

# Universo del Alfabeto Binario '0,1'

Estudiante    González Fonseca José Juan  
Docente      Juárez Martínez Genaro  
Materia      Teoría de la Computación

## Introducción

En esta práctica se realizó la codificación del universo de las cadenas binarias ( $\Sigma^n$ ) para cualquier valor de 'n' dado por el usuario o seleccionado automáticamente por la máquina en el intervalo  $[0,1000]$ .

Adicionalmente se realizarán las gráficas correspondientes a la cantidades de ceros y unos de cada cadena así como la gráfica con logaritmo base 10.

## Marco Teórico

### Alfabetos

Un alfabeto  $\Sigma$  es un conjunto finito de símbolos. En esta práctica, se utiliza el alfabeto binario  $\Sigma = \{0, 1\}$ , con el cual se forman todas las combinaciones posibles de ceros y unos.

### Cadenas

Una cadena es una secuencia finita de símbolos tomada de un conjunto llamado alfabeto. Si  $\Sigma$  es un alfabeto, una cadena  $w$  de longitud  $n$  es un elemento de  $\Sigma^n$ . El número de cadenas posibles de longitud  $n$  es  $|\Sigma|^n$ , donde  $|\Sigma|$  es el número de símbolos en el alfabeto.

El universo de cadenas para un alfabeto  $\Sigma$  y una longitud  $n$  es el conjunto de todas las cadenas posibles formadas por los símbolos del alfabeto de longitud  $n$ . En el caso del alfabeto binario  $\Sigma = \{0, 1\}$ , el universo de cadenas es  $\Sigma^n$ , el cual contiene  $2^n$  combinaciones. Este universo representa todas las posibles secuencias binarias de longitud  $n$ , que en esta práctica se generan y analizan.

### Elemento vacío

El elemento vacío, denotado como  $\epsilon$ , es la cadena de longitud cero, que no contiene símbolos. Aunque no tiene contenido, es una cadena válida sobre cualquier alfabeto, perteneciente al conjunto  $\Sigma^0$ .

### Lenguajes

Un lenguaje  $L$  es un conjunto de cadenas sobre un alfabeto  $\Sigma$ . Si  $L \subseteq \Sigma^*$ , el lenguaje incluye todas las cadenas posibles, incluidas las vacías. En esta práctica, se trabaja con cadenas de longitud fija  $n$ , es decir, subconjuntos de  $\Sigma^n$ .

## Desarrollo

Para esta práctica usamos dos programas, uno para la generación de todas las combinaciones posibles para un  $n$  seleccionado por el usuario o elegido aleatoriamente por la máquina. Estas combinaciones son guardadas en un archivo .txt. El segundo programa consiste en la lectura del .txt generado para generar cuatro gráficas: una para la cantidad de ceros del universo, otra para la cantidad de unos y otras dos para sus versiones de logaritmo base 10.

- El usuario puede elegir entre el modo automático o manual.
- Se añade primero el valor de  $\epsilon$  al .txt.
- Se usa un enfoque recursivo para obtener todas las cadenas de una longitud  $n$ , esto se repite para todos los  $n$  y se escriben en un .txt.

- El script de Python encargado de generar las gráficas es llamado desde C++.
- Para generar las gráficas, se lee el archivo .txt y se separan sus valores gracias a las comas; se omite el primer valor ( $\epsilon$ ).
- Contamos la cantidad de ceros y unos de cada combinación y esa lista se usa como parámetro para graficar.
- Graficamos en escala lineal y en la escala logarítmica omitimos el primer logaritmo de cero.

Listing 1: Código en C++ para calculo del universo y generar el .txt

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>    // para manejar archivos
4  #include <cstdlib>    // para rand() y srand()
5  #include <ctime>      // para usar time() y crear valores aleatorios
6
7  using namespace std;
8
9  void generarCombinaciones(string alfabeto, int n, ofstream& archivo, string
   combinacionActual = "", int longitudActual = 0) {
10     // Caso base: si la longitud de la combinacion actual es igual a n, la escribimos
       en el txt
11     if (longitudActual == n) {
12         archivo << combinacionActual << ",";
13         return;
14     }
15
16     // Recorrer cada caracter del alfabeto y generar combinaciones
17     for (char c : alfabeto) {
18         generarCombinaciones(alfabeto, n, archivo, combinacionActual + c,
           longitudActual + 1);
19     }
20 }
21 int main() {
22     srand(time(0)); // Inicializar la semilla para numeros aleatorios
23
24     string alfabeto = "01";
25     int n;
26     int opcion;
27
28     do {
29         cout << "Seleccione una opcion:\n";
30         cout << "1. Ingresar el valor de n manualmente\n";
31         cout << "2. Generar el valor de n aleatoriamente [1,1000]\n";
32         cout << "3. Salir\n";
33         cin >> opcion;
34
35         if (opcion == 1) {
36             cout << "Ingrese el valor de n: ";
37             cin >> n;
38         } else if (opcion == 2) {
39             n = rand() % 1000 + 1; // Generar un valor aleatorio entre 1 y 1000 para
               n
40             cout << "Valor de n generado aleatoriamente: " << n << endl;
41         } else if (opcion == 3) {
42             cout << "Saliendo del programa.\n";
43             return 0;
44         } else {
45             cout << "Opcion no valida. Por favor intente de nuevo.\n";
46             continue; // Volver al inicio del bucle
47         }
48
49         // Crear un archivo con el nombre que indica el valor de n
50         string nombre_archivo = "combinaciones_n" + to_string(n) + ".txt";

```

```

51     ofstream archivo(nombre_archivo);
52
53     if (!archivo) {
54         cout << "Error al crear el archivo.\n";
55         return 1;
56     }
57     archivo << "\u03B5,"; // Agregar la combinacion vacia al inicio del txt (
58         // \u03B5 es el simbolo de epsilon)
59
60     // Generar combinaciones por cada valor de n para formar el universo y
61     // escribirlas en el archivo
62     for (int i = 1; i <= n; i++) {
63         generarCombinaciones(alfabeto, i, archivo);
64         //archivo << endl; // Opcional: Separar las combinaciones por longitud
65         // con un salto de linea
66     }
67
68     archivo.close();
69     cout << "Combinaciones generadas y guardadas en el archivo combinaciones_n"
70         << n << ".txt\n";
71
72     // Llamar al script de Python para graficar
73     string comando = "python graficas.py" + nombre_archivo;
74     system(comando.c_str());
75
76 } while (opcion != 3); // Continuar hasta que el usuario elija salir
77
78 return 0;
79 }

```

Listing 2: Código en Python para la generación de graficas

```

1  import sys
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  def procesar_combinaciones(archivo):
6      # Leer el archivo desde la ruta proporcionada en la terminal
7      with open(archivo, 'r') as f:
8          combinaciones = f.read().split(',') # Separar las combinaciones por coma
9
10     # Remover el primer elemento que es vacio (epsilon)
11     combinaciones = combinaciones[1:]
12
13     # Contadores de ceros y unos
14     ceros = []
15     unos = []
16
17     # Contar ceros y unos en cada combinacion
18     for comb in combinaciones:
19         ceros.append(comb.count('0'))
20         unos.append(comb.count('1'))
21
22     return ceros, unos
23
24  def graficar_ceros(ceros):
25     # Graficar la cantidad de ceros en escala lineal
26     plt.figure(figsize=(10, 5))
27     plt.plot(ceros, color='purple')
28     plt.title('Grafica de la cantidad de ceros')
29     plt.xlabel('Cadena')
30     plt.ylabel('Cantidad de ceros')
31     plt.show()
32
33  def graficar_unos(unos):

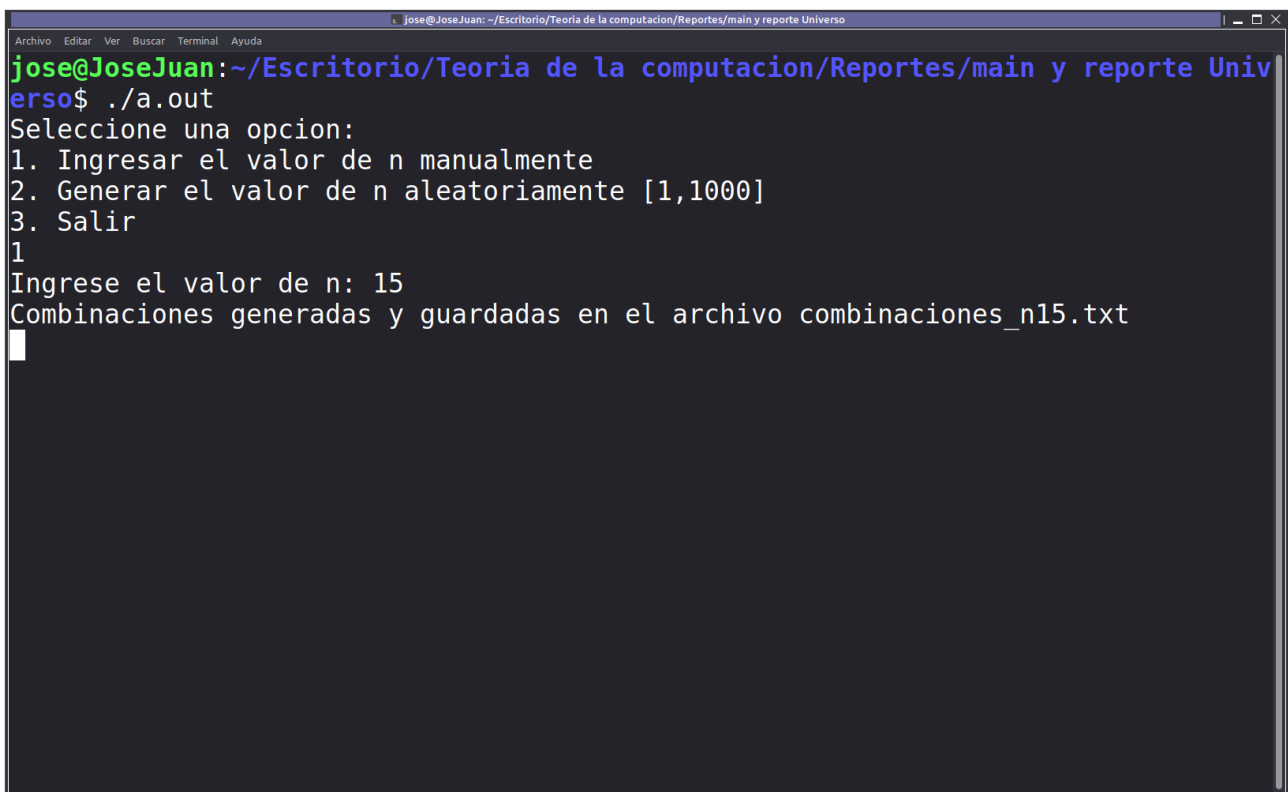
```

```

34     # Graficar la cantidad de unos en escala lineal
35     plt.figure(figsize=(10, 5))
36     plt.plot(unos, color='palegreen')
37     plt.title('Grafica de la cantidad de unos')
38     plt.xlabel('Cadena')
39     plt.ylabel('Cantidad de unos')
40     plt.show()
41
42 def graficar_ceros_logaritmico(ceros):
43     # Graficar en escala logaritmica la cantidad de ceros
44     plt.figure(figsize=(10, 5))
45     plt.plot(np.log10(np.array(ceros) + 1), color='purple') # Sumar 1 para evitar
46         logaritmo de 0
47     plt.title('Grafica del logaritmo base 10 de ceros')
48     plt.xlabel('Cadena')
49     plt.ylabel('logaritmo base 10 de la cantidad de ceros')
50     plt.show()
51
52 def graficar_unos_logaritmico(unos):
53     # Graficar en escala logaritmica la cantidad de unos
54     plt.figure(figsize=(10, 5))
55     plt.plot(np.log10(np.array(unos) + 1), color='palegreen') # Sumar 1 para evitar
56         logaritmo de 0
57     plt.title('Grafica del logaritmo base 10 de unos')
58     plt.xlabel('Cadena')
59     plt.ylabel('logaritmo base 10 de la cantidad de unos')
60     plt.show()
61
62 if __name__ == "__main__":
63     # Verificar que el archivo fue proporcionado como argumento
64     if len(sys.argv) != 2:
65         print("Uso: python graficas.py <ruta_del_archivo_txt>") # Mensaje de error
66         sys.exit(1)
67
68     archivo = sys.argv[1]
69
70     # Procesar combinaciones binarias del archivo
71     ceros, unos = procesar_combinaciones(archivo)
72
73     # Graficar datos para 0s
74     graficar_ceros(ceros)
75
76     # Graficar datos para 1s
77     graficar_unos(unos)
78
79     # Graficar datos en escala logaritmica para 0s
80     graficar_ceros_logaritmico(ceros)
81
82     # Graficar datos en escala logaritmica para 1s
83     graficar_unos_logaritmico(unos)

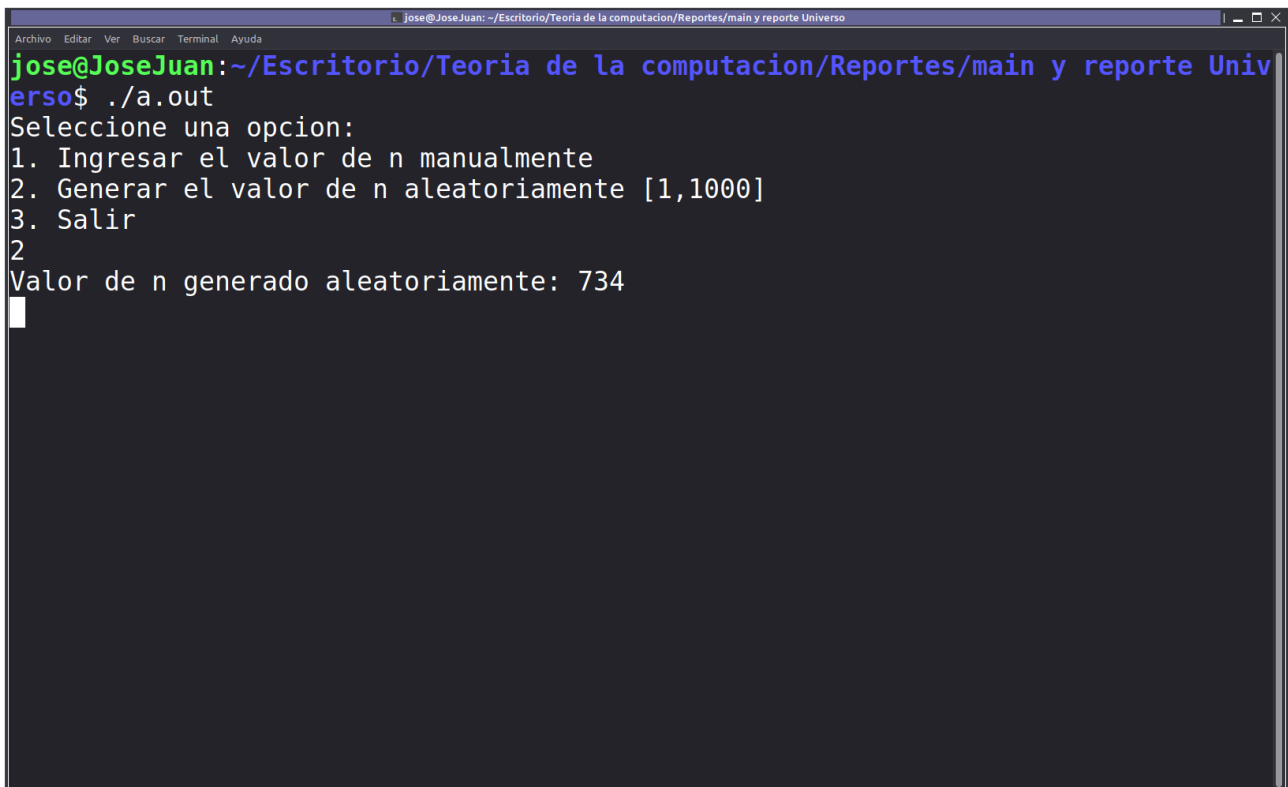
```

## Resultados



```
jose@JoseJuan: ~/Escritorio/Teoria de la computacion/Reportes/main y reporte Univ
erso$ ./a.out
Seleccione una opcion:
1. Ingresar el valor de n manualmente
2. Generar el valor de n aleatoriamente [1,1000]
3. Salir
1
Ingrese el valor de n: 15
Combinaciones generadas y guardadas en el archivo combinaciones_n15.txt
```

Figure 1: Ejecución Manual



```
jose@JoseJuan: ~/Escritorio/Teoria de la computacion/Reportes/main y reporte Univ
erso$ ./a.out
Seleccione una opcion:
1. Ingresar el valor de n manualmente
2. Generar el valor de n aleatoriamente [1,1000]
3. Salir
2
Valor de n generado aleatoriamente: 734
```

Figure 2: Ejecución Automatica

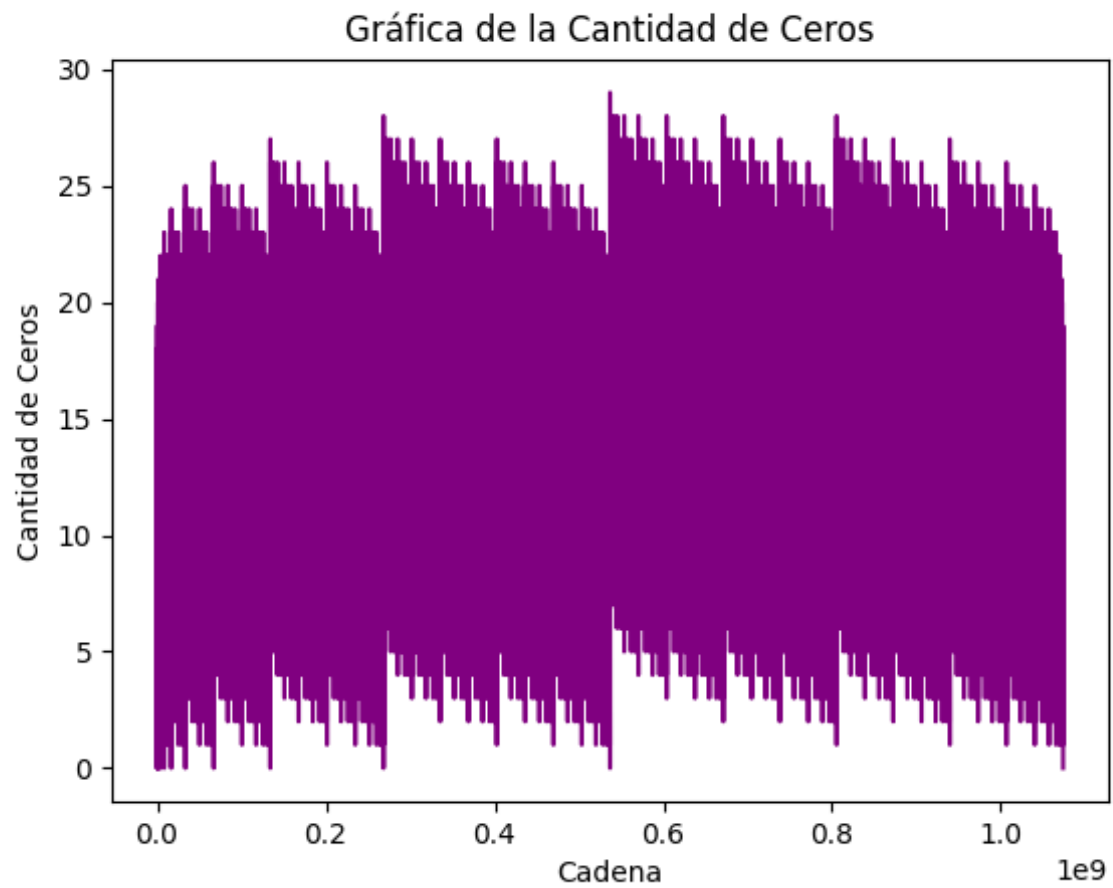


Figure 3: Gráfica cantidad de ceros para  $N=29$

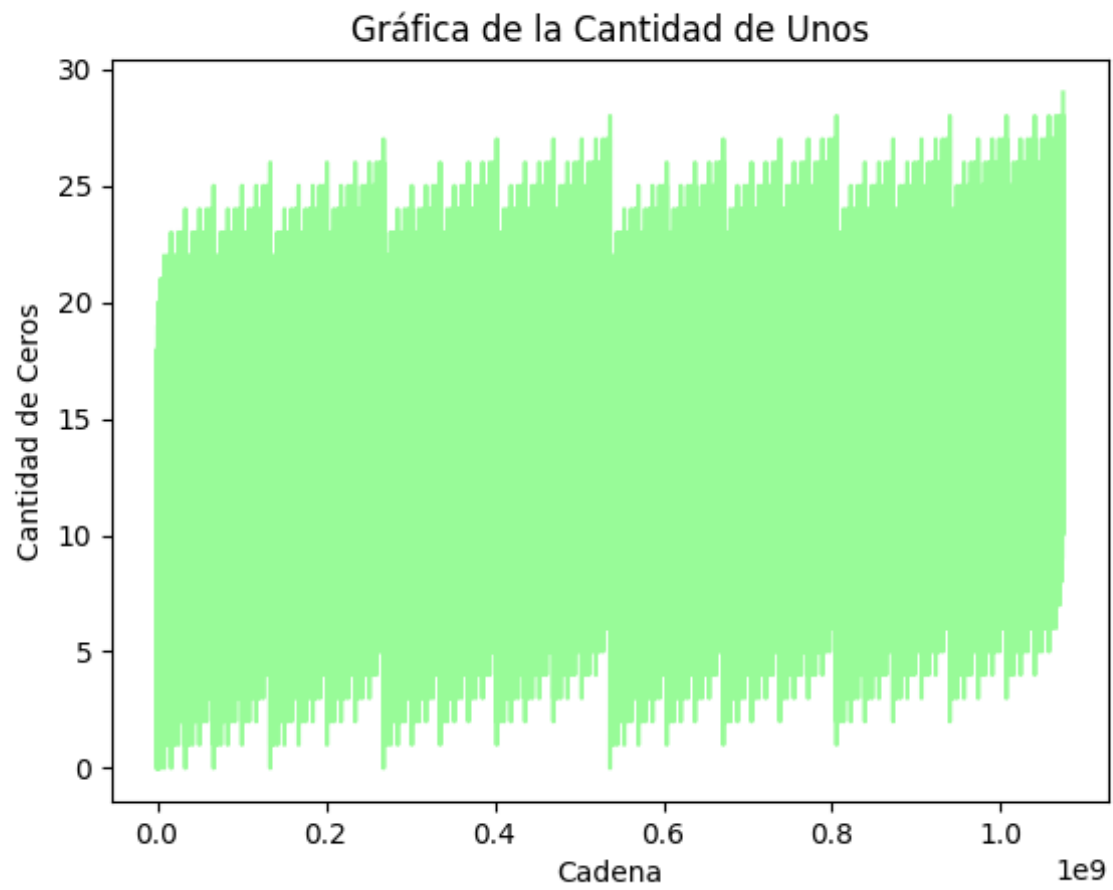


Figure 4: Grafica cantidad de unos para  $N=29$

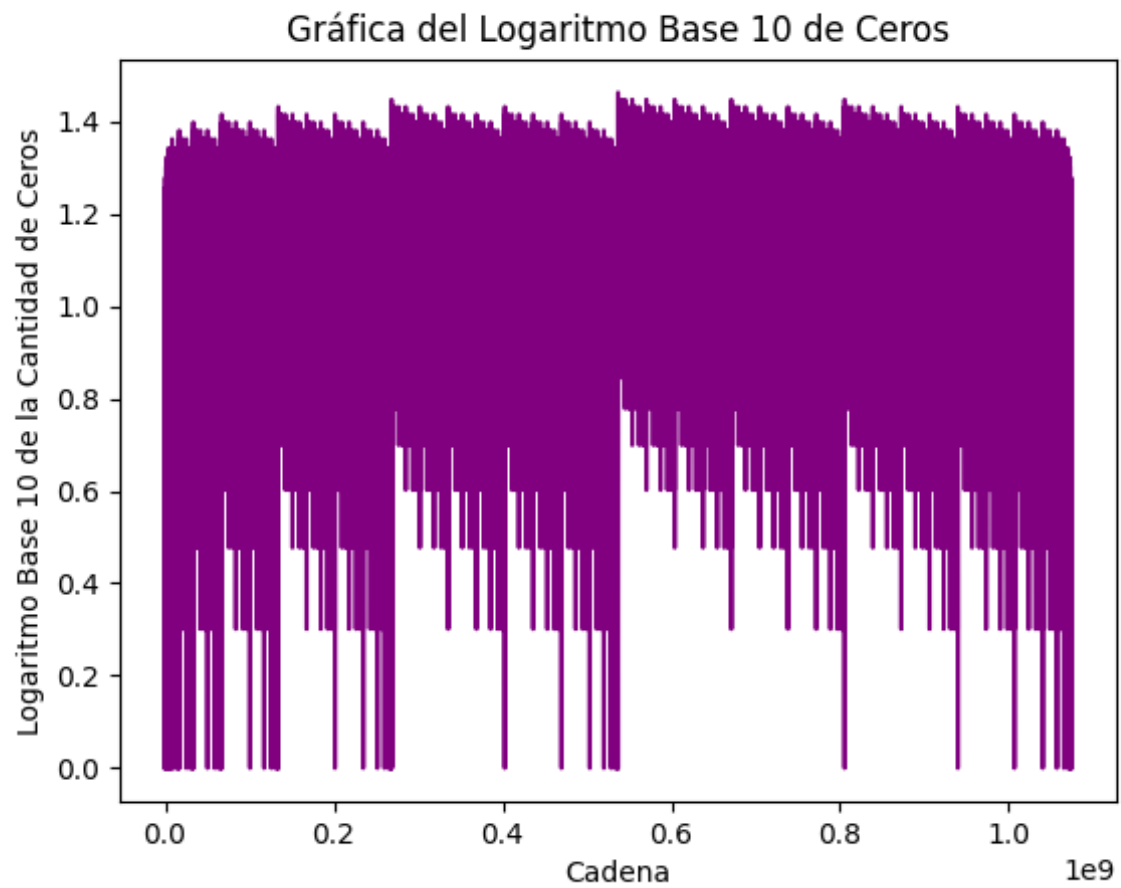


Figure 5: Gráfica logaritmo base 10 de ceros para  $N=29$



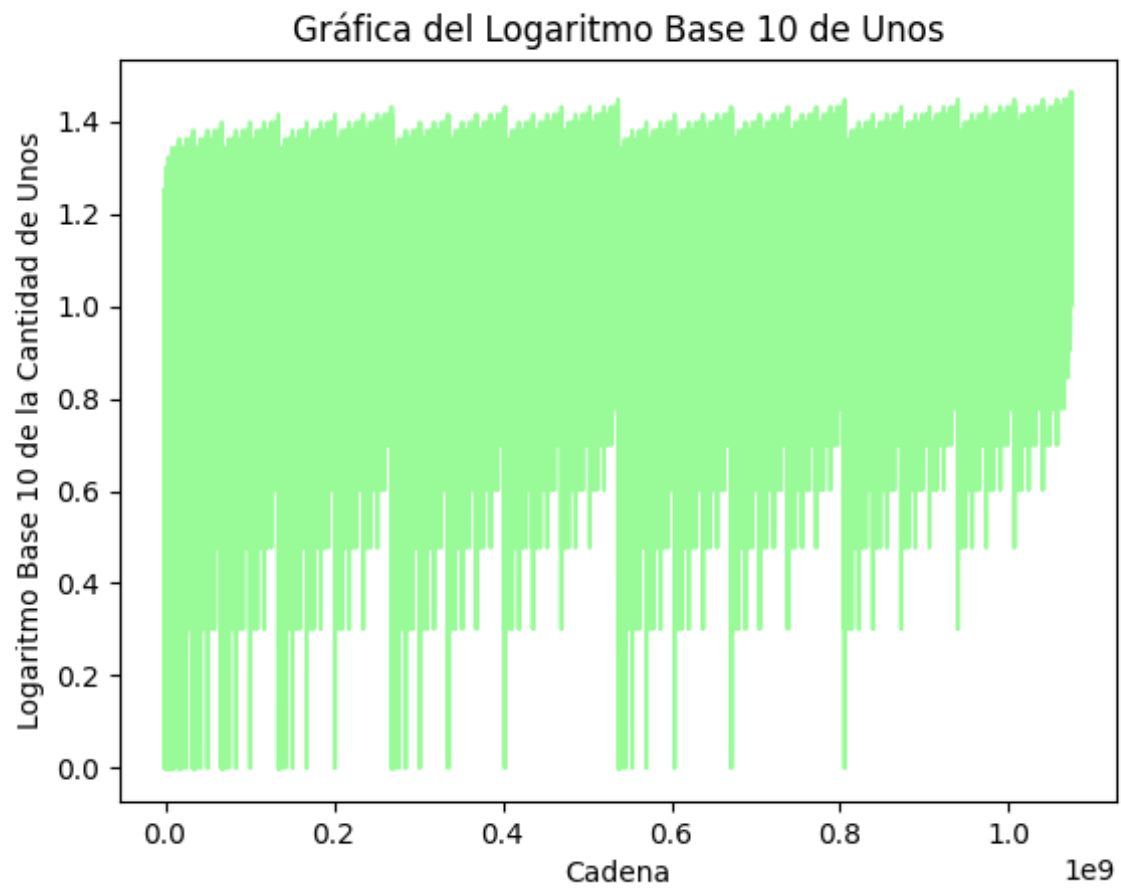


Figure 6: Gráfica logaritmo base 10 de unos para  $N=29$

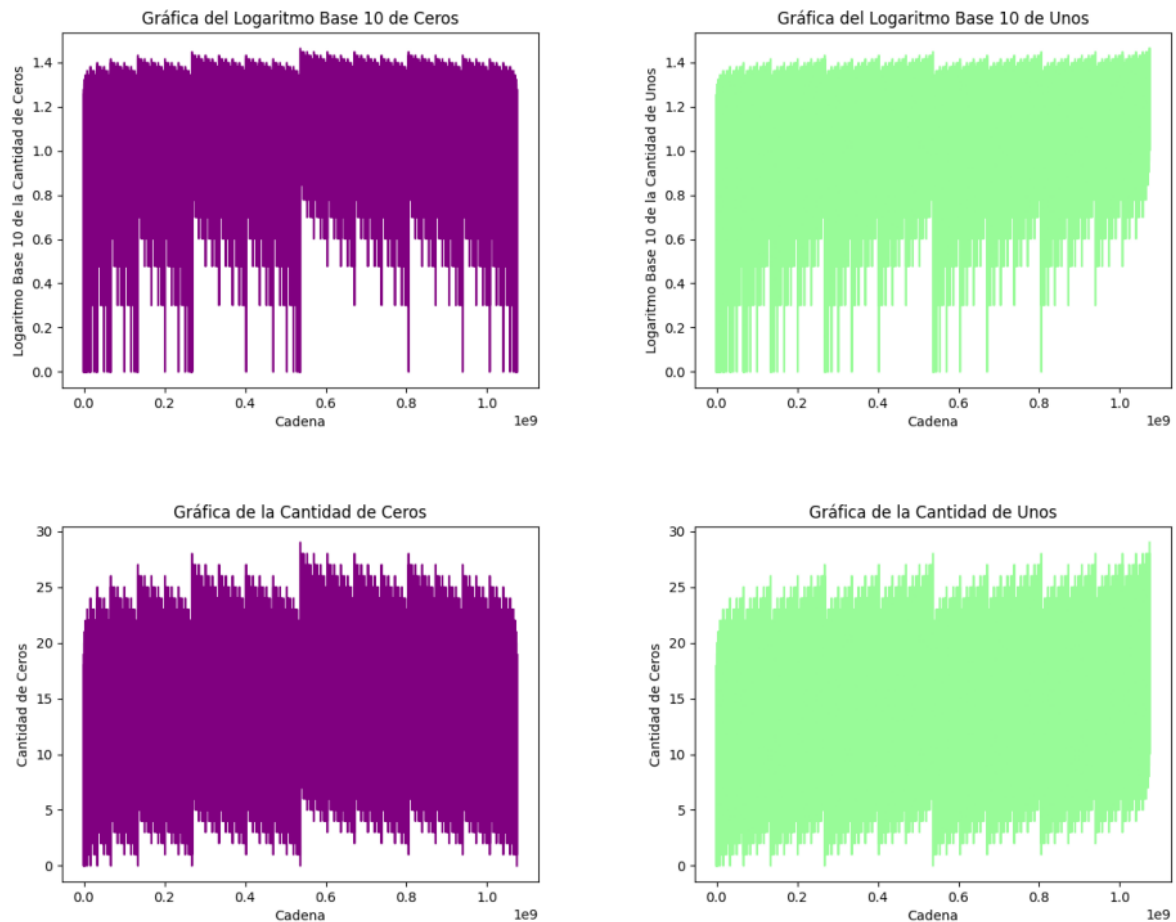


Figure 7: Comparativa de todas las gráficas

```

GNU nano 6.2 combinaciones n15.txt *
ε,0,1,00,01,10,11,000,001,010,011,100,101,110,111,0000,0001,0010>
0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111,0000>
00010,00011,00100,00101,00110,00111,01000,01001,01010,01011,0110>
01110,01111,10000,10001,10010,10011,10100,10101,10110,10111,1100>
11010,11011,11100,11101,11110,11111,000000,000001,000010,000011,>
000101,000110,000111,001000,001001,001010,001011,001100,001101,0>
001111,010000,010001,010010,010011,010100,010101,010110,010111,>
011000,011001,011010,011011,011100,011101,011110,011111,100000,>
100001,100010,100011,100100,100101,100110,100111,101000,101001,>
101010,101011,101100,101101,101110,101111,110000,110001,110010,>
110011,110100,110101,110110,110111,111000,111001,111010,111011,>
111100,111101,111110,111111,0000000,0000001,0000010,0000011,0000>
100,0000101,0000110,0000111,0001000,0001001,0001010,0001011,000>
^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar

```

Figure 8: Ejemplo de Visualización del txt

## Conclusiones

A lo largo de esta práctica, abordamos la codificación del universo del alfabeto binario. La generación exhaustiva de combinaciones binarias mediante un enfoque recursivo nos permitió ver la naturaleza exponencial del problema —con  $2^n$  posibles cadenas para un valor de longitud  $n$ — además de ver la existencia elemento vacío  $\epsilon$  como parte integrante del conjunto  $\Sigma^0$ .

El análisis de las frecuencias de ceros y unos en las cadenas generadas permite entender la distribución de los símbolos en el contexto de la teoría de lenguajes formales. A través de las gráficas generadas, se evidencia el comportamiento de la cantidad de ceros y unos en relación con la longitud de las cadenas, lo que nos permite un análisis visual de lo que está pasando.