

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Ciudad de México, México, 13 de octubre de 2024 .

## Programa Universo

Ponce Fragoso Emmanuel

Teoria de Computaciòn  
Genaro Juarez  
5BM2

# Índice

<b>1. Inicio</b>	<b>2</b>
1.1. Objetivo de la práctica . . . . .	2
1.2. Objetivo específico . . . . .	2
1.3. Introducción . . . . .	2
1.4. Metodología . . . . .	2
<b>2. Desarrollo</b>	<b>4</b>
2.1. Generador_universo.py . . . . .	4
2.1.1. Librerías Importadas . . . . .	4
2.1.2. Función generar_cadenas(n) . . . . .	4
2.1.3. Función contar_unos(cadena) . . . . .	4
2.1.4. Función generar_archivo_salida_organizado(n, nombre_archivo) . . . . .	5
2.1.5. Función generar_csv_organizado(n, nombre_archivo)) . . . . .	5
2.1.6. Función menu() . . . . .	6
2.1.7. Ejecución del Programa . . . . .	6
2.2. Graficar_universo.py . . . . .	7
2.2.1. Librerías Importadas . . . . .	7
2.2.2. Cargar el DataFrame desde un archivo CSV . . . . .	7
2.2.3. Convertir las columnas a valores numéricos . . . . .	7
2.2.4. Crear un lienzo (Canvas) para los gráficos . . . . .	7
2.2.5. Generar la imagen de los '1's y '0's . . . . .	7
2.2.6. Aplicar transformaciones logarítmicas . . . . .	8
2.2.7. Generar la imagen logarítmica de los '1's y '0's . . . . .	8
2.2.8. Mensaje Final . . . . .	9
<b>3. Salidas</b>	<b>10</b>
3.1. Complejidad Temporal . . . . .	10
3.1.1. Generación de cadenas binarias . . . . .	10
3.1.2. Conteo de '1's y '0's en cada cadena . . . . .	10
3.1.3. Generación del archivo CSV . . . . .	10
3.1.4. Visualización de los datos . . . . .	10
3.1.5. Complejidad total . . . . .	10
3.2. OutUniverso.txt . . . . .	11
3.3. plotDataSet.csv . . . . .	12
3.4. Graficas de Datos y Logaritmicas Base 10 de '1's y '0's . . . . .	13
3.4.1. Grafica de <i>OnesNumber</i> . . . . .	13
3.4.2. Grafica de <i>ZeroesNumber</i> . . . . .	14
<b>4. Conclusion</b>	<b>15</b>
<b>5. Referencias</b>	<b>15</b>

# 1. Inicio

## 1.1. Objetivo de la práctica

Con la presente práctica se busca implementar en código los conceptos aprendidos en la clase de Teoría de la Computación, más específicamente para poder programar el universo de las cadenas binarias ( $\Sigma^n$ ). Dada una  $n$  que introduzca el usuario o que el programa lo determine automáticamente. El rango de  $n$  debe de estar en el intervalo de  $[0, 1000]$ .

## 1.2. Objetivo específico

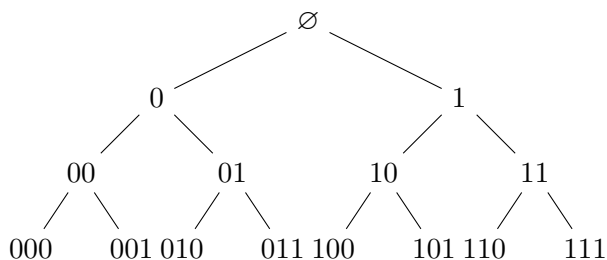
El objetivo es desarrollar un programa que utilice técnicas de programación dinámica y programación lineal para generar todas las posibles secuencias de longitud  $n$  a partir de secuencias de longitud  $n - 1$ . El propósito es investigar y comprender cómo la descomposición de problemas complejos en subproblemas más pequeños y su resolución de manera recursiva puede llevar a soluciones óptimas y eficientes del problema original. El enfoque de esta práctica está en aprovechar las propiedades de las cadenas para reducir la complejidad temporal en el proceso de generación, permitiendo una ejecución más eficiente y una comprensión más profunda de la aplicación de estas técnicas en la resolución de problemas de optimización.

## 1.3. Introducción

La programación dinámica es una técnica de resolución de problemas que se basa en dividir un problema grande en subproblemas más sencillos, los cuales se abordan de manera recursiva. Este enfoque permite que las soluciones de los subproblemas se utilicen para construir la solución final del problema original. Se destaca por su capacidad para mejorar la eficiencia en la resolución de problemas de optimización, evitando la repetición de cálculos innecesarios.

## 1.4. Metodología

Para desarrollar el algoritmo que calcula  $\Sigma^n$ , se implementó una técnica basada en programación dinámica. Esto permite que las cadenas de longitud  $n$  se construyan a partir de las cadenas de longitud  $n - 1$ . Asimismo, para optimizar el cálculo de  $\Sigma^n$ , se utilizó un enfoque de programación lineal. Este método asegura que las secuencias más largas se formen agregando un dígito a las secuencias ya existentes de longitud  $n - 1$ .



Es importante destacar que en cada nivel del árbol, la siguiente cadena se genera tomando la cadena anterior y añadiendo un dígito binario, sea 0 o 1. Este patrón es clave para reducir la complejidad del cálculo. Considerando ciertos razgos:

1. Debe de correr en modo automático (todo) y de forma manual.
2. El programa debe de preguntar si quiere calcular otra  $N$  o no y salir hasta que se le especifique.
3. La salida debe ser expresada en notación de conjunto y debe ir a un archivo de texto.

4. Del archivo de salida, graficar el número de unos y ceros de cada cadena. En el eje de las "x" se representan las cadenas y en el eje de las Y el número de unos/ceros que tiene esa cadena. Específicamente en el reporte, explicar, calcular y graficar cuando  $n=29$ .  
Adicionalmente, con los mismos valores, calcular la gráfica pero ahora con logaritmo base 10.

## 2. Desarrollo

### 2.1. Generador\_universo.py

#### 2.1.1. Librerías Importadas

```
import os
import random
```

#### Librería OS

Esta librería permite interactuar con el entorno es muy útil para hacer tareas relacionadas con archivos o directorios, como asegurar de que un archivo o carpeta existe antes de crear o leerlo.

#### Librería random

Se usa para generar números aleatorios. En este código, se emplea para elegir un valor aleatorio para  $n$  (la longitud de las cadenas binarias) cuando el usuario selecciona una opción aleatoria en el menú.

#### 2.1.2. Función generar\_cadenas(n)

Esta función genera todas las cadenas binarias posibles de longitud  $n$ . Se hace de manera recursiva, lo que significa que llama a sí misma hasta que llega al caso base donde  $n = 0$ .

```
# Generador para crear cadenas de longitud n
def generar_cadenas(n):
    if n == 0:
        yield ''
    else:
        for cadena in generar_cadenas(n - 1):
            yield cadena + '0'
            yield cadena + '1'
```

#### Fucionamiento de la Función:

**Caso Base:** Cuando  $n$  es 0, la función genera una cadena vacía (").

**Caso Recursivo:** Cuando  $n$  es mayor que 0, la función toma todas las cadenas generadas en la llamada recursiva anterior y le añade '0' y un '1', creando todas las combinaciones posibles dependiendo de  $n$ .

#### Ejemplo con $n = 2$ :

1. Para  $n = 2$ , la función va a *generar\_cadenas(1)*, que generaría las cadenas ['0', '1'].
2. Luego, la función agregaría '0' y '1' a cada cadena de longitud 1, resultando en ['00', '01', '10', '11'].

#### 2.1.3. Función contar\_unos(cadena)

Esta función toma una cadena binaria cuenta cuántos '1's y '0's contiene la cadena.

```
# Funcion para contar el numero de '1's y '0's en una cadena
def contar_unos(cadena):
    return cadena.count('1'), cadena.count('0')
```

#### Fucionamiento de la Función:

1. Usa el método *.count()* de las cadenas en Python para contar el número de veces que aparece el carácter '1' y el carácter '0'.

2. Retorna una tupla con dos valores: el número de '1's y el número de '0's.

### Ejemplo:

```
contar_unos('101') # Retorna (2, 1) porque hay dos '1's y un '0'.
```

#### 2.1.4. Función generar\_archivo\_salida\_organizado(n, nombre\_archivo)

Generar un archivo de texto con las cadenas binarias organizadas por su longitud, comenzando con un símbolo para la lista vacía 'e', y escribir cada cadena junto con el número de '1's y '0's que contiene.

```
# Funcion para generar el archivo de salida con la 'e' al inicio y las cadenas
organizadas por longitud
def generar_archivo_salida_organizado(n, nombre_archivo="Universo/OutUniverso.txt"):
    with open(nombre_archivo, 'w') as archivo:
        archivo.write("{\n")
        archivo.write("e\n") #Poner 'e' al principio
        for longitud in range(1, n + 1): # Iterar por las longitudes de cadena desde 1
            hasta n
            for cadena in generar_cadenas(longitud):
                unos, ceros = contar_unos(cadena)
                archivo.write(f"{cadena}, {unos}, {ceros}\n")
            archivo.write("}\n")
    print("Generacion de cadenas organizada completada.")
```

### Fucionamiento de la Función:

#### Escribe una estructura en el archivo:

Abre o crea un archivo de salida, *Universo/OutUniverso.txt* en modo de escritura ('w').

1. Empieza con un corchete para la lista.
2. Añade el símbolo 'e' para la lista.
3. Para cada cadena de longitud de 1 a n, escribe la cadena junto con el conteo de '1's y '0's.
4. Cierra la estructura con un corchete al final.

#### 2.1.5. Función generar\_csv\_organizado(n, nombre\_archivo))

Generar un archivo CSV diseñado para las cadenas binarias generadas en *OutUniverso.txt*.

```
# Funcion para generar el archivo CSV
def generar_csv_organizado(n, nombre_archivo="Universo/plotDataSet.csv"):
    print(f"Generando cadenas organizadas por longitud para n = {n}...")
    generar_archivo_salida_organizado(n)
    with open("Universo/OutUniverso.txt", "r") as archivo:
        lineas = archivo.readlines()

    print("Generando archivo CSV...")
    with open(nombre_archivo, 'w') as csv_file:
        csv_file.write("id,OnesNumber,ZeroesNumber\n")
        for idx, linea in enumerate(lineas[2:-1]): # Saltar la primera linea
            cadena, unos, ceros = linea.strip().split(", ")
            csv_file.write(f"{idx},{unos},{ceros}\n")
    print(f"Archivo CSV generado: {nombre_archivo}")
```

### Fucionamiento de la Función:

Se llama la función: *generar\_archivo\_salida\_organizado(n)* para crear el archivo de texto con las cadenas y los conteos de '1's y '0's.

Lee ese archivo (*OutUniverso.txt*) línea por línea.

#### Creación de CSV:

Se pone el en la primera linea del CVS: **id,OnesNumber,ZeroesNumber**.

Para cada cadena, el CSV registra como: el número de '1's y el número de '0's.  
Imprime un mensaje avisando al usuario que el CSV ha sido creado.

### 2.1.6. Función menu()

Ofrecer un menú interactivo al usuario para tener opciones:

1. Generar cadenas de longitud aleatoria entre 1 y 1000.
2. Elegir manualmente un valor para n entre 1 y 1000.
3. Salir del programa.

```
# Funcion para mostrar el menu y obtener la entrada del usuario
def menu():
    while True:
        print("\nMenu:")
        print("1. Generar cadenas con n aleatorio entre 1-1000")
        print("2. Elegir n manualmente (1-1000)")
        print("3. Salir")

        opcion = input("Seleccione una opcion: ")

        if opcion == '1':
            n = random.randint(1, 1000) # Limitar el rango para evitar problemas de
                memoria
            print(f"Generando cadenas con n = {n}")
            generar_csv_organizado(n)
        elif opcion == '2':
            n = int(input("Introduce el valor de n (1-1000): "))
            if 1 <= n <= 1000:
                generar_csv_organizado(n)
            else:
                print("Por favor, ingresa un numero entre 1 y 1000.")
        elif opcion == '3':
            print("Cerrando el programa...")
            break
        else:
            print("Opcion invalida. Intenta de nuevo.")
```

#### Funcionamiento de la Función:

El menú se presenta en un bucle infinito (*whileTrue*), por lo que seguirá pidiendo al usuario que elija una opción hasta que seleccione la opción de salir ('3').

Según la opción seleccionada:

**Opción 1:** Elige un n aleatorio entre 1 y 1000 y genera los archivos.

**Opción 2:** Permite al usuario ingresar manualmente el valor de n y generar los archivos.

**Opción 3:** Termina el programa.

### 2.1.7. Ejecución del Programa

Se ejecuta el *main* con la función *menu()* cuando el programa se ejecuta directamente.

```
# Ejecucion del programa
if __name__ == "__main__":
    menu()
```

## 2.2. Graficar\_universo.py

### 2.2.1. Librerías Importadas

```
import pandas as pd
import datashader as ds
from datashader import transfer_functions as tf
from datashader.utils import export_image
import numpy as np
```

#### Librería pandas

La librería es para manipulación y análisis de datos y en este programa, se usa para cargar un archivo CSV en un DataFrame y convertir las columnas en datos numéricos.

#### Librería datashader

La librería para crear visualizaciones eficientes de grandes conjuntos de datos y genera imágenes basadas como gráficos de dispersión.

**Las funciones clave usadas aquí son:**

*Canvas*: Define el área de la imagen (ancho y alto).

*transfer\_functions.shade*: Colorea la agregación generada por el lienzo.

*export\_image*: Exporta la imagen generada a un archivo de imagen.

#### Librería numpy:

La librería para realizar operaciones numéricas y este programa se va a utilizar para aplicar transformaciones logarítmicas a los datos.

### 2.2.2. Cargar el DataFrame desde un archivo CSV

Cargar el archivo CSV, en un DataFrame de pandas, el archivo contiene información sobre las cadenas binarias generadas del archivo de texto, que contiene el número de '1's y '0's en cada cadena.

```
# Cargar el DataFrame desde un archivo CSV
df = pd.read_csv('Universo/plotDataSet.csv')
```

### 2.2.3. Convertir las columnas a valores numéricos

Las columnas 'id', 'OnesNumber' y 'ZeroesNumber' del DataFrame están en formato numérico. Si hay algún valor no convertible a número, se reemplaza con *NaN(errors='coerce')*.

```
df['id'] = pd.to_numeric(df['id'], errors='coerce')
df['OnesNumber'] = pd.to_numeric(df['OnesNumber'], errors='coerce')
df['ZeroesNumber'] = pd.to_numeric(df['ZeroesNumber'], errors='coerce')
```

### 2.2.4. Crear un lienzo (Canvas) para los gráficos

Crear un lienzo (canvas) de tamaño 800x400 píxeles donde se dibujarán los puntos para los gráficos.

```
# Crear las imagenes individuales
canvas = ds.Canvas(plot_width=800, plot_height=400)
```

### 2.2.5. Generar la imagen de los '1's y '0's

Generar un gráfico de dispersión en el que se representa para los '1's:

1. El eje X ('id') con los identificadores de las cadenas.
2. El eje Y ('OnesNumber') con el número de '1's en cada cadena.



```
# Unos (en azul)
agg_ones = canvas.points(df, 'id', 'OnesNumber')
img_ones = tf.shade(agg_ones, cmap=["blue"], how='eq_hist')
export_image(img_ones, filename='ones_image')
```

Generar un gráfico de dispersión en el que se representa para los '0's:

1. El eje X ('id') con los identificadores de las cadenas.
2. El eje Y ('ZeroesNumber') con el número de '1's en cada cadena.

```
# Ceros (en rojo)
agg_zeros = canvas.points(df, 'id', 'ZeroesNumber')
img_zeros = tf.shade(agg_zeros, cmap=["red"], how='eq_hist')
export_image(img_zeros, filename='zeros_image')
```

### Fucionamiento de la Función:

**canvas.points():** Genera una matriz de agregación de puntos donde los valores se distribuyen en función de los ejes seleccionados ('id' y 'OnesNumber' o 'ZeroesNumber').

**tf.shade():** Aplica un mapa de colores a la agregación, para los 'OnesNumber' de color azul y para los 'ZeroesNumber' de color rojo.

**export\_image():** Exporta la imagen resultante como un archivo llamado 'ones\_image.png' y 'zeros\_image.png'.

## 2.2.6. Aplicar transformaciones logarítmicas

Se aplica una transformación logarítmica a los valores de 'OnesNumber' y 'ZeroesNumber'. para la visualización es útil cuando los datos tienen una gran cantidad.

```
df['log_OnesNumber'] = np.log10(df['OnesNumber'] + 1) # Evitar log(0)
df['log_ZeroesNumber'] = np.log10(df['ZeroesNumber'] + 1)
```

### Fucionamiento de la Función:

Para evitar problemas con el logaritmo de cero ( $\log(0)$ ), se le suma 1 a cada valor de las columnas antes de aplicar `np.log10()`.

**Se crean dos nuevas columnas en el DataFrame:**

'log\_OnesNumber': El logaritmo en base 10 del número de '1's.

'log\_ZeroesNumber': El logaritmo en base 10 del número de '0's.

## 2.2.7. Generar la imagen logarítmica de los '1's y '0's

Crear un gráfico logarítmico para los '1's transformados. Aquí los valores logarítmicos de los '1's se muestran en el eje Y y los identificadores de las cadenas en el eje X.

```
agg_log_ones = canvas.points(df, 'id', 'log_OnesNumber')
img_log_ones = tf.shade(agg_log_ones, cmap=["blue"], how='eq_hist')
export_image(img_log_ones, filename='log_ones_image')
```

Crear un gráfico logarítmico para los '0's transformados. Aquí los valores logarítmicos de los '0's se muestran en el eje Y y los identificadores de las cadenas en el eje X. '0's se muestran en el eje Y y los identificadores de las cadenas en el eje X.

```
agg_log_zeros = canvas.points(df, 'id', 'log_ZeroesNumber')
img_log_zeros = tf.shade(agg_log_zeros, cmap=["red"], how='eq_hist')
export_image(img_log_zeros, filename='log_zeros_image')
```

### Fucionamiento de la Función:

Utiliza 'log\_OnesNumber' o 'log\_ZerosNumber' como el eje Y y mantiene el identificador 'id' en el eje X.

Se aplica la misma técnica de sombreado y escala de colores para azul o rojo, exportando las imagenes a

*'log\_ones\_image.png'* o *'log\_zeros\_image.png'*.

### 2.2.8. Mensaje Final

Imprime un mensaje para indicar que todas las imágenes han sido generadas y exportadas correctamente.

```
print("Imágenes generadas.")
```

### 3. Salidas

#### 3.1. Complejidad Temporal

La complejidad temporal de los programas es la generación de cadenas binarias y las operaciones asociadas con el manejo de archivos CSV.

##### 3.1.1. Generación de cadenas binarias

La función `generar_cadenas(n)` genera todas las posibles combinaciones binarias de longitud `n`. Esta operación tiene una complejidad de:

$$O(2^n)$$

##### 3.1.2. Conteo de '1's y '0's en cada cadena

Para cada cadena generada, la función `contar_unos(cadena)` cuenta los '1's y los '0's en la cadena. Dado que esta operación recorre cada carácter de la cadena, tiene una complejidad de:

$$O(n)$$

##### 3.1.3. Generación del archivo CSV

La función `generar_csv_organizado(n)` crea un archivo CSV que contiene la información de todas las cadenas binarias generadas. Para ello, se realizan las siguientes operaciones:

- Generar todas las cadenas binarias posibles ( $O(2^n)$ ).
- Contar los '1's y '0's en cada cadena ( $O(n)$  por cadena).
- Escribir cada línea en el archivo ( $O(1)$  por cadena).

La complejidad total de esta operación es:

$$O(n \cdot 2^n)$$

##### 3.1.4. Visualización de los datos

El proceso de visualización mediante `datashader` implica crear gráficos en base a los datos del archivo CSV. La función `Canvas.points()` tiene una complejidad de:

$$O(N)$$

##### 3.1.5. Complejidad total

La complejidad total del programa está dominada por la generación de las cadenas binarias y el conteo de los '1's y '0's, lo que resulta en una complejidad de:

$$O(n \cdot 2^n)$$

### 3.2. OutUniverso.txt

Este es el archivo de salida del algoritmo. En este se plasma la unión de las  $n$ -ésimas potencias, al conjunto resultante lo llamamos Universo. Está especificado en términos de conjuntos, donde la cadena vacía es representada con  $e$ .

A continuación se indexa el contenido del archivo para una longitud  $n = 29$ .

$$\sum_0 \cup \sum_1 \cup \sum_2 \cup \sum_3 \cup \sum_5 \cup \sum_6 \dots$$

$$Con : \sum \{0, 1\}$$

```
OutUniverso.txt {
1 {
2 e
3 0, 0, 1
4 1, 1, 0
5 00, 0, 2
6 01, 1, 1
7 10, 1, 1
8 11, 2, 0
9 000, 0, 3
10 001, 1, 2
11 010, 1, 2
12 011, 2, 1
13 100, 1, 2
14 101, 2, 1
15 110, 2, 1
16 111, 3, 0
17 0000, 0, 4
18 0001, 1, 3
19 0010, 1, 3
20 0011, 2, 2
21 0100, 1, 3
22 0101, 2, 2
23 0110, 2, 2
24 0111, 3, 1
25 1000, 1, 3
26 1001, 2, 2
27 1010, 2, 2
28 1011, 3, 1
29 1100, 2, 2
30 1101, 3, 1
31 1110, 3, 1
32 1111, 4, 0
```

Figura 1: Archivo de *OutUniverse.txt* en n=29, hasta mostrando n=4.

[illegible]

Figura 2: Archivo de *OutUniver*so.txt en n=29, parte final.

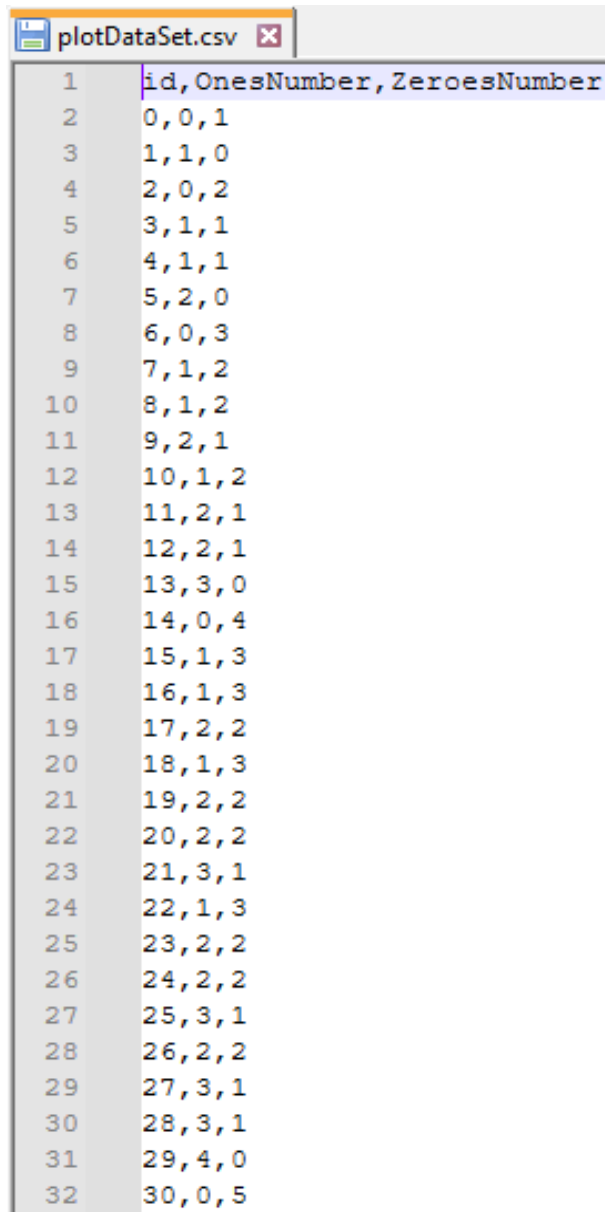
### 3.3. plotDataSet.csv

Este archivo es generado por el programa `graficar_universo.py`. En este, hay dos columnas: `id`, `OnesNumber` y `ZeroesNumber`

Es un archivo temporal utilizado para generar las imágenes de los gráficos. Internamente tiene la siguiente estructura:

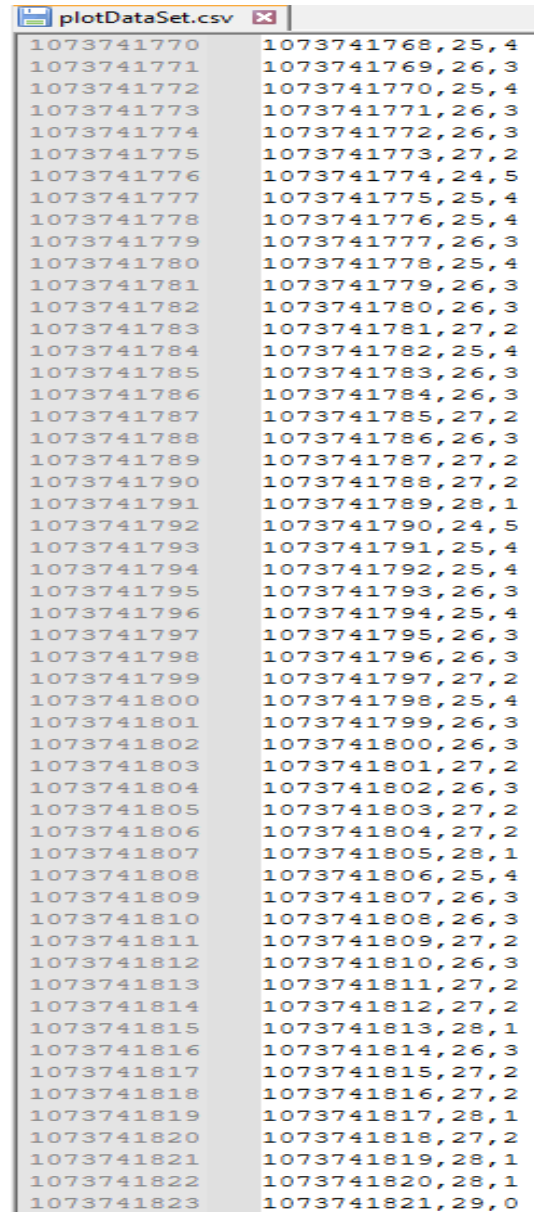
`id, OnesNumber, ZeroesNumber`

A continuación, se muestra un ejemplo de cómo se estructuran los datos en el archivo:



	<code>id, OnesNumber, ZeroesNumber</code>
1	
2	0, 0, 1
3	1, 1, 0
4	2, 0, 2
5	3, 1, 1
6	4, 1, 1
7	5, 2, 0
8	6, 0, 3
9	7, 1, 2
10	8, 1, 2
11	9, 2, 1
12	10, 1, 2
13	11, 2, 1
14	12, 2, 1
15	13, 3, 0
16	14, 0, 4
17	15, 1, 3
18	16, 1, 3
19	17, 2, 2
20	18, 1, 3
21	19, 2, 2
22	20, 2, 2
23	21, 3, 1
24	22, 1, 3
25	23, 2, 2
26	24, 2, 2
27	25, 3, 1
28	26, 2, 2
29	27, 3, 1
30	28, 3, 1
31	29, 4, 0
32	30, 0, 5

Figura 3: Archivo de `plotDataSet.csv` en `n=29`, mostrando hasta `n=4`.



1073741770	1073741768, 25, 4
1073741771	1073741769, 26, 3
1073741772	1073741770, 25, 4
1073741773	1073741771, 26, 3
1073741774	1073741772, 26, 3
1073741775	1073741773, 27, 2
1073741776	1073741774, 24, 5
1073741777	1073741775, 25, 4
1073741778	1073741776, 25, 4
1073741779	1073741777, 26, 3
1073741780	1073741778, 25, 4
1073741781	1073741779, 26, 3
1073741782	1073741780, 26, 3
1073741783	1073741781, 27, 2
1073741784	1073741782, 25, 4
1073741785	1073741783, 26, 3
1073741786	1073741784, 26, 3
1073741787	1073741785, 27, 2
1073741788	1073741786, 26, 3
1073741789	1073741787, 27, 2
1073741790	1073741788, 27, 2
1073741791	1073741789, 28, 1
1073741792	1073741790, 24, 5
1073741793	1073741791, 25, 4
1073741794	1073741792, 25, 4
1073741795	1073741793, 26, 3
1073741796	1073741794, 25, 4
1073741797	1073741795, 26, 3
1073741798	1073741796, 26, 3
1073741799	1073741797, 27, 2
1073741800	1073741798, 25, 4
1073741801	1073741799, 26, 3
1073741802	1073741800, 26, 3
1073741803	1073741801, 27, 2
1073741804	1073741802, 26, 3
1073741805	1073741803, 27, 2
1073741806	1073741804, 27, 2
1073741807	1073741805, 28, 1
1073741808	1073741806, 25, 4
1073741809	1073741807, 26, 3
1073741810	1073741808, 26, 3
1073741811	1073741809, 27, 2
1073741812	1073741810, 26, 3
1073741813	1073741811, 27, 2
1073741814	1073741812, 27, 2
1073741815	1073741813, 28, 1
1073741816	1073741814, 26, 3
1073741817	1073741815, 27, 2
1073741818	1073741816, 27, 2
1073741819	1073741817, 28, 1
1073741820	1073741818, 27, 2
1073741821	1073741819, 28, 1
1073741822	1073741820, 28, 1
1073741823	1073741821, 29, 0

Figura 4: Archivo de `plotDataSet.csv` en `n=29`, parte final.

3.4. Graficas de Datos y Logaritmicas Base 10 de '1's y '0's

3.4.1. Grafica de *OnesNumber*

Se trata del la distribución del numero de '1's en cada cadena del universo.

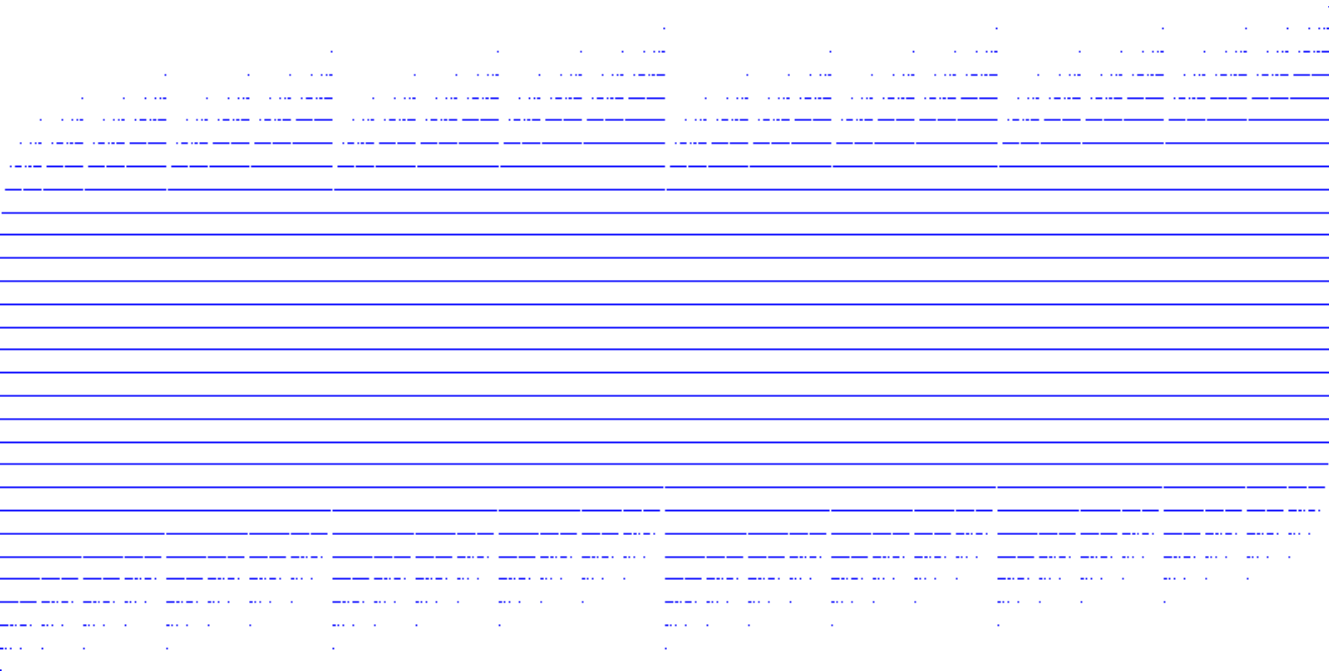


Figura 5: Grafica de Cadenas n=29.

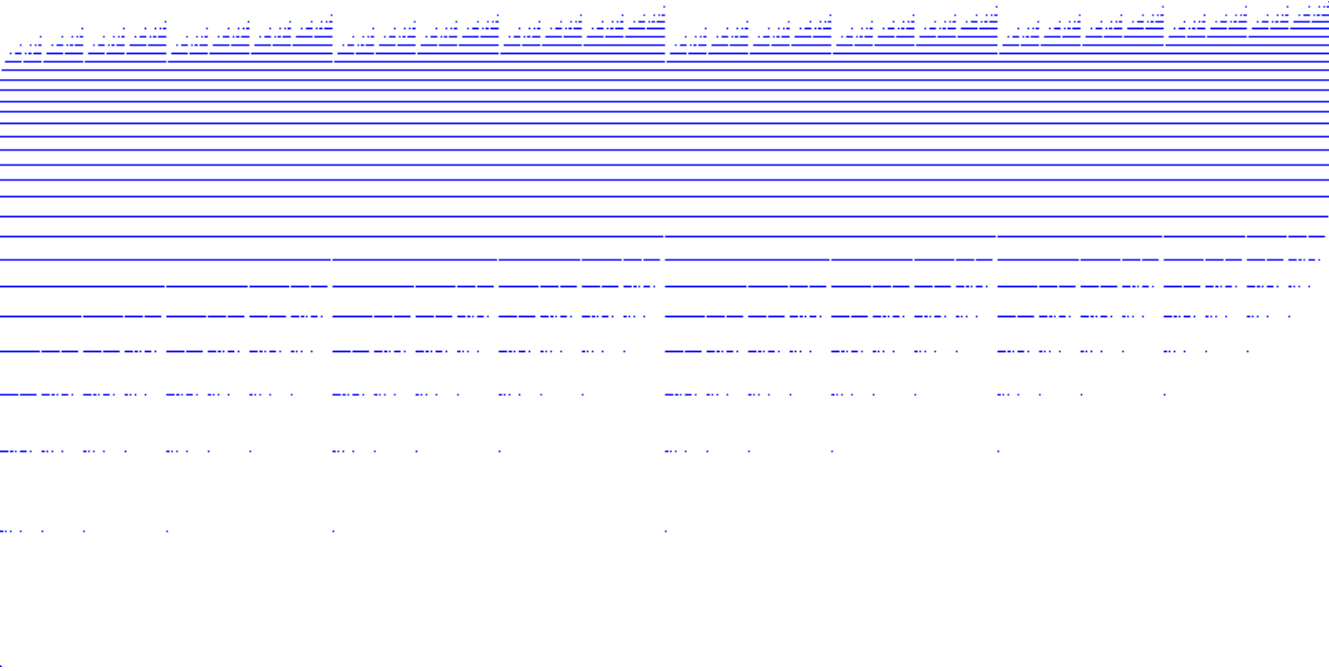


Figura 6: Grafica de n=29 en Logaritmo base 10.

### 3.4.2. Grafica de *ZeroesNumber*

Se trata del la distribución del numero de '0's en cada cadena del universo.

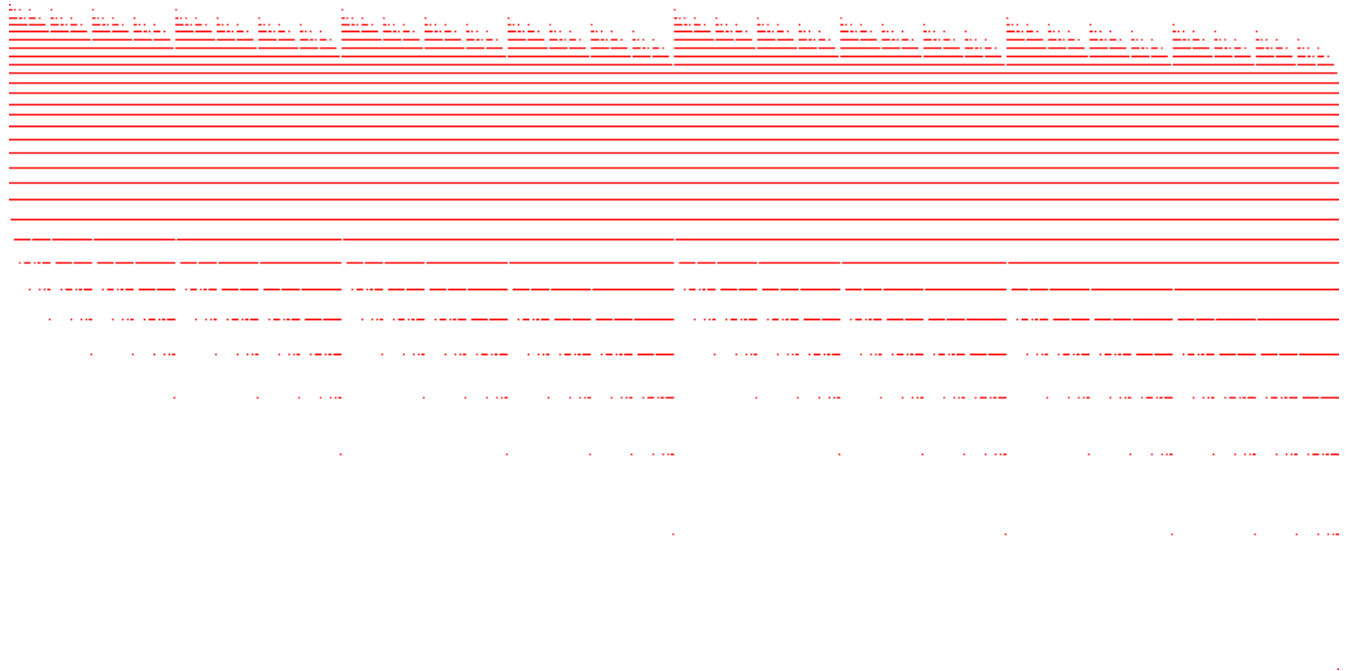


Figura 7: Grafica de Cadenas n=29.

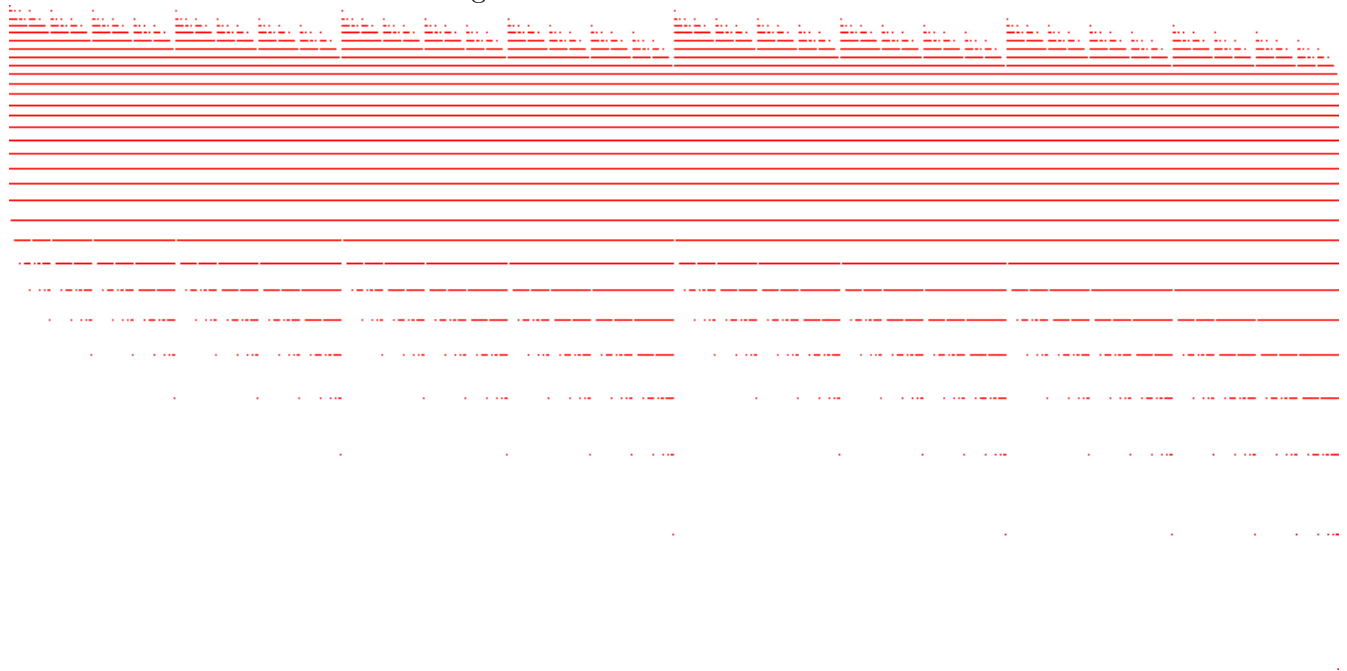


Figura 8: Grafica de n=29 en Logaritmo base 10.

## 4. Conclusion

Dado este programa llamado Universo sobre generación, análisis y visualización de cadenas binarias de longitud variable, centrándose en el conteo de '1's y '0's en cada cadena, se encarga de crear todas las posibles combinaciones de cadenas binarias de longitud  $n$ , que puede ser un número aleatorio entre 1 y 1000 o un valor específico proporcionado por el usuario. Cada cadena generada se analiza para contar el número de '1's y '0's, y esta información se almacena en un archivo de texto estructurado y el archivo CSV creado para las graficas como datashader, para crear representaciones gráficas de los datos. Se generan gráficos que muestran el número de '1's y '0's en cada cadena, así como gráficos logarítmicos que permiten una mejor visualización de la distribución de estos valores, especialmente cuando hay grandes variaciones en los conteos.

Además, este enfoque es especialmente valioso en contextos educativos, donde se puede la programación, combinatoria y visualización de datos. Tambien integración de la generación de cadenas binarias y su visualización.

## 5. Referencias

@bookhopcroft2001introduction, author = John E. Hopcroft and Rajeev D. Ullman and Jeffrey D. Ullman, title = Introduction to Automata Theory, Languages, and Computation, publisher = Addison-Wesley, year = 2001, url = <https://www-2.dc.uba.ar/staff/becher/Hopcroft-Motwani-Ullman-2001.pdf>

@miscullman2010jeffslides, author = Jeffrey D. Ullman, title = Jeff's Slides for CS154, Spring 2010, year = 2010, note = Stanford University, url = <http://infolab.stanford.edu/ullman/ialc/spr10/spr10.html>LECTURE