1.- Creación del diccionario (Combinaciones Binarias).

Felipe Sánchez Martínez
Escuela Superior de Cómputo, IPN

Septiembre de 2023

Índice general

1. Planteamiento del problema. **1**

2. Marco Teórico. **3**
- 2.1. Los Conceptos Centrales de la Teoría de Autómatas . . . 3
  - 2.1.1. Alfabetos . . . 3
  - 2.1.2. Cadenas . . . 3
  - 2.1.3. Lenguajes . . . 4

3. Desarollo del problema. **5**
  - 3.0.1. Approach. . . . 5
  - 3.0.2. Complejidad . . . 6
  - 3.0.3. Codigo fuente. . . . 6
  - 3.0.4. Casos de prueba. . . . 11

4. Conclusión. **19**

5. Referencias. **21**

 Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 1

Planteamiento del problema.

La representación binaria es una parte fundamental de las ciencias de la computación, ya que forma la base de la comunicación y el procesamiento de datos en sistemas digitales. En el mundo de la informática, cada número, carácter o símbolo se puede expresar mediante una combinación de dígitos binarios, es decir, unos y ceros.

La siguiente práctica muestra el planteamiento de la creación de un diccionario que aborda la representación de todas las secuencias binarias de longitud 'k', así como sus antecesoras, para poder formar su diccionario. El diccionario consistirá en una lista exhaustiva de todas las combinaciones posibles de dígitos binarios de cierta longitud 'k', desde las secuencias más simples hasta las más complejas. Por ejemplo, comenzando con la cadena vacía ϵ, hasta llegar a tener todas las combinaciones posibles dadas por el valor de 2^k y sus antecesoras $2^{k-1} + 2^{k-2} + \ldots + 2^1$.

Así se podrá formar el lenguaje binario, ampliamente utilizado en la programación y la electrónica, que se basa en estas combinaciones de dígitos binarios. Cada secuencia en el diccionario representa una palabra en este lenguaje, y su comprensión es esencial para entender cómo funcionan las operaciones binarias, como la adición, la multiplicación y la representación de datos.

En esta práctica, exploraremos la generación de este diccionario y analizaremos cómo se relaciona con el lenguaje binario, mediante su representación en una gráfica de los números de 1's y cómo se comporta al aplicarle una función logarítmica.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 2

Marco Teórico.

2.1. Los Conceptos Centrales de la Teoría de Autómatas

2.1.1. Alfabetos

Un alfabeto es un conjunto finito y no vacío de símbolos. Convencionalmente, utilizamos el símbolo Σ para representar un alfabeto. Algunos ejemplos comunes de alfabetos son:

1. $\Sigma = \{0, 1\}$, el alfabeto binario.
2. $\Sigma = \{a, b, \ldots, z\}$, el conjunto de todas las letras minúsculas.
3. El conjunto de todos los caracteres ASCII o el conjunto de todos los caracteres ASCII imprimibles.

2.1.2. Cadenas

Una cadena (o a veces palabra) es una secuencia finita de símbolos elegidos de algún alfabeto. Por ejemplo, 01101 es una cadena del alfabeto binario $\Sigma = \{0, 1\}$. La cadena 111 es otra cadena elegida de este alfabeto.

La Cadena Vacía

La cadena vacía es la cadena que no contiene ningún símbolo. Esta cadena, denotada como e, es una cadena que puede ser elegida de cualquier alfabeto.

Longitud de una Cadena

A menudo es útil clasificar las cadenas por su longitud, es decir, el número de posiciones para los símbolos en la cadena. Por ejemplo, 01101 tiene una longitud de 5. Es común decir que la longitud de una cadena es 'el número de símbolos' en la cadena; esta afirmación es aceptada coloquialmente pero no es estrictamente correcta. Así, en la cadena 01101, solo hay dos símbolos, 0 y 1, pero hay cinco posiciones para símbolos, y su longitud es 5. Sin embargo, generalmente se puede usar .[el] número de símbolosçuando se quiere decir ”número de posiciones”. La notación estándar para la longitud de una cadena w es $|w|$. Por ejemplo, $|011| = 3$ y $|| = 0$.

2.1.3. Lenguajes

Un conjunto de cadenas, todas elegidas de algún Σ^*, donde Σ es un alfabeto particular, se llama un lenguaje. Si Σ es un alfabeto y $L \subseteq \Sigma^*$, entonces L es un lenguaje sobre Σ. Nótese que un lenguaje sobre Σ no necesita incluir cadenas con todos los símbolos de Σ, por lo que una vez que hemos establecido que L es un lenguaje sobre Σ, también sabemos que es un lenguaje sobre cualquier alfabeto que sea un superconjunto de Σ.

La elección del término "lenguaje"puede parecer extraña. Sin embargo, los lenguajes comunes pueden verse como conjuntos de cadenas. Un ejemplo es el inglés, donde la colección de palabras en inglés legales es un conjunto de cadenas sobre el alfabeto que consiste en todas las letras. Otro ejemplo es C, o cualquier otro lenguaje de programación, donde los programas legales son un subconjunto de las posibles cadenas que se pueden formar a partir del alfabeto del lenguaje. Este alfabeto es un subconjunto de los caracteres ASCII. El alfabeto exacto puede diferir ligeramente entre diferentes lenguajes de programación, pero generalmente incluye letras mayúsculas y minúsculas, dígitos, signos de puntuación y símbolos matemáticos.

Sin embargo, también existen muchos otros lenguajes que aparecen cuando estudiamos autómatas. Algunos son ejemplos abstractos, como:

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 3

Desarollo del problema.

3.0.1. Approach.

Se decidió implementar el algoritmo utilizando dos ciclos 'for'. Por ejemplo, para un valor de $k = 3$, donde k representa la longitud de la cadena, la salida deseada debería tener el siguiente formato:

$$\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$$

En este caso, el primer ciclo abarcaría el número máximo que se puede formar con k bits (2^k), mientras que el segundo ciclo completaría el conjunto hasta ese punto.

```python
number_zeros_left = 0

for i in range(1, k_length + 1):
    number_zeros_left += 1
    bits = 2**i
    for j in range(bits):
        if i == k_length and j == bits - 1:
            output.write(bin(j)[2::].zfill(number_zeros_left))
        else:
            output.write(bin(j)[2::].zfill(number_zeros_left)+ ', ')
```

Una vez que se haya completado el código fuente, se desea graficar la cantidad de unos por cadena. Para esto, se utilizará la función de peso de Hamming para reducir el tiempo de ejecución en el cálculo del número de unos por cadena. Los resultados se almacenarán en un archivo '.csv' para su posterior manipulación con la biblioteca Pandas.

```python
def hammingWeight(n):
    """
    :type n: int
    :rtype: int
```

```
    """
    mask = 1
    count = 0
    while n != 0:
        if (mask & n == 1):
            count = count + 1
        n = n>>1

    return count
```

3.0.2. Complejidad

Complejidad Temporal

La complejidad del algoritmo se divide en dos componentes principales: la generación de las cadenas y la función de Hamming.

Para la generación de las cadenas, la complejidad temporal es:

$$T : O(2^k + 2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0) = O(2^k)$$

Por otro lado, al considerar la función de Hamming, la complejidad temporal es:

$$T : O(k \cdot 2^k)$$

En conjunto, la complejidad temporal total del algoritmo es:

$$O(2^k) + O(k \cdot 2^k) = O(2^k)$$

Complejidad Espacial

Dentro del algoritmo, la generación del output no se considera en la complejidad espacial, lo que resulta en una complejidad espacial de:

$$S : O(1)$$

Sin embargo, al momento de graficar, se utiliza memoria RAM, lo que eleva la complejidad espacial a:

$$S : O(2^k)$$

El uso de un archivo .csv en combinación con las bibliotecas Pandas y Matplotlib permite una gestión más eficiente de la memoria en RAM al evitar cargar grandes conjuntos de datos completamente en memoria. Sin embargo, es importante tener en cuenta que la lectura y escritura de archivos .csv desde y hacia el disco pueden tener un impacto en el rendimiento en términos de velocidad de acceso al disco."

3.0.3. Codigo fuente.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

```python
import random
import time
# graph libs
import pandas as pd
import matplotlib.pyplot as plt
# args
import argparse
import math

def main(args):

    while True:

        k_length = input("Pls give me the 'k' for the alphabet or just press 'enter': ")
        if k_length == '':
            k_length = random.randint(0, 1000)
        else:
            k_length = int(k_length)

        print(f"Your k is: {k_length}")


        # OUTPUT FILES
        output = open('Outputf_BinaryStrings.txt', 'w', encoding='utf-8')



        # BASIC INFO
        output.write("{0, 1}")
        output.write(f"^{k_length} = ")
        output.write("{\u03B5, ")


        # LOGIC
        number_zeros_left = 0

        for i in range(1, k_length + 1):
            number_zeros_left += 1
            bits = 2**i
            for j in range(bits):
                if i == k_length and j == bits - 1:
                    output.write(bin(j)[2::].zfill(number_zeros_left))
                else:
                    output.write(bin(j)[2::].zfill(number_zeros_left)+ ', ')

        output.write('}')
```

```

        bits = 2**k_length
        # CLOSE FILES
        output.close()

        # """
        # k =3

        # output :  {e, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110,
           111}

        # T: O(2^k + 2^k-1 + 2^k-2 + ... + 2^1 + 2^0) = O(2^k)
        # O: O(1)   only output.txt

        # """


        # # PLOT
        # '''
        # Data frame:
        # # k = 3
        # # bits = 2**3
        # bits = 8
        # chain          number_of_1s
        # 1                0          -> epsilon
        # 2                0          -> '000'
        # 3                1          -> '001'
        # 4                1          -> '010'
        # 5                2          -> '011'
        # 6                1          -> '100'
        # 7                2          -> '101'
        # 8                2          -> '110'
        # 9                3          -> '111'

        # '''
        # OUTPUT FILES
        df = open('Binary_Strings.txt', 'w')
        blog = open('Binary_Strings_Log.txt', 'w')

        # BASIC INFO
        df.write('chain' + ',' + 'number_of_1s\n')
        df.write('1' + ','+ '0 \n')

        blog.write('chain' + ',' + 'number_of_1s_baselog10\n')



```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

```python
        for i in range(2, bits + 2):
            df.write(str(i) + ',' + str(hammingWeight(i - 2)) + '\n')
            if hammingWeight(i - 2) != 0:
                blog.write(str(i) + ',' + str(round(math.log10(hammingWeight(i - 2)), 4)) + '\n')

        # CLOSE FILES
        df.close()
        blog.close()

        # SAVE DATA FRAME
        df = pd.read_csv('Binary_Strings.txt')

        # PLotting
        plt.plot(df['chain'], df['number_of_1s'], 'bo')
        plt.title('Binary Strings')
        plt.xlabel('Chain')
        plt.ylabel('Number of 1s')
        plt.show()

        # SAVE DATA FRAME
        blog = pd.read_csv('Binary_Strings_Log.txt')

        # PLotting
        plt.plot(blog['chain'], blog['number_of_1s_baselog10'], 'bo')
        plt.title('Binary Strings base Log10')
        plt.xlabel('Chain')
        plt.ylabel('Number of 1s')
        plt.show()

        if input("Do you want to continue? (y/n): ") == 'n':
            break
        print("-"*60)


def hammingWeight(n):
    """
    :type n: int
    :rtype: int
    """
    mask = 1
    count = 0
    while n != 0:
        if (mask & n == 1):
            count = count + 1
        n = n>>1

```

```python
    return count



def parse_args():
    # setup arg parser
    parser = argparse.ArgumentParser()

    # random default value
    default_k_length = random.randint(0, 1000)


    # add arguments
    parser.add_argument("k_length",
                        type=int, help="number of primes numbers to check",
                        default=default_k_length, nargs='?')
    # parse args
    args = parser.parse_args()

    # return args
    return args

# run script
if __name__ == "__main__":
    # add space in logs
    print("\n\n")
    print("*" * 60)
    start = time.time()

    # parse args
    args = parse_args()

    # run main function
    main(args)

    end = time.time()
    print("\n\n")
    print("Total time taken: {}s (Wall time)".format(end - start))
    # print("max number of K length: {}".format(args.k_length))
    # add space in logs
    print("*" * 60)
    print("\n\n")
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

3.0.4. Casos de prueba.

Sin nigun Input.

- **Output en archivo .txt:**

$$\{0,1\}^{11} = \{\epsilon, 0, 1, 00, 01, 10, ..., 11111111011, 11111111100, 11111111101, 11111111110, 11111111111\}$$

- **Output en archivo .csv:**

```
chain,number_of_1s
1,0
2,0
3,1
4,1
5,2
6,1
7,2
8,2
9,3
10,1
11,2
12,2
...
2043,9
2044,9
2045,10
2046,9
2047,10
2048,10
2049,11
```

- **Output en archivo .csv en base log10:**

```
chain,number_of_1s_baselog10
3,0.0
4,0.0
5,0.3010299956639812
6,0.0
7,0.3010299956639812
8,0.3010299956639812
9,0.47712125471966244
10,0.0
...
2043,0.9542425094393249
2044,0.9542425094393249
2045,1.0
2046,0.9542425094393249
```

```
2047,1.0
2048,1.0
2049,1.0413926851582251
```

- **Mensaje en la terminal:**

```
C:\Users\CristoRey\Escom\SEM_5>py Binary_Strings.py
************************************************************
Pls give me the 'k' for the alphabet or just press 'enter':
Your k is: 11
Do you want to continue? (y/n): n



Total time taken: 16.247236251831055s (Wall time)
************************************************************
```

- **Gráfica:**

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

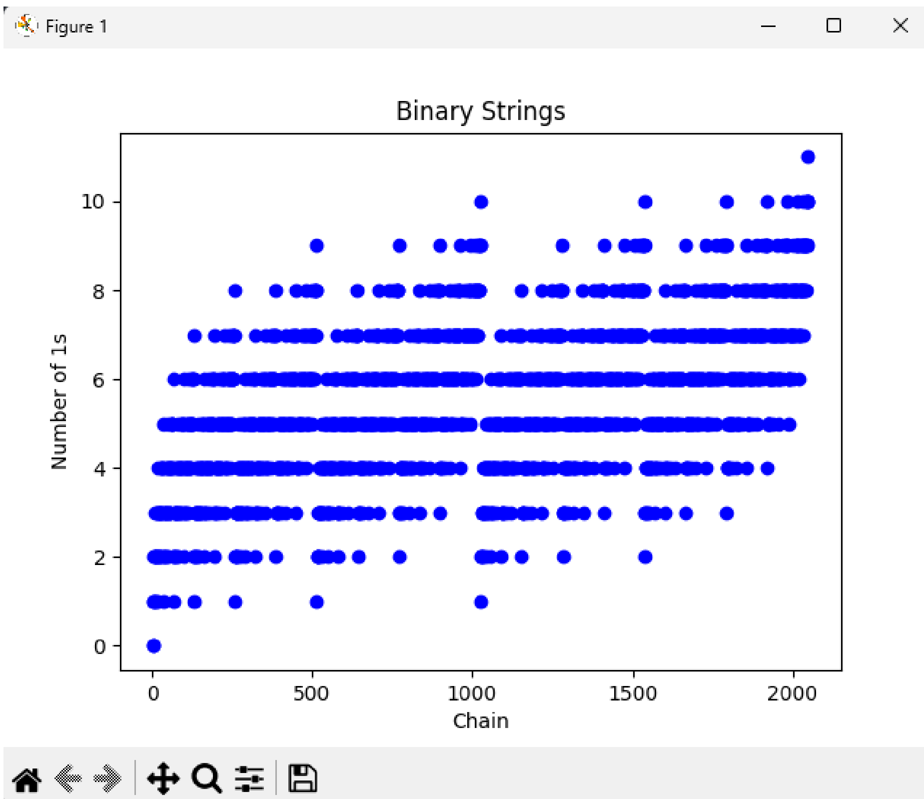

Figura 3.1: Gráfica de puntos con K = random(11).

- **Gráfica base log10:**

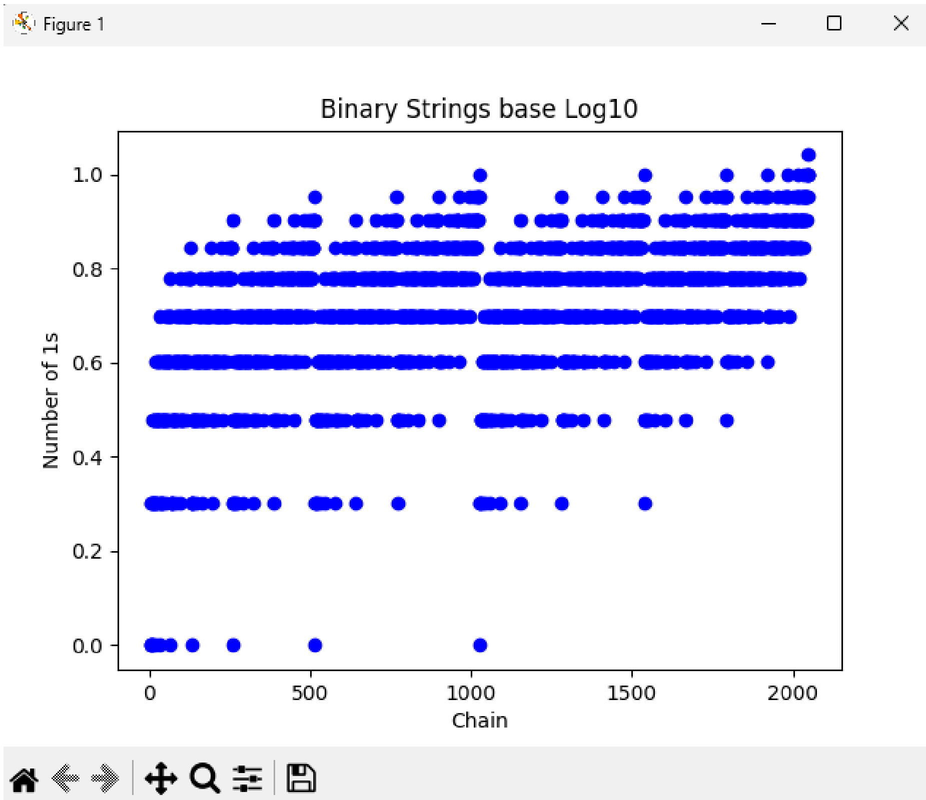

Figura 3.2: Gráfica de puntos con K = 11 y log10.

Tamaño de la cadena igual a 3

- **Output en archivo .txt:**

$$\{0,1\}^3 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111\}$$

- **Output en archivo .csv:**

```
chain,number_of_1s
1,0
2,0
3,1
4,1
5,2
6,1
7,2
8,2
9,3
```


Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Output en archivo .csv en base log10:**

```
chain,number_of_1s
chain,number_of_1s_baselog10
3,0.0
4,0.0
5,0.3010299956639812
6,0.0
7,0.3010299956639812
8,0.3010299956639812
9,0.47712125471966244
```

- **Mensaje en la terminal:**

```
C:\Users\CristoRey\Escom\SEM_5>py Binary_Strings.py
************************************************************
Pls give me the 'k' for the alphabet or just press 'enter': 3
Your k is: 3
Do you want to continue? (y/n): n



Total time taken: 9.055204629898071s (Wall time)
************************************************************
```

- **Gráfica:**

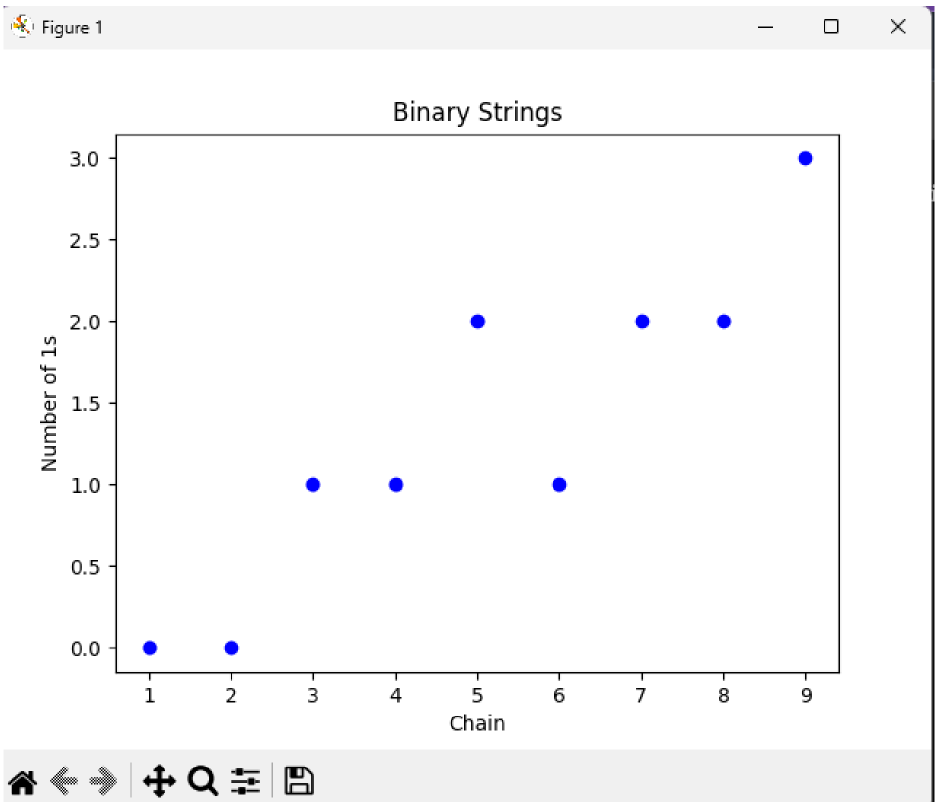

Figura 3.3: Gráfica de puntos con K = 3.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

- **Gráfica base log10:**

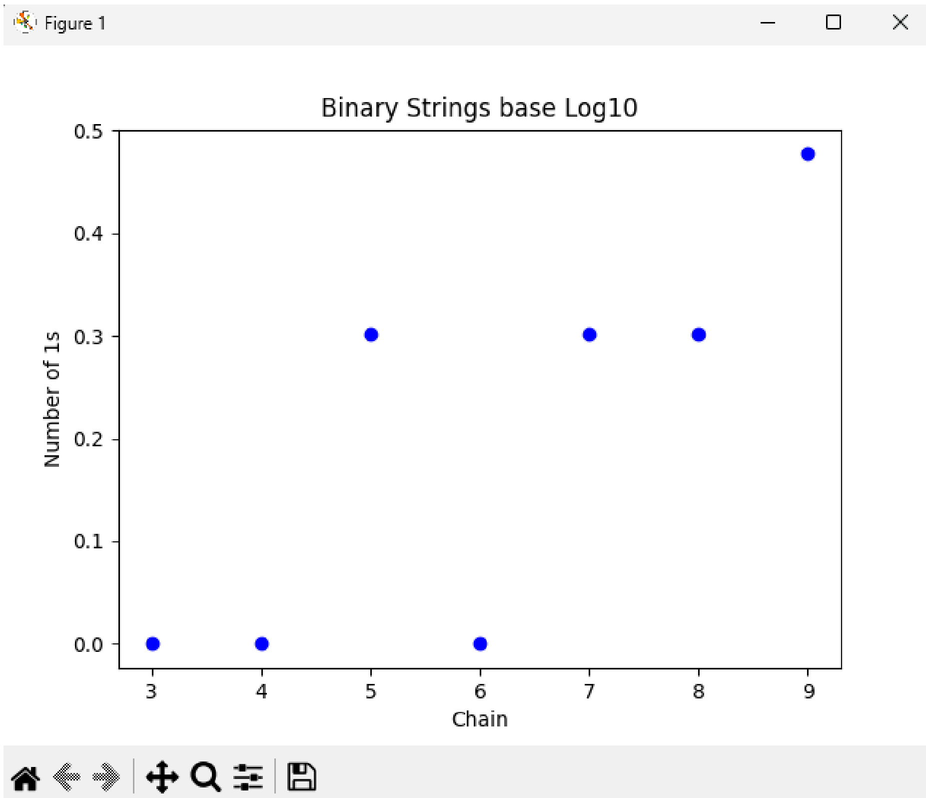

Figura 3.4: Gráfica de puntos con K = 3 y log10.

Tamaño de la cadena igual a 28

- **Output en archivo .txt:**

  Usando un valor de K igual a 28, se crea un archivo .txt con un tamaño superior a los 15 GB, lo cual no es posible abrir en un editor de código convencional.

- **Mensaje en la terminal:**

```
C:\Users\CristoRey\Escom\SEM_5>py Binary_Strings.py
************************************************************
Pls give me the 'k' for the alphabet or just press 'enter': 28
Your k is: 28
Do you want to continue? (y/n): n



Total time taken: 2203.679901599884s (Wall time)
************************************************************
```

- **Gráfica:**

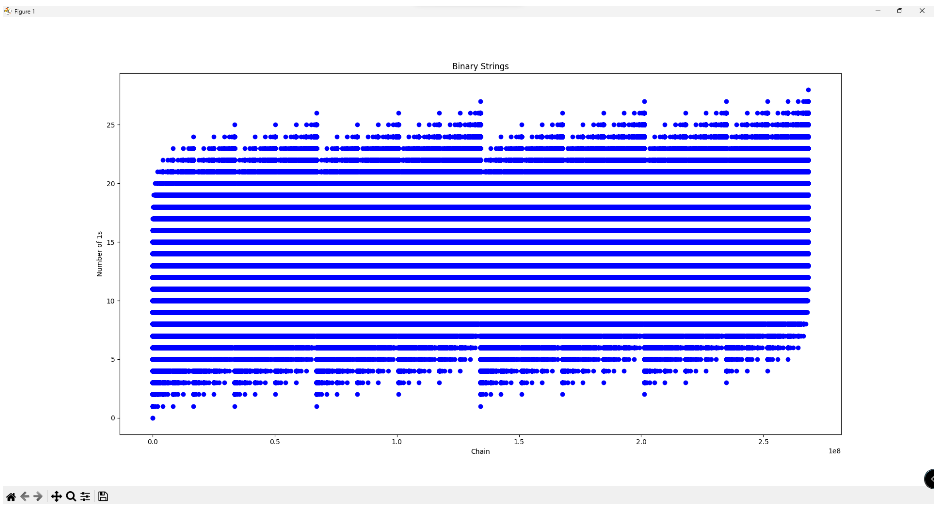

Figura 3.5: Gráfica de puntos con K = 28.

- **Gráfica base log10:**

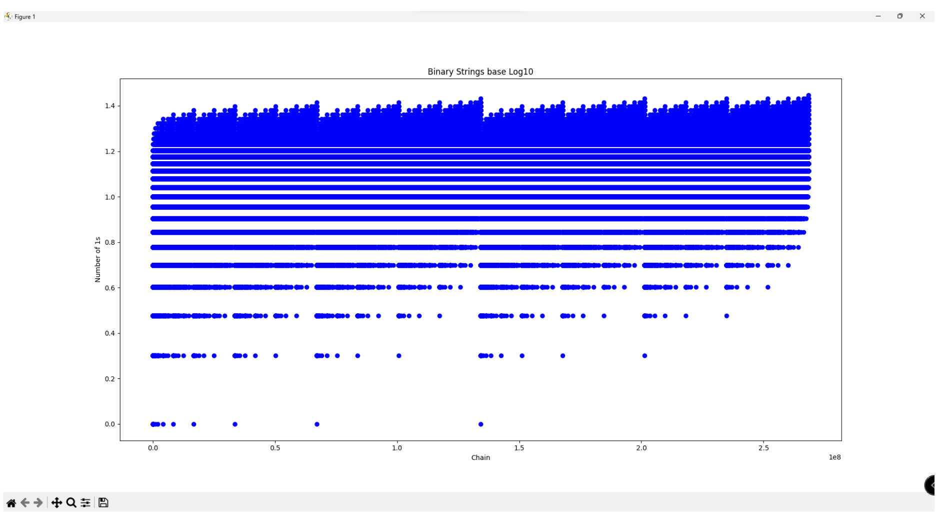

Figura 3.6: Gráfica de puntos con K = 28 y log10.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 4

Conclusión.

La creación de un diccionario que comprende todas las posibles combinaciones de longitud 'k' en el sistema binario nos ha proporcionado una visión clara y exhaustiva de las secuencias binarias, desde las más simples hasta las más complejas, y nos ha permitido comprender cómo se generan y representan las palabras en el lenguaje binario.

Dentro de la programación del problema y su complejidad, nos dimos cuenta de que para 'k' igual a 28, el número de combinaciones totales crece de manera gigantesca en cuanto al almacenamiento en el disco. Aunque utilizamos herramientas para manejar una gran cantidad de datos y algoritmos para reducir la complejidad temporal, podemos decir que, aun así, es bastante tardado visualizar una gráfica. Además, observando la gráfica, es demasiado complejo poder visualizar una función con el número de 1's de las cadenas y de igual forma con el algoritmo. Del mismo modo, si usamos una 'k' más baja, nos resulta igual de complicado poder visualizar una función dentro de la gráfica. Posiblemente, utilizando algoritmos enfocados en encontrar el comportamiento de la gráfica, podremos descubrir su naturaleza.

Av. Juan de Dios Bátiz esq. Miguel Othón de Mendizabal S/N Col. Lindavista, GAM, CDMX ✉ sanchez.martinez.felipe18@gmail.com

CAPÍTULO 5

Referencias.

- Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson.