

Reporte del primer parcial

Carlos Tonatihu Barrera Pérez
Profesor: Genaro Juárez Martínez
Teoría Computacional
Grupo: 2CM4

10 de septiembre de 2016

Índice

1. Alfabeto	2
1.1. Descripción del problema	2
1.2. Código	2
1.3. Pruebas	5
2. Números primos	8
2.1. Descripción del problema	8
2.2. Código	8
2.3. Pruebas	10
3. AFD Palabras con terminación 'ere'	13
3.1. Descripción del problema	13
3.2. Código	13
3.3. Pruebas	19
4. AFD Paridad en números binarios	22
4.1. Descripción del problema	22
4.2. Código	22
4.3. Pruebas	28
5. Protocolo de transmisión	30
5.1. Descripción del problema	30
5.2. Código	30
5.3. Pruebas	36
6. AFND Números binarios con terminación '01'	38
6.1. Descripción del problema	38
6.2. Código	38
6.3. Pruebas	44

1. Alfabeto

1.1. Descripción del problema

El objetivo de esta practica es el generar las potencias del alfabeto binario $\Sigma = \{0, 1\}$ desde $k = 0$ hasta un k seleccionado con un máximo de $k = 1000$, para después guardar en un archivo todas las cadenas que se pudieron formar bajo estas condiciones, es decir:

$$\Sigma^+ = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^{1000}$$

Es importante señalar que este conjunto solo es un subconjunto de Σ^* que representa todas las cadenas que se pueden formar con este alfabeto binario. El programa cuenta con modo manual (el usuario ingresa un k) y automático (genera su propio k).

1.2. Código

El código del programa fue realizado en C.
Archivo: alfafeto.h

```
//alfafeto.h
#ifndef __ALFABETO__
#define __ALFABETO__

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

static const int CONTINUAR = 1;
int iniciar();
int abrir_archivo(FILE **);
int generar_palabras();

#endif
```

Archivo: alfabeto.c

```
//alfabeto.c
#include "alfabeto.h"

int generar_palabras(int potencia_k) {
    FILE *archivo = NULL;
    int long_cadena;
    int i;
    int posicion;
    int *cadena_aux = NULL;

    abrir_archivo(&archivo);
    fputs("S = {e", archivo);
```

```

for (long_cadena = 1; long_cadena <= potencia_k; long_cadena++) {
    cadena_aux = (int*) calloc(long_cadena, sizeof(int));
    if(cadena_aux == NULL) {
        printf("%s\n", "Error en el calloc");
        exit(0);
    }
    while(CONTINUAR) {
        fputc(',', archivo);
        for(i = 0; i < long_cadena; i++) {
            fputc(cadena_aux[i] + '0', archivo);
        }
        for(posicion = 0; posicion < long_cadena; posicion++) {
            cadena_aux[posicion] += 1;
            if(cadena_aux[posicion] > 1) {
                cadena_aux[posicion] = 0;
            } else {
                break;
            }
        }
        if(posicion >= long_cadena) {
            free(cadena_aux);
            break;
        }
    }
    printf("Va en 2^%d\n", long_cadena);
}

fputs("}", archivo);
fclose(archivo);
return 1;
}

int abrir_archivo(FILE **archivo) {
    *archivo = fopen("palabras.txt", "w");
    if (*archivo == NULL) {
        printf("%s\n", "No se pudo abrir");
        exit(0);
    }
    return 1;
}

```

Archivo: main.c

```

//albafeto.h
#include "main.c"

int iniciar();

```

```

int menu();
int menu_continuar();
int random_potencia_k();

int main(int argc, char const *argv[]) {
    printf("%s\n", "*****Alfabeto*****");
    iniciar();
    return 0;
}

int iniciar() {
    srand(time(NULL));
    int continuar = 1;
    int potencia_k = 1;
    int manual = 1;
    while(continuar) {
        manual = menu();
        if (manual == 1) {
            printf("%s\n", "Ingresa el valor de k: ");
            scanf("%d", &potencia_k);
        } else if (manual == 2) {
            potencia_k = random_potencia_k();
        } else {
            break;
        }
        printf("El valor de k es: %d\n", potencia_k);
        generar_palabras(potencia_k);
        printf("%s\n", "Cadenas guardadas en el archivo palabras.txt");
        continuar = menu_continuar();
    }
    printf("\n%s\n", "Saliendo...");
    return 1;
}

int menu_continuar() {
    int opcion;
    printf("Intentar otra vez?\nSi = 1 NO = 0\n");
    scanf(" %d", &opcion);
    return opcion;
}

int menu() {
    int opcion;
    printf("Que quieres hacer?\n1.-Manual\n2.-Automatico\n3.-Salir\n");
    scanf(" %d", &opcion);
    return opcion;
}

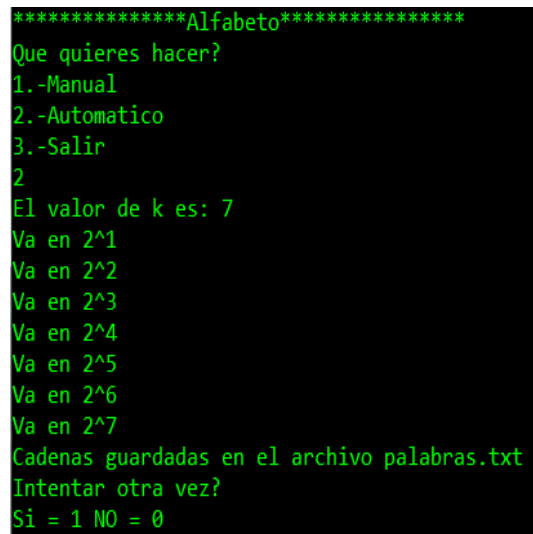
```

```
int random_potencia_k() {  
    //numero = rand () % (N-M+1) + M;  
    // Cambiar el valor del random k a 1000  
    int potencia_k = 1 + rand() % (1000 + 1 - 1);  
    return potencia_k;  
}
```

1.3. Pruebas

Las pruebas están divididas en modo automático y manual, en ambos dada una k se generan todas las cadenas de longitud 1 hasta k .

Modo automático.



```
*****Alfabeto*****  
Que quieres hacer?  
1.-Manual  
2.-Automatico  
3.-Salir  
2  
El valor de k es: 7  
Va en 2^1  
Va en 2^2  
Va en 2^3  
Va en 2^4  
Va en 2^5  
Va en 2^6  
Va en 2^7  
Cadenas guardadas en el archivo palabras.txt  
Intentar otra vez?  
Si = 1 NO = 0
```

Figura 1: Alfabeto con $k = 7$.

```

S =
[e,0,1,00,10,01,11,000,100,010,110,001,101,011,111,0000,1000,0100,1100,0010,1010,0110,1110,0001,1001,0101,1101,0011,1011
,0111,1111,00000,10000,01000,11000,00100,10100,01100,11100,00010,10010,01010,11010,00110,10110,01110,11110,00001,10001,0
1001,11001,00101,10101,01101,11101,00011,10011,01011,11011,00111,10111,01111,11111,000000,100000,010000,110000,001000,10
1000,011000,111000,000100,100100,010100,110100,001100,101100,011100,111100,000010,100010,010010,110010,001010,101010,011
010,111010,000110,100110,010110,110110,001110,101110,011110,111110,000001,100001,010001,110001,001001,101001,011001,1110
01,000101,100101,010101,110101,001101,101101,011101,111101,000011,100011,010011,110011,001011,101011,011011,111011,00011
1,100111,010111,110111,001111,101111,011111,111111,0000000,1000000,0100000,1100000,0010000,1010000,0110000,1110000,00010
00,1001000,0101000,1101000,0011000,1011000,0111000,1111000,0000100,1000100,0100100,1100100,0010100,1010100,0110100,11101
00,0001100,1001100,0101100,1101100,0011100,1011100,0111100,1111100,0000010,1000010,0100010,1100010,0010010,1010010,01100
10,1110010,0001010,1001010,0101010,1101010,0011010,1011010,0111010,1111010,0000110,1000110,0100110,1100110,0010110,10101
10,0110110,1110110,0001110,1001110,0101110,1101110,0011110,1011110,0111110,1111110,0000001,1000001,0100001,1100001,00100
01,1010001,0110001,1110001,0001001,1001001,0101001,1101001,0011001,1011001,0111001,1111001,0000101,1000101,0100101,11001
01,0010101,1010101,0110101,1110101,0001101,1001101,0101101,1101101,0011101,1011101,0111101,1111101,0000011,1000011,01000
11,1100011,0010011,1010011,0110011,1110011,0001011,1001011,0101011,1101011,0011011,1011011,0111011,1111011,0000111,10001
11,0100111,1100111,0010111,1010111,0110111,1110111,0001111,1001111,0101111,1101111,0011111,1011111,0111111,1111111]

```

Figura 2: Archivo generado.

Modo manual.

```

*****Alfabeto*****
Que quieres hacer?
1.-Manual
2.-Automatico
3.-Salir
1
Ingresa el valor de k:
9
El valor de k es: 9
Va en 2^1
Va en 2^2
Va en 2^3
Va en 2^4
Va en 2^5
Va en 2^6
Va en 2^7
Va en 2^8
Va en 2^9
Cadenas guardadas en el archivo palabras.txt
Intentar otra vez?
Si = 1 NO = 0

```

Figura 3: Alfabeto con $k = 9$.

Figura 4: Archivo generado.

2. Números primos

2.1. Descripción del problema

Desarrollar un programa que encuentre todos los números primos en el intervalo $0 \leq n \leq 1000$ imprimirlos en pantalla junto con su representación en binario además de contar la cantidad de ceros y unos en dicho número, finalmente guarda los números primos en su forma binaria en un archivo txt. Cuenta con modo manual (el usuario ingresa un número n) y automático (el programa utiliza un n aleatorio).

2.2. Código

El código del programa fue realizado en Python 3.5.
Archivo: primos.py

```
# primos.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import math, random

separador = '*'*50

def iniciar():
    maximo = 1
    continuar = True

while continuar:
    archivo = open('primos.txt', 'w')
    lista_primos = []
    lista_binarios = []
    manual = imprimir_menu()
    if manual == 1:
        maximo = int(input("Escribe un numero entre 1 y 1000 "))
    elif manual == 2:
        maximo = random_maximo()
    else:
        break
    print("El limite es: ", maximo)
    lista_primos = calcular_primos(maximo)
    lista_binarios = conversion_binaria(lista_primos, archivo)
    print(lista_primos)
    print('*'*50)
    print(lista_binarios)
    contar_repeticiones(lista_binarios, lista_primos)
    print('Numeros primos en binarios guardados en primos.txt')
    archivo.close()
    opcion = input("Reintentar s/n: ")
    if opcion.lower() != 's':
```

```

        continuar = False

print('Saliendo...')

def random_maximo():
    return random.randint(1, 1000)

def calcular_primos(maximo):
    lista_primos = []
    es_primo = False
    if maximo < 2:
        es_primo = False
    else:
        lista_primos.append(2)
        for numero_actual in range(2, maximo + 1):
            raiz = math.sqrt(numero_actual)
            if raiz == round(raiz):
                es_primo = False
            else:
                for num in lista_primos:
                    if num > math.ceil(raiz):
                        break
                    if numero_actual % num == 0:
                        es_primo = False
                        break
                else:
                    es_primo = True
            if es_primo:
                lista_primos.append(numero_actual)
    return lista_primos

def imprimir_menu():
    print('\n\n%sMenu%s' % (separador, separador))
    print("""
1.- Manual
2.- Automatico
3.- Salir
""")
    try:
        opcion = int(input("Selecciona una opcion valida: "))
        return opcion
    except Exception as e:
        print('Error ', e)
        return 0

def conversion_binaria(lista_primos, archivo):
    lista_binarios = []
    archivo.write('{')

```

```

for numero in lista_primos:
    lista_binarios.append(bin(numero)[2:])
    archivo.write('%s, ' % bin(numero)[2:])

archivo.write('}')
return lista_binarios

def contar_repeticiones(lista_binarios, lista_primos):
    total = []
    i = 0
    for valor in lista_binarios:
        ceros, unos = 0, 0
        for digito in valor:
            if digito == '0':
                ceros += 1
            else:
                unos += 1
        total.append({'Numero': lista_primos[i], 'Ceros': ceros, 'Unos': unos})
        i += 1
    for numero in total:
        print('Numero: %s No. Ceros: %s No. Unos: %s' % (numero['Numero'],
            numero['Ceros'], numero['Unos']))

iniciar()

```

2.3. Pruebas

Las pruebas están divididas en modo automático y manual.
Modo automático.

```

USER@TONA MINGW32 ~/Documents/tona/Git/teoria-computacional/numeros-primos (master)
$ python primos.py
*****Menu*****
1.- Manual
2.- Automatico
3.- Salir
Selecciona una opcion valida: 2
El limite es: 41
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]
*****
['10', '11', '101', '111', '1011', '1101', '10001', '10011', '10111', '11101', '11111', '100101', '101001']
Numero: 2 No. Ceros: 1 No. Unos: 1
Numero: 3 No. Ceros: 0 No. Unos: 2
Numero: 5 No. Ceros: 1 No. Unos: 2
Numero: 7 No. Ceros: 0 No. Unos: 3
Numero: 11 No. Ceros: 1 No. Unos: 3
Numero: 13 No. Ceros: 1 No. Unos: 3
Numero: 17 No. Ceros: 3 No. Unos: 2
Numero: 19 No. Ceros: 2 No. Unos: 3
Numero: 23 No. Ceros: 1 No. Unos: 4
Numero: 29 No. Ceros: 1 No. Unos: 4
Numero: 31 No. Ceros: 0 No. Unos: 5
Numero: 37 No. Ceros: 3 No. Unos: 3
Numero: 41 No. Ceros: 3 No. Unos: 3
Numeros primos en binarios guardados en primos.txt
Reintentar s/n:

```

Figura 5: Modo automático n=41.

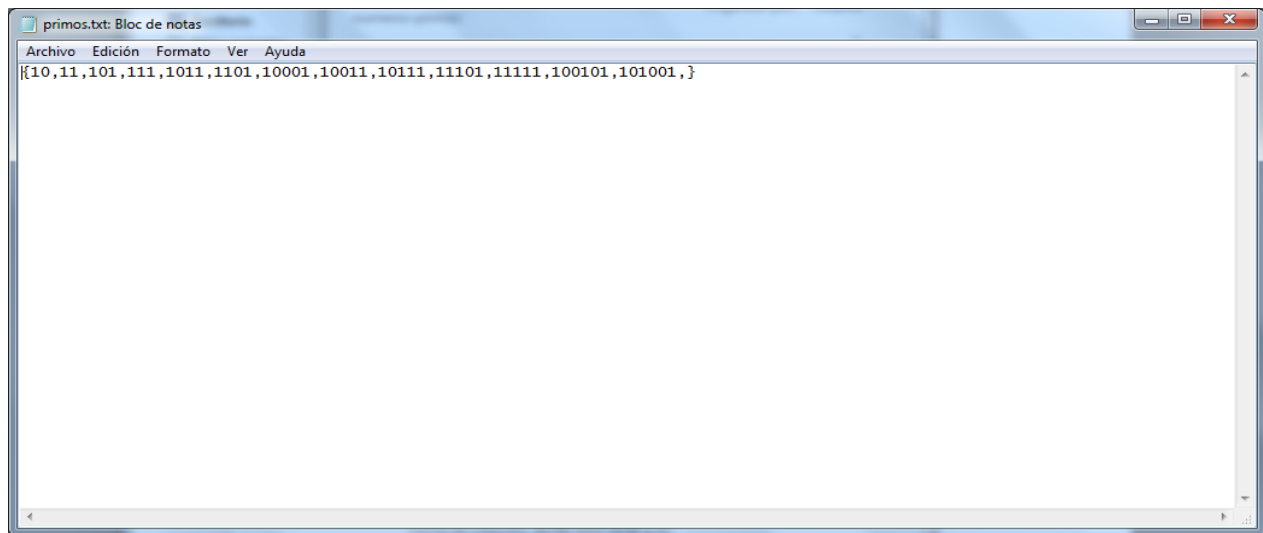


Figura 6: Números primos en binario en el archivo.

Modo manual.

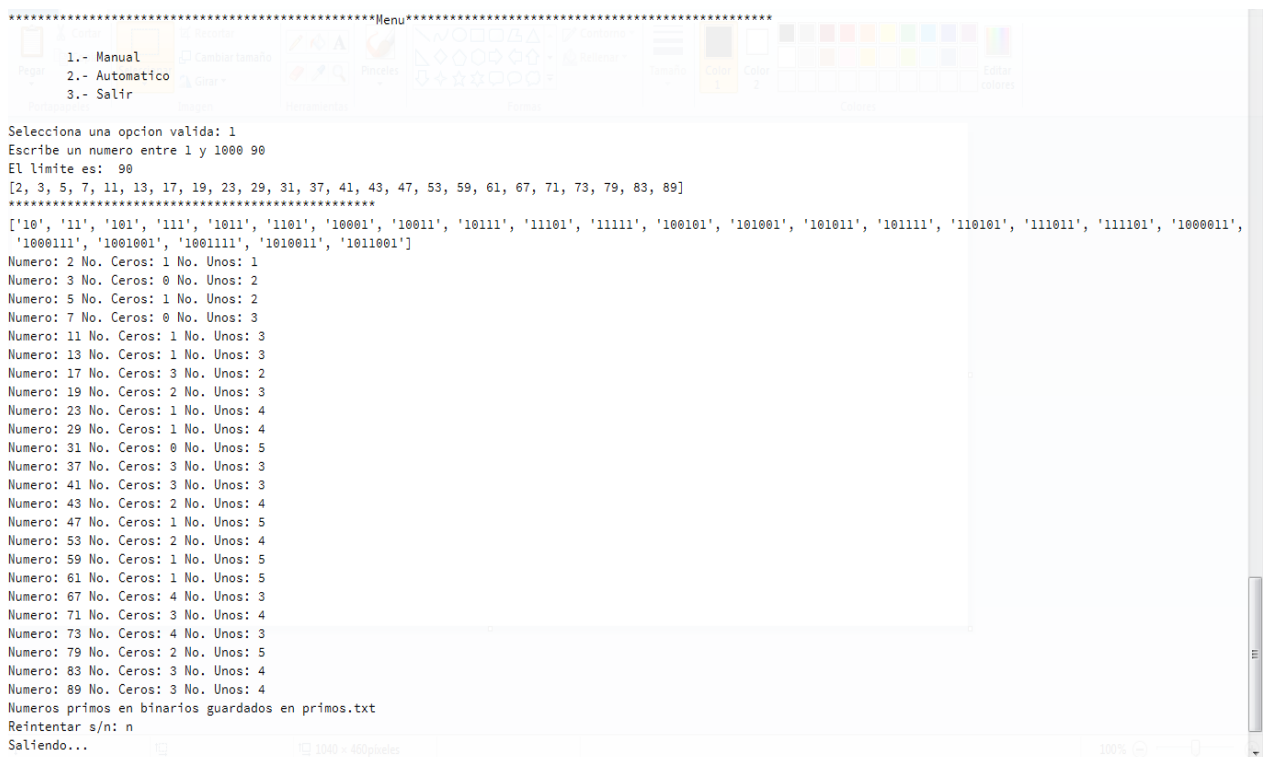


Figura 7: Modo automático n=90.

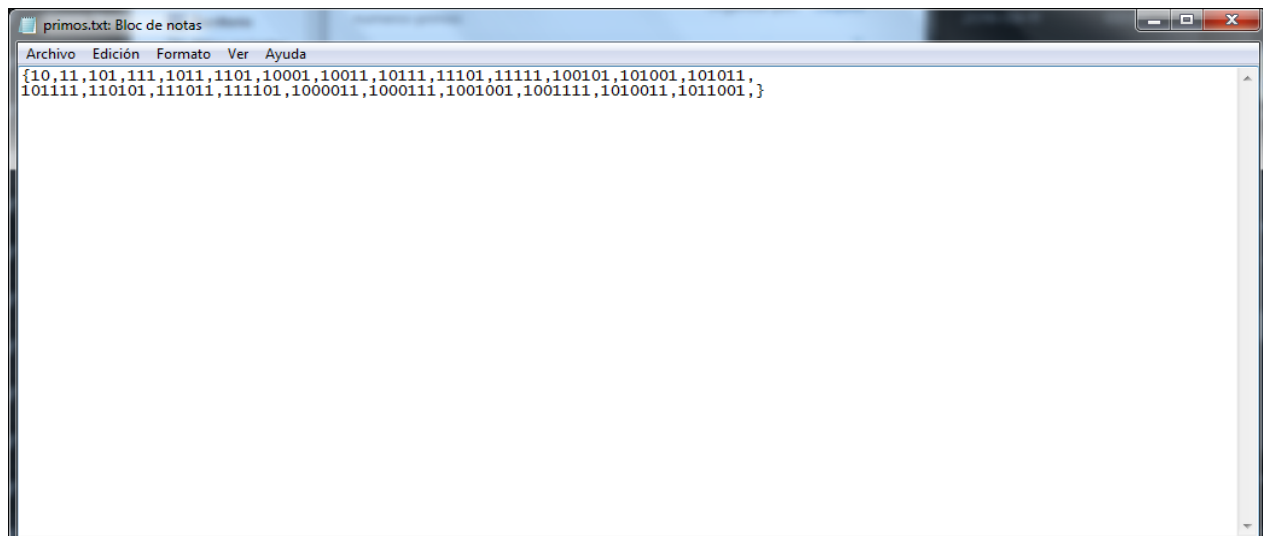


Figura 8: Números primos en binario en el archivo.

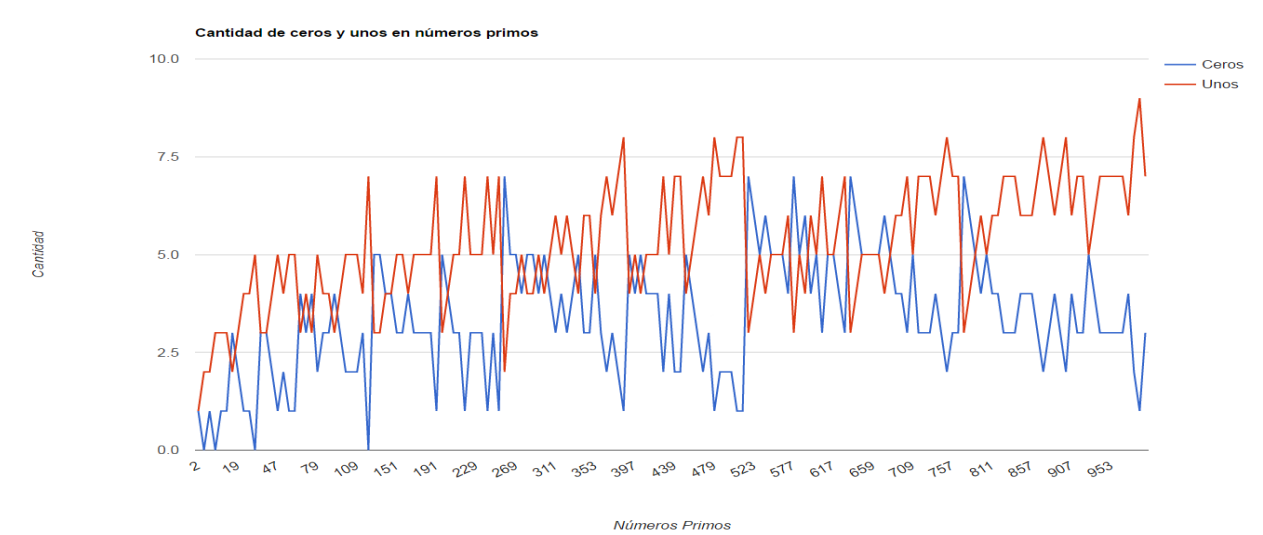


Figura 9: Cantidad de ceros y unos encontrados entre 1 y 1000.

3. AFD Palabras con terminación 'ere'

3.1. Descripción del problema

Desarrollar un autómata finito determinista capaz de encontrar las palabras con terminación 'ere' ya sea leyendo un archivo txt o en una línea de texto que el usuario ingresa, y que dichas palabras se muestren en pantalla y en el caso del archivo de texto imprimir la línea y el número de palabra (por línea) en el que fue encontrada dicha palabra. Es importante señalar que todo aquello que no es un símbolo del alfabeto inglés, $\Sigma = \{a, b, \dots, z, A, B, \dots, Z\}$, es tomado como un espacio. Además, debe tener una opción para visualizar el siguiente diagrama.

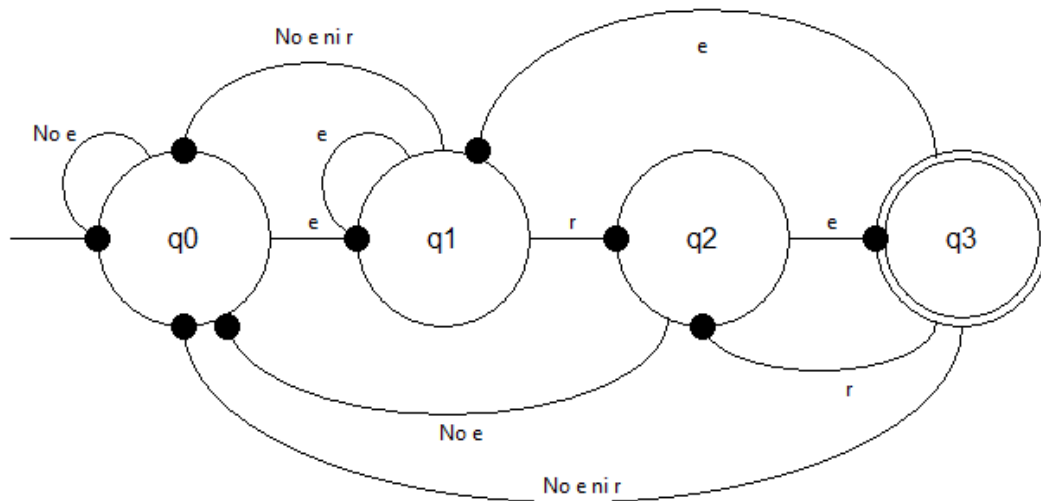


Figura 10: Diagrama de transiciones del autómata 'ere'.

3.2. Código

El código fue realizado en Python 3.5.
Archivo: main_ere.py

```
#main_ere.py
# -*- coding: utf-8 -*-
from __future__ import print_function
from automata_ere import verificar_palabras
from diagrama_ere import Diagrama

separador = ' '*50

def iniciar():
    continuar = True
    while continuar:
        opcion = imprimir_menu()
```

```

    if opcion == 1:
        entrada_consola()
    elif opcion == 2:
        entrada_archivo()
    elif opcion == 3:
        ver_diagrama()
    else:
        break
    print('*' * 100)
    opcion = input("Reintentar s/n: ")
    if opcion.lower() != 's':
        continuar = False

print('Saliendo del programa...')

def imprimir_menu():
    print("""Es importante mencionar que en este programa cualquier simbolo que
        no sea una letra en el alfabeto ingles separa una palabra de otra""")
    print('\n\nsMenu%s' % (separador, separador))
    print("""
1.- Entrada en consola
2.- Ingresar nombre del archivo
3.- Ver diagrama de estados
4.- Salir
""")
    try:
        opcion = int(input("Selecciona una opcion valida: "))
        return opcion
    except Exception as e:
        print('Error ', e)
        return 0

def entrada_consola():
    texto = input("Escribe el texto: ")
    texto += ' '
    palabras_ere = []
    verificar_palabras(texto, palabras_ere)
    print('\n', palabras_ere)

def entrada_archivo():
    archivo = input("Ingresa el nombre del archivo: ")
    try:
        archivo_abierto = open(archivo, 'r')
    except Exception as e:
        print('Error al abrir archivo: ', e)
        return 0

linea_palabras = []

```

```

num_linea = 1
palabras_ere = []
posiciones = []
for linea in archivo_abierto:
    verificar_palabras(linea, palabras_ere, posiciones)
    linea_palabras.append({'Linea': num_linea, 'Palabras': palabras_ere,
        'Posiciones': posiciones})
    num_linea += 1
    palabras_ere = []
    posiciones = []

imprimir_archivo(linea_palabras)
archivo_abierto.close()

def imprimir_archivo(linea_palabras):
    print('\n\n')
    for elemento in linea_palabras:
        if len(elemento['Palabras']) > 0:
            print('Numero de linea: ', elemento['Linea'])
            i = 0
            for palabra in elemento['Palabras']:
                print('\tPalabra: %s No. Palabra: %s' % (palabra,
                    elemento['Posiciones'][i]))
                i += 1

def ver_diagrama():
    print('Mostrando diagrama del automata. Cierre la ventana para continuar')
    try:
        diagrama_ere = Diagrama()
        diagrama_ere.master.title('Diagrama del automata ere')
        diagrama_ere.mainloop()
    except Exception as e:
        print("Error", e)

iniciar()

```

Archivo: automata_ere.py

```

#automata_ere.py
# -*- coding: utf-8 -*-
from __future__ import print_function

def verificar_palabras(texto, palabras_ere, posiciones = []):
    palabra_aux = ''
    estado = 0
    num_palabra = 1
    for simbolo in texto:
        simbolo_aux = simbolo.lower()

```



```

if simbolo == '\n':
    simbolo = '\\n'
print('-> delta(q%s,%s)' % (estado, simbolo), end="\t")

estado = automata(estado, simbolo_aux)

if (ord(simbolo_aux) < 123 and ord(simbolo_aux) > 96):
    palabra_aux += simbolo
    if estado == 4:
        estado = 0
    else:
        if estado == 4:
            palabras_ere.append(palabra_aux)
            posiciones.append(num_palabra)
            estado = 0
        palabra_aux = ''
        if simbolo == ' ':
            num_palabra += 1

def automata(estado, simbolo):
    if estado == 0:
        estado = estado_cero(simbolo)
    elif estado == 1:
        estado = estado_uno(simbolo)
    elif estado == 2:
        estado = estado_dos(simbolo)
    elif estado == 3:
        estado = estado_tres(simbolo)
    else:
        print('Simbolo extraño: ', simbolo)

    return estado

def estado_cero(simbolo):
    if simbolo == 'e':
        return 1
    else:
        return 0

def estado_uno(simbolo):
    if simbolo == 'r':
        return 2
    elif simbolo == 'e':
        return 1
    else:
        return 0

def estado_dos(simbolo):

```

```

    if simbolo == 'e':
        return 3
    else:
        return 0

def estado_tres(simbolo):
    if simbolo == 'r':
        return 2
    elif simbolo == 'e':
        return 1
    else:
        return 4

```

Archivo: diagrama_ere.py

```

#diagrama_ere.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import tkinter as tk

class Diagrama(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master, background='white')
        self.pack(fill=tk.BOTH, expand=tk.YES)
        self.dibujarDiagrama()
        self.centrarVentana()

    def dibujarDiagrama(self):
        canvas = tk.Canvas(self, bg='white')
        datos = {}
        datos['coordenadas'] = [100, 100, 200, 200]
        datos['canvas'] = canvas
        self.circulos_flechas(datos)

        self.crear_arco(canvas, [150, 50, 300, 150])
        canvas.create_text(150+75, 50-15, text='No e ni r')
        self.crear_arco(canvas, [320, 20, 585, 190])
        canvas.create_text(320+130, 50-10, text='e')
        # reflexiva
        extra = {'start': 30, 'extend': 235}
        self.crear_arco(canvas, [80, 90, 135, 150], extra)
        canvas.create_text(80-5, 90, text='No e')
        self.crear_arco(canvas, [230, 90, 285, 150], extra)
        canvas.create_text(230, 90, text='e')

        #de cabeza
        extra = {'start': 0, 'extend': -180}
        self.crear_arco(canvas, [150, 100, 600, 300], extra)

```

```

canvas.create_text(600-220, 300-10, text='No e ni r')
self.crear_arco(canvas, [175, 140, 430, 250], extra)
canvas.create_text(430-120, 300-40, text='No e')
self.crear_arco(canvas, [450, 170, 585, 225], extra)
canvas.create_text(585-50, 225+10, text='r')

canvas.pack(fill=tk.BOTH, expand=1)

def circulos_flechas(self, arg):
    coordenadas = arg['coordenadas']
    canvas = arg['canvas']
    for x in range(4):
        self.escribirTexto(canvas, x, coordenadas)
        self.dibujarCirculo(canvas, coordenadas)
        self.dibujarFlecha(canvas, [coordenadas[0]-50, 150,
            coordenadas[2]-100, 150])
        coordenadas[0] += 150
        coordenadas[2] += 150

    self.dibujarCirculo(canvas, [coordenadas[0]-150+5, 105,
        coordenadas[2]-155, 195]) # circulo interior

def escribirTexto(self, canvas, x, coordenadas):
    text = ''
    text_flecha = ''
    if x == 0:
        text = 'q%s' % x
        text_flecha = ''
    elif x == 1:
        text = 'q%s' % x
        text_flecha = 'e'
    elif x == 2:
        text = 'q%s' % x
        text_flecha = 'r'
    elif x == 3:
        text = 'q%s' % x
        text_flecha = 'e'
    else:
        print('otro')

    canvas.create_text(coordenadas[0]+50, coordenadas[1]+50, font=('15'),
        text=text)
    canvas.create_text(coordenadas[0]-25, coordenadas[1]+40, text=text_flecha)

def dibujarCirculo(self, canvas, coordenadas):
    circulo = canvas.create_oval(coordenadas)

def dibujarFlecha(self, canvas, coordenadas):

```

```

linea = canvas.create_line(coordenadas)
canvas.create_oval(coordenadas[2]-7, coordenadas[1]-7,
                  coordenadas[2]+7, coordenadas[1]+7, fill = 'black')

def crear_arco(self, canvas, coordenadas, extra=None):
    if extra != None:
        arco = canvas.create_arc(coordenadas, start=extra['start'],
                                extent=extra['extend'], style='arc')
        if extra['extend'] == -180:
            canvas.create_oval(coordenadas[0]-7, 100+100-7, coordenadas[0]+7,
                              100+100+7, fill = 'black')
    else:
        arco = canvas.create_arc(coordenadas, start=0, extent=180, style='arc')
        canvas.create_oval(coordenadas[0]-7, 100-7, coordenadas[0]+7, 100+7,
                          fill = 'black')

def centrarVentana(self):
    ancho, altura = 700, 350
    ancho_pantalla = self.winfo_screenwidth()
    altura_pantalla = self.winfo_screenheight()
    posicion_x = (ancho_pantalla - ancho)/2
    posicion_y = (altura_pantalla - altura)/2
    self.master.geometry('%dx%d+%d+%d' % (ancho, altura, posicion_x,
                                          posicion_y))

```

3.3. Pruebas

Pruebas de las opciones del menú.
Modo de consola.

```

MINGW32/c/Users/USER/Documents/tona/Git/teoria-computacional/automata-ere
USER@TONA MINGW32 ~/Documents/tona/Git/teoria-computacional/automata-ere (master)
$ python main.py
Es importante mencionar que en este programa cualquier simbolo que no sea una letra en el alfabeto ingl s separa una palabra d
e otra

*****Menu*****

1.- Entrada en consola
2.- Ingresar nombre del archivo
3.- Ver diagrama de estados
4.- Salir

Selecciona una opcion valida: 1
Escribe el texto: yesterday there were a big red bird right there where we met
-> delta(q0,y) -> delta(q0,e) -> delta(q1,s) -> delta(q0,t) -> delta(q0,e) -> delta(q1,r) -> delta(q2,d) -> delta(q0,a)
-> delta(q0,y) -> delta(q0,) -> delta(q0,t) -> delta(q0,h) -> delta(q0,e) -> delta(q1,r) -> delta(q2,e) -> delta(q3,)
-> delta(q0,w) -> delta(q0,e) -> delta(q1,r) -> delta(q2,e) -> delta(q3,) -> delta(q0,a) -> delta(q0,) -> delta(q0,b)
-> delta(q0,i) -> delta(q0,g) -> delta(q0,) -> delta(q0,r) -> delta(q0,e) -> delta(q1,d) -> delta(q0,) -> delta(q0,b)
-> delta(q0,i) -> delta(q0,r) -> delta(q0,d) -> delta(q0,) -> delta(q0,r) -> delta(q0,i) -> delta(q0,g) -> delta(q0,h)
-> delta(q0,t) -> delta(q0,) -> delta(q0,t) -> delta(q0,h) -> delta(q0,e) -> delta(q1,r) -> delta(q2,e) -> delta(q3,)
-> delta(q0,w) -> delta(q0,h) -> delta(q0,e) -> delta(q1,r) -> delta(q2,e) -> delta(q3,) -> delta(q0,w) -> delta(q0,e)
-> delta(q1,) -> delta(q0,m) -> delta(q0,e) -> delta(q1,t) -> delta(q0,)

['there', 'were', 'there', 'where']
*****
Reintentar s/n:

```

Figura 11: Historia del aut mata y las palabras con terminaci n 'ere'.

Modo archivo.

```
MINGW32/c/Users/USER/Documents/tona/Git/teoria-computacional/automata-ere
USER@TONA MINGW32 ~/Documents/tona/Git/teoria-computacional/automata-ere (master)
$ python main.py
Es importante mencionar que en este programa cualquier símbolo que no sea una letra en el alfabeto inglés separa una palabra de otra
ra

*****Menu*****
1.- Entrada en consola
2.- Ingresar nombre del archivo
3.- Ver diagrama de estados
4.- Salir

Selecciona una opcion valida: 2
Ingresa el nombre del archivo: tolkien.txt
-> delta(q0,L) -> delta(q0,a) -> delta(q0,m) -> delta(q0,e) -> delta(q1,n) -> delta(q0,t) -> delta(q0, ) -> delta(q0,F) ->
delta(q0,o) -> delta(q0,r) -> delta(q0, ) -> delta(q0,B) -> delta(q0,o) -> delta(q0,r) -> delta(q0,o) -> delta(q0,m) ->
delta(q0,i) -> delta(q0,r) -> delta(q0, ) -> delta(q0,B) -> delta(q0,y) -> delta(q0, ) -> delta(q0,J) -> delta(q0,.) ->
delta(q0,R) -> delta(q0,.) -> delta(q0,R) -> delta(q0, ) -> delta(q0,T) -> delta(q0,o) -> delta(q0,l) -> delta(q0,k) ->
delta(q0,i) -> delta(q0,e) -> delta(q1,n) -> delta(q0,\n) -> delta(q0,\n) -> delta(q0,T) -> delta(q0,h) -> delta(q0,r) ->
delta(q0,o) -> delta(q0,u) -> delta(q0,g) -> delta(q0,h) -> delta(q0, ) -> delta(q0,R) -> delta(q0,o) -> delta(q0,h) ->
delta(q0,a) -> delta(q0,n) -> delta(q0, ) -> delta(q0,o) -> delta(q0,v) -> delta(q0,e) -> delta(q1,r) -> delta(q2, ) ->
delta(q0,f) -> delta(q0,e) -> delta(q1,n) -> delta(q0, ) -> delta(q0,a) -> delta(q0,n) -> delta(q0,d) -> delta(q0, ) ->
delta(q0,f) -> delta(q0,i) -> delta(q0,e) -> delta(q1,l) -> delta(q0,d) -> delta(q0, ) -> delta(q0,w) -> delta(q0,h) ->
delta(q0,e) -> delta(q1,r) -> delta(q2,e) -> delta(q3, ) -> delta(q0,t) -> delta(q0,h) -> delta(q0,e) -> delta(q1, ) ->
delta(q0,l) -> delta(q0,o) -> delta(q0,n) -> delta(q0,g) -> delta(q0, ) -> delta(q0,g) -> delta(q0,r) -> delta(q0,a) ->
delta(q0,s) -> delta(q0,s) -> delta(q0, ) -> delta(q0,g) -> delta(q0,r) -> delta(q0,o) -> delta(q0,w) -> delta(q0,s) ->
delta(q0,.) -> delta(q0,\n) -> delta(q0,T) -> delta(q0,h) -> delta(q0,e) -> delta(q1, ) -> delta(q0,W) -> delta(q0,e) ->
delta(q1,s) -> delta(q0,t) -> delta(q0, ) -> delta(q0,W) -> delta(q0,i) -> delta(q0,n) -> delta(q0,d) -> delta(q0, ) ->
delta(q0,c) -> delta(q0,o) -> delta(q0,m) -> delta(q0,e) -> delta(q1,s) -> delta(q0, ) -> delta(q0,w) -> delta(q0,a) ->
delta(q0,l) -> delta(q0,k) -> delta(q0,i) -> delta(q0,n) -> delta(q0,g) -> delta(q0,.) -> delta(q0, ) -> delta(q0,a) ->
delta(q0,n) -> delta(q0,d) -> delta(q0, ) -> delta(q0,a) -> delta(q0,b) -> delta(q0,o) -> delta(q0,u) -> delta(q0, ) ->

delta(q1, ) -> delta(q0,h) -> delta(q0,i) -> delta(q0,m) -> delta(q0, ) -> delta(q0,u) -> delta(q0,p) -> delta(q0,o) ->
delta(q0,n) -> delta(q0, ) -> delta(q0,i) -> delta(q0,t) -> delta(q0,s) -> delta(q0, ) -> delta(q0,b) -> delta(q0,r) ->
delta(q0,e) -> delta(q1,a) -> delta(q0,s) -> delta(q0,t) -> delta(q0,.) -> delta(q0,') -> delta(q0,\n) -> delta(q0,') ->
delta(q0,o) -> delta(q0, ) -> delta(q0,B) -> delta(q0,o) -> delta(q0,r) -> delta(q0,o) -> delta(q0,m) -> delta(q0,i) ->
delta(q0,r) -> delta(q0,!) -> delta(q0, ) -> delta(q0,T) -> delta(q0,h) -> delta(q0,e) -> delta(q1, ) -> delta(q0,T) ->
delta(q0,o) -> delta(q0,w) -> delta(q0,e) -> delta(q1,r) -> delta(q2, ) -> delta(q0,o) -> delta(q0,f) -> delta(q0, ) ->
delta(q0,G) -> delta(q0,u) -> delta(q0,a) -> delta(q0,r) -> delta(q0,d) -> delta(q0, ) -> delta(q0,s) -> delta(q0,h) ->
delta(q0,a) -> delta(q0,l) -> delta(q0, ) -> delta(q0, ) -> delta(q0,e) -> delta(q1,v) -> delta(q0,e) -> delta(q1,r) ->
delta(q2, ) -> delta(q0,n) -> delta(q0,o) -> delta(q0,r) -> delta(q0,t) -> delta(q0,h) -> delta(q0,w) -> delta(q0,a) ->
delta(q0,r) -> delta(q0,d) -> delta(q0, ) -> delta(q0,g) -> delta(q0,a) -> delta(q0,z) -> delta(q0,e) -> delta(q1,\n) ->
delta(q0,T) -> delta(q0,o) -> delta(q0, ) -> delta(q0,R) -> delta(q0,a) -> delta(q0,u) -> delta(q0,r) -> delta(q0,o) ->
delta(q0,s) -> delta(q0,.) -> delta(q0, ) -> delta(q0,g) -> delta(q0,o) -> delta(q0,l) -> delta(q0,d) -> delta(q0,e) ->
delta(q1,n) -> delta(q0, ) -> delta(q0,R) -> delta(q0,a) -> delta(q0,u) -> delta(q0,r) -> delta(q0,o) -> delta(q0,s) ->
delta(q0,-) -> delta(q0,f) -> delta(q0,a) -> delta(q0,l) -> delta(q0,l) -> delta(q0,s) -> delta(q0,.) -> delta(q0, ) ->
delta(q0,u) -> delta(q0,n) -> delta(q0,t) -> delta(q0,i) -> delta(q0,l) -> delta(q0, ) -> delta(q0,t) -> delta(q0,h) ->
delta(q0,e) -> delta(q1, ) -> delta(q0,e) -> delta(q1,n) -> delta(q0,d) -> delta(q0, ) -> delta(q0,o) -> delta(q0,f) ->
delta(q0, ) -> delta(q0,d) -> delta(q0,a) -> delta(q0,y) -> delta(q0,s) -> delta(q0,.) -> delta(q0,') -> delta(q0,\n) ->

Numero de línea: 3
Palabra: where No. Palabra: 7
Numero de línea: 12
Palabra: where No. Palabra: 9
Numero de línea: 17
Palabra: Where No. Palabra: 1
Numero de línea: 18
Palabra: where No. Palabra: 5
Palabra: there No. Palabra: 11
Numero de línea: 29
Palabra: There No. Palabra: 8
*****
Reintentar s/n:
```

Figura 12: Parte de la historia del autómata y las palabras con terminación 'ere'.

Diagrama.

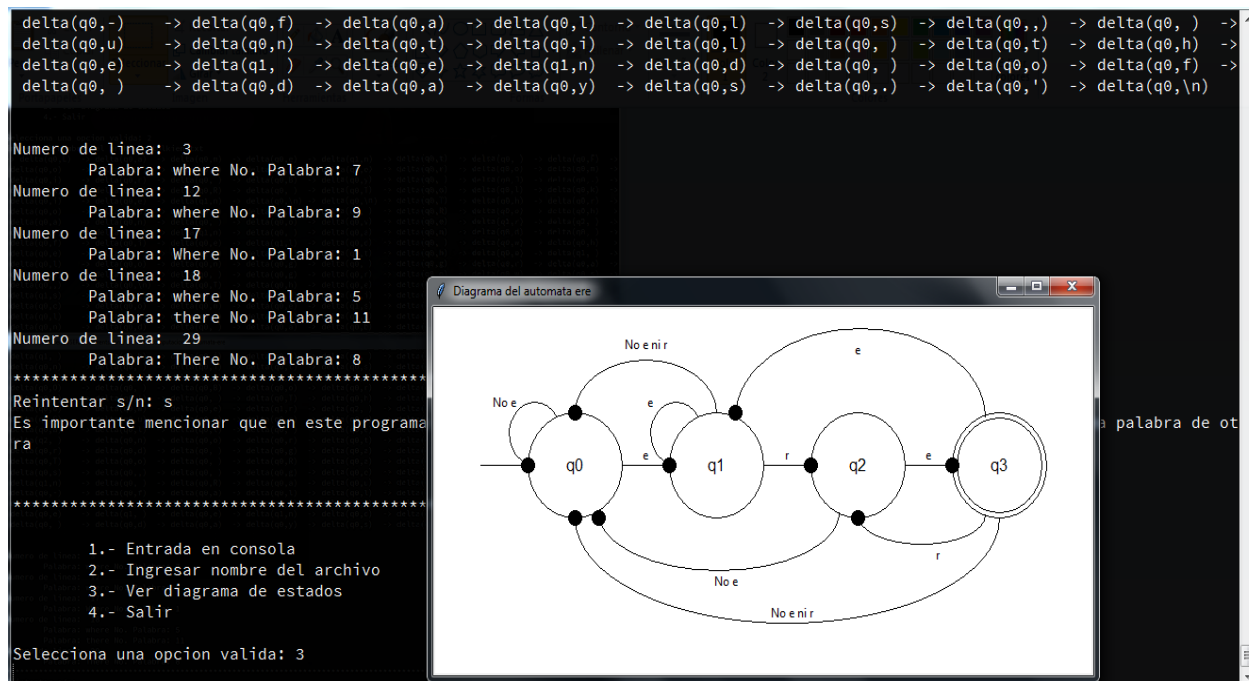


Figura 13: Diagrama de transiciones del autómata 'ere'.

4. AFD Paridad en números binarios

4.1. Descripción del problema

Diseñar y programar un autómata finito determinista que acepte el lenguaje:

$$L = \{w \mid w \text{ tiene un número par de ceros y un número par de unos}\}[1]$$

Es decir, los números binarios de entrada se generan de manera automática (cadena de longitud $n \mid 1 \leq n \leq 1000$) o manual y después se imprime si es una cadena válida o no y en ambos casos imprimir su historia. Además, mostrar el siguiente diagrama.

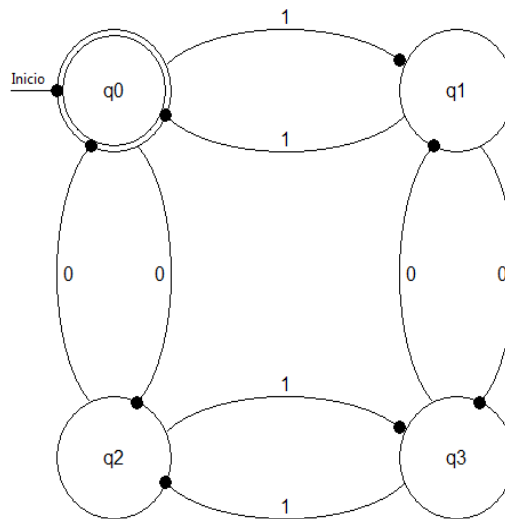


Figura 14: Diagrama de transiciones del autómata. [1]

4.2. Código

El código fue realizado en Python 3.5.

Archivo: main_paridad.py

```
#main_paridad.py
# -*- coding: utf-8 -*-
from __future__ import print_function
from automata_paridad import ejecutar_automata
from diagrama_paridad import Diagrama
import random

separador = '*' * 50
def iniciar():
    continuar = True
    while continuar:
        opcion = imprimir_menu()
```

```

    if opcion == 1:
        ejecutar_manual()
    elif opcion == 2:
        ejecutar_random()
    elif opcion == 3:
        ver_diagrama()
    else:
        break
    print('\n', separador)
    opcion = input("Reintentar s/n: ")
    if opcion.lower() != 's':
        continuar = False

print('Saliendo del programa...')

def imprimir_menu():
    print('\n\n%sMenu%s' % (separador, separador))
    print("""
1.- Entrada en consola (Manual)
2.- Numero aleatorio (automatico)
3.- Ver diagrama de transiciones
4.- Salir
""")
    try:
        opcion = int(input("Selecciona una opcion valida: "))
        return opcion
    except Exception as e:
        print('Error ', e)
        return 0

def ejecutar_random():
    i = 0
    longitud_random = random.randint(1, 1000)
    numero_binario = ''
    while i < longitud_random:
        numero_binario += random.choice(['0', '1'])
        i += 1

    print("El numero aleatorio es: ", numero_binario)
    probar_paridad(numero_binario)

def ejecutar_manual():
    numero_binario = input("Escribe un numero binario: ")
    probar_paridad(numero_binario)

def probar_paridad(numero_binario):
    resultado = ejecutar_automata(numero_binario)
    print('\n')

```



```

if resultado:
    print('El numero %s, es valido' % numero_binario)
else:
    print('El numero: %s, no es valido' % numero_binario)

def ver_diagrama():
    print('Mostrando diagrama del automata. Cierre la ventana para continuar')
    try:
        diagrama_paridad = Diagrama()
        diagrama_paridad.master.title('Diagrama del automata paridad')
        diagrama_paridad.mainloop()
    except Exception as e:
        print("Error", e)

iniciar()

```

Archivo: automata_paridad.py

```

#automata_paridad.py
def ejecutar_automata(cadena):
    estado = 0
    for simbolo in cadena:
        print('-> delta(q%s,%s)' % (estado, simbolo), end="\t")
        estado = automata(estado, simbolo)
        if estado == -1:
            break
    if estado == 0:
        print('-> delta(q%s, )' % estado, end="\t")
        return True
    return False

def automata(estado, simbolo):
    if estado == 0:
        estado = estado_cero(simbolo)
    elif estado == 1:
        estado = estado_uno(simbolo)
    elif estado == 2:
        estado = estado_dos(simbolo)
    elif estado == 3:
        estado = estado_tres(simbolo)
    else:
        print('Simbolo extraño ', simbolo)
        return -1
    return estado

def estado_cero(simbolo):
    if simbolo == '0':
        return 2

```

```

elif simbolo == '1':
    return 1
else:
    return -1

def estado_uno(simbolo):
    if simbolo == '0':
        return 3
    elif simbolo == '1':
        return 0
    else:
        return -1

def estado_dos(simbolo):
    if simbolo == '0':
        return 0
    elif simbolo == '1':
        return 3
    else:
        return -1

def estado_tres(simbolo):
    if simbolo == '0':
        return 1
    elif simbolo == '1':
        return 2
    else:
        return -1

```

Archivo: diagrama_paridad.py

```

#diagrama_paridad.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import tkinter as tk

class Diagrama(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master, background='white')
        self.pack(fill=tk.BOTH, expand=tk.YES)
        self.canvas = tk.Canvas(self, bg='white')
        self.canvas.pack(fill=tk.BOTH, expand=1)
        self.dibujarDiagrama()
        self.centrarVentana()

    def dibujarDiagrama(self):
        self.dibujarFlecha([10, 100, 50, 100])
        self.dibujarCirculo([55, 55, 145, 145])

```

```

for x in range(4):
    if x == 0:
        coordenadas = [50, 50, 150, 150]
        text_circulo = 'q%s' % x
        self.dibujarFlechaHorizontal(coordenadas[:])
        self.dibujarFlechaVertical(coordenadas[:])
    elif x == 1:
        coordenadas = [350, 50, 450, 150]
        text_circulo = 'q%s' % x
        self.dibujarFlechaVertical(coordenadas[:])
    elif x == 2:
        coordenadas = [50, 350, 150, 450]
        text_circulo = 'q%s' % x
        self.dibujarFlechaHorizontal(coordenadas[:])
    elif x == 3:
        coordenadas = [350, 350, 450, 450]
        text_circulo = 'q%s' % x

    else:
        print('Na')

    self.dibujarCirculo(coordenadas)
    self.canvas.create_text(coordenadas[0]+50, coordenadas[1]+50,
        font=('15'), text=text_circulo)

def dibujarCirculo(self, arg):
    circulo = self.canvas.create_oval(arg)

def dibujarFlechaHorizontal(self, coordenadas):
    coordenadas[0] += 85
    coordenadas[2] += 215
    x = ((coordenadas[2] - coordenadas[0])/2) + coordenadas[0]

    self.canvas.create_text(x, coordenadas[1]-10, font=('15'), text='1')
    self.canvas.create_text(x, coordenadas[3]-10, font=('15'), text='1')

    self.canvas.create_arc(coordenadas, start=25, extent=130, style='arc')
    self.canvas.create_arc(coordenadas, start=-25, extent=-130, style='arc')

    self.canvas.create_oval(coordenadas[2]-5-15, coordenadas[1]-5+25,
        coordenadas[2]+5-15, coordenadas[1]+5+25, fill = 'black')
    self.canvas.create_oval(coordenadas[0]-5+10, coordenadas[3]-5-30,
        coordenadas[0]+5+10, coordenadas[3]+5-30, fill = 'black')

def dibujarFlechaVertical(self, coordenadas):
    coordenadas[1] += 85
    coordenadas[3] += 215

```

```

y = ((coordenadas[3] - coordenadas[1])/2) + coordenadas[1]

self.canvas.create_text(coordenadas[0]+10, y, font=('15'), text='0')
self.canvas.create_text(coordenadas[2]-10, y, font=('15'), text='0')

self.canvas.create_arc(coordenadas, start=-65, extent=130, style='arc')
self.canvas.create_arc(coordenadas, start=115, extent=130, style='arc')

self.canvas.create_oval(coordenadas[0]-5+30, coordenadas[3]-5-220,
    coordenadas[0]+5+30, coordenadas[3]+5-220, fill = 'black')
self.canvas.create_oval(coordenadas[2]-5-30, coordenadas[1]-5+220,
    coordenadas[2]+5-30, coordenadas[1]+5+220, fill = 'black')

def dibujarFlecha(self, coordenadas):
    linea = self.canvas.create_line(coordenadas)
    self.canvas.create_text(coordenadas[0]+15, coordenadas[1]-10,
        text='Inicio')
    self.canvas.create_oval(coordenadas[2]-5, coordenadas[1]-5,
        coordenadas[2]+5, coordenadas[1]+5, fill = 'black')

def centrarVentana(self):
    ancho, altura = 500, 500
    ancho_pantalla = self.winfo_screenwidth()
    altura_pantalla = self.winfo_screenheight()
    posicion_x = (ancho_pantalla - ancho)/2
    posicion_y = (altura_pantalla - altura)/2
    self.master.geometry('%dx%d+%d+%d' % (ancho, altura, posicion_x,
        posicion_y))

```

4.3. Pruebas

Pruebas de las opciones del menú.
Modo manual.

```
*****Menu*****
1.- Entrada en consola (Manual)
2.- Numero aleatorio (automatico)
3.- Ver diagrama de transiciones
4.- Salir

Selecciona una opcion valida: 1
Escribe un numero binario: 01010100011101
-> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) ->
delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1)

El numero: 01010100011101, no es valido

*****
Reintentar s/n: s

*****Menu*****
1.- Entrada en consola (Manual)
2.- Numero aleatorio (automatico)
3.- Ver diagrama de transiciones
4.- Salir

Selecciona una opcion valida: 1
Escribe un numero binario: 01010100011101
-> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) ->
delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) ->
delta(q0, )

El numero 01010100011101, es valido
```

Figura 15: Historia del autómata

Modo automático.

```
USER@TONA MINGW32 ~/Documents/tona/Git/teoria-computacional/automata-paridad (master)
$ python main_paridad.py
*****Menu*****
1.- Entrada en consola (Manual)
2.- Numero aleatorio (automatico)
3.- Ver diagrama de transiciones
4.- Salir

Selecciona una opcion valida: 2
El numero aleatorio es: 101011010010111100001110
-> delta(q0,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1) ->
delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1) -> delta(q1,1) ->
delta(q0,0) -> delta(q2,0) -> delta(q0,0) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0)

El numero: 101011010010111100001110, no es valido

*****
Reintentar s/n:
```

Figura 16: Historia del autómata

Diagrama.

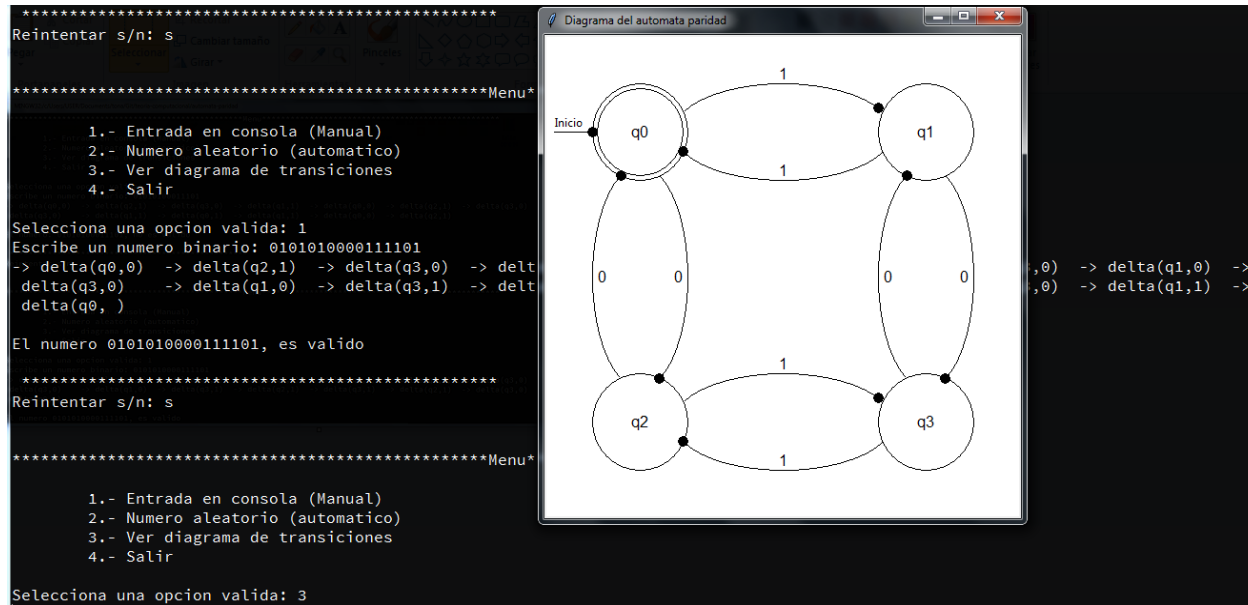


Figura 17: Diagrama de transiciones del autómata

5. Protocolo de transmisión

5.1. Descripción del problema

Desarrollar un programa que genere 50 cadenas de 32 caracteres que sean guardadas en un archivo, para después ser evaluadas por un autómata, en este caso el de paridad binaria y guardar las cadenas binarias validas en otro archivo, siguiendo el siguiente diagrama.

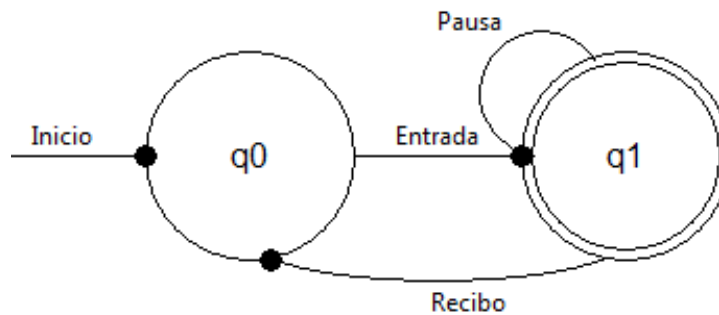


Figura 18: Diagrama de transiciones del autómata. [2]

5.2. Código

El código fue realizado en Python 3.5.
Archivo: main_protocolo.py

```
#main_protocolo.py
# -*- coding: utf-8 -*-
from __future__ import print_function
from diagrama_protocolo import Diagrama
from protocolo import generar_cadenas, pausar_protocolo, verificar_cadenas
import random

separador = '*'*50

def iniciar():
    continuar = True
    while continuar:
        opcion = imprimir_menu()
        if opcion == 1:
            correr_protocolo()
        elif opcion == 2:
            ver_diagrama()
        else:
            break # Sal del programa
    print('*' * 100)
```

```

    opcion = input("Reintentar s/n: ")
    if opcion.lower() != 's':
        continuar = False

print('\nSaliendo del programa...')

def imprimir_menu():
    print('\n\n%sMenu%s' % (separador, separador))
    print("""
1.- Correr protocolo
2.- Ver diagrama
3.- Salir
""")
    try:
        opcion = int(input("Selecciona una opcion valida: "))
        return opcion
    except Exception as e:
        print('Error ', e)
        return 0

def correr_protocolo():
    condicion = True
    ARCHIVO_PALABRAS = 'palabras.txt'
    ARCHIVO_PALABRAS_VALIDAS = 'validas.txt'
    TIEMPO_PAUSA = 2
    palabras = []
    while condicion:
        archivo = open('validas.txt', 'w')
        print('\nGenerando cadenas...')
        generar_cadenas(ARCHIVO_PALABRAS)
        print('\nEnviando las cadenas...')
        pausar_protocolo(TIEMPO_PAUSA)
        print('\nVerificando las cadenas las cadenas...')
        verificar_cadenas(ARCHIVO_PALABRAS, palabras)
        print('\nRegresando las cadenas...')
        print('\nPalabras validas: ', palabras)
        for palabra in palabras:
            archivo.write(palabra)
            archivo.write(' ')
        palabras = []
        condicion = random.choice([True, False])
        archivo.close()

def ver_diagrama():
    print('Mostrando diagrama del automata. Cierre la ventana para continuar')
    try:
        diagrama_ere = Diagrama()
        diagrama_ere.master.title('Diagrama del protocolo')

```



```
    diagrama_ere.mainloop()
except Exception as e:
    print("Error", e)
```

```
iniciar()
```

Archivo: protocolo.py

```
#protocolo.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import time, random

def pausar_protocolo(segundos):
    time.sleep(segundos)

def generar_cadenas(archivo):
    try:
        archivo_abierto = open(archivo, 'a')
    except Exception as e:
        print('Error al abrir archivo ', e)
    i = 0
    LONGITUD = 32
    numero_binario = ''

    for x in range(50):
        while i < LONGITUD:
            numero_binario += random.choice(['0', '1'])
            i += 1
        archivo_abierto.write(numero_binario)
        archivo_abierto.write(' ')
        i = 0
        numero_binario = ''

    archivo_abierto.close()

def verificar_cadenas(archivo, palabras):
    try:
        archivo_abierto = open(archivo, 'r')
    except Exception as e:
        print('Error al abrir archivo ', e)

    texto = archivo_abierto.read()
    estado = 0
    palabra_aux = ''
    for simbolo in texto:
        print('-> delta(q%s,%s)' % (estado, simbolo), end="\t")
```

```

    if simbolo == ' ':
        if estado == 0:
            palabras.append(palabra_aux)
            palabra_aux = ''
            estado = 0
        else:
            palabra_aux += simbolo
            estado = automata(estado, simbolo)

def automata(estado, simbolo):
    if estado == 0:
        estado = estado_cero(simbolo)
    elif estado == 1:
        estado = estado_uno(simbolo)
    elif estado == 2:
        estado = estado_dos(simbolo)
    elif estado == 3:
        estado = estado_tres(simbolo)
    else:
        print('Simbolo extraño ', simbolo)
        return -1
    return estado

def estado_cero(simbolo):
    if simbolo == '0':
        return 2
    elif simbolo == '1':
        return 1
    else:
        return -1

def estado_uno(simbolo):
    if simbolo == '0':
        return 3
    elif simbolo == '1':
        return 0
    else:
        return -1

def estado_dos(simbolo):
    if simbolo == '0':
        return 0
    elif simbolo == '1':
        return 3
    else:
        return -1

def estado_tres(simbolo):

```

```

if simbolo == '0':
    return 1
elif simbolo == '1':
    return 2
else:
    return -1

```

Archivo: diagrama_protocolo.py

```

#diagrama_protocolo.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import tkinter as tk

class Diagrama(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master, background='white')
        self.pack(fill=tk.BOTH, expand=tk.YES)
        self.canvas = tk.Canvas(self, bg='white')
        self.canvas.pack(fill=tk.BOTH, expand=1)
        self.dibujarDiagrama()
        self.centrarVentana()

    def dibujarDiagrama(self):
        coordenadas = [70, 100, 170, 200]
        for x in range(2):
            self.escribirTexto(x, coordenadas)
            self.dibujarCirculo(coordenadas)
            self.dibujarFlecha([coordenadas[0]-80, 150, coordenadas[2]-100, 150])
            coordenadas[0] += 180
            coordenadas[2] += 180

        self.dibujarCirculo([coordenadas[0]-180+5, 105, coordenadas[2]-180-5,
            195]) # circulo interior
        self.canvas.create_arc(230, 90, 290, 150, start=30, extent=235,
            style='arc')
        self.canvas.create_text(225, 85, text='Pausa')

        self.canvas.create_arc(120, 170, 300, 210, start=-30, extent=-120,
            style='arc')
        self.canvas.create_oval(130-5, 200-5, 130+5, 200+5, fill = 'black')
        self.canvas.create_text(225, 220, text='Recibo')

    def escribirTexto(self, x, coordenadas):
        text = ''
        text_flecha = ''
        if x == 0:
            text = 'Listo'

```

```

        text_flecha = 'Inicio'
    elif x == 1:
        text = 'Enviando'
        text_flecha = 'Entrada'
    else:
        print('otro')

    self.canvas.create_text(coordenadas[0]+50, coordenadas[1]+50,
        font=('15'), text=text)
    self.canvas.create_text(coordenadas[0]-40, coordenadas[1]+40,
        text=text_flecha)

def dibujarCirculo(self, arg):
    circulo = self.canvas.create_oval(arg)

def dibujarFlecha(self, coordenadas):
    linea = self.canvas.create_line(coordenadas)
    self.canvas.create_oval(coordenadas[2]-5, coordenadas[1]-5,
        coordenadas[2]+5, coordenadas[1]+5, fill = 'black')

def centrarVentana(self):
    ancho, altura = 400, 300
    ancho_pantalla = self.winfo_screenwidth()
    altura_pantalla = self.winfo_screenheight()
    posicion_x = (ancho_pantalla - ancho)/2
    posicion_y = (altura_pantalla - altura)/2
    self.master.geometry('%dx%d+%d+%d' % (ancho, altura, posicion_x,
        posicion_y))

```

5.3. Pruebas

Pruebas de las opciones del menú.
Ejecutar protocolo.

```
Símbolo del sistema - python main_protocolo.py

*****Menu*****

1.- Correr protocolo
2.- Ver diagrama
3.- Salir

Selecciona una opcion valida: 1

Generando cadenas...

Enviando las cadenas...

Verificando las cadenas las cadenas...
-> delta(q0,1) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,1) -> delta(q0,0) -> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1)
-> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q3, ) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1)
-> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q1,1) -> delta(q0,1) -> delta(q3,1)
-> delta(q3,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0)
-> delta(q3,0) -> delta(q1,0) -> delta(q3, ) -> delta(q0,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q1,0) -> delta(q2,1)
-> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q3,0)
-> delta(q0,0) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3, )
-> delta(q0,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,1)
-> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,0)
-> delta(q0,1) -> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,0)
-> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3, ) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,1) -> delta(q2,0) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1) -> delta(q3, ) -> delta(q0,0) -> delta(q2,1)
-> delta(q3,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1)
-> delta(q1,1) -> delta(q0,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q2,0)
-> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q3,0)
-> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3, ) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,0) -> delta(q1,1) -> delta(q0,1)
-> delta(q1,1) -> delta(q0,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q2,0)
-> delta(q3,0) -> delta(q1,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q2,0)
-> delta(q1,1) -> delta(q0,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q2,0)
-> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q3,0)
-> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3, ) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0)
-> delta(q1,1) -> delta(q2,0) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1) -> delta(q3, ) -> delta(q0,0) -> delta(q2,1)
-> delta(q3,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1)
-> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1)
-> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0)
-> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,0)
-> delta(q1,0) -> delta(q3,1) -> delta(q2,0) -> delta(q0,0) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1)
-> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q1,1) -> delta(q0, )
-> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1) -> delta(q0,0)
-> delta(q3,0) -> delta(q1,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,1) -> delta(q0,0) -> delta(q2,1) -> delta(q3,1)
-> delta(q2,0) -> delta(q0,0) -> delta(q2,0) -> delta(q0,1) -> delta(q1,1) -> delta(q0, ) -> delta(q0,1) -> delta(q1,1) -> delta(q0,1)
-> delta(q1,1) -> delta(q0,0) -> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0)
-> delta(q3,1) -> delta(q2,1) -> delta(q3,1) -> delta(q2,0) -> delta(q0,1) -> delta(q1,0) -> delta(q3,0) -> delta(q1,1) -> delta(q0,1)
-> delta(q1,0) -> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1) -> delta(q3,0) -> delta(q1,0) -> delta(q3,1) -> delta(q2,1)
-> delta(q3,0) -> delta(q1,1) -> delta(q0, )

Regresando las cadenas...

Palabras validas: ['0111110111001111011111011111101', '1010011011110011101000000110011', '001001001000101001011010111111', '10110010111111',
'001010010011001101', '11010111001000100001001001001000', '1001100100000100111000111010100', '01110111001011100011111100100100', '00011000110101010000100100100100', '0010001100100100100100100100100100', '00100000111011000010110011101011', '01111001000100010011011000001000', '001111110010101011101100001100', '00010000111101100010110011101011', '1001101111010110000011001001101', '0100111101010111101101100000100', '1110101111101010111100001100', '0000000111101100010110011101011', '101010100010100101010101110000', '01011111010000000010110110111', '001111100010101011110001101101', '111000011001001010101011101', '1101010001101111000111001100011', '11110111110011101001100011001101']

Reintentar s/n: █
```

Figura 19: Parte de la historia del protocolo. [2]

```

diagrama_protocolo.py protocolo.py validas.txt
1 0111101110011110111110011111101 10100110111100111010000000110011
• 0010010010001010010110101011111 1011001011111001010010011001101
• 11010111001000100001001001001000 10011001000001001111000111010100
• 01110111001011100011111100100100 00011000110101101000110011001010
• 00100111100111001110011100110011 1101101011111101010010100011011
• 01111001000100010011011000000100 0011111100100000011001010101110
• 00010000111101010000001110110000 10010110111010110000011001001101
• 01001111010101011110110100000100 1110101111110101011011100001100
• 00000001111011000101100111101011 10101010001010100101010101110000
• 0101111010000000001011011011011 00111110000010110111110001101101
• 11100001100100101001010100111101 11010100011101111000111001100011
• 1110111110011101001100011001101

```

Figura 20: Palabras validas.[2]

Diagrama.

```

Simbolo del sistema - python main_protocolo.py
1101000110011001010', '0010011110011100111011000110011', '110110111111101010010100011011', '01111001000100010011011000000100', '001111110010
000001100101011110', '0001000011110101000000110110000', '10010110111010110000011001001101', '010011110101011110110100000100', '11101011111
10101011101100001100', '00000001111011000101100111101011', '101010100010101001010101110000', '0101111101000000001011011011', '00111111000
0010110111110001101101', '11100001100100101001010100111101', '11010100011101111000111001100011', '11110111110011101001100011001101']
*****
Reintentar s/n: n
Saliendo del programa...
C:\Users\USER\Documents\tona\Git\teoria-computacional\protocolo>python main_protocolo.py
*****Menu*****
1.- Correr protocolo
2.- Ver diagrama
3.- Salir
Selecciona una opcion valida: 3
Saliendo del programa...
C:\Users\USER\Documents\tona\Git\teoria-computacional\
*****Menu*****
1.- Correr protocolo
2.- Ver diagrama
3.- Salir
Selecciona una opcion valida: 2
Mostrando diagrama del automata. Cierre la ventana para continuar

```

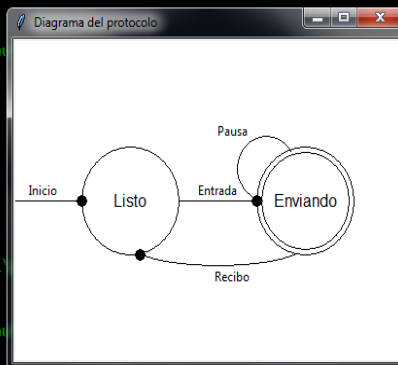


Figura 21: Diagrama de transiciones del autómata.

6. AFND Números binarios con terminación '01'

6.1. Descripción del problema

Desarrollar un autómata finito no determinista, que acepte todas y sólo las cadenas formadas por ceros y unos que terminan en 01. Asimismo, imprimir la tabla de transiciones (historia) y que la entrada de cadenas sea de forma manual o automática, la cadena aleatoria debe de tener una longitud $n \mid 1 \leq n \leq 1000$. Y que contenga la opción de mostrar el siguiente diagrama.

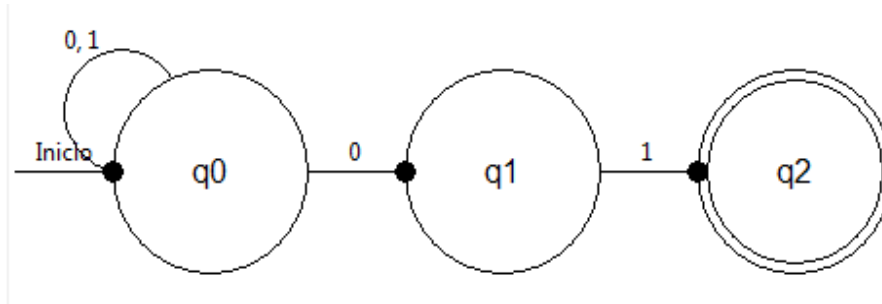


Figura 22: Diagrama de transiciones del autómata. [1]

6.2. Código

El código fue realizado en Python 3.5.
Archivo: main_cero_uno.py

```
#main_cero_uno.py
# -*- coding: utf-8 -*-
from __future__ import print_function
from automata_cero_uno import automata
from diagrama_cero_uno import Diagrama
import random

separador = '*'*50

def iniciar():
    continuar = True
    while continuar:
        opcion = imprimir_menu()
        if opcion == 1:
            entrada_consola()
        elif opcion == 2:
            ejecutar_random()
        elif opcion == 3:
            ver_diagrama()
        else:
```

```

        break # Sal del programa
    print('*' * 100)
    opcion = input("Reintentar s/n: ")
    if opcion.lower() != 's':
        continuar = False

    print('Saliendo del programa...')

def imprimir_menu():
    print('\n\n%sMenu%s' % (separador, separador))
    print("""
1.- Entrada en consola (Manual)
2.- Aleatorio (Automatico)
3.- Ver diagrama
4.- Salir
""")
    try:
        opcion = int(input("Selecciona una opcion valida: "))
        return opcion
    except Exception as e:
        print('Error ', e)
        return 0

def entrada_consola():
    texto = input("Escribe el numero binario: ")
    automata(texto)

def ejecutar_random():
    i = 0
    longitud_random = random.randint(1, 1000)
    numero_binario = ''
    while i < longitud_random:
        numero_binario += random.choice(['0', '1'])
        i += 1

    print("El numero aleatorio es: ", numero_binario)
    automata(numero_binario)

def ver_diagrama():
    print('Mostrando diagrama del automata. Cierre la ventana para continuar')
    try:
        diagrama_ere = Diagrama()
        diagrama_ere.master.title('Diagrama del automata cero-uno')
        diagrama_ere.mainloop()
    except Exception as e:
        print("Error", e)

```


iniciar()

Archivo: automata_cero_uno.py

```
#automata_cero_uno.py
# -*- coding: utf-8 -*-
from __future__ import print_function

def automata(text):
    array_tabla = []
    array = ['0']
    i = 0
    array_tabla.append(['0'])
    temporal_cero = []
    temporal_otro = []
    temporal = []
    for simbolo in text:
        array = array_tabla[i]
        for estado_evaluacion in array:
            if simbolo == '0' and estado_evaluacion == '0':
                temporal_cero = evaluar_estados(estado_evaluacion, simbolo)
            else:
                temporal_otro.append(evaluar_estados(estado_evaluacion, simbolo))

            if(len(temporal_cero) != 0):
                temporal.append(temporal_cero[0])
                temporal += temporal_otro[:]
                temporal.append(temporal_cero[1])
            else:
                temporal += temporal_cero[:]
                temporal += temporal_otro[:]
        array_tabla.append(temporal[:])
        temporal_cero = []
        temporal_otro = []
        temporal = []
        i += 1

rellenar_tabla(array_tabla, text)
letra = 0
text += ' '
print('')
for linea in array_tabla:
    print(text[letra], end=' | ')
    for valor in linea:
        if(valor == '-1'):
            print("x", end=' ')
```

```

        else:
            print("q%s" %valor, end=' ')
        print()
        letra +=1

if array_tabla[len(text)-1][len(array_tabla[len(text)-1])-1] == '2':
    print('El numero: %s es una cadena valida' % text)
else:
    print('El numero: %s NO es una cadena valida' % text)

def rellenar_tabla(array_tabla, text):
    longitud = len(text)
    ultima_fila = len(array_tabla[longitud])
    ceros = False
    if array_tabla[longitud][ultima_fila-1] == '2':
        ceros = True
    for fila in array_tabla:
        while True:
            if len(fila) >= ultima_fila:
                break
            if (len(fila)+1 >= ultima_fila) and ceros:
                fila.append('0')
            else:
                fila.append('-1')

def evaluar_estados(estado, simbolo):
    if estado == '0':
        estado = estado_cero(simbolo)
    elif estado == '1':
        estado = estado_uno(simbolo)
    elif estado == '2':
        estado = estado_dos(simbolo)

    return estado

def estado_cero(simbolo):
    if simbolo == '1':
        return '0'
    elif simbolo == '0':
        return ['0', '1']

def estado_uno(simbolo):
    if simbolo == '1':
        return '2'
    else:
        return '-1'

def estado_dos(simbolo):

```

```
return '-1'
```

Archivo: diagrama_cero_uno.py

```
#diagrama_cero_uno.py
# -*- coding: utf-8 -*-
from __future__ import print_function
import tkinter as tk

class Diagrama(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master, background='white')
        self.pack(fill=tk.BOTH, expand=tk.YES)
        self.canvas = tk.Canvas(self, bg='white')
        self.canvas.pack(fill=tk.BOTH, expand=1)
        self.dibujarDiagrama()
        self.centrarVentana()

    def dibujarDiagrama(self):
        coordenadas = [55, 100, 155, 200]
        for x in range(3):
            self.escribirTexto(x, coordenadas)
            self.dibujarCirculo(coordenadas)
            self.dibujarFlecha([coordenadas[0]-50, 150, coordenadas[2]-100,
                               150])
            coordenadas[0] += 150
            coordenadas[2] += 150

        self.dibujarCirculo([coordenadas[0]-150+5, 105, coordenadas[2]-155,
                             195]) # circulo interior
        self.canvas.create_arc(30, 90, 90, 150, start=30, extent=235,
                               style='arc')
        self.canvas.create_text(40, 85, text='0, 1')

    def escribirTexto(self, x, coordenadas):
        text = ''
        text_flecha = ''
        if x == 0:
            text = 'q%s' % x
            text_flecha = 'Inicio'
        elif x == 1:
            text = 'q%s' % x
            text_flecha = '0'
        elif x == 2:
            text = 'q%s' % x
            text_flecha = '1'
        else:
            print('otro')
```

```

self.canvas.create_text(coordenadas[0]+50, coordenadas[1]+50,
    font=('15'), text=text)
self.canvas.create_text(coordenadas[0]-25, coordenadas[1]+40,
    text=text_flecha)

def dibujarCirculo(self, arg):
    circulo = self.canvas.create_oval(arg)

def dibujarFlecha(self, coordenadas):
    linea = self.canvas.create_line(coordenadas)
    self.canvas.create_oval(coordenadas[2]-5, coordenadas[1]-5,
        coordenadas[2]+5, coordenadas[1]+5, fill = 'black')

def centrarVentana(self):
    ancho, altura = 500, 300
    ancho_pantalla = self.winfo_screenwidth()
    altura_pantalla = self.winfo_screenheight()
    posicion_x = (ancho_pantalla - ancho)/2
    posicion_y = (altura_pantalla - altura)/2
    self.master.geometry('%dx%d+%d+%d' % (ancho, altura, posicion_x,
        posicion_y))

```

6.3. Pruebas

Pruebas de las opciones del menú.
Modo manual

```
MINGW32/c/Users/USER/Documents/tona/Git/teoria-computacional/automata-cero-uno
USER@TONA MINGW32 ~/Documents/tona/Git/teoria-computacional/automata-cero-uno (master)
$ python main_cero_uno.py

*****Menu*****
1.- Entrada en consola (Manual)
2.- Aleatorio (Automatico)
3.- Ver diagrama
4.- Salir

Selecciona una opcion valida: 1
Escribe el numero binario: 01010100011101

0 | q0 x x x x x q0
1 | q0 q1 x x x x x q0
0 | q0 q2 x x x x x q0
1 | q0 x q1 x x x x x q0
0 | q0 x q2 x x x x x q0
1 | q0 x x q1 x x x x x q0
0 | q0 x x q2 x x x x x q0
0 | q0 x x x q1 x x x x x q0
1 | q0 x x x x q1 x x x x x q0
1 | q0 x x x x x q2 q0
1 | q0 x x x x x x q0
0 | q0 x x x x x x q0
1 | q0 x x x x x x q1
0 | q0 x x x x x x q2

El numero: 01010100011101 es una cadena valida
Reintentar s/n: |
```

Figura 23: Historia del autómata en una tabla

Modo automático

```
MINGW32/c/Users/USER/Documents/tona/Git/teoria-computacional/automata-cero-uno
Reintentar s/n: s

*****Menu*****
1.- Entrada en consola (Manual)
2.- Aleatorio (Automatico)
3.- Ver diagrama
4.- Salir

Selecciona una opcion valida: 2
El numero aleatorio es: 01100101101110111 Fila 1]

0 | q0 x x x x x
1 | q0 q1 x x x x
1 | q0 q2 x x x x
0 | q0 x x x x x
0 | q0 x q1 x x x
1 | q0 x x q1 x x x
0 | q0 x x q2 x x x
1 | q0 x x x q1 x x
1 | q0 x x x q2 x x
1 | q0 x x x x q1 x
1 | q0 x x x x q2 x
1 | q0 x x x x x
0 | q0 x x x x x
1 | q0 x x x x x q1
1 | q0 x x x x x q2
1 | q0 x x x x x
0 | q0 x x x x x

El numero: 01100101101110111: NO es una cadena valida
Reintentar s/n: |
```

Figura 24: Historia del autómata en una tabla

Diagrama

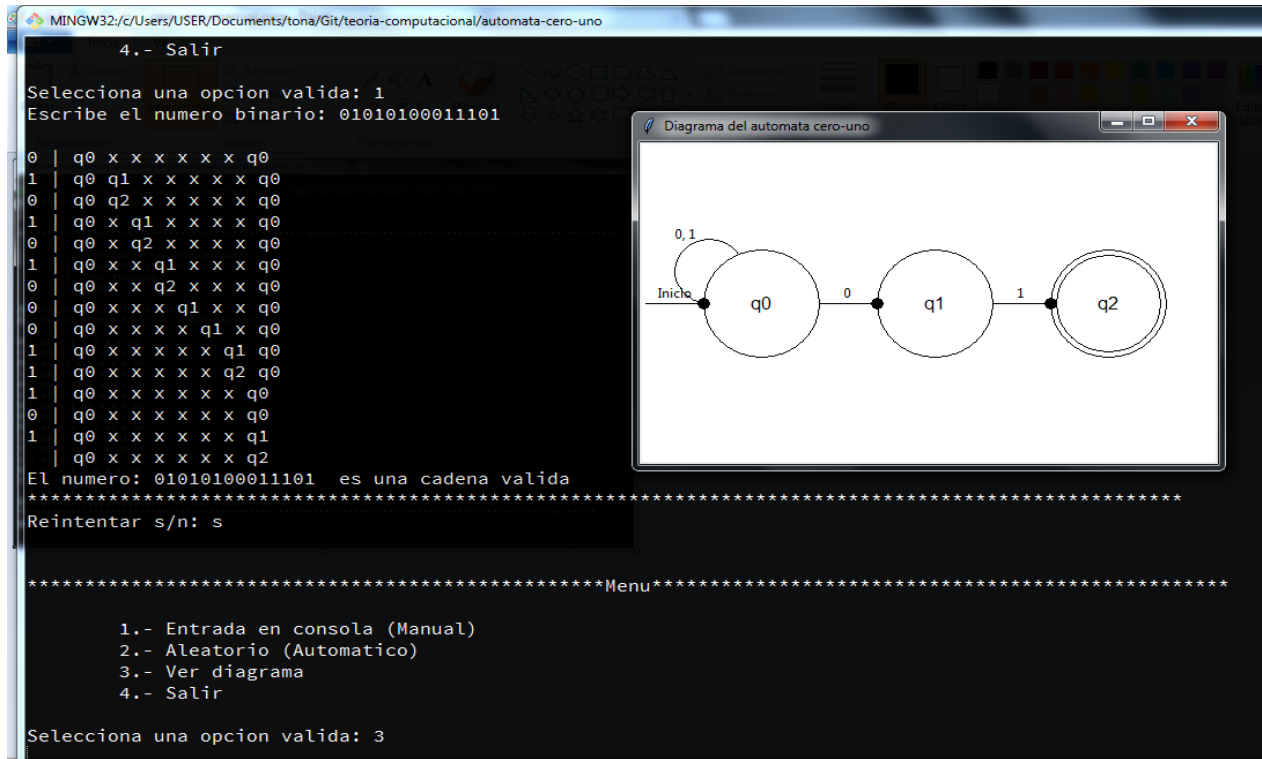


Figura 25: Diagrama del autómata.

Referencias

- [1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introducción a La Teoría De Autómatas, Lenguajes Y Computación*. Addison-Wesley, 2007.
- [2] J. D. Ullman, “Finite Automata.” <http://infolab.stanford.edu/~ullman/ialc/spr10/slides/fa1.pdf>, 2010. [Consultado: 2016-09-10].