



**INSTITUTO POLITÉCNICO
NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



**Reporte de práctica:
Universo de cadenas binarias**

Grupo:
4CM2

Nombre del alumno:
Maldonado Hernández Alexander

Unidad de aprendizaje:
Teoría de la computación

Maestro:
Genaro Juárez Martínez

Fecha de entrega:
7 de abril de 2025

Índice

1. Introducción	1
2. Desarrollo	2
2.1. Implementación del Programa	2
2.2. Gráficas	4
3. Conclusiones	6
4. Referencias	7
5. Anexos	8
5.1. Anexo 1. Código fuente del programa	8
5.2. Anexo 2. Código para la graficación	10

1. Introducción

En teoría de lenguajes formales y ciencias de la computación, el universo de cadenas binarias de longitud n , denotado como Σ^n , representa el conjunto de todas las posibles combinaciones de bits (0 y 1) para una longitud dada. Este concepto fundamental tiene aplicaciones en diversas áreas como criptografía, teoría de la información, diseño de circuitos lógicos, procesamiento de señales y algoritmos de optimización. En particular, la generación y análisis de cadenas binarias desempeña un papel clave en problemas de conteo, representación de espacios de búsqueda y modelado de sistemas basados en reglas binarias.

Uno de los problemas fundamentales en la teoría de la computación es el análisis de estructuras combinatorias y la forma en que las combinaciones binarias pueden ser utilizadas en la resolución de problemas. La exploración del universo de cadenas binarias permite estudiar patrones emergentes en la distribución de bits, lo que resulta útil para el diseño de algoritmos eficientes y la optimización de representaciones de datos. Además, esta exploración es aplicable en el campo del aprendizaje automático, donde la representación binaria de conjuntos de datos juega un papel crucial en el desarrollo de modelos computacionales.

El objetivo de esta práctica es implementar un programa que genere todas las cadenas binarias posibles para una longitud n dada, dentro del rango $[0, 1000]$. Dicho programa debe contar con la capacidad de ejecutarse en dos modos: automático y manual. En el modo automático, el programa debe generar el conjunto completo de cadenas sin intervención del usuario, mientras que en el modo manual, debe permitir la entrada de valores personalizados para n . Este enfoque flexible facilita la exploración y el análisis del comportamiento de las cadenas binarias en distintos contextos. Adicionalmente, se analizará la distribución del número de unos en las cadenas generadas para $n = 29$. Este análisis se llevará a cabo utilizando representaciones gráficas tanto en escala lineal como logarítmica, lo que permitirá observar patrones combinatorios de interés. La visualización de la densidad de unos en las cadenas contribuirá a comprender la distribución estadística de los bits dentro del conjunto total de combinaciones posibles y podrá revelar estructuras subyacentes en la organización de los datos binarios.

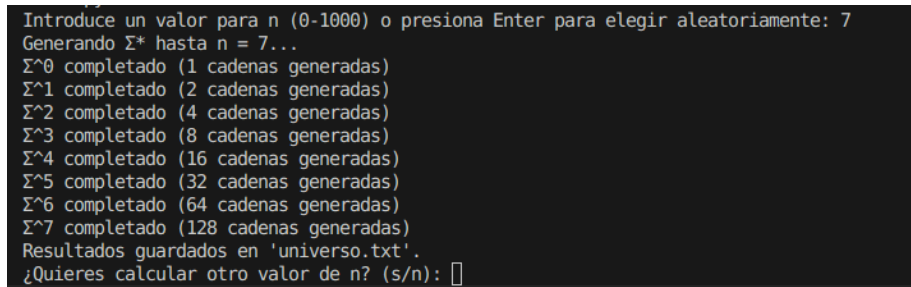
2. Desarrollo

2.1. Implementación del Programa

El programa fue desarrollado en Python debido a su versatilidad y capacidades para manejar estructuras de datos y generación de gráficos. Las principales características implementadas son:

- Generación de todas las cadenas binarias de longitud n y la unión de valores de n inferiores usando la enumeración exhaustiva
- Modos de operación: automático (genera n aleatoria) y manual (usuario introduce n)
- Validación del rango $n \in [0, 1000]$
- Salida en notación de conjuntos a archivo de texto
- Cálculo del número de unos por cadena y el logaritmo natural de estos resultados

El código fuente del programa se puede encontrar en el Anexo 1. Además, a continuación se muestra una foto de los archivos que crea el programa, como se encuentran formateados y como se ve la interfaz del programa en la terminal:



```
Introduce un valor para n (0-1000) o presiona Enter para elegir aleatoriamente: 7
Generando  $\Sigma^*$  hasta n = 7...
 $\Sigma^0$  completado (1 cadenas generadas)
 $\Sigma^1$  completado (2 cadenas generadas)
 $\Sigma^2$  completado (4 cadenas generadas)
 $\Sigma^3$  completado (8 cadenas generadas)
 $\Sigma^4$  completado (16 cadenas generadas)
 $\Sigma^5$  completado (32 cadenas generadas)
 $\Sigma^6$  completado (64 cadenas generadas)
 $\Sigma^7$  completado (128 cadenas generadas)
Resultados guardados en 'universo.txt'.
¿Quieres calcular otro valor de n? (s/n):
```

Figura 1: Interfaz del programa en la terminal

Tal como se ve en la Figura 1, tras elegir un valor de n (o que sea elegido aleatoriamente si así se desea) el programa calculará todas las cadenas posibles para cada Σ , además de mostrar el progreso en tiempo real de los sigmas que ya ha calculado.

Por otro lado, se puede observar en la Figura 2 y en la Figura 3 los archivos que se generan tras ejecutar el programa. Por un lado, se genera un archivo "universo.txt", el cual muestra todas las cadenas posibles para cada valor de Σ^n . Por otro lado, también se genera un archivo "datos.txt" en el cual se obtiene el número de unos en cada cadena, el logaritmo natural de estos y el peso que tenía el documento del universo al momento de escribir ese dato, todo esto con el propósito de la graficación de los datos.

```

universo.txt
1   $\Sigma^0 = \{\epsilon\}$ 
2   $\Sigma^1 = \{0,1\}$ 
3   $\Sigma^2 = \{00,01,10,11\}$ 
4   $\Sigma^3 = \{000,001,010,011,100,101,110,111\}$ 
5   $\Sigma^4 = \{0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111\}$ 
6

```

Figura 2: Formato del archivo universo.txt

```

datos.txt > data
1  tamaño,unos,log_unos
2  0.00,0,0
3  0.00,0,0
4  0.00,1,0.0
5  0.00,0,0
6  0.00,1,0.0
7  0.00,1,0.0
8  0.00,2,0.6931471805599453
9  0.00,0,0
10 0.00,1,0.0
11 0.00,1,0.0
12 0.00,2,0.6931471805599453
13 0.00,1,0.0
14 0.00,2,0.6931471805599453
15 0.00,2,0.6931471805599453
16 0.00,3,1.0986122886681098
17 0.00,0,0
18 0.00,1,0.0
19 0.00,1,0.0
20 0.00,2,0.6931471805599453
21 0.00,1,0.0
22 0.00,2,0.6931471805599453
23 0.00,2,0.6931471805599453
24 0.00,3,1.0986122886681098
25 0.00,1,0.0
26 0.00,2,0.6931471805599453
27 0.00,2,0.6931471805599453
28 0.00,3,1.0986122886681098
29 0.00,2,0.6931471805599453
30 0.00,3,1.0986122886681098
31 0.00,3,1.0986122886681098
32 0.00,4,1.3862943611198906

```

Figura 3: Formato del archivo datos.txt

2.2. Gráficas

Para la realización de las gráficas para visualizar la distribución de unos en las cadenas, se tuvo que dividir la información en dos gráficas, ya que de otra manera era imposible graficar los datos. Ya que mi computadora solo cuenta con 16GB de RAM, por lo que cargar el archivo de 30.1GB de espacio era inviable. A continuación en la Figura 4 y la Figura 5 se ve la distribución de unos para las cadenas.

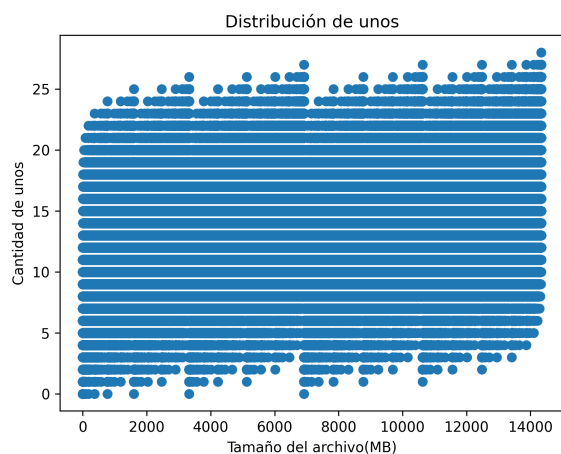


Figura 4: Distribución del número de unos para cadenas con $n=29$

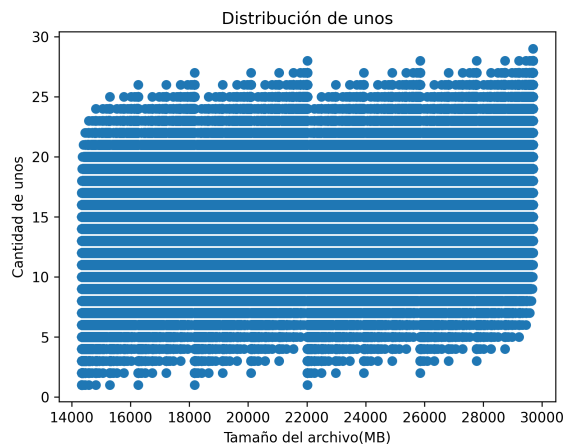


Figura 5: Distribución del número de unos para cadenas con $n=29$ (Continuación)

Por otro lado, podemos observar en la Figura 6 y la Figura 7 la misma distribución pero con los valores logarítmicos.

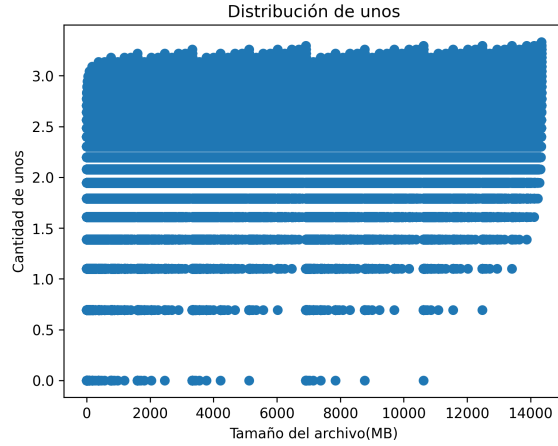


Figura 6: Distribución logarítmica del número de unos para $n=29$

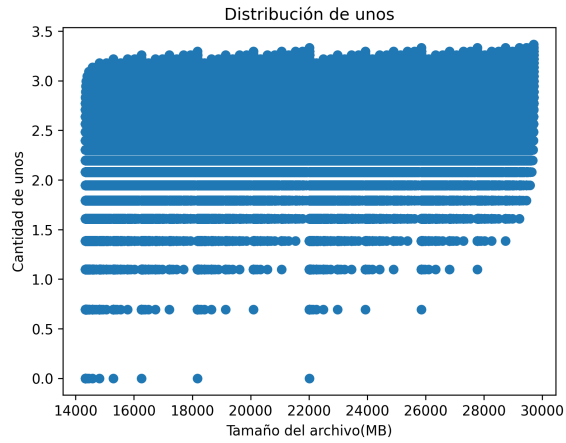


Figura 7: Distribución logarítmica del número de unos para $n=29$ (Continuación)

La gráfica logarítmica revela la naturaleza simétrica de la distribución binomial y cómo los valores del extremo inferior (cadenas con muy pocos unos) son exponencialmente menos frecuentes. El código que se usó para la graficación se puede encontrar en el Anexo 2. Por último, es importante recalcar que el código está diseñado para graficar $n = 29$, por lo que con valores de n mayores o menores podría no funcionar.

3. Conclusiones

La presente práctica permitió explorar el universo de cadenas binarias desde una perspectiva computacional, implementando un programa que genera y analiza combinaciones de bits para diferentes valores de n . A través de esta implementación, se logró no solo validar la correcta generación de cadenas, sino también analizar la distribución de unos en las mismas, utilizando representaciones gráficas en escalas lineales y logarítmicas. Dichas representaciones evidenciaron patrones estadísticos relevantes, como la distribución binomial de los unos en las cadenas y su comportamiento simétrico en el dominio logarítmico.

Asimismo, el desarrollo de esta práctica destacó la importancia del manejo eficiente de datos a gran escala, ya que la generación del universo completo para $n = 29$ representó un reto computacional debido al tamaño exponencial de los resultados. La necesidad de segmentar la información para su procesamiento y visualización puso de manifiesto las limitaciones de hardware y la relevancia de técnicas de optimización en el análisis de grandes volúmenes de datos.

En conclusión, esta práctica reforzó conceptos fundamentales de la teoría de la computación y su aplicación en problemas de conteo y análisis combinatorio. Además, evidenció la utilidad de herramientas computacionales en la exploración de espacios de búsqueda y en la interpretación de datos mediante visualización gráfica. El enfoque adoptado no solo permitió comprender mejor las propiedades del universo de cadenas binarias, sino que también sentó bases para futuras aplicaciones en campos como criptografía, inteligencia artificial y optimización de algoritmos.

4. Referencias

- Collette. (2014, 31 de octubre). *HDF5 for Python — h5py 3.13.0 documentation*. Consultado el 3 de abril de 2025, desde <https://docs.h5py.org/en/stable/>
- Vaex. (2016, 10 de mayo). *What is Vaex? — vaex 4.17.0 documentation*. Consultado el 3 de abril de 2025, desde <https://vaex.readthedocs.io/en/latest/>
- Vaex. (2018, 12 de diciembre). *Vaex: A Fast DataFrame for Python*. Consultado el 3 de abril de 2025, desde <https://vaex.io/>

5. Anexos

5.1. Anexo 1. Código fuente del programa

```
import random
import os
import math

def generar_universo(n):
    #datos.txt sera el archivo con el que graficaremos los datos
    with open("universo.txt", "w") as f, open("datos.txt", "w") as d:

        tamaño_f = 0
        unos = 0
        d.write("tamaño,unos,log_unos\n")

        for i in range(n + 1):
            f.write(f"^{i} = {{")
            for numero in range(2 ** i):
                cadena = bin(numero)[2:].zfill(i) if i > 0 else ""
                ↪ #Cadena vacía para ^0

                if numero < (2 ** i) - 1:
                    f.write(f"{cadena}" + ",")
                else:
                    f.write(f"{cadena}")

            f.flush()
            tamaño_f = os.path.getsize("universo.txt") / (1024 **
                ↪ 2)

            unos = cadena.count("1")
            log_unos = math.log(unos) if unos > 0 else unos
            d.write(f"{tamaño_f:.2f},{unos},{log_unos}\n")

        f.write("}\n")
        print(f"^{i} completado ({2**i} cadenas generadas)")
        ↪ #Mostrar progreso
```

```

def main():
    while True:
        entrada = input("Introduce un valor para n (0-1000) o
        ↪ presiona Enter para elegir aleatoriamente: ")

        if entrada == "":
            n = random.randint(0, 10)
            print(f"Se ha elegido aleatoriamente n = {n}")
        else:
            try:
                n = int(entrada)
                if not (0 <= n <= 1000):
                    print("Por favor, introduce un número entre 0 y
                    ↪ 1000.")
                    continue
            except ValueError:
                print("Entrada inválida. Introduce un número
                ↪ entero.")
                continue

        print(f"Generando * hasta n = {n}...")
        generar_universo(n)
        print(f"Resultados guardados en 'universo.txt'.")

        while True:
            repetir = input("¿Quieres calcular otro valor de n?
            ↪ (s/n): ").strip().lower()
            if repetir in ["s", "n"]:
                break
            else:
                print("Entrada inválida. Seleccione \"s\" o \"n\".")

        if repetir == "n":
            break

if __name__ == "__main__":
    main()

```

5.2. Anexo 2. Código para la graficación

```
import vaex
import matplotlib.pyplot as plt
import numpy as np

def txt_a_hdf5():
    df = vaex.from_csv('datos.txt', sep=',', convert='datos.hdf5')

def graficar_datos():
    df = vaex.open("datos.hdf5")[0:536870911]

    #Grafica normal parte 1
    plt.figure()
    plt.title("Distribución de unos")
    df.viz.scatter(x='tamano', y='unos', length_check=False,
        ↪ xlabel='Tamaño del archivo(MB)', ylabel='Cantidad de unos')
    plt.savefig("grafica_1", dpi=300, bbox_inches='tight')

    #Grafica con logaritmo parte 1
    plt.figure()
    plt.title("Distribución de unos")
    df.viz.scatter(x='tamano', y='log_unos', length_check=False,
        ↪ xlabel='Tamaño del archivo(MB)', ylabel='Cantidad de unos')
    plt.savefig("graficaLog_1", dpi=300, bbox_inches='tight')

    df = vaex.open("datos.hdf5")[536870912:1073741823]

    #Grafica normal parte 2
    plt.figure()
    plt.title("Distribución de unos")
    df.viz.scatter(x='tamano', y='unos', length_check=False,
        ↪ xlabel='Tamaño del archivo(MB)', ylabel='Cantidad de unos')
    plt.savefig("grafica_2", dpi=300, bbox_inches='tight')

    #Grafica con logaritmo parte 2
    plt.figure()
    plt.title("Distribución de unos")
```

```

df.viz.scatter(x='tamano', y='log_unos', length_check=False,
    ↪ xlabel='Tamaño del archivo(MB)', ylabel='Cantidad de unos')
plt.savefig("graficaLog_2", dpi=300, bbox_inches='tight')

def main():
    txt_a_hdf5()
    graficar_datos()

if __name__ == "__main__":
    main()

```