        std::cout<< fila[0];
11    }
12 }
13 nuevo.EstadoP04Inicial = TablaEstados [0][0];
14 return nuevo;
15 }
```

Listing 44: Archivo: *automataP04.cpp* — Funcion `crearAutomataP04`

### 5.3.7. *palabrasP04.cpp*

La función `encontrarPalabrasReservadas` busca palabras reservadas en un texto utilizando un autómata finito. Recibe como entrada el autómata `automataP04` y una lista de palabras `palabrasEnArchivo`. Primero, crea un conjunto de estados finales a partir de los estados finales del autómata. Luego, itera sobre cada palabra en la lista de palabras. Para cada palabra, itera sobre sus caracteres y verifica si hay una transición válida desde el estado actual del autómata utilizando los caracteres de la palabra. Si encuentra una transición válida para cada carácter de la palabra, verifica si el estado actual es un estado final. Si lo es, agrega la palabra a la lista de palabras encontradas. Finalmente, devuelve la lista de palabras encontradas.

```
1 #include "PRACTICA04.h"
2 //std::vector<std::pair<std::string, std::pair<int, int>>>
3 ListaPalabras encontrarPalabrasReservadas(const AutomataP04& automataP04, const
4     ListaPalabras& palabrasEnArchivo) {
5     ListaPalabras palabras_encontradas;
6     std::unordered_set<int> estados_finales(automataP04.EstadoP04Final.begin(),
7         automataP04.EstadoP04Final.end());
8     int posicion = automataP04.EstadoP04Inicial;
9     bool transicionEncontrada = false;
10
11    for (const auto& palabra : palabrasEnArchivo.lista) {
12        const std::string& palabraActual = palabra.cadena;
13        for(int i = 0 ; i < palabraActual.size() ; i++){
14            transicionEncontrada = false;
15            for(auto& transicion : automataP04.grafo[posicion].transiciones){
16                if (palabraActual[i] == transicion.first) {
17                    posicion = transicion.second;
18                    transicionEncontrada = true;
19                }
20            }
21            if(!transicionEncontrada){
22                break;
23            }
24        }
25        if(estados_finales.count(posicion)){
26            palabras_encontradas.lista.push_back(palabra);
27        }
28        posicion = automataP04.EstadoP04Inicial;
29    }
30
31    return palabras_encontradas;
32 }
```

Listing 45: Archivo: *PRACTICA03.h* — Funcion `encontrarPalabrasReservadas`

### 5.3.8. *matlabP04.cpp*

La función `EjecutarMatlabP04` ejecuta un script de MATLAB utilizando la ruta proporcionada como argumento. Primero, utiliza la función `system` para ejecutar la ruta como un comando del sistema. Luego, verifica si la ejecución fue exitosa. Si la ejecución devuelve un código de error igual a cero, muestra un mensaje indicando que la ruta se ejecutó correctamente; de lo contrario, muestra un mensaje de error. Después, espera 30 segundos antes de pasar al siguiente paso del programa.

```
1 #include "PRACTICA04.h"
2
3 void EjecutarMatlabP04(std::string& ruta, int& error) {
4     error = system(ruta.c_str());
5
6     // Verificar si la ejecucion fue exitosa
7     if (error == 0) {
8         std::cout << "La ruta se ejecuto correctamente." << std::endl;
9     } else {
10        std::cerr << "Error al ejecutar la ruta." << std::endl;
11    }
12
13    // Esperar un tiempo adicional
14    std::cout << "Esperando 30 segundos antes de pasar al siguiente paso del
15        programa..." << std::endl;
16    Sleep(30000); // Espera 20000 milisegundos (equivalente a 20 segundos)
}
```

Listing 46: Archivo: *matlabP04* — Funcion: *EjecutarMatlabP04*

### 5.3.9. *leerPagWebs.m*

El script **leerPagWebs** comienza limpiando la ventana de comandos, cerrando todas las figuras y limpiando las variables del espacio de trabajo en MATLAB. Luego, especifica la ruta del archivo que contiene la URL de la página web que se va a analizar.

A continuación, lee la URL desde el archivo de texto especificado y luego utiliza la función **webread** para obtener el contenido HTML de esa URL. Posteriormente, extrae el texto visible del contenido HTML utilizando la función **extractHTMLText** y lo convierte a minúsculas.

Después, se especifica la ruta del archivo donde se almacenará el texto extraído de la página web. El script escribe el texto en el archivo destino y muestra un mensaje indicando que el texto se ha extraído y almacenado correctamente.

Finalmente, cierra todas las figuras en MATLAB y finaliza la ejecución del script.

```
1 clc
2 clear
3 close all
4
5 % Especifica la ruta del archivo que contiene la URL
6 archivo_url = "URL.txt";
7
8 % Lee la URL desde el archivo de texto
9 fid = fopen(archivo_url, 'r');
10 url = fgetl(fid);
11 fclose(fid);
12
13 % Lee el contenido HTML desde la URL
14 html_content = webread(url);
15
16 % Extrae el texto visible del contenido HTML
17 text_content = lower(extractHTMLText(html_content));
18
19 % Especifica la ruta del archivo donde deseas almacenar el texto
20 archivo_destino = 'textoWEB.txt';
21
22 % Escribe el texto en el archivo destino
23 fid = fopen(archivo_destino, 'w');
24 fprintf(fid, '%s', text_content);
25 fclose(fid);
26
27 disp('Texto extraido y almacenado con exito.');
```

Listing 47: Archivo: *leerPagWebs* — Script

### 5.3.10. *GraficoAutomata.m*

El script `graficoAutomata` comienza limpiando la ventana de comandos, el espacio de trabajo y cerrando todas las figuras abiertas en MATLAB. Luego, define algunas variables y lee el archivo CSV que contiene las transiciones del autómata.

Después, define los estados del autómata como un arreglo de matrices y dibuja los círculos representando cada estado, así como las líneas de conexión entre los estados y las transiciones. Los círculos se representan mediante la función `rectangle` de MATLAB, y las líneas de conexión se trazan utilizando la función `plot`.

El script también etiqueta cada estado con su respectivo número y muestra las transiciones con diferentes colores según el tipo de transición. Finalmente, ajusta los ejes, el fondo del gráfico y el título del plot antes de finalizar la ejecución del script.

Después de una pausa de 10 segundos para permitir que el usuario observe el gráfico, el script cierra todas las figuras y finaliza la ejecución de MATLAB.

```
1 clear
2 clc
3 close all;
4
5 %% Definiciones
6
7 %Leemos el CSV de las transiciones
8 data = readtable("TablaEstadosP04.csv");
9 cabeceras = data.Properties.VariableNames;
10 % Definimos el radio para los estados
11 r = 2;
12
13 % Define los estados como un arreglo de matrices
14 estados = [
15     [ 0,  0]; % Estado 0
16     [10,  0]; % Estado 01
17     [20, -10]; % Estado 02
18     [25, -5]; % Estado 03
19     [30, -10]; % Estado 04
20     [35, -5]; % Estado 05
21     [40, -10]; % Estado 06
22     [45, -5]; % Estado 07
23     [25, -10]; % Estado 08
24     [30, -15]; % Estado 09
25     [35, -20]; % Estado 10
26     [40, -25]; % Estado 11
27     [45, -30]; % Estado 12
28     [45, -25]; % Estado 13
29     [45, -20]; % Estado 14
30     [45, -15]; % Estado 15
31     [45, -10]; % Estado 16
32     [10,  10]; % Estado 17
33     [20,  20]; % Estado 18
34     [30,  30]; % Estado 19
35     [40,  40]; % Estado 20
36     [50,  30]; % Estado 21
37     [40,  30]; % Estado 22
38     [20,  10]; % Estado 23
39     [30,  10]; % Estado 24
40     [40, 20]; % Estado 25
41     [50, 20]; % Estado 26
42     [50, 10]; % Estado 27
43     [10, -30]; % Estado 28
44     [15, -30]; % Estado 29
45     [20, -30]; % Estado 30
46     [25, -30]; % Estado 31
47     [30, -30]; % Estado 32
48     [35, -30]; % Estado 33
49     [10, -40]; % Estado 34
50     [15, -40]; % Estado 35
51     [20, -40]; % Estado 36
52     [25, -40]; % Estado 37
53     [30, -40]; % Estado 38
```

```
54 [35,-40]; % Estado 39
55 [40,-40] % Estado 40
56 ];
57
58 % Dibujar el escenario
59 figure;
60 hold on;
61
62 %% Dibujar los circulos y las lineas de conexion al estado 0
63 for i = 1:size(estados, 1)
64 centro = estados(i,:);
65 rectangle('Position', [centro(1)-r, centro(2)-r, 2*r, 2*r], 'Curvature', [1,
66 1], 'FaceColor', 'w', 'EdgeColor', 'k', 'LineWidth', 2);
67 if ismember(i, [8, 17, 23, 28, 34, 41])
68 rectangle('Position', [centro(1)-r+0.8, centro(2)-r+0.8, 1.2*r, 1.2*r],
69 'Curvature', [1, 1], 'FaceColor', 'w', 'EdgeColor', 'k', 'LineWidth',
70 1);
71 end
72 text(centro(1) - r/3.5, centro(2), sprintf('%02d', i), 'FontName', 'Times New
73 Roman', 'FontSize', 12, 'FontAngle', 'italic', 'Color', 'k');
74
75 if i ~= 1
76 % Calcular el punto de interseccion entre la linea de conexion y el
77 % ciculo del Estado 0
78 p0 = estados(1,:);
79 p1 = centro;
80 d = sqrt((p1(1) - p0(1))^2 + (p1(2) - p0(2))^2);
81 x = p0(1) + (p1(1) - p0(1)) * r / d;
82 y = p0(2) + (p1(2) - p0(2)) * r / d;
83
84 % Dibujar la conexion al Estado 0
85 if ~ismember(i, [2,18,29,35])
86 %text(centro(1)-r, centro(2), '\Sigma', 'Color', 'red');
87 plot([centro(1)-r, x], [centro(2), y], 'r--', 'LineWidth', 2, 'Color',
88 [1, 0, 0, 0.1], 'DisplayName', '\Sigma');
89 end
90 end
91
92 %obtener las transiciones
93 filaTransiciones = data{i, 2:17};
94 for j = 1 : numel(filaTransiciones)
95 if filaTransiciones(j) ~= 0
96 punto0 = centro;
97 punto1 = estados(filaTransiciones(j) +1, :);
98 dis = sqrt((punto1(1)-punto0(1)+r)^2 + (punto1(2)-punto0(2)-r)^2 );
99 x1 = punto0(1) + (punto1(1) - punto0(1)) * r/dis;
100 y2 = punto0(2) + (punto1(2) - punto0(2)) * r/dis;
101 % Calcular el punto medio entre los puntos inicial y final de la linea
102 medio_x = (centro(1) - r + punto1(1)) / 2;
103 medio_y = (centro(2) + punto1(2)) / 2;
104
105 % Obtener la cabecera correspondiente
106 cabecera = cabeceras{j + 1};
107
108 % Dibujar la conexion al estadoDestino
109 if filaTransiciones(j) == 1&&i~=1
110 plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'm--',
111 'LineWidth', 2, 'Color', [1, 0, 1, 0.2], 'DisplayName', 'E');
112 % Dibujar el texto en el punto medio de la linea
113 %text(medio_x, medio_y+0.5, cabecera, 'Color', [1, 0, 1, 0.1],
114 'HorizontalAlignment', 'center', 'VerticalAlignment',
115 'middle');
116 elseif filaTransiciones(j) == 28&&i~=1
117 plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'c--',
118 'LineWidth', 2, 'Color', [0, 1, 1, 0.1], 'DisplayName', 'C');
119 %text(medio_x, medio_y+0.5, cabecera, 'Color', [0, 1, 1, 0.3],
120 'HorizontalAlignment', 'center', 'VerticalAlignment',
121 'middle');
122 elseif filaTransiciones(j) == 17&&i~=1
123 plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'y--',
124 'LineWidth', 2, 'Color', [1, 1, 0, 0.1], 'DisplayName', 'R');
125 %text(medio_x, medio_y+0.5, cabecera, 'Color', [1, 1, 0, 0.3],
126 'HorizontalAlignment', 'center', 'VerticalAlignment',
```

```
        'middle');
113    elseif filaTransiciones(j) == 34&&i~=1
114        plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'g--',
115            'LineWidth', 2, 'Color', [0, 1, 0, 0.1], 'DisplayName', 'M');
116        %text(medio_x, medio_y+0.5, cabecera, 'Color', [0, 1, 0, 0.3],
117            'HorizontalAlignment', 'center', 'VerticalAlignment',
118            'middle');
119    else
120        plot([centro(1), punto1(1)], [centro(2), punto1(2)], 'b',
121            'LineWidth', 2, 'Color', [0, 0, 1, 1], 'DisplayName', 'Ruta');
122        text(medio_x, medio_y+0.5, cabecera, 'Color', 'blue',
123            'HorizontalAlignment', 'center', 'VerticalAlignment',
124            'middle');
125    end
126 end
127 end
128 legend('\Sigma', 'E', 'R', 'C', 'M');
129 %% Ajustes de los ejes y fondo del ploteo
130 axis equal;% Ajusta la relacion de aspecto para que los circulos se vean como
131 %circulos
132 axis off;
133 grid off; % Activa la cuadricula para una mejor visualizacion
134 %% Ajustes del titulo del ploteo
135 [titulo,subtitulo] = title('Automata Practica 04','Automata FInito Determinista',
136     'FontName','Times New Roman');
137 titulo.FontName = 'Times New Roman';
138 subtitle.FontName = 'Times New Roman';
139 titulo.FontSize = 16;
140 titulo.FontWeight = "bold";
141 subtitle.FontAngle = 'italic';
142 hold off;
143 %% Paso 4: Terminar el programa
144 pause(10);
145 close all
146 exit();
```

Listing 48: Archivo: *GraficoAutomataP04* — Script

## 5.4. Compendio

La función `main` del archivo *Compendio01.cpp* controla la ejecución de diferentes programas de práctica de forma aleatoria, cada uno representado por una opción numerada del 1 al 3. Estas opciones corresponden a los programas 01, 03 y 04 respectivamente.

En resumen, esta función realiza lo siguiente:

1. Inicializa variables y semilla para la generación de números aleatorios.
2. Ejecuta un bucle que itera un número aleatorio de veces, determinado por `iteraciones`.
3. En cada iteración, elige aleatoriamente uno de los tres programas de práctica para ejecutar.
4. Ejecuta el programa correspondiente según la opción elegida.
5. Muestra mensajes informativos sobre la ejecución del programa.
6. Finalmente, muestra un mensaje de agradecimiento y devuelve un posible error al finalizar la ejecución.

Se puede considerar como el controlador principal del programa que organiza y gestiona la ejecución de los distintos programas de práctica, vistos en las subsecciones anteriores, de manera aleatoria.

Definición para el reporte: La función `main` controla la ejecución aleatoria de varios programas de práctica, cada uno representado por una opción numérica. Se encarga de inicializar variables, ejecutar los programas seleccionados y mostrar mensajes informativos sobre el proceso. Es el punto de entrada principal del programa, coordinando la ejecución de las diferentes partes de la práctica de forma dinámica.

```
1 #include "PRACTICA01.h"
2 #include "PRACTICA03.h"
3 #include "PRACTICA04.h"
4
5 int main() {
6     srand(time(NULL));
7     //Variables propias del programa
8     int error = 0;
9     int iteraciones = rand() % (10 - 1 + 1) + 1;
10    int opcion;
11
12    //Variables PRACTICA01
13    int potencia;
14    char s0 = '#';
15    char s1 = '.';
16
17    //Variables PRACTICA03
18    std::string cadena = "";
19    Tablero tablero = crearTablero(0);
20    Tablero tablero2 = crearTablero(1);
21
22    //Variables PRACTICA04
23    std::string rutaCSV = "TablaEstadosP04.csv";
24    AutomataP04 automataP04 = crearAutomataP04(rutaCSV);
25    std::string URL = "";
26    std::string rutaAutomata = "MATLAB -r run('GraficoAutomata.m')";
27
28    std::cout << "EL numero de Iteraciones es: " << iteraciones << std::endl;
29    for(iteraciones; iteraciones > 0 ; iteraciones--) {
30        std::cout << std::endl;
31
32        opcion = rand() % (3 - 1 + 1) + 1;
33
34        std::cout << "La opcion fue el 'PROGRAMA 0" << opcion << "' se procede a
35        ejecutar..." << std::endl;
36        switch(opcion) {
37            case 1:
38                randomPotencia(potencia);
39                ejecucion(potencia, 1, s0, s1);
40                graficar(error);
41                break;
42            case 2:
43                cadena = generarCadenaP03("");
44                ejecucionP03(cadena, tablero, tablero2, error);
45                animarP03(error);
46                break;
47            case 3:
48                ejecucionP04(URL, automataP04, error);
49                EjecutarMatlabP04(rutaAutomata, error);
50                break;
51        }
52        std::cout << "Ejecutado el 'PROGRAMA 0" << opcion << "' quedan: " <<
53        iteraciones-1 << " iteraciones" << std::endl;
54    }
55    std::cout << "#-#-#-#-#-_GRACIASPOR USAR EL PROGRAMA___#-#-#-#-#" ;
56
57    return error;
58 }
```

Listing 49: Archivo: *Compendio01.cpp* — Funcion: `main`

## 6 Resultados

## 6.1. Resultados: Universo de cadenas binarias.

A continuación se dejan los resultados dados por el programa con una potencia igual a veinte y ocho:

```
Windows PowerShell
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingresar la opcion deseada: 1
Ingresar la potencia deseada: 28
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingrrese la opcion deseada: 4
Número de iteraciones realizadas: 536870911
Tiempo transcurrido: 17 minutos, 45 segundos, 376 milisegundos
Desea Graficar el resultado(Y[y]/N[n]): y
La ruta se ejecuto correctamente.
Esperando 18 segundos antes de cerrar MATLAB...
BIENVENIDO A LA PRACTICA 01.
OPCIONES A REALIZAR
1- Ingresar potencia.
2- Dar potencia random.
3- Cambiar caracteres.
4- Ejecutar Programa.
0- Salir del Programa.
Ingresar la opcion deseada:
```

Figura 1: Ejecucion de la Practica 01 en terminal.

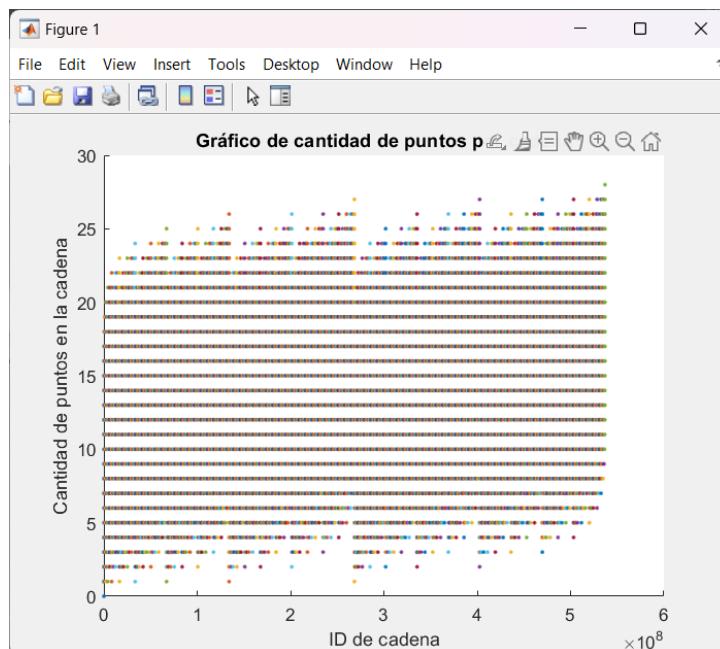


Figura 2: Grafico resultante con potencia = 28

Este es el resultado en el *universo.txt* con una potencia igual a cinco:

## 6.2. Resultados: Tablero de ajedrez

A continuación se dejan los resultados dados por el programa con 20 turnos:

```
Windows PowerShell

Ingrese la opcion deseada: 2
La cadena generade fue de longitud: 17 denotada como: rwwrrwwrrwwrwwrww
BIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar cadena.
2- Generar cadena random.
3- Ejecutar Programa.
0- Salir del Programa.

Ingrese la opcion deseada: 2
La cadena generade fue de longitud: 20 denotada como: wwwrrrwrwrrwrwwrwwrwww
BIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar cadena.
2- Generar cadena random.
3- Ejecutar Programa.
0- Salir del Programa.

Ingrese la opcion deseada: 3
La cadena generade fue de longitud: 20 denotada como: rrwwwwrrwwrrwwrwrwrr
Tiempo transcurrido: 46 minutos, 37 segundos, 485 milisegundos
Se han guardado las rutas en el archivo jugadas.txt.txt'
Se han guardado las rutas en el archivo J01.csv.csv'
Se han guardado las rutas en el archivo jugadas2.txt.txt'
Se han guardado las rutas en el archivo J02.csv.csv'

BIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar cadena.
2- Generar cadena random.
3- Ejecutar Programa.
0- Salir del Programa.

Ingrese la opcion deseada:
```

Figura 3: Ejecucion de la Practica 03 en terminal.

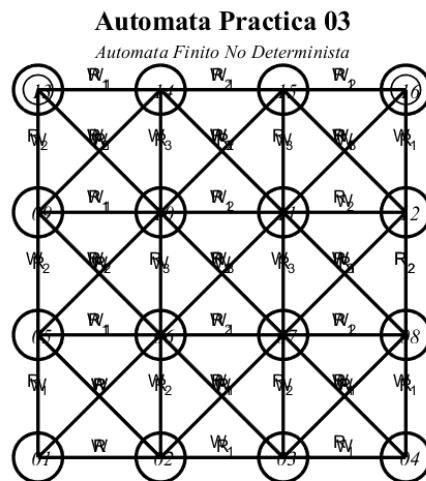


Figura 4: Automata Resultante P04

### Animación Practica 03

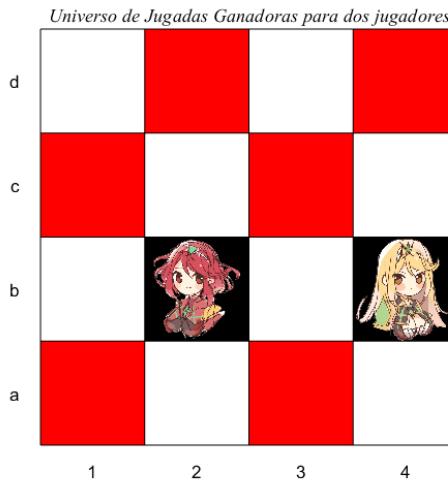


Figura 5: Tablero de ajedrez

Estas son las primeras cincuenta rutas dentro del archivo *jugadas.txt* con una la cadena «wwwwwwrrwwrrwwrrw», el total de rutas es: 253435 (contando desde 1);

```

1 Se encontraron las siguientes jugadas para la cadea <<wwwwwwrrwwrrwwrrw>>:
2 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q0 -> Q5
   -> Q6 -> Q11 -> Q15
3 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q0 -> Q5
   -> Q9 -> Q14 -> Q15
4 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q2
   -> Q6 -> Q11 -> Q15
5 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q8
   -> Q9 -> Q14 -> Q15
6 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q10
   -> Q6 -> Q11 -> Q15
7 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q10
   -> Q9 -> Q14 -> Q15
8 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q10
   -> Q11 -> Q14 -> Q15
9 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q5 -> Q10
   -> Q14 -> Q11 -> Q15
10 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q8 -> Q5
    -> Q6 -> Q11 -> Q15
11 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q8 -> Q5
    -> Q9 -> Q14 -> Q15
12 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q8 -> Q13
    -> Q9 -> Q14 -> Q15
13 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q4 -> Q8 -> Q13
    -> Q14 -> Q11 -> Q15
14 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q2 -> Q5
    -> Q6 -> Q11 -> Q15
15 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q2 -> Q5
    -> Q9 -> Q14 -> Q15
16 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q2 -> Q7
    -> Q6 -> Q11 -> Q15
17 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q2 -> Q7
    -> Q11 -> Q14 -> Q15
18 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q2
    -> Q6 -> Q11 -> Q15
19 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q8
    -> Q9 -> Q14 -> Q15
20 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q10
    -> Q6 -> Q11 -> Q15
21 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q10
    -> Q9 -> Q14 -> Q15
22 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q10
    -> Q11 -> Q14 -> Q15
23 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q5 -> Q10
    -> Q14 -> Q11 -> Q15

```

```
24 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q7 -> Q2  
-> Q6 -> Q11 -> Q15  
25 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q7 -> Q10  
-> Q6 -> Q11 -> Q15  
26 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q7 -> Q10  
-> Q9 -> Q14 -> Q15  
27 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q7 -> Q10  
-> Q11 -> Q14 -> Q15  
28 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q7 -> Q10  
-> Q14 -> Q11 -> Q15  
29 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q5  
-> Q6 -> Q11 -> Q15  
30 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q5  
-> Q9 -> Q14 -> Q15  
31 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q7  
-> Q6 -> Q11 -> Q15  
32 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q7  
-> Q11 -> Q14 -> Q15  
33 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q13  
-> Q9 -> Q14 -> Q15  
34 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q13  
-> Q14 -> Q11 -> Q15  
35 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q15  
-> Q11 -> Q14 -> Q15  
36 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q1 -> Q6 -> Q10 -> Q15  
-> Q14 -> Q11 -> Q15  
37 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q0 -> Q5  
-> Q6 -> Q11 -> Q15  
38 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q0 -> Q5  
-> Q9 -> Q14 -> Q15  
39 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q2 -> Q5  
-> Q6 -> Q11 -> Q15  
40 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q2 -> Q5  
-> Q9 -> Q14 -> Q15  
41 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q2 -> Q7  
-> Q6 -> Q11 -> Q15  
42 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q2 -> Q7  
-> Q11 -> Q14 -> Q15  
43 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q2  
-> Q6 -> Q11 -> Q15  
44 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q8  
-> Q9 -> Q14 -> Q15  
45 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q10  
-> Q6 -> Q11 -> Q15  
46 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q10  
-> Q9 -> Q14 -> Q15  
47 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q10  
-> Q11 -> Q14 -> Q15  
48 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q1 -> Q5 -> Q10  
-> Q14 -> Q11 -> Q15  
49 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q9 -> Q5 -> Q2  
-> Q6 -> Q11 -> Q15  
50 Ruta: Q0 -> Q5 -> Q0 -> Q5 -> Q0 -> Q1 -> Q4 -> Q0 -> Q5 -> Q4 -> Q9 -> Q5 -> Q8  
-> Q9 -> Q14 -> Q15
```

### 6.3. Resultados: Buscador de palabras

La tabla de estados del automata DFA resultante fue:

ESTADOS	E	S	C	U	L	A	T	D	I	N	R	O	F	M	Z		E.F
0	1	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
1	1	2	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
2	1	0	3	0	0	0	8	0	0	0	17	0	0	34	0	0	0
3	1	0	28	4	0	0	0	0	0	0	17	0	0	34	0	0	0
4	5	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
5	1	0	28	0	6	0	0	0	0	0	17	0	0	34	0	0	0
6	1	0	28	0	0	7	0	0	0	0	17	0	0	34	0	0	0
7	1	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	1
8	1	0	28	9	0	0	0	0	0	0	17	0	0	34	0	0	0
9	1	0	28	0	0	0	0	0	10	0	0	17	0	0	34	0	0
10	1	0	28	0	0	0	0	0	11	0	17	0	0	34	0	0	0
11	1	0	28	0	0	12	0	0	0	0	17	0	0	34	0	0	0
12	1	0	28	0	0	0	0	0	0	13	17	0	0	34	0	0	0
13	1	0	28	0	0	0	14	0	0	0	17	0	0	34	0	0	0
14	15	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
15	1	16	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
16	1	0	3	0	0	0	8	0	0	0	17	0	0	34	0	0	1
17	18	0	28	0	0	0	0	0	23	0	17	0	0	34	0	0	0
18	1	0	28	0	0	0	0	0	0	19	17	0	0	34	0	0	0
19	1	0	20	0	0	0	0	0	0	0	17	0	0	34	0	0	0
20	1	0	28	0	0	0	0	0	0	0	17	21	0	34	0	0	0
21	1	0	28	0	0	0	0	0	0	0	22	0	0	34	0	0	0
22	18	0	28	0	0	0	0	0	23	0	17	0	0	34	0	0	1
23	1	0	28	0	0	0	0	0	0	0	17	0	24	34	0	0	0
24	1	0	28	0	25	0	0	0	0	0	17	0	0	34	0	0	0
25	26	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
26	1	27	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
27	1	0	3	0	0	0	8	0	0	0	17	0	0	34	0	0	1
28	1	0	28	0	0	0	0	0	0	0	29	0	0	34	0	0	0
29	1	0	28	0	0	0	0	0	30	0	17	0	0	34	0	0	0
30	1	0	28	0	0	0	0	0	0	0	17	0	0	31	0	0	0
31	32	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	0
32	1	0	28	0	0	0	0	0	0	33	17	0	0	34	0	0	0
33	1	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	1
34	1	0	28	0	0	35	0	0	0	0	17	0	0	34	0	0	0
35	1	0	28	0	0	0	36	0	0	0	17	0	0	34	0	0	0
36	1	0	28	0	0	37	0	0	0	0	17	0	0	34	0	0	0
37	1	0	28	0	0	0	0	0	0	38	17	0	0	34	0	0	0
38	1	0	28	0	0	0	0	0	0	0	17	0	0	34	39	0	0
39	1	0	28	0	0	40	0	0	0	0	17	0	0	34	0	0	0
40	1	0	28	0	0	0	0	0	0	0	17	0	0	34	0	0	1

Cuadro 3: Tabla de Estados del DFA

A continuación se dejan los resultados dados por el programa con el url: «Noticia»:

```

C:\Users\emeric\Documents\G... + v
7162273340BBIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar URL.
2- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada: 1
Ingrese una cadena (URL): https://wwwnoticias.com/nacional/2024/4/2/investigar-morena-por-presunto-financiamiento-del-crimen-sus-campanas-oposicion-633387.
URL guardado exitosamente en el archivo.
BIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar URL.
2- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada: 2
El URL se guardo exitosamente en el archivo
La ruta se ejecuto correctamente.
Esperando 20 segundos antes de pasar al siguiente paso del programa...
Tiempo transcurrido: 0 minutos, 0 segundos, 1 milisegundos
Se han guardado las palabras en el archivo palabrasEncontradas.txt.txt
Desea Graficar el resultado (Y/y)/N(n): y
La grafica se ejecuto correctamente.
Esperando 20 segundos antes de pasar al siguiente paso del programa...
BIENVENIDO A LA PRACTICA 03.
OPCIONES A REALIZAR
1- Ingresar URL.
2- Ejecutar Programa.
0- Salir del Programa.
Ingrese la opcion deseada:

```

Figura 6: Ejecucion de la Practica 04 en terminal.

### Automata Practica 04

Automata FInito Determinista

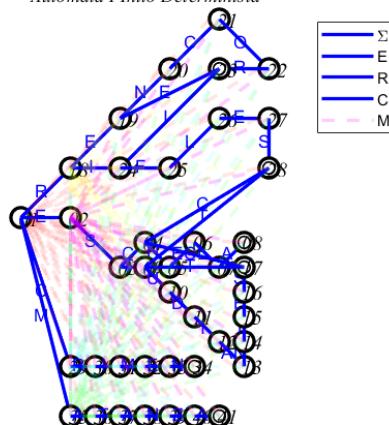


Figura 7: Automata Resultante P04

1	Palabra: crimen Coordenada en X: 287	Coordenada en Y: 11
2	Palabra: crimen Coordenada en X: 63	Coordenada en Y: 29
3	Palabra: crimen Coordenada en X: 52	Coordenada en Y: 33
4	Palabra: crimen Coordenada en X: 209	Coordenada en Y: 35
5	Palabra: crimen Coordenada en X: 32	Coordenada en Y: 41
6	Palabra: crimen , Coordenada en X: 127	Coordenada en Y: 41

## 7 Conclusiones

Durante esta actividad, se desarrollaron tres programas distintos: *Universo*, *Tablero* y *Buscador de Palabras*, cada uno con sus propias características y desafíos. La combinación de lenguajes de programación como C++ y MATLAB permitió abordar eficazmente cada uno de estos problemas, ofreciendo soluciones robustas y eficientes.

Con el programa *Universo*, se exploró la generación y manipulación de cadenas binarias de longitud variable, demostrando la versatilidad y eficiencia de C++ en el manejo de grandes conjuntos de datos. La capacidad de MATLAB para visualizar estos datos facilitó la comprensión y el análisis de los resultados obtenidos.

El programa *Tablero* abordó la simulación de movimientos en un tablero de ajedrez, incorporando reglas de juego y condiciones de victoria. La flexibilidad de C++ y la capacidad de MATLAB para graficar y visualizar datos permitieron crear una solución completa que puede ejecutarse tanto de forma automática como manual.

Por último, el desarrollo del *Buscador de Palabras* se centró en la implementación de un autómata capaz de reconocer un conjunto específico de palabras. Esta aplicación demostró la eficacia de los autómatas en el procesamiento de texto y la utilidad de la combinación de C++ y MATLAB para implementar soluciones complejas.

En conclusión, la combinación de estos lenguajes proporcionó una solución integral para diseñar, implementar y analizar los programas desarrollados en esta actividad, ofreciendo resultados satisfactorios y cumpliendo con los requisitos establecidos.

# Referencias

- [1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introducción a la teoría de autómatas, lenguajes y computación*. Madrid: PEARSON EDUCACIÓN S.A., 2007. Formato: 195 x 250 mm. Páginas: 452.