

# Reporte del primer parcial

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

10 de septiembre de 2016

# Índice

<b>1. Alfabeto</b>	<b>2</b>
1.1. Descripción del problema . . . . .	2
1.2. Código . . . . .	2
1.3. Pruebas . . . . .	5
<b>2. Números primos</b>	<b>8</b>
2.1. Descripción del problema . . . . .	8
2.2. Código . . . . .	8
2.3. Pruebas . . . . .	10
<b>3. AFD Palabras con terminación 'ere'</b>	<b>13</b>
3.1. Descripción del problema . . . . .	13
3.2. Código . . . . .	13
3.3. Pruebas . . . . .	13
<b>4. AFD Paridad en números binarios</b>	<b>14</b>
4.1. Descripción del problema . . . . .	14
4.2. Código . . . . .	14
4.3. Pruebas . . . . .	14
<b>5. Protocolo de transmisión</b>	<b>15</b>
5.1. Descripción del problema . . . . .	15
5.2. Código . . . . .	15
5.3. Pruebas . . . . .	15
<b>6. AFND Números binarios con terminación '01'</b>	<b>16</b>
6.1. Descripción del problema . . . . .	16
6.2. Código . . . . .	16
6.3. Pruebas . . . . .	16

# 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  $\Sigma^*$  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;
}

```

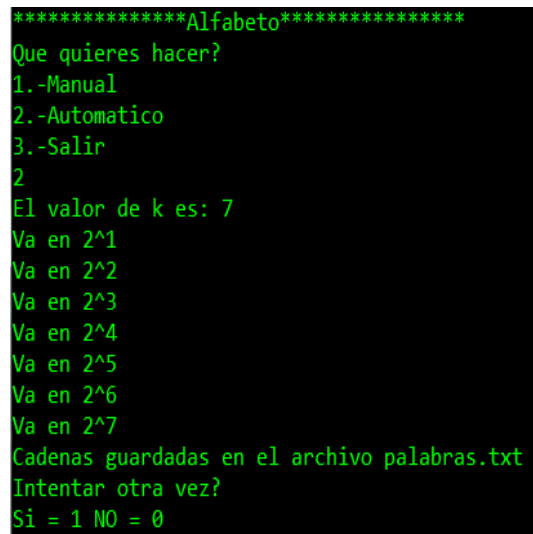
```
int random_potencia_k() {  
    //numero = rand () % (N-M+1) + M;  
    // Cambiar el valor del random k a 1000  
    int potencia_k = 1 + rand() % (1000 + 1 - 1);  
    return potencia_k;  
}
```

---

### 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.

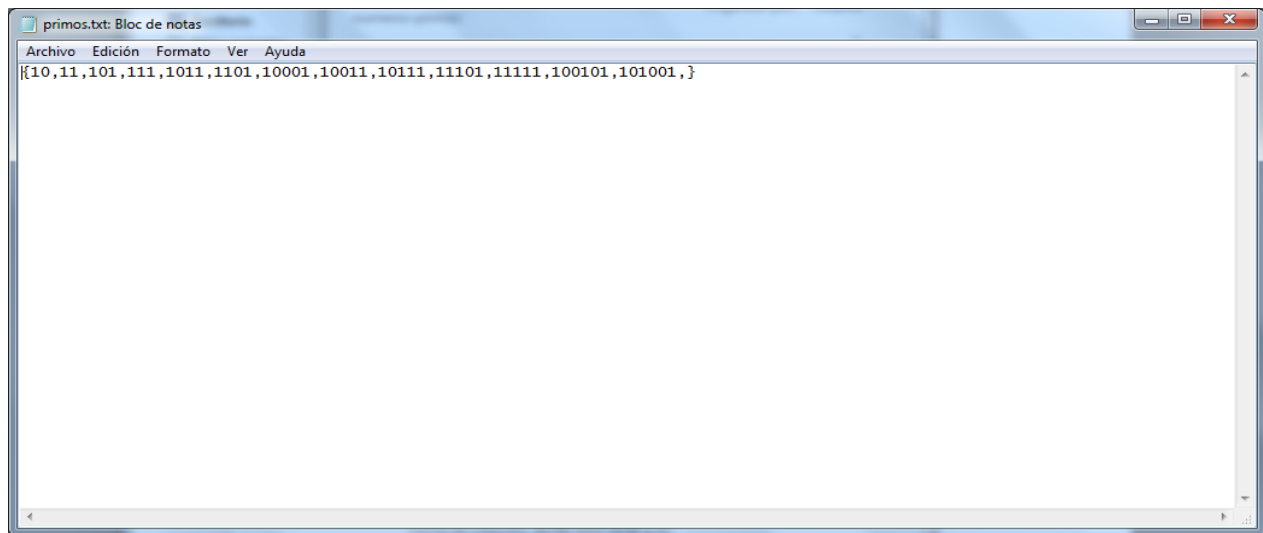


Figura 6: Números primos en binario en el archivo.

## Modo manual.

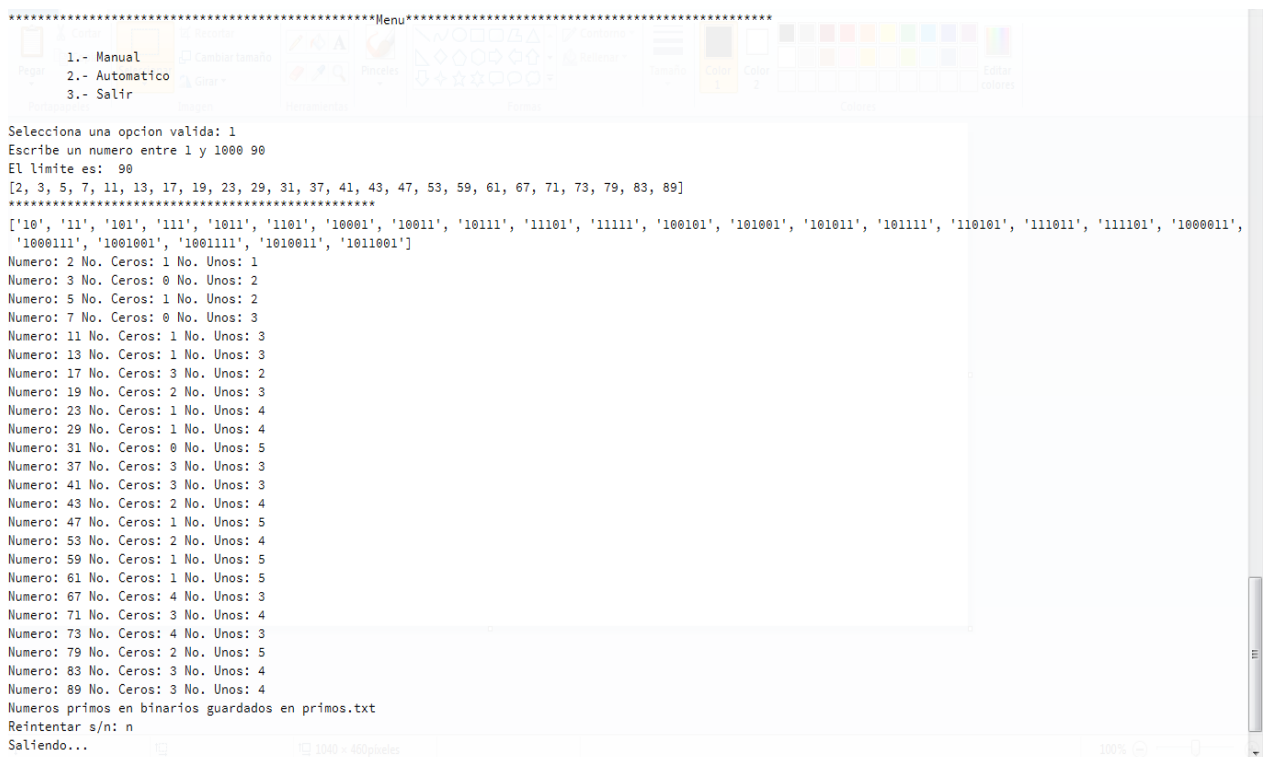


Figura 7: Modo automático n=90.

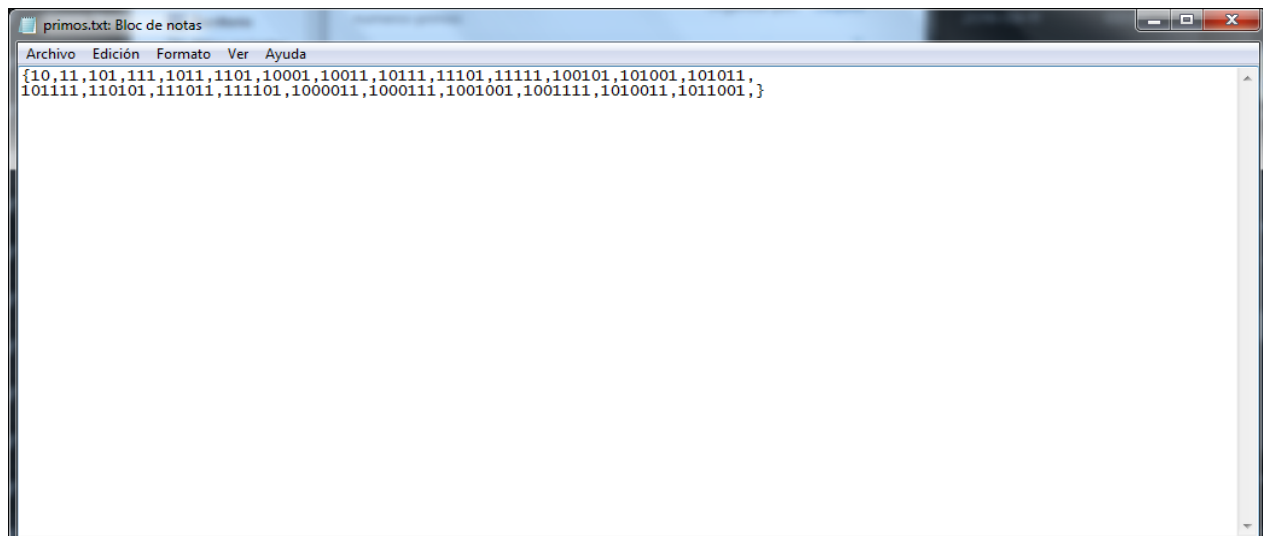


Figura 8: Números primos en binario en el archivo.

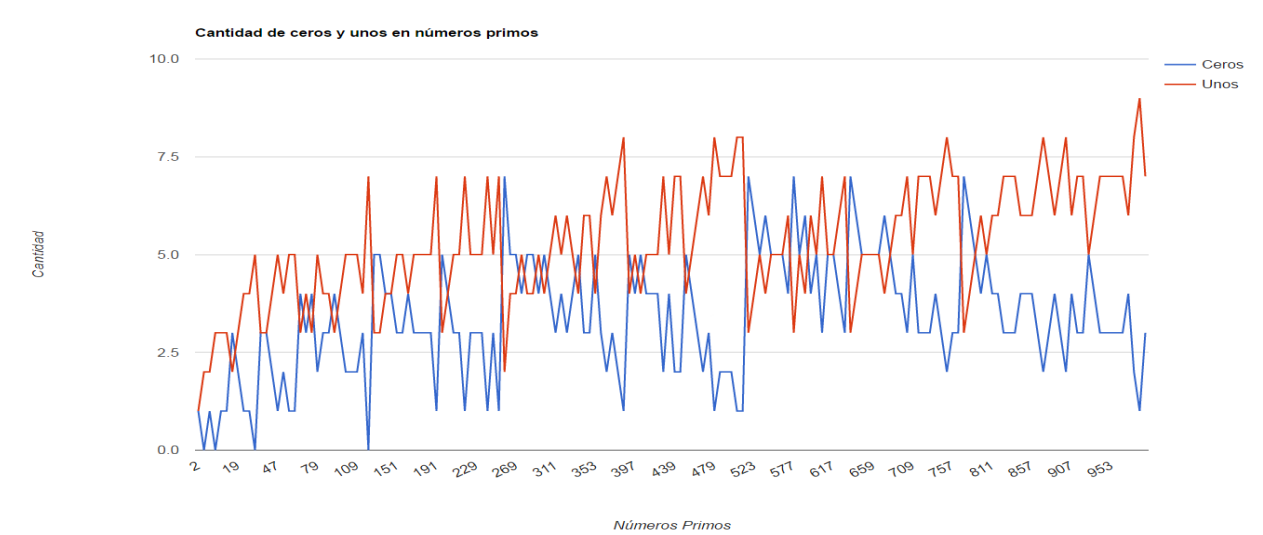


Figura 9: Cantidad de ceros y unos encontrados entre 1 y 1000.

### 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.

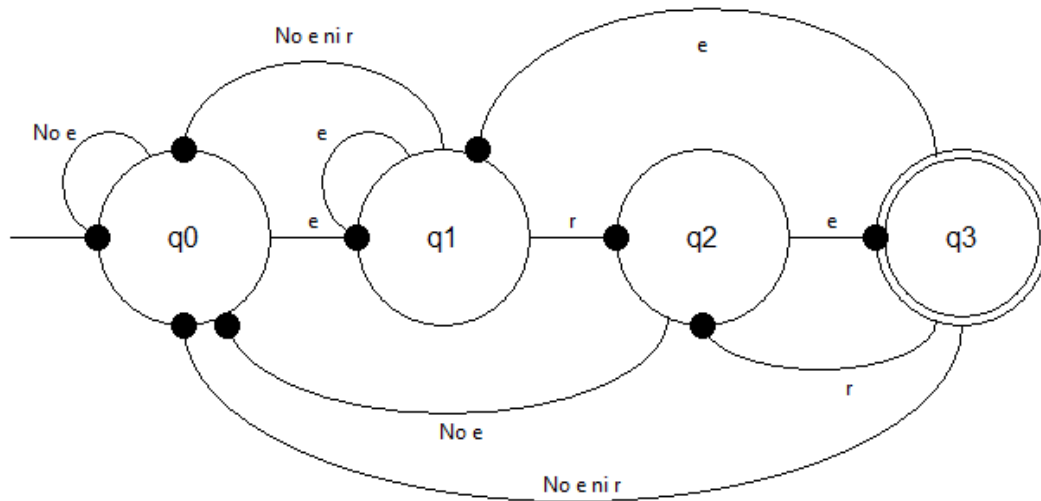


Figura 10: Diagrama de transiciones del autómata 'ere'.

#### 3.2. Código

El código fue realizado en Python 3.5.

#### 3.3. Pruebas

Pruebas de las opciones del menú. Modo automático. Modo manual. Diagrama.

## 4. AFD Paridad en números binarios

### 4.1. Descripción del problema

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

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

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 valida o no y en ambos casos imprimir su historia. Además, mostrar el siguiente diagrama.

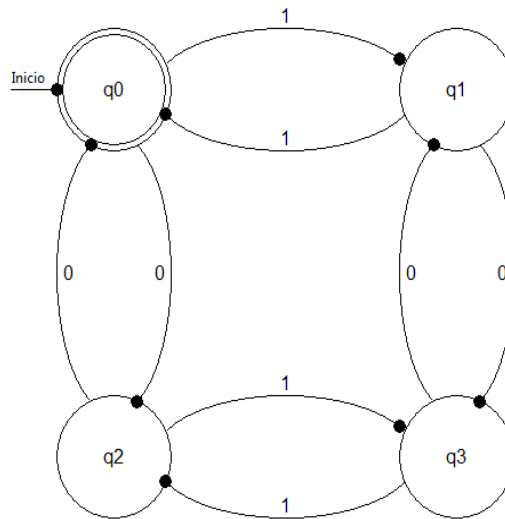


Figura 11: Diagrama de transiciones del autómata. [1]

### 4.2. Código

El código fue realizado en Python 3.5.

### 4.3. Pruebas

Pruebas de las opciones del menú. Modo automático. Modo manual. Diagrama.

## 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 autómata de paridad binaria y guardar las cadenas binarias en otro archivo, siguiendo el siguiente diagrama.

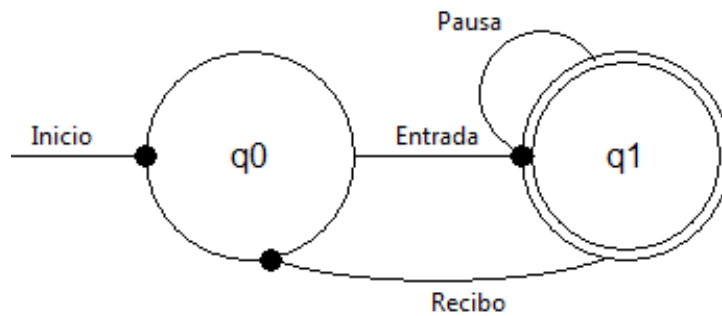


Figura 12: Diagrama de transiciones del autómata. [2]

### 5.2. Código

El código fue realizado en Python 3.5.

### 5.3. Pruebas

Pruebas de las opciones del menú. Modo automático. Diagrama.



## 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 automática debe de tener una longitud  $n \mid 1 \leq n \leq 1000$ . Y que contenga la opción de mostrar el siguiente diagrama.

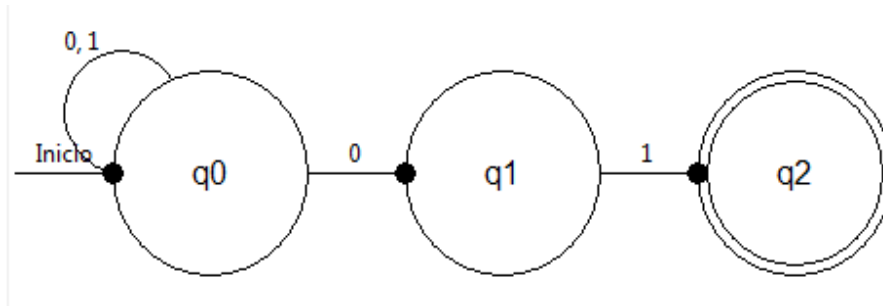


Figura 13: Diagrama de transiciones del autómata. [1]

### 6.2. Código

El código fue realizado en Python 3.5.

### 6.3. Pruebas

Pruebas de las opciones del menú. Modo automático. Modo manual. Diagrama.

## 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].