

INSTITUTO POLITÉCNICO NACIONAL

---

ESCUELA SUPERIOR DE CÓMPUTO

TÍTULO DEL REPORTE

P R O G R A M A  
M Á Q U I N A D E  
T U R I N G

QUE PARA OBTENER UN 10 EN EL REPORTE:

PRESENTA:

CONNOR URBANO MENDOZA

DOCENTE:

JUÁREZ MARTÍNEZ GENARO

INSTITUTO POLITÉCNICO NACIONAL



Estados Unidos Mexicanos

Ciudad de México

2023

# Índice general

<b>Introducción</b>	<b>III</b>
<b>1 Desarrollo</b>	<b>1</b>
1.1 Análisis del problema principal . . . . .	1
1.2 Límites del problema . . . . .	2
1.3 Estrategia para atacar el problema . . . . .	3
1.4 Implementación . . . . .	4
<b>2 Análisis de Resultados</b>	<b>21</b>
2.1 Capturas del programa en ejecución . . . . .	21
<b>3 Conclusión</b>	<b>30</b>
3.1 Complejidades . . . . .	31
<b>4 Bibliografías</b>	<b>32</b>
<b>5 Anexos</b>	<b>33</b>
5.1 Código LATEX de este documento . . . . .	33
5.2 maquina_turing.py . . . . .	33

# Índice de figuras

1 Tabla de estados para el problema. . . . .	IV
----------------------------------------------	----

1.1	Inicio del código, librería random y funciones básicas. . . . .	10
1.2	Función de recorrido de turing. . . . .	14
1.3	Función de turing MG para casos grandes. . . . .	16
1.4	Main del programa. . . . .	20
2.1	Visualización del programa en terminal caso 1. . . . .	22
2.2	Visualización de la primera parte de animaciones. . . . .	23
2.3	Visualización de la segunda parte de animaciones. . . . .	24
2.4	Vista del archivo de salida 'turing.txt'. . . . .	25
2.5	Visualización del programa en terminal caso 2. . . . .	26
2.6	Visualización de memoria utilizada por el archivo para caso 2. . .	27
2.7	Visualización del inicio de archivo de salida turing.txt. . . . .	28
2.8	Visualización del final de archivo de salida turing.txt. . . . .	29

# Introducción

En esta práctica, vamos a trabajar con una máquina de Turing diseñada para reconocer el lenguaje  $0^n 1^n \mid n \geq 1$ , es decir, cadenas formadas por una secuencia de ceros seguida por una secuencia de unos, donde el número de ceros y unos es el mismo y mayor o igual a uno. Esta máquina de Turing se basa en el libro de John Hopcroft, en el ejercicio 8.2 de la segunda edición.

La máquina de Turing que construiremos seguirá un proceso específico para transformar la entrada dada. Comenzando desde el extremo izquierdo de la cinta, la máquina cambiará secuencialmente los ceros por la letra X y se moverá hacia la derecha, pasando por encima de cualquier cero o letra Y que encuentre, hasta que encuentre un uno. En ese momento, cambiará el uno por la letra Y y se moverá hacia la izquierda, pasando sobre cualquier letra Y o cero hasta que encuentre una X. En esta situación, buscará un cero colocado inmediatamente a la derecha y, si lo encuentra, lo cambiará por una X y repetirá el proceso, cambiando el uno correspondiente por una Y.

Si la entrada no cumple con el patrón  $0^n 1^n$ , la máquina de Turing se detendrá sin aceptar la cadena. Sin embargo, si la máquina logra cambiar todos los ceros por X en la misma iteración en la que cambia el último uno por una Y, entonces se determina que la entrada es de la forma  $0^n 1^n$  y la máquina la acepta.

En esta práctica, implementaremos la máquina de Turing en Python, permitiendo que el usuario ingrese una cadena o generando una automáticamente. El programa generará un archivo de texto como salida, que contendrá las descripciones instantáneas en cada paso de la computación. También animaremos la máquina de Turing para cadenas de longitud menor o igual a 10 caracteres.

A continuación, se presenta el código de la máquina de Turing y se proporcio-

na en el reporte, junto con el archivo fuente del código.

Esta práctica se basa en conceptos de teoría de la computación, específicamente en el estudio de gramáticas formales y sus derivaciones. El propósito es aplicar estos conceptos para implementar un programa que pueda procesar y generar cadenas válidas según una gramática específica.

La tabla de transiciones a utilizar en el desarrollo de esta práctica es:

State	Symbol				
	0	1	$X$	$Y$	$B$
$q_0$	$(q_1, X, R)$	—	—	$(q_3, Y, R)$	—
$q_1$	$(q_1, 0, R)$	$(q_2, Y, L)$	—	$(q_1, Y, R)$	—
$q_2$	$(q_2, 0, L)$	—	$(q_0, X, R)$	$(q_2, Y, L)$	—
$q_3$	—	—	—	$(q_3, Y, R)$	$(q_4, B, R)$
$q_4$	—	—	—	—	—

Figura 1: Tabla de estados para el problema.

# Capítulo 1

## Desarrollo

### 1.1. Análisis del problema principal

Al resolver el problema de la máquina de Turing y programarla, se deben considerar los siguientes aspectos:

1. Definición del lenguaje: Es importante comprender claramente el lenguaje que se desea reconocer con la máquina de Turing. En el caso específico mencionado, el lenguaje es  $0^n 1^n \mid n \geq 1$ , lo que significa que debe haber una secuencia de ceros seguida de una secuencia igual de unos.
2. Tabla de transiciones: Se debe crear una tabla de transiciones que defina las reglas para cambiar de un estado a otro en función del símbolo leído en la cinta. La tabla de transiciones debe contener información sobre el estado actual, el símbolo leído, el estado siguiente, el símbolo a escribir en la cinta y la dirección en la que se moverá el cabezal.
3. Implementación de los estados y transiciones: Es necesario implementar los diferentes estados de la máquina de Turing y programar las transiciones correspondientes según las reglas establecidas en la tabla de transiciones. Esto implica manejar correctamente el cambio de estado, la lectura y escritura de símbolos en la cinta, y el movimiento del cabezal.

4. Manejo de la entrada y salida: Se debe tener en cuenta cómo se proporcionará la entrada a la máquina de Turing y cómo se mostrarán los pasos de computación en la salida. Puede ser necesario leer la cadena definida por el usuario o generar una automáticamente, y registrar las descripciones instantáneas en cada paso en un archivo de texto.
5. Validación y aceptación: Durante la ejecución del programa de la máquina de Turing, se deben verificar las condiciones de aceptación o rechazo de una entrada. En el caso mencionado, la máquina de Turing aceptará si todos los ceros se han cambiado por X y el último uno se ha cambiado por Y en la misma iteración. Si la entrada no cumple con estas condiciones, la máquina de Turing se detendrá sin aceptar.

El análisis de estos problemas, facilita la programación de la máquina de Turing y se asegura una solución adecuada para el problema planteado.

## 1.2. Límites del problema

Los límites del problema establecidos por el profesor e implícitamente en esta práctica son los siguientes:

1. Longitud máxima de la cadena: El programa debe ser capaz de recibir una cadena definida por el usuario o generada automáticamente por la máquina, pero la longitud de la cadena no debe exceder los 1000 caracteres. Esto implica que la máquina de Turing debe estar diseñada y programada para manejar cadenas de hasta 1000 caracteres como máximo.
2. Animación con cadenas menores a 10 caracteres: Para facilitar la comprensión y visualización del funcionamiento de la máquina de Turing, se debe animar su ejecución con cadenas que tengan una longitud igual o menor a 10 caracteres. Esto implica que se deberá realizar una representación gráfica o visual de la cinta y el cabezal, mostrando los movimientos y cambios de símbolos durante la ejecución de la máquina.

3. Salida en archivo de texto: La salida del programa debe ser redirigida a un archivo de texto. Es decir, en cada paso de la computación, se debe generar una descripción instantánea que se registrará en un archivo de texto. Esta salida permitirá seguir el proceso de ejecución de la máquina de Turing y analizar su comportamiento.

Estos límites establecidos por el profesor definen las restricciones específicas para la implementación y presentación de la solución del problema de la máquina de Turing.

### 1.3. Estrategia para atacar el problema

A continuación, se describen las estrategias utilizadas para solucionar el problema de la máquina de Turing:

1. El interfaz gráfica: El código utiliza la biblioteca pygame para crear una ventana y dibujar los elementos gráficos necesarios, como los cuadros de la cinta y los botones.
2. Representación de la cinta: La cinta de la máquina de Turing se representa como una lista en el código, donde cada elemento de la lista corresponde a un símbolo en la cinta.
3. Reglas de transición: Las reglas de transición de la máquina de Turing se definen en una lista llamada "rules". Cada regla se representa como una tupla de la forma (estado\_actual, simbolo\_actual) ->(nuevo\_simbolo, dirección, nuevo\_estado).
4. Ejecución de la máquina de Turing: La función turing\_M se encarga de ejecutar la máquina de Turing. Recibe los parámetros iniciales, como el estado inicial, el símbolo blanco, las reglas de transición y la cinta. Luego, sigue las reglas de transición para procesar la cinta hasta alcanzar un estado final.



## 1.4. Implementación

Aquí se explica qué hace cada parte del código:

1. Se importan los módulos necesarios: `random` para generar números aleatorios y `pygame` para la creación de la interfaz gráfica del juego.
2. Se inicializa el módulo `pygame` mediante `pygame.init()` para poder utilizar sus funciones y recursos.
3. Se definen las variables `WIDTH` y `HEIGHT` para el ancho y altura de la ventana del juego.
4. Se crea la ventana del juego con las dimensiones especificadas en `WIDTH` y `HEIGHT` mediante `pygame.display.set_mode((WIDTH, HEIGHT))`. Además, se establece el título de la ventana como 'Maquina de turing' mediante `pygame.display.set_caption('Maquina de turing')`.
5. Se crean dos objetos `Font` de `pygame` para definir el tipo y tamaño de la fuente utilizada en el juego.
6. Se crea un objeto `Clock` de `pygame` llamado `timer` para controlar la velocidad de actualización del juego.
7. Se define la variable `fps` para establecer la cantidad de fotogramas por segundo a los que se actualizará el juego.
8. Se definen las variables `ubicacion`, `turn_step` y `selection`, que se utilizan para almacenar información y controlar los turnos en el juego.

9. Se define la función `dibujar_boton()` que dibuja un botón en la pantalla del juego utilizando la función `pygame.draw.rect()`. El botón se muestra como un rectángulo verde con el texto 'Siguiente' en el centro. La función devuelve un objeto `Rect` de `pygame` que representa el área del botón.
10. Se define la función `mostrar_transiciones(recorrido, estado)` que se encarga de mostrar las transiciones de la Máquina de Turing en la pantalla del juego. La función recibe dos argumentos: `recorrido`, que es una cadena que representa las transiciones realizadas, y `estado`, que es una cadena que representa el estado actual de la Máquina de Turing. La función utiliza varias operaciones de `pygame` para dibujar las transiciones en la pantalla, como dibujar rectángulos, texto y cargar imágenes.
  - a) Esta función se encarga de mostrar las transiciones de la Máquina de Turing en la pantalla del juego.
  - b) Recibe dos argumentos: `recorrido`, que es una cadena que representa las transiciones realizadas, y `estado`, que es una cadena que representa el estado actual de la Máquina de Turing.
  - c) La función recorre cada transición del `recorrido` y las muestra en la pantalla.
  - d) Para cada transición, se dibuja un cuadro en la posición correspondiente en la pantalla y se muestra el valor de la transición en el centro del cuadro.
  - e) Si una transición comienza con `[`, significa que es una transición especial. En este caso, se realizan acciones adicionales, como cambiar el color del cuadro, agregar marcadores o mostrar el estado actual de la Máquina de Turing.
  - f) Para mostrar el estado actual, se utiliza la función `blit()` de `pygame` para pegar el texto del estado en la posición adecuada dentro del cuadro.

- g) Al final de la función, se muestra en la pantalla el conjunto de cuadros con las transiciones y los elementos especiales.
11. Se define la función `dibujar_tablero()` que se encarga de dibujar el tablero de la Máquina de Turing en la pantalla del juego. La función utiliza operaciones de `pygame` para dibujar los cuadros del tablero en la pantalla.
- a) Esta función se encarga de dibujar el tablero de la Máquina de Turing en la pantalla del juego. Utiliza operaciones de `pygame` para dibujar una cuadrícula de cuadros en la pantalla que representan el tablero.
  - b) Cada cuadro se dibuja como un rectángulo con un tamaño determinado y un color específico.
  - c) La función itera sobre los cuadros del tablero y los dibuja en la pantalla.
  - d) Además de dibujar los cuadros, se añade un borde de color negro alrededor de cada cuadro para resaltarlos visualmente.
  - e) Al finalizar la función, se muestra en la pantalla el tablero completo con sus cuadros y bordes.

El fragmento de código que acabamos de explicar pertenece a la figura 1.1. Observar figura 1.1.

```
1 #Teoria de la computacion
2 #Maquina de turing
3 #Alumno: Connor Urbano Mendoza
4 import random
5 import pygame
6
7 pygame.init() #Acceso al paquete pygame
8 #Ancho
9 WIDTH = 1300
10 #Altura
```

```
11 HEIGHT = 700
12 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
13 pygame.display.set_caption('Maquina de turing')
14 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
15 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo de
    fuente 2 del juego
16 timer = pygame.time.Clock()#velocidad de actualizacion de
    nuestro juego a 60 fps
17 fps=60
18
19 #Variables e imagenes del juego
20 ubicacion = ['recuadro']
21
22 #Variables de turnos cambiantes
23 turn_step = 0
24 selection= 100
25 #Cargar imagenes en juego
26 cuadro = pygame.image.load('C:\\Users\\soyco\\OneDrive\\
    Documents\\ESCOM\\sem4\\Teoria\\P2\\turing\\img\\cuadro.
    png')
27 cuadro = pygame.transform.scale(cuadro, (101,101))
28
29 cuadrado = [cuadro]
30
31 lista_piezas = ['recuadro']
32
33 #ver variables/contador flash
34 boton_presionado = False
35
36 def dibujar_boton():
37     boton_width = 150
38     boton_height = 45
39     boton_x = (WIDTH - boton_width) // 2
40     boton_y = (HEIGHT - boton_height - 20)-100
41
42     # Dibujar el boton como un rectangulo en la pantalla
43     boton_rect=pygame.Rect(boton_x, boton_y, boton_width,
        boton_height)
```

```

44     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
45         , boton_width, boton_height))
46     texto = font.render("Siguiente", True, (0, 0, 0))
47     texto_rect = texto.get_rect(center=(boton_x +
48         boton_width // 2, boton_y + boton_height // 2))
49     screen.blit(texto, texto_rect)
50     return boton_rect
51
52 #Funcion para dibujar el contenido de la maquina de turing
53 def mostrar_transiciones(recorrido,estado):
54
55     #recorrido="X,0,0,1,[1],1,1,1,"
56     #estado="q2"
57     cuadro_size = 100 # Tama o de cada cuadro del tablero
58     x = 0
59     y = 255
60
61     font = pygame.font.Font(None, 100) # Fuente y tama o
62     del n mero de casilla
63
64     transiciones = recorrido.split(',') # Obtener las
65     transiciones del recorrido
66
67     for i, transicion in enumerate(transiciones):
68         if i >= 12:
69             break # Salir del bucle si se han mostrado
70             todas las transiciones posibles
71
72         x = x + 100
73         if transicion.startswith('['):
74             # Realizar alguna acci n especial para la
75             transici n que comienza con '['
76             # Por ejemplo, cambiar el color del recuadro o
77             agregar un marcador adicional
78             numero_texto = font.render("^", True, 'yellow')
79             # Crear superficie de texto con la
80             transici n
81             numero_rect = numero_texto.get_rect(center=((x,
82                 y+110))) # Posici n del texto en el
83             centro del recuadro
84             screen.blit(numero_texto, numero_rect) # Pegar

```

```

73         el texto en la pantalla
numero_texto = font.render("|", True, 'yellow')
        # Crear superficie de texto con la
        transici n
74 numero_rect = numero_texto.get_rect(center=((x,
y+120))) # Posici n del texto en el
        centro del recuadro
75 screen.blit(numero_texto, numero_rect) # Pegar
        el texto en la pantalla
76 numero_texto = font.render(estado, True, '
yellow') # Crear superficie de texto con la
        transici n
77 numero_rect = numero_texto.get_rect(center=((x,
y+190))) # Posici n del texto en el
        centro del recuadro
78 screen.blit(numero_texto, numero_rect) # Pegar
        el texto en la pantalla
79 index=lista_piezas.index('recuadro')
80 screen.blit(cuadrado[index], (x-50,y-55))
81
82 numero_texto = font.render(transicion, True, 'black
') # Crear superficie de texto con la
        transici n
83 numero_rect = numero_texto.get_rect(center=((x, y))
) # Posici n del texto en el centro del
        recuadro
84 screen.blit(numero_texto, numero_rect) # Pegar el
        texto en la pantalla
85
86 #Funcion para dibujar tablero
87 def dibujar_tablero():
88     cuadro_size = 100 # Tama o de cada cuadro del tablero
89     tablero_width = 12 * cuadro_size # Ancho total del
        tablero
90     tablero_height = 1 * cuadro_size # Altura total del
        tablero
91     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
        para centrar el tablero
92     tablero_y = ((HEIGHT - tablero_height) // 2 )-200 #
        Posici n Y para centrar el tablero

```

```
93
94     for i in range(12): # Iterar 12 veces para un tablero
95         de 1x12
96         columna = i % 12
97         fila = 1
98         x = tablero_x + columna * cuadro_size
99         y = tablero_y + fila * cuadro_size
100         if fila % 2 == 0:
101             color = 'white' if columna % 2 == 0 else (226,
102                 255, 255)
103         else:
104             color = (226, 255, 255) if columna % 2 == 0
105                 else 'white'
106         pygame.draw.rect(screen, color, [x, y, cuadro_size,
107             cuadro_size])
108         pygame.draw.rect(screen, 'black', [x, y,
109             cuadro_size, cuadro_size], 2) # Agregar borde
110             de color
```

Figura 1.1: Inicio del código, librería random y funciones básicas.

La función `turing_M()` es una implementación de una Máquina de Turing que realiza una simulación y guarda los resultados en un archivo de texto. A continuación, explicaré en detalle cómo funciona y qué hace cada parte del código:

#### 1. Parámetros de la función:

- a) `state`: Representa el estado actual de la Máquina de Turing.
- b) `blank`: Es el símbolo en blanco del alfabeto de la cinta.
- c) `rules`: Es una lista de reglas de transición que define cómo la Máquina de Turing se mueve y cambia de estado.

- d)* tape: Es la cinta de la Máquina de Turing, representada como una lista de símbolos.
- e)* final: Representa el estado final o de aceptación de la Máquina de Turing.
- f)* pos: Es la posición actual del cabezal de lectura/escritura en la cinta.

## 2. Inicialización y validación de parámetros:

- a)* La función asigna el estado inicial *st* con el valor de *state*.
- b)* Si la cinta está vacía (*tape* es una lista vacía), se asigna un símbolo en blanco (*blank*) como único elemento de la cinta.
- c)* Se verifica y corrige la posición *pos* del cabezal de lectura/escritura. Si es negativa, se ajusta al tamaño de la cinta.
- d)* Si la posición está fuera del rango válido de la cinta, se muestra un mensaje de error y se termina el programa con `SystemExit(1)`.

## 3. Preparación de las reglas de transición:

- a)* La función asigna el estado inicial *st* con el valor de *state*. La variable *rules* se transforma de una lista de reglas de transición a un diccionario para facilitar el acceso y búsqueda.
- b)* Cada regla de transición de la forma (*s0*, *v0*, *v1*, *dr*, *s1*) se convierte en una entrada del diccionario donde la clave es una tupla (*s0*, *v0*) y el valor es una tupla (*v1*, *dr*, *s1*).

## 4. Ciclo principal de ejecución:



- a) El bucle principal se ejecuta continuamente hasta que se alcanza el estado final (final) o si no se encuentra una regla de transición para el estado y símbolo actuales.
- b) En cada iteración del bucle, se guarda el estado actual (st) y la configuración de la cinta en un archivo de texto especificado (turing.txt). Cada línea del archivo contiene el estado y los símbolos de la cinta separados por comas. El símbolo en la posición actual del cabezal de lectura/escritura se muestra entre corchetes.
- c) Si se alcanza el estado final, se muestra el mensaje 'Cadena válida' y se rompe el bucle.
- d) Si no se encuentra una regla de transición para el estado y símbolo actuales, se muestra el mensaje 'Cadena inválida' y se rompe el bucle.

5. Ejecución de las reglas de transición:

- a) Si se encuentra una regla de transición para el estado y símbolo actuales, se obtiene la tupla (v1, dr, s1) correspondiente.
- b) Se actualiza el símbolo de la cinta en la posición actual (pos) con el nuevo símbolo v1.
- c) Luego, se mueve el cabezal de lectura/escritura según la dirección indicada por dr. Si es 'left', se mueve hacia la izquierda disminuyendo pos. Si es 'right', se mueve hacia la derecha aumentando pos. Si la posición está fuera del rango de la cinta, se agrega un símbolo en blanco al final o al principio de la cinta, según corresponda.
- d) Luego, se mueve el cabezal de lectura/escritura según la dirección indicada por dr. Si es 'left', se mueve hacia la izquierda disminuyendo pos. Si es 'right', se mueve hacia la derecha aumentando pos. Si la posición está fuera del rango de la cinta, se agrega un símbolo en blanco

al final o al principio de la cinta, según corresponda.

- e) El estado actual (st) se actualiza con el nuevo estado s1 obtenido de la regla de transición.

Observar figura 1.2.

```

1 def turing_M (state = None, #estados de la maquina de
   turing
2         blank = None, #simbolo blanco de el alfabeto
   dela cinta
3         rules = [], #reglas de transicion
4         tape = [], #cinta
5         final = None, #estado valido y/o final
6         pos = 0):#posicion siguiente de la maquina de
   turing
7
8     st = state
9     if not tape: tape = [blank]
10    if pos < 0 : pos += len(tape)
11    if pos >= len(tape) or pos < 0 :
12        print("Se inicializa mal la posicion")
13        SystemExit(1)
14
15    rules = dict(((s0, v0), (v1, dr, s1)) for (s0, v0, v1,
   dr, s1) in rules)
16    """
17        Estado S mbolo le do S mbolo escrito
18        Mov. Estado sig.
19        qn(s0) 1,0,X,Y,B(v0) 1,0,X,Y,B(v1) R o L(
20        dr) qn(s1)
21    """
22    while True:
23        with open('C:\\Users\\soyco\\OneDrive\\Documents\\
   ESCOM\\sem4\\Teoria\\P2\\turing\\output\\turing.
   txt', 'a') as archivo:
24            archivo.write(st+ '\n')
25            #print (st, '\t', end=" ")
26            for i, v in enumerate(tape):

```

```
25         if i==pos:
26             #print ("[%s]"%(v,),end=" ")
27             archivo.write('['+v+']')
28         else:
29             #print (v, end=" ")
30             archivo.write(v+',')
31     #print()
32     archivo.write('\n')
33     if st == final:
34         print("Cadena valida.")
35         break
36     if (st, tape[pos]) not in rules:
37         print("Cadena invalida.")
38         break
39
40     (v1,dr,s1) = rules [(st, tape[pos])]
41     tape[pos]=v1 #rescribe el simbolo de la cinta
42
43     #movimiento del cabezal
44     if dr == 'left':
45         if pos > 0: pos -= 1
46         else: tape.insert(0, blank)
47     if dr == 'right':
48         pos += 1
49         if pos >= len(tape): tape.append(blank)
50     st = s1
```

Figura 1.2: Función de recorrido de turing.

La función turing MG hace exactamente lo mismo que la primera, solo que el formato de guardado para el archivo de salida es diferente, esto se hace debido a que en la primera maquina está destinada a casos menores a 10 (donde se requiere una animación). Sin embargo, para casos muy grandes y sin animación lo mejor es guardar la información de manera lineal en el archivo de salida, de esta manera no desperdiciamos tantos recursos de memoria. Observar figura 1.3.

```

1 def turing_MG (state = None, #estados de la maquina de
   turing
2         blank = None, #simbolo blanco de el alfabeto
   dela cinta
3         rules = [], #reglas de transicion
4         tape = [], #cinta
5         final = None, #estado valido y/o final
6         pos = 0):#posicion siguiente de la maquina de
   turing
7
8     st = state
9     if not tape: tape = [blank]
10    if pos < 0 : pos += len(tape)
11    if pos >= len(tape) or pos < 0 :
12        print("Se inicializa mal la posicion")
13        SystemExit(1)
14
15    rules = dict(((s0, v0), (v1, dr, s1)) for (s0, v0, v1,
   dr, s1) in rules)
16    """
17        Estado S mbolo le do S mbolo escrito
18        Mov. Estado sig.
19        qn(s0) 1,0,X,Y,B(v0) 1,0,X,Y,B(v1) R o L(
20        dr) qn(s1)
21    """
22    while True:
23        with open('C:\\Users\\soyco\\OneDrive\\Documents\\
   ESCOM\\sem4\\Teoria\\P2\\turing\\output\\turing.
   txt', 'a') as archivo:
24            archivo.write('|-'+st+ '->')
25            #print (st, '\t', end=" ")
26            for i, v in enumerate(tape):

```

```

25         if i==pos:
26             #print ("[%s]"%(v,),end=" ")
27             archivo.write('['+v+']','')
28         else:
29             #print (v, end=" ")
30             archivo.write(v+',')
31     #print()
32     if st == final:
33         print("Cadena valida.")
34         break
35     if (st, tape[pos]) not in rules:
36         print("Cadena invalida.")
37         break
38
39     (v1,dr,s1) = rules [(st, tape[pos])]
40     tape[pos]=v1 #rescribe el simbolo de la cinta
41
42     #movimiento del cabezal
43     if dr == 'left':
44         if pos > 0: pos -= 1
45         else: tape.insert(0, blank)
46     if dr == 'right':
47         pos += 1
48         if pos >= len(tape): tape.append(blank)
49     st = s1

```

Figura 1.3: Función de turing MG para casos grandes.

Esta parte de código realiza las siguientes acciones:

1. Abre el archivo "turing.txt.<sup>en</sup> modo de escritura ('w') utilizando la sentencia with open('C:/ Users/ soyco/ OneDrive/ Documents/ ESCOM/ sem4/ Teoria/ P2/ turing/ output/ turing.txt', 'w') as archivo. La declaración with garantiza que el archivo se cerrará correctamente después de su uso.

2. La sentencia `pass` se utiliza para indicar que no se realiza ninguna acción en este bloque de código. Es un marcador de posición que se utiliza cuando se requiere sintácticamente un bloque de código, pero no se desea ejecutar ninguna instrucción.
3. Se imprime 'Maquina de turing' en la consola utilizando la función `print()`.
4. Se imprime el menú en la consola utilizando la función `print()`:
5. Se solicita al usuario que elija una opción ingresando un número utilizando la función `input()`. El valor ingresado se asigna a la variable `option`. Si la opción seleccionada es '1', se ejecuta el siguiente bloque de código:
  - a) Se solicita al usuario que ingrese una cadena menor a 11 caracteres utilizando la función `input()`. El valor ingresado se asigna a la variable `input_string`.
  - b) Se llama a la función `turing_M()` con los siguientes argumentos: a) `state = 'q0'`: estado inicial de la máquina de Turing. b) `blank = 'B'`: símbolo blanco del alfabeto de la cinta. c) `tape = list(input_string)`: inserta los elementos de la cadena en la cinta como una lista. d) `final = 'q4'`: estado válido y/o final. e) `rules`: reglas de transición representadas como una lista de listas de cadenas divididas por espacios.
  - c) Después de la llamada a `turing_M()`, se anima la máquina de Turing mediante un ciclo `while`.
  - d) Se lee el contenido del archivo "turing.txt" utilizando la sentencia `with open('C:/ Users/ soyco/ OneDrive/ Documents/ ESCOM/ sem4/ Teoria/ P2/ turing/ output/ turing.txt', 'r')` as `archivo`: y se almacena en la variable `lineas`.

- e) Se ejecuta un ciclo while que se repetirá mientras la variable run sea verdadera.
- f) Dentro del ciclo, se obtiene el estado y el recorrido de la máquina de Turing desde las líneas correspondientes en líneas.
- g) Se llama a las funciones dibujar\_tablero() y dibujar\_boton() para dibujar el tablero y el botón respectivamente en una interfaz gráfica.
- h) Se verifica si se hizo clic en el botón utilizando la función collide-point() de Pygame.
- i) Se incrementa el valor de leerlinea en 2 para avanzar a las siguientes líneas en líneas.
- j) Finalmente, se actualiza la pantalla y se verifica si se ha cerrado la ventana del programa.

Si la opción seleccionada es '2', se ejecuta el siguiente bloque de código:

- a) Se genera automáticamente una cadena utilizando la función generate\_auto\_string() y se asigna a la variable input\_string.
  - b) Se imprime la cadena generada automáticamente utilizando la función print().
  - c) El proceso de animación y verificación se lleva a cabo de manera similar a la opción '1'.
6. Si la opción seleccionada no es '1' ni '2', se imprime 'Opción inválida. Inténtalo de nuevo.' en la consola.

7. Se imprime "Programa terminó..<sup>en</sup> la consola.

Esta interfaz permite al usuario interactuar con la máquina mediante opciones del menú. Dependiendo de la opción seleccionada, se ingresa manualmente una cadena o se genera automáticamente, y luego se ejecuta la máquina de Turing utilizando las reglas de transición especificadas. Observar figura 1.4.

```

1  def turing_MG (state = None, #estados de la maquina de
    turing
2      blank = None, #simbolo blanco de el alfabeto
        dela cinta
3      rules = [], #reglas de transicion
4      tape = [], #cinta
5      final = None, #estado valido y/o final
6      pos = 0):#posicion siguiente de la maquina de
        turing
7
8      st = state
9      if not tape: tape = [blank]
10     if pos < 0 : pos += len(tape)
11     if pos >= len(tape) or pos < 0 :
12         print("Se inicializa mal la posicion")
13         SystemExit(1)
14
15     rules = dict(((s0, v0), (v1, dr, s1)) for (s0, v0, v1,
        dr, s1) in rules)
16     """
17         Estado  S mbolo le do  S mbolo escrito
18         Mov.    Estado sig.
19         qn(s0)  1,0,X,Y,B(v0)  1,0,X,Y,B(v1)      R o L(
20         dr)    qn(s1)
21     """
22     while True:
23         with open('C:\\Users\\soyco\\OneDrive\\Documents\\
24             ESCOM\\sem4\\Teoria\\P2\\turing\\output\\turing.
25             txt', 'a') as archivo:
26             archivo.write('|-'+st+ '->')
27             #print (st, '\t', end=" ")
28             for i, v in enumerate(tape):

```



```
25         if i==pos:
26             #print ("[%s]"%(v,),end=" ")
27             archivo.write(['+v+'],')
28         else:
29             #print (v, end=" ")
30             archivo.write(v+',')
31     #print()
32     if st == final:
33         print("Cadena valida.")
34         break
35     if (st, tape[pos]) not in rules:
36         print("Cadena invalida.")
37         break
38
39     (v1,dr,s1) = rules [(st, tape[pos])]
40     tape[pos]=v1 #rescribe el simbolo de la cinta
41
42     #movimiento del cabezal
43     if dr == 'left':
44         if pos > 0: pos -= 1
45         else: tape.insert(0, blank)
46     if dr == 'right':
47         pos += 1
48         if pos >= len(tape): tape.append(blank)
49     st = s1
```

Figura 1.4: Main del programa.

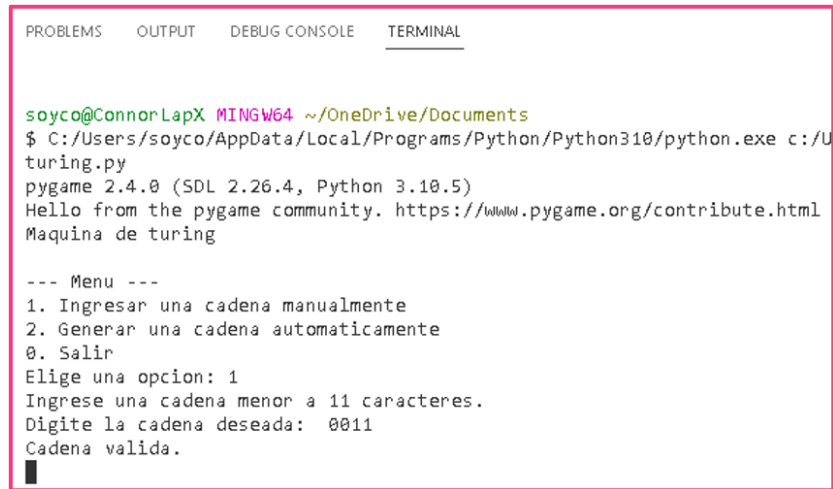
## **Capítulo 2**

### **Análisis de Resultados**

#### **2.1. Capturas del programa en ejecución**

A continuación se presenta en orden el proceso de ejecución del programa, donde primeramente se muestra el código en ejecución para la parte de la animación y otro ejemplo más grande donde no se incluye animación.

1. Iniciamos el programa, donde nos pide que introduzcamos una opción en el menú de inicio, seleccionamos opción 1 y luego digitamos '0011' que será nuestra cadena a evaluar. El programa nos indica que la cadena fue evaluada y es valida. Observar la Figura 2.1.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
soyco@ConnorLapX MINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/U
turing.py
pygame 2.4.0 (SDL 2.26.4, Python 3.10.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Maquina de turing

--- Menu ---
1. Ingresar una cadena manualmente
2. Generar una cadena automaticamente
0. Salir
Elige una opcion: 1
Ingrese una cadena menor a 11 caracteres.
Digite la cadena deseada: 0011
Cadena valida.
█
```

Figura 2.1: Visualización del programa en terminal caso 1.

2. Aquí se puede ver la primera parte de la animación, donde vemos todas las transiciones de la animación, esta animación va cambiando conforme le demos clic al botón 'siguiente'. Observar la Figura 2.2.



Figura 2.2: Visualización de la primera parte de animaciones.

3. Aquí se puede ver la segunda parte de la animación, donde vemos todas las transiciones de la animación, esta animación va cambiando conforme le demos clic al botón 'siguiente'. En esta última parte podemos ver como llegamos al estado final q4, ya si el programa finaliza. Observar la Figura 2.3.

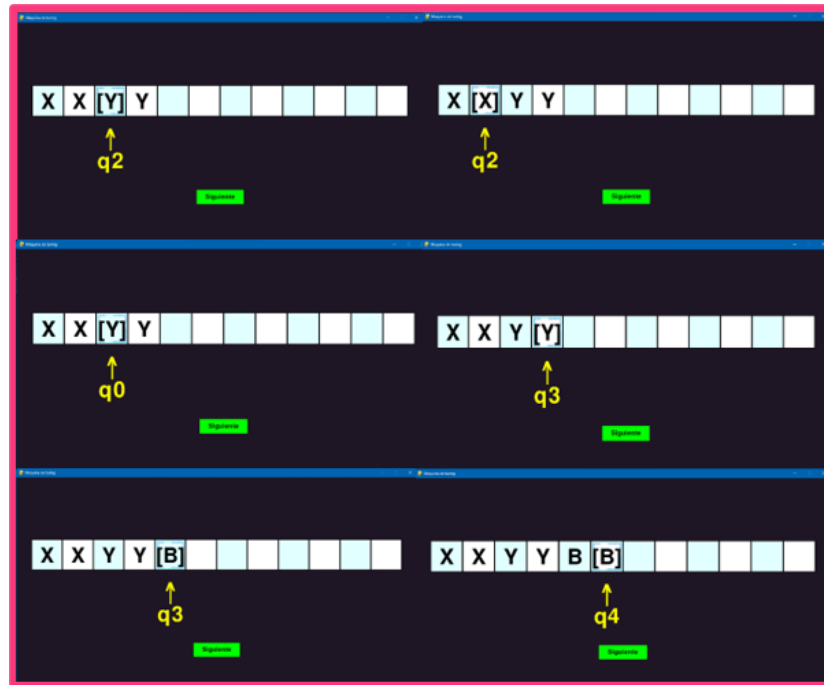
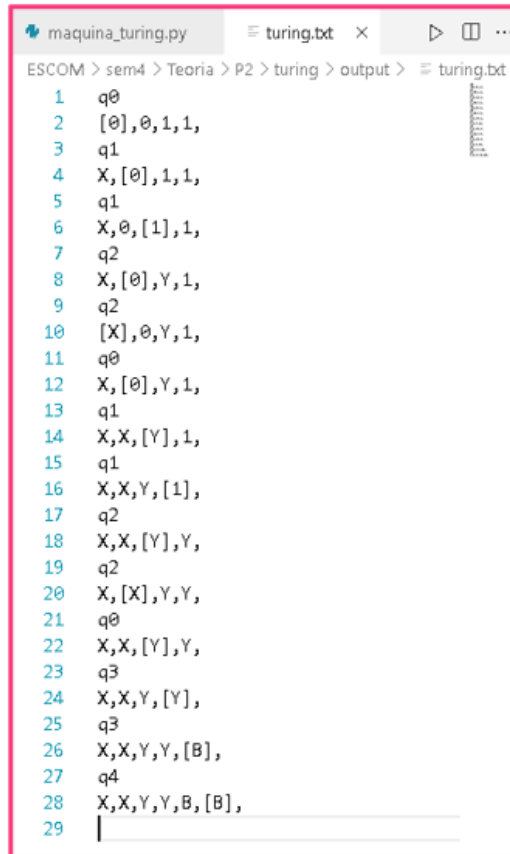


Figura 2.3: Visualización de la segunda parte de animaciones.

4. Aquí podemos ver el archivo de salida turing.txt, que nos mostrara cada paso de las evaluaciones de la máquina de turing. Observar la Figura 2.4.



```
maquina_turing.py  turing.txt  x  ▶  □  ...
ESCOM > sem4 > Teoria > P2 > turing > output > turing.txt
1  q0
2  [0],0,1,1,
3  q1
4  X,[0],1,1,
5  q1
6  X,0,[1],1,
7  q2
8  X,[0],Y,1,
9  q2
10 [X],0,Y,1,
11 q0
12 X,[0],Y,1,
13 q1
14 X,X,[Y],1,
15 q1
16 X,X,Y,[1],
17 q2
18 X,X,[Y],Y,
19 q2
20 X,[X],Y,Y,
21 q0
22 X,X,[Y],Y,
23 q3
24 X,X,Y,[Y],
25 q3
26 X,X,Y,Y,[B],
27 q4
28 X,X,Y,Y,B,[B],
29 |
```

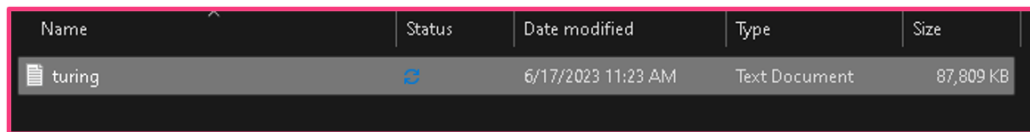
Figura 2.4: Vista del archivo de salida 'turing.txt'.

5. Iniciamos nuevamente el programa, esta vez seleccionamos opción 2 y nos genera una cadena aleatoria, para este caso nos genero una cadena de tamaño 446, posteriormente la evalúa y termina el programa. Observar la figura 2.5.

[illegible]

Figura 2.5: Visualización del programa en terminal caso 2.

6. Vemos que el archivo de salida 'turing.txt' llegó a pesar cerca de 90MB, lo que significa que la complejidad del algoritmo incrementa exponencialmente conforme el tamaño de la cadena crece y por ende entre mayor tamaño de cadena, más recursos de memoria serán necesarios. Observar la figura 2.6.



Name	Status	Date modified	Type	Size
turing		6/17/2023 11:23 AM	Text Document	87,809 KB

Figura 2.6: Visualización de memoria utilizada por el archivo para caso 2.



7. Aquí se puede ver el inicio del archivo de 'turing.txt', donde podemos ver que inicia correctamente con el cabezal en la primera posición. Cabe recalcar que se tuvo que hacer uso de una herramienta especial para abrir archivos grandes de información. Observar la figura 2.7.

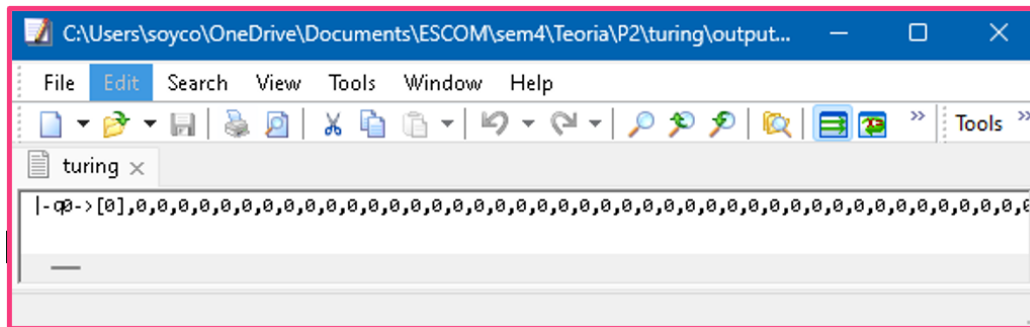


Figura 2.7: Visualización del inicio de archivo de salida turing.txt.

8. Aquí se puede ver el final del archivo de 'turing.txt', donde podemos ver en la primer parte como termina correctamente con los dos espacios en blanco, que nos indica que se llegó al estado q4, y en la parte inferior podemos ver como realmente todos los ceros y unos fueron reescritos correctamente. Observar la figura 2.7.

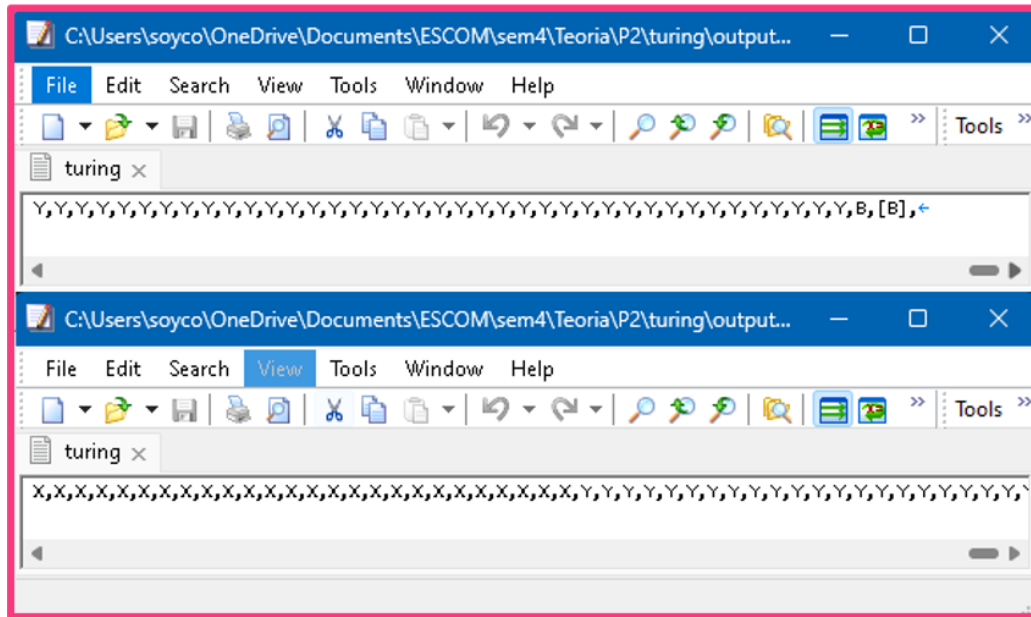


Figura 2.8: Visualización del final de archivo de salida turing.txt.

## Capítulo 3

### Conclusión

La conclusión de desarrollar y solucionar el problema de la máquina de Turing es que se trata de un modelo computacional muy poderoso y versátil. Al implementar una máquina de Turing para resolver un problema específico, se obtienen varias conclusiones:

1. **Universalidad:** La máquina de Turing es capaz de simular cualquier algoritmo computacional, lo que demuestra su capacidad para resolver problemas teóricos y prácticos de manera general.
2. **Complejidad:** La máquina de Turing permite analizar la complejidad de un problema al medir la cantidad de operaciones necesarias para resolverlo. Esto es fundamental para evaluar la eficiencia de los algoritmos y comprender la viabilidad de las soluciones propuestas.
3. **Representación abstracta:** La máquina de Turing proporciona una abstracción poderosa para representar y modelar problemas complejos. Permite separar el concepto del problema en una cinta y un cabezal que interactúa con ella, lo cual facilita el diseño de soluciones.
4. **Limitaciones y alcance:** La máquina de Turing tiene sus limitaciones, especialmente en términos de la indecibilidad de algunos problemas o la imposibilidad de resolverlos de manera eficiente. Sin embargo, también tiene

un amplio alcance y puede abordar una variedad de problemas teóricos y prácticos.

En conclusión general, el desarrollo y la solución de problemas utilizando la máquina de Turing permiten comprender y analizar la computabilidad y complejidad de los problemas. Además, brindan una base sólida para el diseño y análisis de algoritmos, y ayudan a investigar los límites de lo que se puede computar.

### **3.1. Complejidades**

La complejidad tanto espacial como temporal del algoritmo dependerá fielmente del tamaño de la entrada de ceros y unos y de las reglas de transición definidas en la máquina de Turing. Pero se espera que sea un comportamiento exponencial conforme el tamaño de la entrada incrementa.

# Capítulo 4

## Bibliografías

1. Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation (2nd ed.). Pearson.
2. Sipser, M. (2012). Introduction to the Theory of Computation (3rd ed.). Cengage Learning.
3. Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 42(2), 230-265.

# Capítulo 5

## Anexos

### 5.1. Código LATEX de este documento

Dirección GitHub:<https://github.com/Connor-UM-18/Teoria-Computacional---Turing.git>

Dirección Overleaf:<https://www.overleaf.com/2428945132mnyyfwyhddjq>

### 5.2. `maquina_turing.py`

Se presenta el código implementado para la solución al problema con extensión `.py`. Donde es necesario tener importada la librería `random` y `pygame`.

```
1 #Teoria de la computacion
2 #Maquina de turing
3 #Alumno: Connor Urbano Mendoza
4 import random
5 import pygame
6
7 pygame.init() #Acceso al paquete pygame
8 #Ancho
9 WIDTH = 1300
10 #Altura
11 HEIGHT = 700
```

```
12 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
13 pygame.display.set_caption('Maquina de turing')
14 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
15 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo de
    fuente 2 del juego
16 timer = pygame.time.Clock()#velocidad de actualizacion de
    nuestro juego a 60 fps
17 fps=60
18
19 #Variables e imagenes del juego
20 ubicacion = ['recuadro']
21
22 #Variables de turnos cambiantes
23 turn_step = 0
24 selection= 100
25 #Cargar imagenes en juego
26 cuadro = pygame.image.load('C:\\Users\\soyco\\OneDrive\\
    Documents\\ESCOM\\sem4\\Teoria\\P2\\turing\\img\\cuadro.
    png')
27 cuadro = pygame.transform.scale(cuadro, (101,101))
28
29 cuadrado = [cuadro]
30
31 lista_piezas = ['recuadro']
32
33 #ver variables/contador flash
34 boton_presionado = False
35
36 def dibujar_boton():
37     boton_width = 150
38     boton_height = 45
39     boton_x = (WIDTH - boton_width) // 2
40     boton_y = (HEIGHT - boton_height - 20)-100
41
42     # Dibujar el bot n como un rect ngulo en la pantalla
43     boton_rect=pygame.Rect(boton_x, boton_y, boton_width,
        boton_height)
44     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
```

```

    , boton_width, boton_height))
45 texto = font.render("Siguiente", True, (0, 0, 0))
46 texto_rect = texto.get_rect(center=(boton_x +
    boton_width // 2, boton_y + boton_height // 2))
47 screen.blit(texto, texto_rect)
48 return boton_rect
49
50 #Funcion para dibujar el contenido de la maquina de turing
51 def mostrar_transiciones(recorrido,estado):
52
53     #recorrido="X,0,0,1,[1],1,1,1,"
54     #estado="q2"
55     cuadro_size = 100 # Tama o de cada cuadro del tablero
56     x = 0
57     y = 255
58
59     font = pygame.font.Font(None, 100) # Fuente y tama o
    del n mero de casilla
60
61     transiciones = recorrido.split(',') # Obtener las
    transiciones del recorrido
62
63     for i, transicion in enumerate(transiciones):
64         if i >= 12:
65             break # Salir del bucle si se han mostrado
                todas las transiciones posibles
66         x = x + 100
67         if transicion.startswith('['):
68             # Realizar alguna acci n especial para la
                transici n que comienza con '['
69             # Por ejemplo, cambiar el color del recuadro o
                agregar un marcador adicional
70             numero_texto = font.render("^", True, 'yellow')
                # Crear superficie de texto con la
                transici n
71             numero_rect = numero_texto.get_rect(center=((x,
                y+110))) # Posici n del texto en el
                centro del recuadro
72             screen.blit(numero_texto, numero_rect) # Pegar
                el texto en la pantalla

```



```

73         numero_texto = font.render("|", True, 'yellow')
           # Crear superficie de texto con la
           transici n
74         numero_rect = numero_texto.get_rect(center=((x,
           y+120))) # Posici n del texto en el
           centro del recuadro
75         screen.blit(numero_texto, numero_rect) # Pegar
           el texto en la pantalla
76         numero_texto = font.render(estado, True, '
           yellow') # Crear superficie de texto con la
           transici n
77         numero_rect = numero_texto.get_rect(center=((x,
           y+190))) # Posici n del texto en el
           centro del recuadro
78         screen.blit(numero_texto, numero_rect) # Pegar
           el texto en la pantalla
79         index=lista_piezas.index('recuadro')
80         screen.blit(cuadrado[index], (x-50,y-55))
81
82         numero_texto = font.render(transicion, True, 'black
           ') # Crear superficie de texto con la
           transici n
83         numero_rect = numero_texto.get_rect(center=((x, y))
           ) # Posici n del texto en el centro del
           recuadro
84         screen.blit(numero_texto, numero_rect) # Pegar el
           texto en la pantalla
85
86 #Funcion para dibujar tablero
87 def dibujar_tablero():
88     cuadro_size = 100 # Tama o de cada cuadro del tablero
89     tablero_width = 12 * cuadro_size # Ancho total del
           tablero
90     tablero_height = 1 * cuadro_size # Altura total del
           tablero
91     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
           para centrar el tablero
92     tablero_y = ((HEIGHT - tablero_height) // 2 )-200 #
           Posici n Y para centrar el tablero
93

```

```

94     for i in range(12): # Iterar 12 veces para un tablero
95         de 1x12
96         columna = i % 12
97         fila = 1
98         x = tablero_x + columna * cuadro_size
99         y = tablero_y + fila * cuadro_size
100         if fila % 2 == 0:
101             color = 'white' if columna % 2 == 0 else (226,
102                 255, 255)
103         else:
104             color = (226, 255, 255) if columna % 2 == 0
105                 else 'white'
106         pygame.draw.rect(screen, color, [x, y, cuadro_size,
107             cuadro_size])
108         pygame.draw.rect(screen, 'black', [x, y,
109             cuadro_size, cuadro_size], 2) # Agregar borde
110             de color
111
112 # Funci n para generar automticamente una cadena v lida
113 def generate_auto_string():
114     n = random.randint(0, 500)
115     input_string = '0' * n + '1' * n
116     return input_string
117
118 def turing_M (state = None, #estados de la maquina de
119     turing
120         blank = None, #simbolo blanco de el alfabeto
121             dela cinta
122         rules = [], #reglas de transicion
123         tape = [], #cinta
124         final = None, #estado valido y/o final
125         pos = 0):#posicion siguiente de la maquina de
126     turing
127
128     st = state
129     if not tape: tape = [blank]
130     if pos < 0 : pos += len(tape)
131     if pos >= len(tape) or pos < 0 :
132         print("Se inicializa mal la posicion")

```

```

125         SystemExit(1)
126
127     rules = dict(((s0, v0), (v1, dr, s1)) for (s0, v0, v1,
128         dr, s1) in rules)
129     """
130     Estado  S mbolo le do S mbolo escrito
131     Mov.    Estado sig.
132     qn(s0)  1,0,X,Y,B(v0)  1,0,X,Y,B(v1)      R o L(
133     dr)     qn(s1)
134     """
135     while True:
136         with open('C:\\Users\\soyco\\OneDrive\\Documents\\
137             ESCOM\\sem4\\Teoria\\P2\\turing\\output\\turing.
138             txt', 'a') as archivo:
139             archivo.write(st+ '\n')
140             #print (st, '\t', end=" ")
141             for i, v in enumerate(tape):
142                 if i==pos:
143                     #print ("[%s]"%(v,),end=" ")
144                     archivo.write('['+v+',')
145                 else:
146                     #print (v, end=" ")
147                     archivo.write(v+',')
148             #print()
149             archivo.write('\n')
150             if st == final:
151                 print("Cadena valida.")
152                 break
153             if (st, tape[pos]) not in rules:
154                 print("Cadena invalida.")
155                 break
156
157             (v1,dr,s1) = rules [(st, tape[pos])]
158             tape[pos]=v1 #rescribe el simbolo de la cinta
159
160     #movimiento del cabezal
161     if dr == 'left':
162         if pos > 0: pos -= 1
163         else: tape.insert(0, blank)
164     if dr == 'right':

```

```

160         pos += 1
161         if pos >= len(tape): tape.append(blank)
162         st = s1
163
164 def turing_MG (state = None, #estados de la maquina de
    turing
165         blank = None, #simbolo blanco de el alfabeto
        dela cinta
166         rules = [], #reglas de transicion
167         tape = [], #cinta
168         final = None, #estado valido y/o final
169         pos = 0):#posicion siguiente de la maquina de
        turing
170
171     st = state
172     if not tape: tape = [blank]
173     if pos < 0 : pos += len(tape)
174     if pos >= len(tape) or pos < 0 :
175         print("Se inicializa mal la posicion")
176         SystemExit(1)
177
178     rules = dict(((s0, v0), (v1, dr, s1)) for (s0, v0, v1,
        dr, s1) in rules)
179     """
180         Estado S mbolo le do S mbolo escrito
        Mov. Estado sig.
181         qn(s0) 1,0,X,Y,B(v0) 1,0,X,Y,B(v1) R o L(
        dr) qn(s1)
182     """
183     while True:
184         with open('C:\\Users\\soyco\\OneDrive\\Documents\\
            ESCOM\\sem4\\Teoria\\P2\\turing\\output\\turing.
            txt', 'a') as archivo:
185             archivo.write('|-'+st+'->')
186             #print (st, '\t', end=" ")
187             for i, v in enumerate(tape):
188                 if i==pos:
189                     #print ("[%s]"%(v,),end=" ")
190                     archivo.write('['+v+'],')
191             else:

```

```

192         #print (v, end=" ")
193         archivo.write(v+',')
194     #print()
195     if st == final:
196         print("Cadena valida.")
197         break
198     if (st, tape[pos]) not in rules:
199         print("Cadena invalida.")
200         break
201
202     (v1,dr,s1) = rules [(st, tape[pos])]
203     tape[pos]=v1 #rescribe el simbolo de la cinta
204
205     #movimiento del cabezal
206     if dr == 'left':
207         if pos > 0: pos -= 1
208         else: tape.insert(0, blank)
209     if dr == 'right':
210         pos += 1
211         if pos >= len(tape): tape.append(blank)
212     st = s1
213
214 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
    sem4\\Teoria\\P2\\turing\\output\\turing.txt', 'w') as
    archivo:
215     pass
216 print("Maquina de turing")
217 print("\n--- Menu ---")
218 print("1. Ingresar una cadena manualmente")
219 print("2. Generar una cadena automaticamente")
220 print("0. Salir")
221
222 option = input("Elige una opcion: ")
223
224 if option == '1':
225     input_string = input("Ingrese una cadena menor a 11
        caracteres. \nDigite la cadena deseada: ")
226     #se puede cambiar las reglasde transicion para otra
        maquina de turing
227     turing_M (state = 'q0', #estado inicial de la maquina

```

```

228         de turing
                blank = 'B', #simbolo blanco de el alfabeto
                        dela cinta
229         tape = list(input_string), #inserta los
                elementos en la cinta
230         final = 'q4', #estado valido y/o final
231         rules = map(tuple, #reglas de transicion
232                     [
233                         "q0 0 X right q1".split(),
234                         "q0 Y Y right q3".split(),
235                         "q1 0 0 right q1".split(),
236                         "q1 1 Y left q2".split(),
237                         "q1 Y Y right q1".split(),
238                         "q2 0 0 left q2".split(),
239                         "q2 X X right q0".split(),
240                         "q2 Y Y left q2".split(),
241                         "q3 Y Y right q3".split(),
242                         "q3 B B right q4".split(),
243                     ]
244                 )
245     )
246     #Animamos la maquina de turing
247     #lista_estados = [int(num) for num in recorrido.split
248         (",")]
249     coordenadas=(235,85)
250     nueva_coordenada=(0,0)
251     run=True
252     contador=1
253     leerlinea=0
254     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
255         \\sem4\\Teoria\\P2\\turing\\output\\turing.txt', 'r'
256         ) as archivo:
257         lineas = archivo.readlines()
258
259     while run:
260         if len(lineas) == leerlinea:
261             print("Programa termino.")
262             SystemExit(0)
263             timer.tick(fps)
264             # Rellenar la pantalla con el color

```

```

262     screen.fill((30, 22, 37))
263     dibujar_tablero()
264     boton_rect = dibujar_boton()
265     estado = lineas[leerlinea].strip() # Eliminar
        espacios en blanco al inicio y al final de la
        linea
266     recorrido = lineas[leerlinea+1].strip()
267     mostrar_transiciones(recorrido, estado)
268     for event in pygame.event.get():
269         if event.type == pygame.QUIT:
270             run = False
271
272         if event.type == pygame.MOUSEBUTTONDOWN and
            event.button == 1:
273             mouse_pos = pygame.mouse.get_pos()
274             if boton_rect.collidepoint(mouse_pos): #
                Verificar si se hizo clic en el bot n
                leerlinea=leerlinea+2
275             pygame.display.flip()
276     pygame.quit()
277
278
279
280 elif option == '2':
281     input_string = generate_auto_string()
282     print("Cadena generada automaticamente:", input_string)
283     print("Tamano de cadena:", str(len(input_string)))
284     turing_MG (state = 'q0', blank = 'B', tape = list(
        input_string), final = 'q4', rules = map(tuple,
285         #reglas de transicion
286         [
287             "q0 0 X right q1".split(),
288             "q0 Y Y right q3".split(),
289             "q1 0 0 right q1".split(),
290             "q1 1 Y left q2".split(),
291             "q1 Y Y right q1".split(),
292             "q2 0 0 left q2".split(),
293             "q2 X X right q0".split(),
294             "q2 Y Y left q2".split(),
295             "q3 Y Y right q3".split(),
296             "q3 B B right q4".split(),

```

```
297                                     ]
298                                     )
299                                 )
300
301
302 else:
303     print("Opcion invalida. Int ntalo de nuevo.")
304     SystemExit(0)
305
306 print("Programa termino.")
```